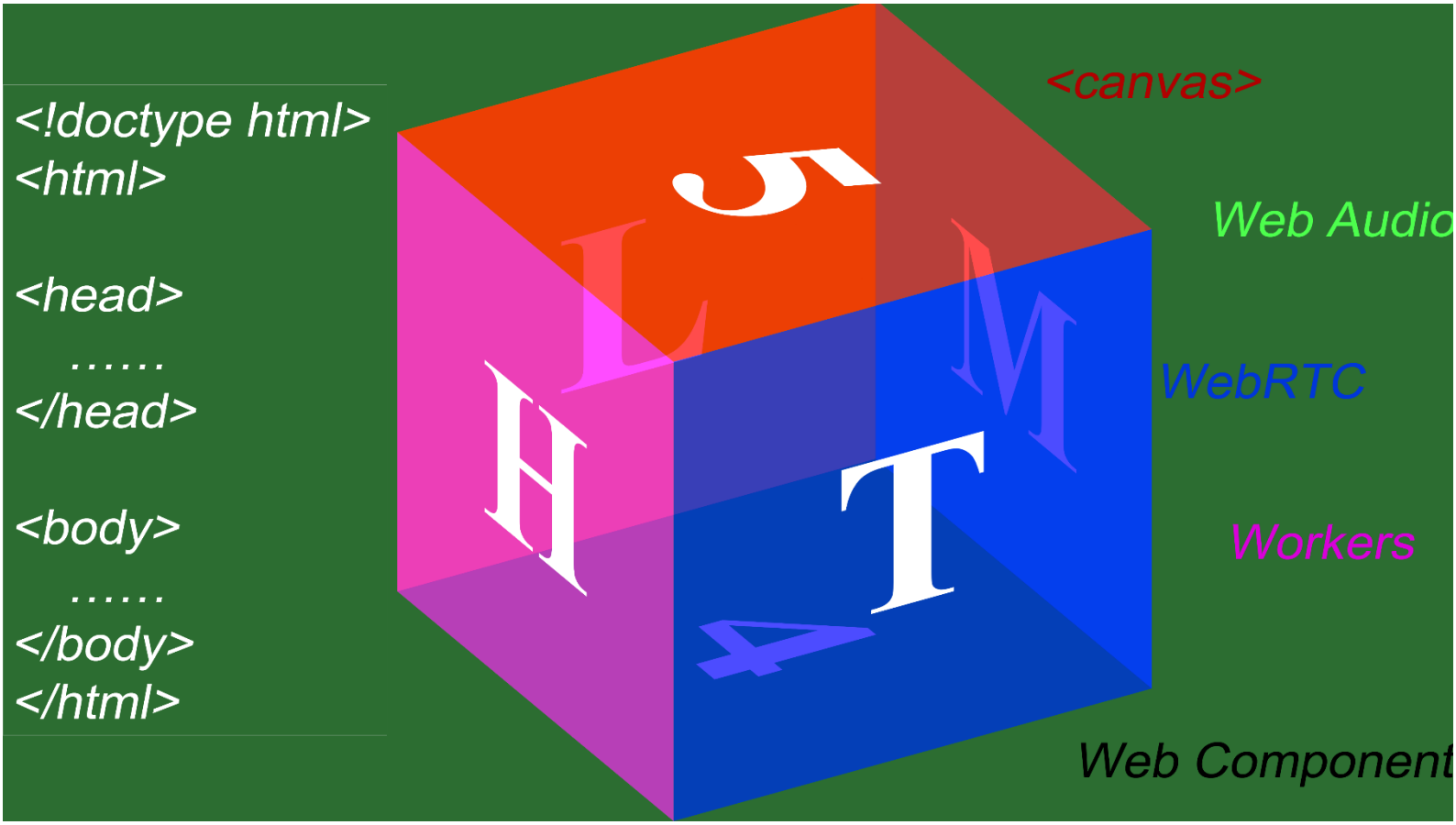
HTML5&CSS3 开发

大起底



作者：亚伯拉罕

目录

前言 **1**

第 1 章 概况 **2**

1.1 入行建站 3

1.2 建站工具 4

1.2.1 浏览器 4

1.2.2 代码编辑器 7

1.2.3 UI 设计工具 8

1.2.4 音视频剪辑工具 8

1.2.5 调试工具 9

1.2.6 传输工具 13

1.2.7 远程登录工具 15

1.3 建站步骤 16

1.3.1 第一步：需求分析 16

1.3.2 第二步：开发网站 16

1.3.3 第三步：注册域名 16

1.3.4 第四步：申请网站空间 18

1.3.5 第五步：上传网页文件 19

1.3.6 第六步：给域名做 DNS 解析 19

1.3.7 第七步：网站推广与运营 20

1.4 搭建网站运行平台 20

1.4.1 开发环境 20

1.4.2 生产环境 25

第 2 章 网站开发基础 **29**

2.1 屏幕与长度单位 30

2.2 浏览网页的过程 37

2.3 网站三层架构 40

2.4 前端三层分离 42

2.5 DOM 模型树 46

第 3 章 网页行为：JAVASCRIPT 基础 **48**

3.1 JavaScript 运行时 49
3.2 JavaScript 基本用法 50
3.2.1 几个常用方法 51
3.2.2 **获取元素对象** 54
3.2.3 操纵网页样式 57
3.2.4 操纵网页结构 57
3.2.5 操纵 COOKIE 60
3.3 页面事件 61
3.4 定时器 65
3.5 Ajax 编程 67
第 4 章 网页结构：HTML5 72
4.1 元素简介 73
4.2 页面结构元素 77
4.3 文档元数据 80
4.4 脚本元素 86
4.5 章节编排 89
4.6 组织内容 94
4.7 文字形式 99
4.8 编辑元素 108
4.9 表单元素 109
4.9.1 <datalist>/<option>/<label> 109
4.9.2 <form>元素 111
4.9.3 <input>元素 115
4.9.4 按钮 129
4.9.5 多行编辑 130
4.9.6 可视化计量 132
4.9.7 进度条 133
4.9.8 下拉框 134
4.9.9 输出结果 135
4.9.10 控件分组 136
4.9.11 综合例子 137

4.10 嵌入内容 139
4.10.1 自适应图片 140
4.10.2 创建图片热点 142
4.10.3 多媒体元素 144
4.10.4 canvas 画布元素 158
4.10.6 嵌入页面 207
4.10.7 SVG 矢量图 209
4.10.8 嵌入外部资源 283
4.11 表格元素 284
4.12 互动元素 290
第 5 章 网页样式：CSS3 295
5.1 CSS3 简介 296
5.2 选择器 297
5.3 样式属性 298
5.4 文字内容 299
5.4.1 字体 299
5.4.2 文本 302
5.4.3 装饰 311
5.5 元素定位 317
5.6 元素尺寸 330
5.6.1 元素的宽和高 330
5.6.2 元素的外边距 333
5.6.3 元素的内边距 334
5.7 元素外观 334
5.7.1 边框 334
5.7.2 背景 348
5.7.3 元素轮廓 360
5.8 列表 361
5.9 表格 362
5.10 网页布局 363
5.10.1 多列布局 363

5.10.2 弹性布局 365
5.10.3 网格布局 378
5.10.4 布局实战：结合 grid 和 flex 401
5.11 变换 405
5.11.1 2D 变换 405
5.11.2 3D 变换 414
5.12 过渡 423
5.13 动画 426
5.14 滚动条 428
5.16 渐变 431
5.16.1 线性浅变 432
5.16.2 径向浅变 437
5.16.3 圆锥渐变 442
5.17 伪元素 447
第 6 章 HTML5 高级应用 456
6.1 拖放 457
6.2 系统剪贴板 464
6.2 地理定位 467
6.3 本地存储 470
6.4 全双工通信 472
6.5 服务器发送事件 475
6.6 独立线程 477
6.7 服务线程 483
6.7.1 ServiceWorkerContainer 485
6.7.2 ServiceWorkerRegistration 487
6.7.3 ServiceWorker 490
6.7.4 ServiceWorkerGlobalScope 491
6.7.5 应用场景 498
6.8 INDEXEDDB 504
6.8.1 概述 505
6.8.2 操作 506

6.9 媒体流 517
6.9.1 MediaDevices 518
6.9.2 MediaStream 521
6.9.3 MediaStreamTrack 522
6.9.4 MediaRecorder 525
6.10 Web 音频 API 527
6.10.1 乐理知识 527
6.10.2 AudioContext 532
6.10.3 AudioNode 534
6.10.4 播放音乐文件 536
6.10.5 控制音量 538
6.10.6 拆分声道 540
6.10.7 音频可视化 541
6.10.8 滤波 544
6.10.9 卷积混响 547
6.10.10 3D 环绕 548
6.10.11 声音扭曲 552
6.11 WebRTC 555
6.11.1 联网方式 555
6.11.2 搭建实验环境 565
6.11.3 搭建 STUN/TURN 567
6.11.4 实现步骤 571
6.12 Web 组件 581
6.12.1 定制元素 581
6.12.2 影子 DOM 588
6.12.3 HTML 模板 593
6.13 Web Crypto API 597
6.13.1 产生摘要 597
6.13.2 管理密钥 598
6.13.3 使用密钥 605

前言

我从事 IT 工作快 30 年了，IT 系统的基础设施层、平台软件层、应用软件层和数据层的方方面面我都涉足过，我算是一个“软/硬”兼施的 IT 全栈工程师啦。现在在大学教计算机课程——Linux 操作系统、云计算和 HTML5 网站开发的课。由于一直没有使我满意的 HTML5 教材，中国大陆出版的书几乎都是抄来抄去的，或者是一些生硬的翻译版本，或者是没有教学经验的纯技术人员写的，有然没有所以然，或者是知识覆盖面狭窄。时至今日，我们都知道，HTML5+CSS3+JavaScript 几乎是一统江湖，可以预计，假以时日，应用都将被移植成网页形式并以 SaaS 模式向外销售，谷歌文档、微软 office 165、Adobe 等公司算是先行者吧。之所以本书取名《HTML5&CSS3 开发大起底》，是因为书中除了覆盖从网站开发到运营全过程外，横向上几乎涉及了全部的知识点——基本元素、css3 样式、画布、矢量图、语义元素、本地存储、多媒体处理、点对点通讯、线程、Web 组件等等。我有很多年的企业实战经验，加上十几年的大学教学总结，计算机理论知识扎实，喜好哲学，由我来完成这么一本大部头是再好不过的了。

本书具备相当的深度，建议你静下心来仔细阅读和思考，相信会使你受益良多。向主保证，书稿全是我一人写就，不涉及任何其他人的版权问题，差不多一年前电子工业出版社有意向我约稿，但是等我交稿时出版社反馈这方面的书销售呈下降趋势，因此不便出版了。为了不浪费自己一年来的心血和汗水，除了简体版外，我还翻译成繁体中文和英文版，这样我就可以在亚马逊上出版我的电子版书籍了。

感谢主，感谢我的家人、朋友和我的同事，我对你们的爱一如既往。

需要随书配套资源的同仁请联系我索取，我的邮件是 183407563@qq.com，或通过微信联系。来信时请注明是简体中文版、繁体中文版还是英文版。作者的网站 https://www.veryopen.org。



我的微信：moodisk



我的公众号：农妇3拳

第 1 章

概况

假如你从网站开发专业毕业了，到了一家不大的公司上班，第一天老板就交给你一个任务说：“小王啊，你刚好学网站开发，你就为我们公司开发一个门户网站，让客户能及时了解公司的业务和产品信息。网站开发和运营成本你不用担心，我会全力支持的。网站内容你向田总要。”

得了，你不但要开发网站，还要让网站跑起来，而且还要让世人通过域名访问网站，还有网站必须稳定可靠，杜绝 404！还有安全保障呢，不要被黑客黑了你的网站。你是不是头都大了？“以前在学校没这么玩过啊！”你挠着后脑勺不知所措。或者你压根没有想到这些，就开始编码了，因为学校的老师就是这么教的！

你能看到这里说明你是幸运的，因为你正在读我写的《HTML5&CSS3 开发大起底》。我从事 IT 工作二十多年了，搞过电脑硬件组装与维修，建设过电脑机房，写过电脑病毒免疫程序，开发过网络服务软件，做过 Linux 内核裁剪，从事过数据库管理工作，开发了众多的电脑程序，五年前涉足网站开发。时至今日，我算是一个“软/硬”兼施的网页全栈工程师了。好了，我已经把建站、跑站、运维都写在本书里了，看完本书，我可以负责任地告诉你：“小王，你老板交给你的任务根本不是一个事儿！”

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

1.1 入行建站

开发网站是容易的，因为用记事本随便写一点什么内容，你就可以在浏览器中浏览了，参见图 1.1；也可以在 vscode 里面自动产生网页框架，然后再添加些文字内容或按钮什么的，最后就能浏览一个正儿八经的页面了，具体参考“§1.2.2 代码编辑器”。但是这些页面又有什么用呢？所以说开发一个有用的网站又是最难的 IT 工作之一，为了开发网站前端，你得掌握 HTML5 元素、CSS3 样式和 JavaScript 语言，为了开发网站后端，你得懂 php 或 java 或 Python 语言或其他一门动态语言，为了存储网站数据，你还得学会至少一个数据库管理系统（如 MariaDB），为了让网站跑起来以便人类和非人类（如网络爬虫）访问你的网站，你还得掌握 Web 服务器和 Linux 操作系统（生产环境中的服务器就别跟我提 windows 了）以及申请域名，为了运营你的网站，你必须懂得如何推广和运维网站。当然你如果想成为一个网页全栈开发工程师，那就更难了，你须掌握的技术可以列出一个长长的清单。

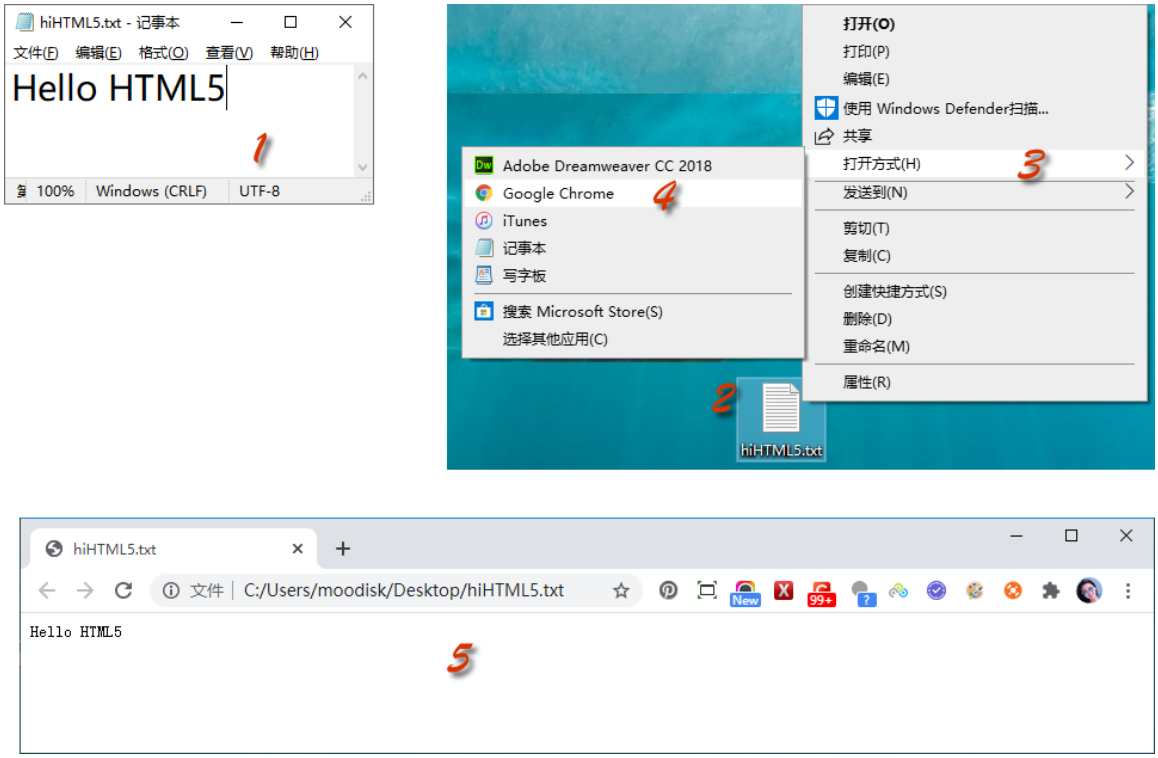


图 1.1 简单页面及浏览

啥？这下你就怕了要改行了？没出息，我告诉你，一旦你掌握了开发网站的整套技术，那就大有用武之地了，就业门路非常宽广。拜 HTML5 和 CSS3 所赐，原来在电脑本地运行的程序，现在都可以用网页实现，包括文字处理、图像处理、游戏、3D、动画、工程软件等，比如微软 office 365、谷歌文档排版、Adobe 全家桶、photopea 图像处理等，而且越来越多的人开始采用 HTML5 开发移动 app 了。flash 已寿终正寝， HTML5+CSS3+JavaScript 将一统天下！

做网站开发除了涉及众多的技术外，你还得照顾人类和各种非人类的感受——阅读代码的人、浏览网站的人、网站运维的人、渲染页面的浏览器、搜索引擎、网络爬虫、读屏器、社交媒体

应用程序、RSS 阅读器以及各种非人类后端程序等等，ta 们都是你的网站的利益相关者。你在写每一行代码时，你不能忽视 ta 们的感受！

HTML5 也是拼了，为了照顾各种非人类的感受，居然增加了很多语义元素，所以同学呀，你也要尽量使用这些语义元素。例如图 1.2 左侧的代码没有照顾到各种非人类的感受，为何？请在本书中寻找答案。

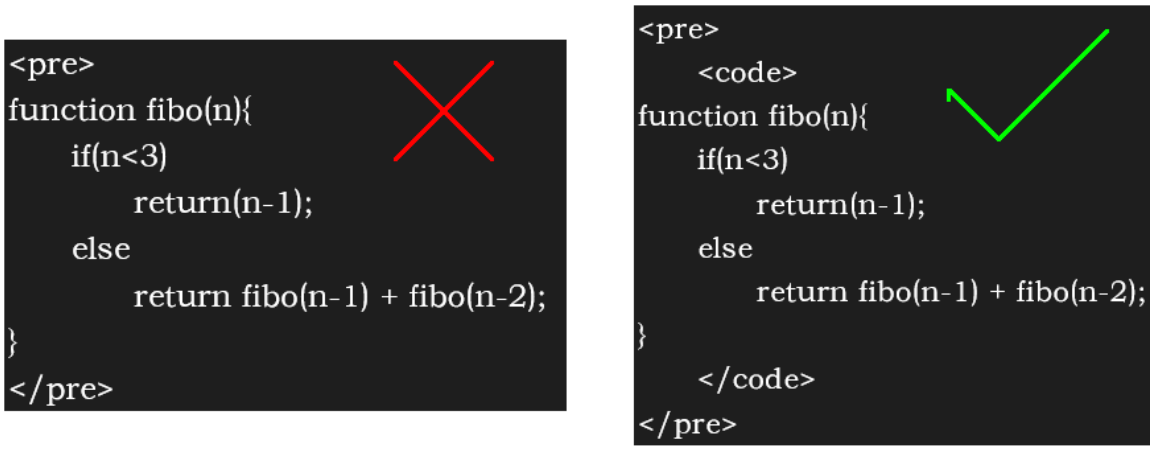


图 1.2 尽量使用语义元素

1.2 建站工具

开发和运营一个网站用到的工具主要有代码编辑器、图片处理工具、音视频剪辑工具、调试工具、上传/下载工具、远程登录工具以及网站运行平台等。

1.2.1 浏览器

其实不光开发网站需要浏览器，我们平时上网都得用到浏览器，浏览器算是日常生活中用得最多的计算机程序了，所以选择一款好用的浏览器很重要。目前最流行的几款浏览器有谷歌的 chrome、mozilla 公司的 firefox、Opera 公司的 opera、微软的 Edge 和苹果的 Safari。前四个浏览器安装后在电脑桌面上的图标见图 1.3。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

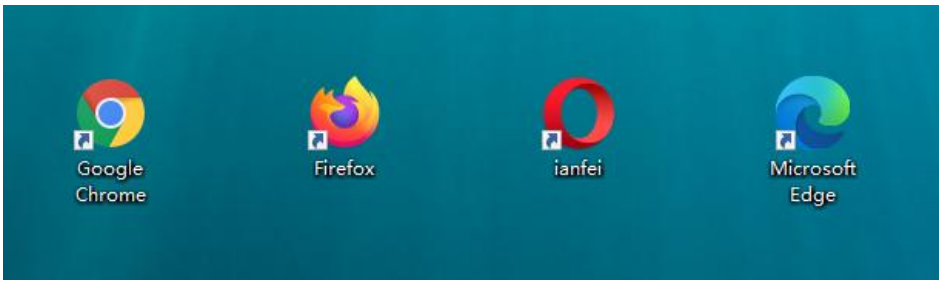


图 1.3 windows 上的四大浏览器图标

启动浏览器后的界面见图 1.4 所示。







图 1.4 windows 上四大浏览器界面

Opera 浏览器有几个额外的特色，它的左侧边栏竟然整合了几个目前流行的即时通讯工具，同时集成了免费的 VPN(不过需要从 https://ftp.opera.com/pub/opera/desktop/下载独立的安装包)。

不过我还是最热衷于谷歌的 chrome 浏览器，其次是火狐浏览器（firefox），这两个浏览器都有很多快捷键,比如按 Ctrl+t 打开一个新标签页,Ctrl+Shift+t 重新打开刚刚关闭的标签页,Ctrl+Tab 切换标签页，F11 全屏模式，F12 打开开发者工具窗口等等。由于 firefox 是公开源代码的，所以在乎安全的朋友更喜欢采用 firefox 浏览器，大名鼎鼎的暗网浏览器（Tor Browser）就是定制 firefox 而来的。

微软的 Edge 浏览器不光换了马甲，而且内核都换成了谷歌公司开源的 Chromium（Chromium 采用了页面渲染内核 Blink，而 Blink 又集成了 JS 引擎 V8）。这样网站开发人员的噩梦就快结束

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

了，大家应该还记得，以前为了提高用户体验，要写一大堆的代码来判断用户使用哪家公司的浏览器，然后根据不同的浏览器编写不同的代码，现在回想起来都有点怕怕。

目前市面上存在四大浏览器内核，分别是 Blink 内核（被 Chome, Edge 和国内大部分浏览器使用）、Gecko 内核（被 Firefox 使用）、Webkit 内容（被苹果 Safari 使用）和 Presto 内核（被 Opera 使用），可以预计 Blink 内核的市场份额会进一步扩大。

1.2.2 代码编辑器

如果你非常熟悉一款编辑器，那么你就继续用它吧，否则强烈建议你使用 Visual Studio Code，简称为 vscode，这是微软公司推出的功能强大的开源编辑器，瞧这界面，都是小时候熟悉的味道。其他的比较流行的编辑器还有 eclipse、hbuilder、notepad++、webstorm、sublime、vim 等。

从 https://code.visualstudio.com 下载最新版的 vscode 并安装，然后安装这些扩展插件：Chinese (Simplified) Language Pack【简体中文汉化包】、HTML CSS Support、IntelliSense for CSS class、PHP IntelliSense、Debugger for Chrome【与 chrome 联动调试】、ESLint【JavaScript 代码排错】、Beautify【代码美化】、JShint。在图 1.5 所示的输入框中搜索需要安装的插件名称，然后点击“install“安装。



图 1.5 在 vscode 中管理扩展插件

在 vscode 编辑器中快速产生 HTML5 网页文件框架的方法：按 Ctrl+n 新建一个文件，然后按 Ctrl+s 保存为以 html 为扩展名的文件，比如 index.html，最后在编辑窗口中输入!再回车即可，注意是半角感叹号，参考图 1.6。

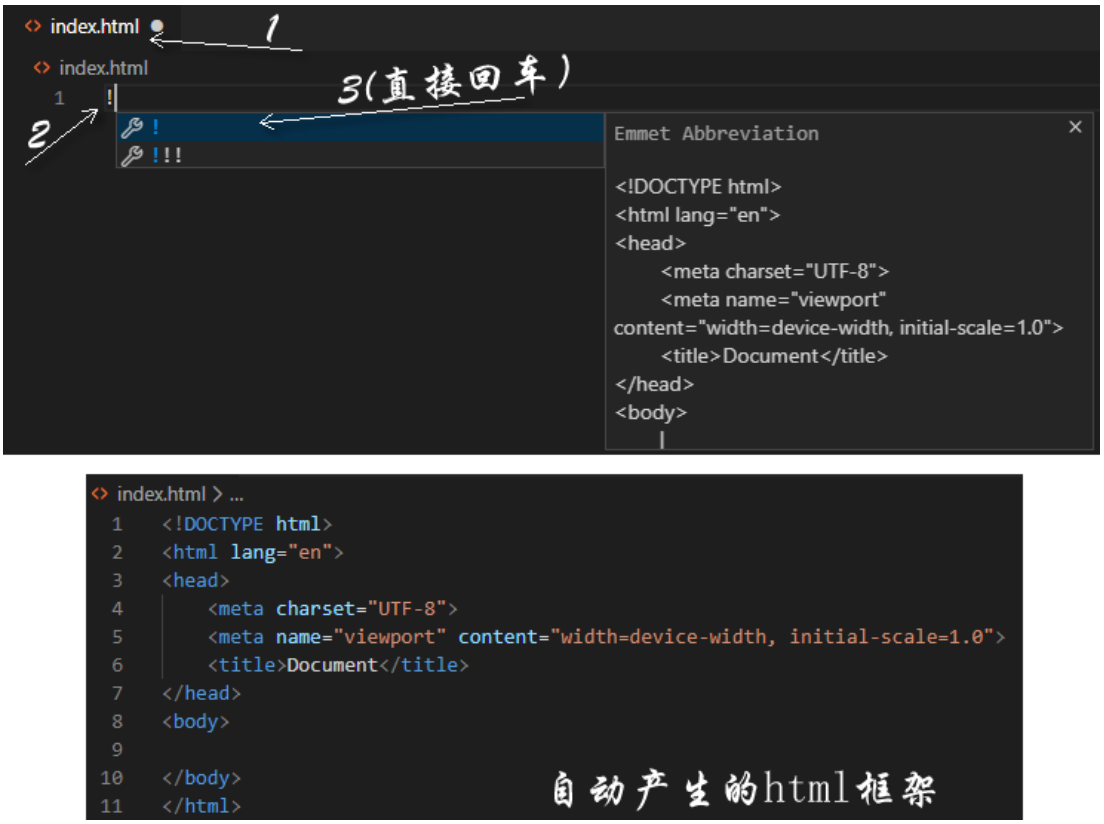


图 1.6 vscode 快速生成页面框架

1.2.3 UI 设计工具

网站 UI 包括页面美工和交互两部分，早期常用图形工具绘制页面和处理各种图片，但是很难完成交互设计。后来人们采用 Axure 原型设计工具完成网站的 UI 设计，这是个绝佳的工具，建议大家学习掌握它。另外在线 UI 设计（https://www.figma.com）能多人协同，效果非常不错。页面美工可采用 Adobe Fireworks 和 Adobe Photoshop，Fireworks 算是 Photoshop 的简化版，容易掌握，完全可以胜任图片的日常编辑工作。如果你已经熟悉 Photoshop，那么就继续用它吧，否则建议学习 Fireworks，毕竟 Photoshop 比 Fireworks 强大得多！

1.2.4 音视频剪辑工具

还是 adobe 公司的 Adobe Premiere 和 Adobe Audition，没办法，adobe 公司几乎垄断了文字/图片加工、音/视后期处理等工具软件市场，而且还有一个不好的消息我要跟你说：Adobe 竟然采用 HTML5 技术云化了公司的全家桶，得付租金在线使用！已经没有能在本地安装的新版本了。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

1.2.5 调试工具

最简单的方法是直接使用浏览器 chrome 或者 firefox 的开发者工具（启动 chrome 网页浏览器，然后按 Ctrl+Shift+i 进入开发者工具，或者参考图 1.7 通过菜单进入）。

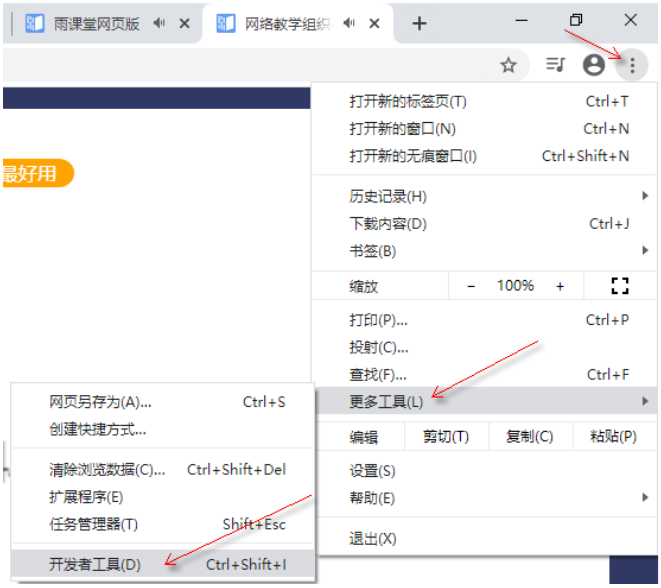


图 1.7 通过菜单打卡法这这工具

开发者工具可以修改 css 代码并看到实时效果，可以编写和调试 JavaScript 语句，可以查看资源请求状况等等，具体使用请参考网上有关文章，这里不再赘述。一个好用的附加功能之一就是截屏整个页面，进入开发者工具后按 Ctrl+Shift+p 打开命令面板，然后输入 Shot 定位到 Capture full size screenshot 并执行它即可，参见图 1.8 所示。

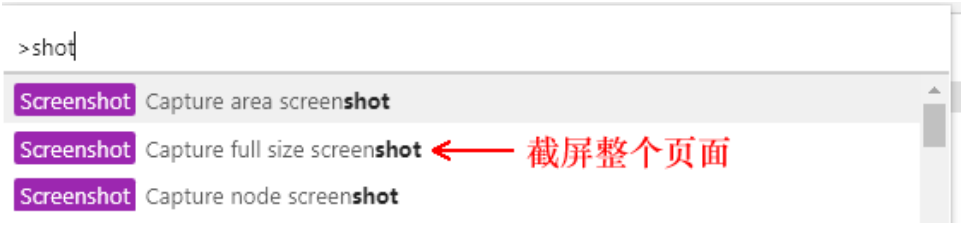


图 1.8 截屏整个页面

另一个好用的功能是模拟各种尺寸的屏幕，进入开发者工具后按 Ctrl+Shift+m 或者点击 图标即可。进入 fixfore 的开发者工具的控制台，这是可以直接输入 js 代码，输入一行回车就执行一行，按 Ctrl+b 快捷键可以在多行和单行输入模式中切换，参见下图：



chrome 浏览器中输入一行代码直接按 Shift+Enter 换行，即可输入多行，最后按 Enter 执行。

高级的调试方法是 chrome 浏览器与 vscode 代码编辑器联合调试，与 vscode 联动调试的大致的步骤如下：

1. 参考图 1.9 在 vscode 中安装 Debugger for Chrome 扩展插件。

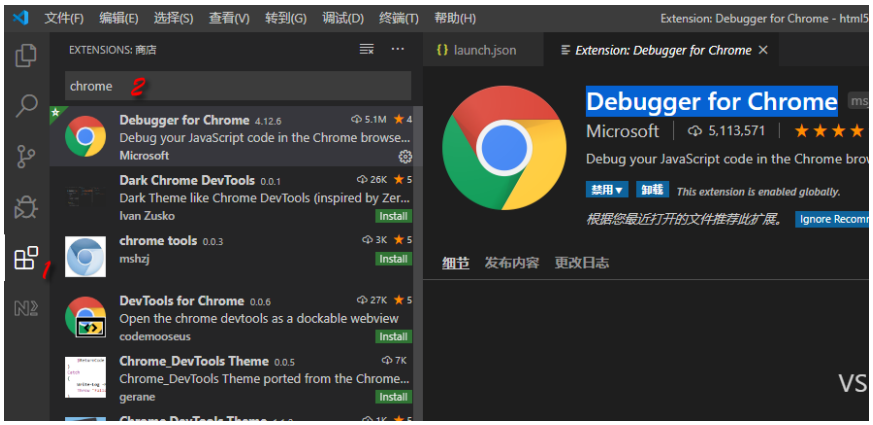


图 1.9 安装调试插件

2. 在 vscode 中打开网页所在的目录。

菜单：文件》打开文件夹，定位到网页文件所在的目录，这样就把目录下的全部文件都导入到 vscode 中了。

3. 参考图 1.10，开启“允许在任何文件中设置断点”。

菜单：文件》首选项》设置》功能》调试，勾选“允许在任何文件中设置断点”。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

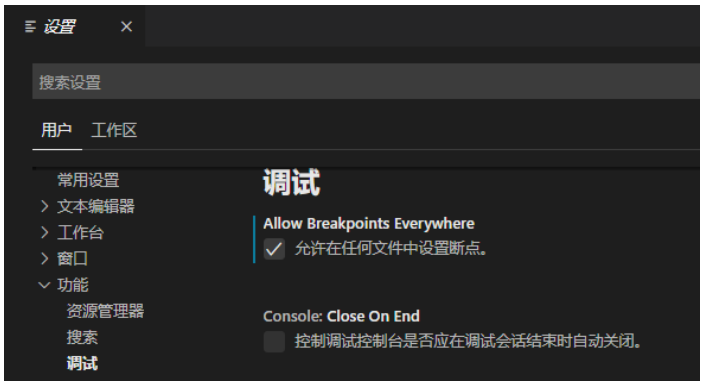


图 1.10 允许在任何文件中设置断点

这样 html 文件中的 JS 代码也可以设置断点了。

4. 参考图 1.11，创建启动配置文件 launch.JSon

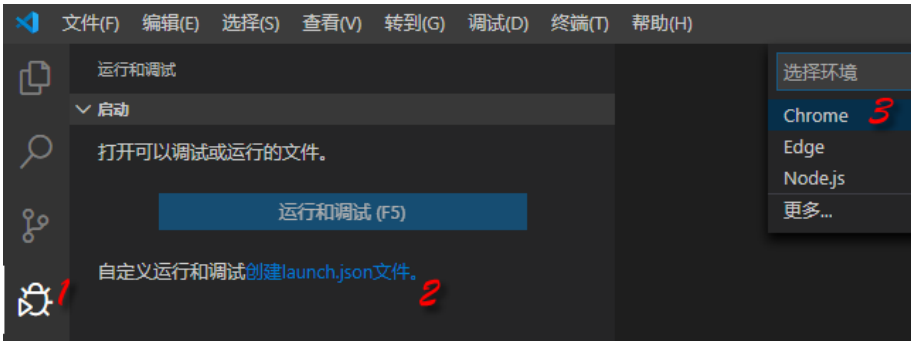


图 1.11 创建启动配置文件

然后参考图 1.12，输入配置信息，并保存：

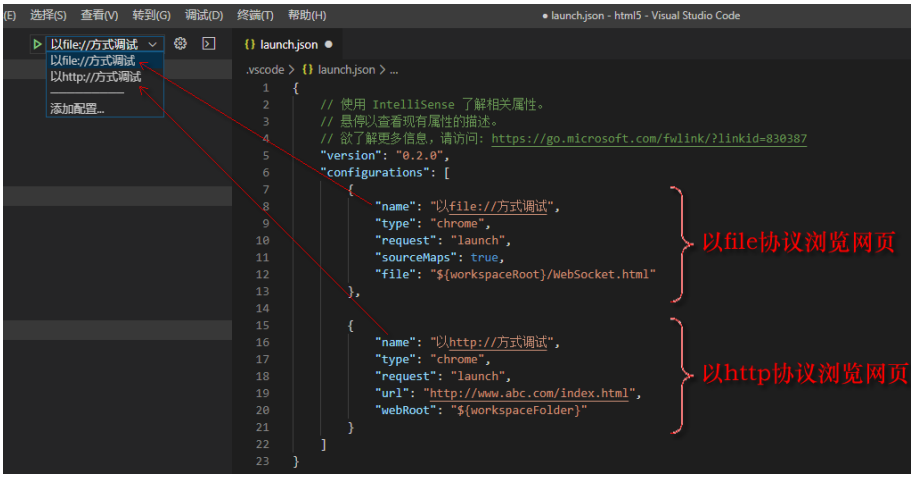


图 1.12 输入启动配置信息

这样我们既可以通过 file://协议调试网页文件，也可以通过 http://协议调试。调试配置文件放在网页文件所在目录的.vscode 子目录下，比如你的网页文件所在的目录是 D:\www，那么配置文件就是 D:\www\.vscode\launch.JSon。

5. 开始调试

在 vscode 中打开要调试的文件，然后在合适的代码前设置断点（在行号前点击鼠标即可），再进入运行和调试界面（点击左侧小臭虫图标或者 Ctrl+Shift+d），见图 1.13 所示。

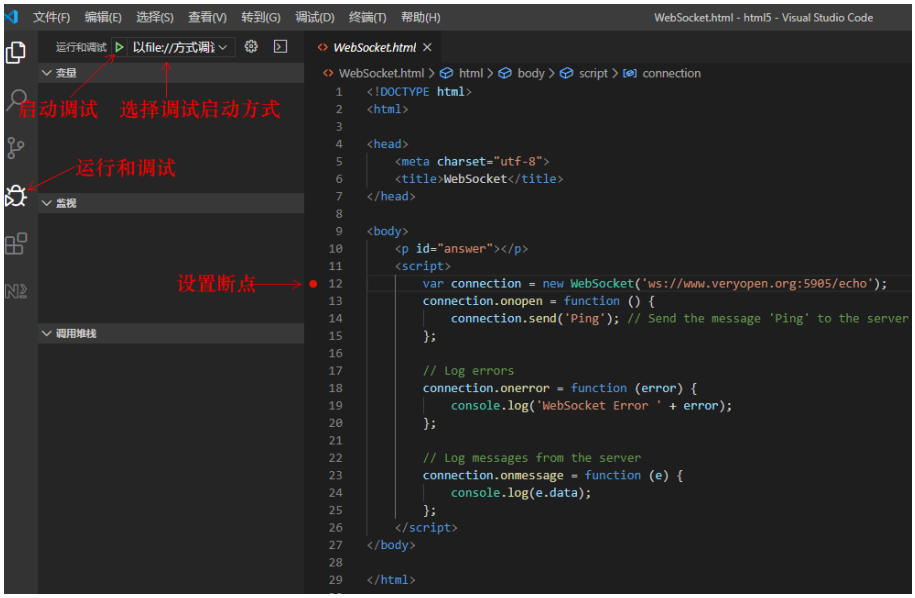
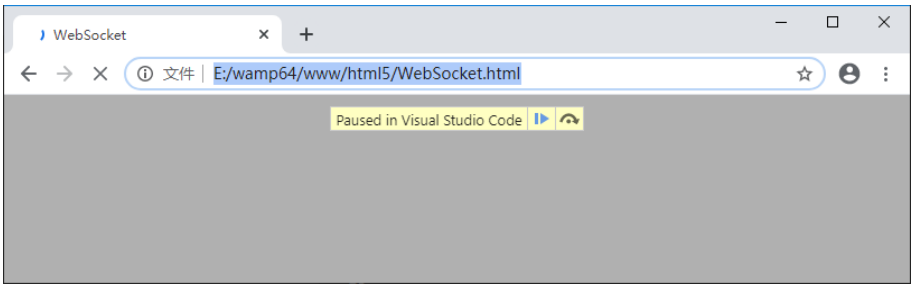


图 1.13 chrome 与 vscode 联调

点击“启动调试“按钮（或者按 F5）之后，会自动启动 chrome 浏览器，同时 vscode 中的代码暂停在第一个断点处，见图 1.14。



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

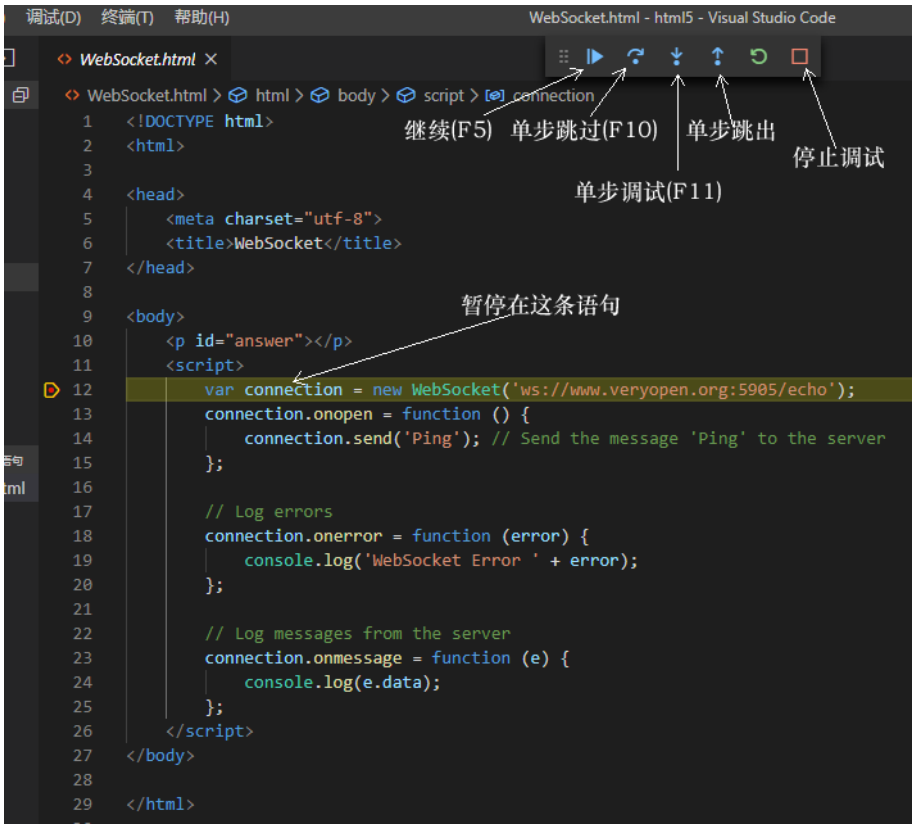


图 1.14 调试中断

点击那个“单步调试(F11)”按钮或者直接按 F11 键执行当前那条语句，每按一次就执行一条语句。点击“继续(F5)“按钮表示不再调试了，把剩下的语句全部执行完毕。“单步跳过(F10)”按钮就是不进入函数，“单步跳出”就是从函数中返回。

1.2.6 传输工具

如果你今天还在用 ftp 工具，那真是 out 了，因为是明文传输，不安全，所以尽量使用基于 ssh 协议的加密传输工具。Windows 上采用 ssh 协议的传输工具有 xftp、Winscp 以及 windows 10 操作系统自带的 scp.exe 和 sftp.exe 命令（默认已经安装了 openssh-client），xftp 非商业用途是免费的，和 xshell 为同一家公司的产品。这里介绍 Winscp 工具，从 https://winscp.ne 网站下载并安装，然后启动它，界面参见图 1.15。

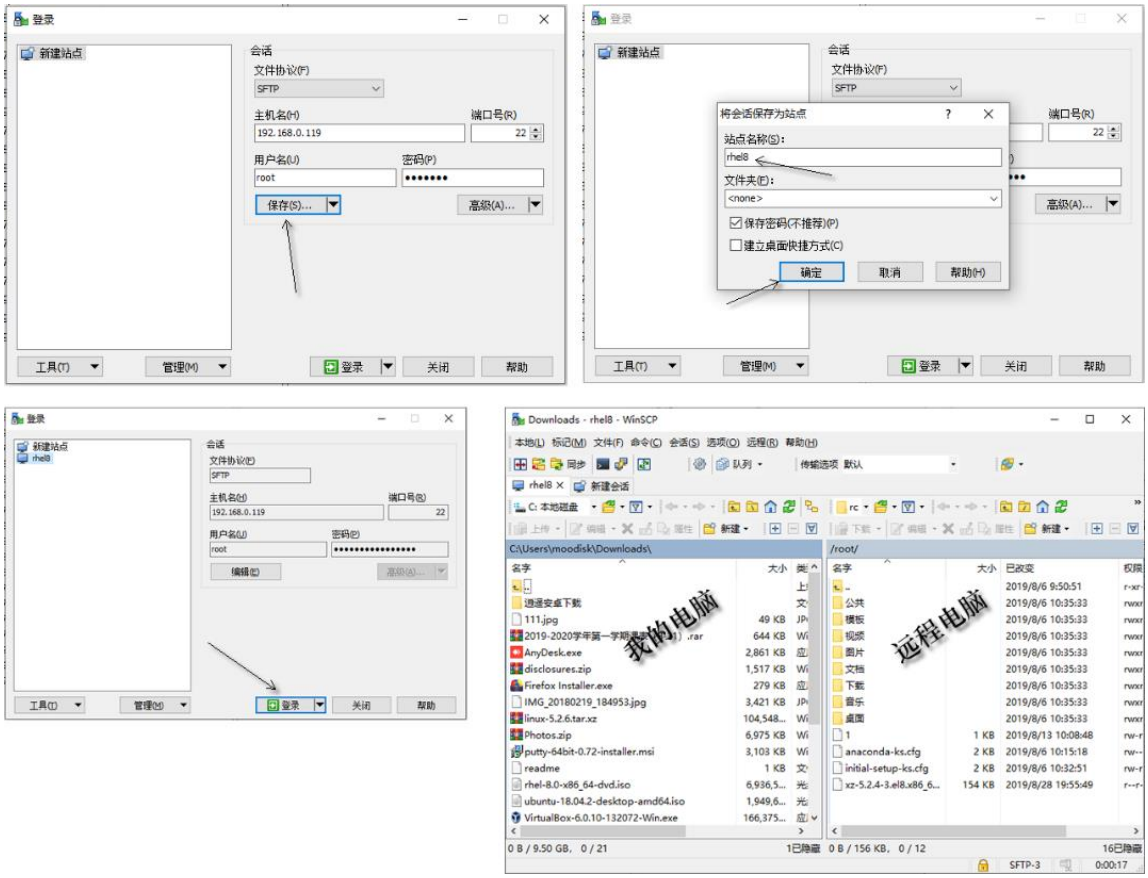
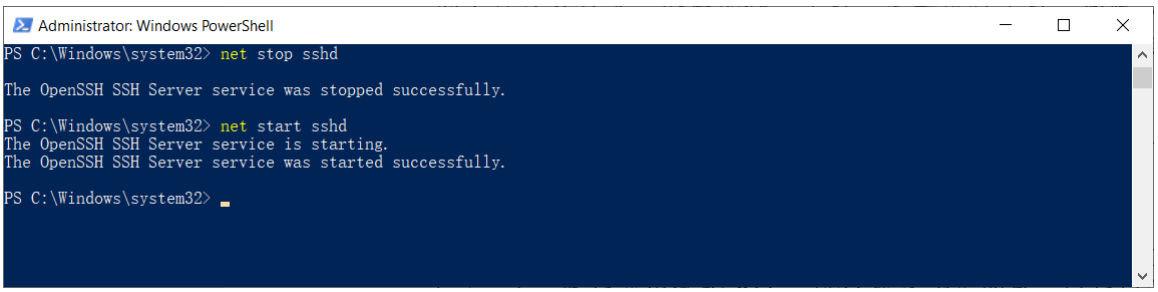


图 1.15 windows 上的 WinScp 传输工具

图 1.24 右下角部分，左边是本地硬盘窗口，右边是 ssh 服务器的窗口，用鼠标右击右边窗口中的文件或者目录，然后选择“下载”菜单进行下载；同样可以右击左边窗口的文件或者目录，然后选择“上传”进行上传操作。可以按住 Ctrl 键，然后点鼠标选择多个文件和目录，这样就能下载或上传多个文件或目录。

上面假设了远程机器启用了 ssh 服务，如果没启用，请参考下面的方法启用：

对于 linux 操作系统，就安装 openssh-server 软件包；对于 windows 10 操作系统，菜单：开始（start）》设置（Settings）》应用（Apps）》可选功能（Optional Features）》添加功能（Add a feature），然后选中添加“OpenSSH 服务器”即可，以后以管理员运行 PowerShell 后，执行命令 net start sshd 启动 ssh 服务，参考下图：



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

1.2.7 远程登录工具

基于 SSH 协议的 Windows 登陆工具有 putty.exe、Xshell、SecureCRT、SSH Secure Shell Clien 和 MobaXterm，windows 10 已经自带 ssh 命令，都为字符界面登陆工具。我喜欢使用 putty.exe，它小巧而稳定，见图 1.16 所示。

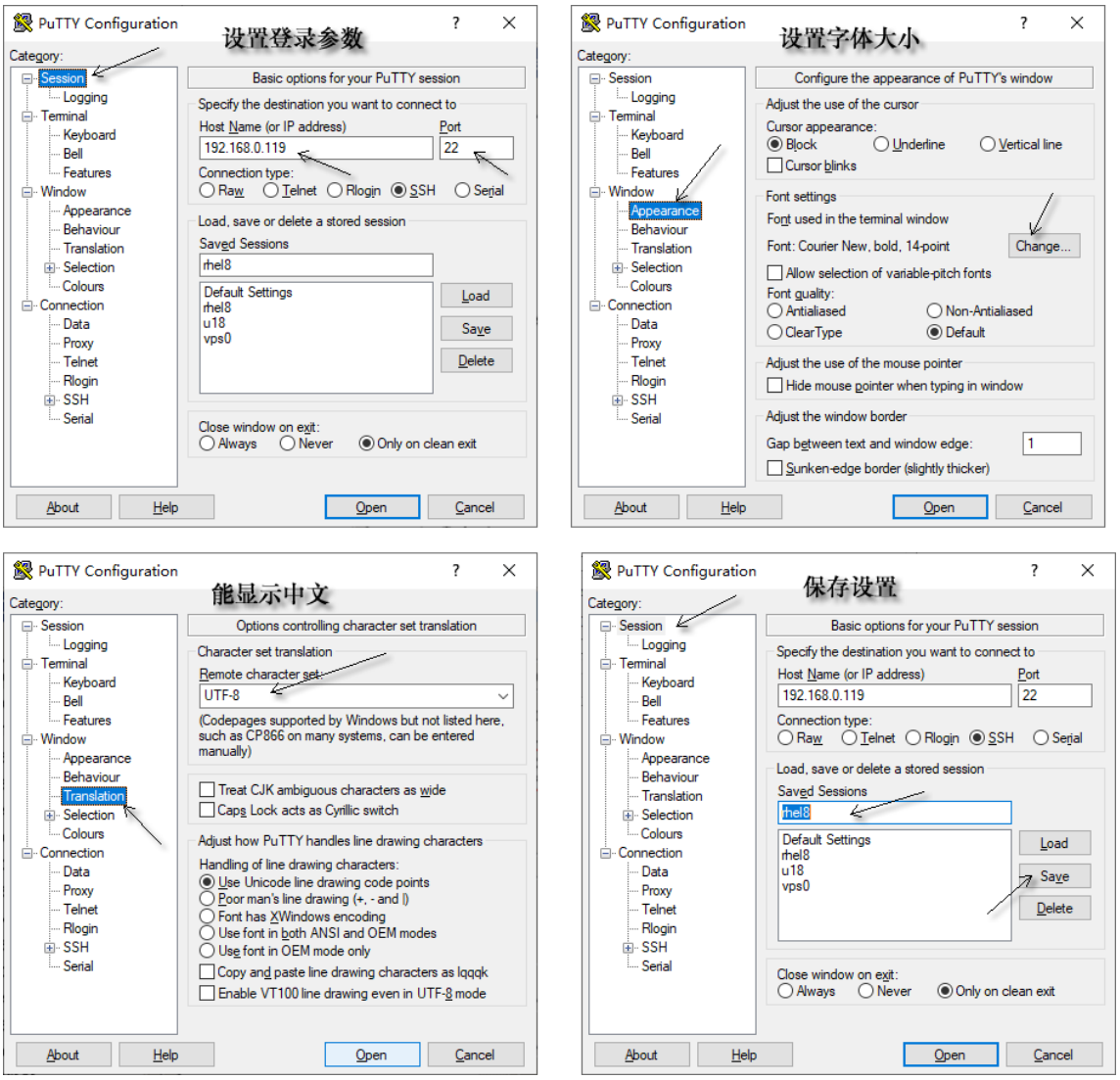
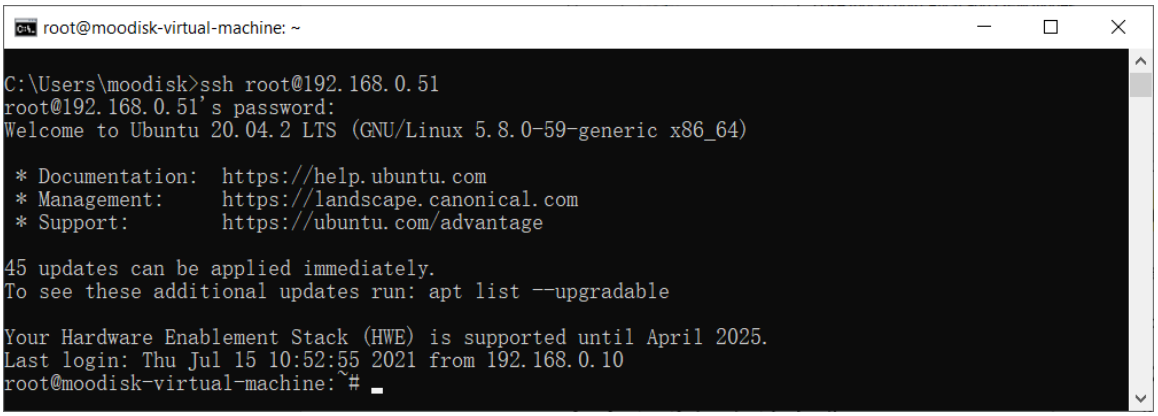


图 1.16 Windows 上的 putty 登陆工具

对于经常要登录的电脑，可以取一个名字并保存参数，以后只要双击这个名字就可以登录了。登录后按鼠标右键就是粘贴，按住鼠标选中屏幕上的内容，然后松开鼠标就是拷贝。MobaXterm 整合了登录和文件传输功能，支持多种协议，非商业用途免费。Xshell 是商业软件，但是个人、家庭和学校允许免费使用，可以从这里下载免费版 https://www.netsarang.com/zh/free-for-home-school/，Xshell 比 putty.exe 功能要强大得多。

至于 windows 10 自带的 ssh.exe 命令，直接在 windows 的 cmd 窗口中使用 ssh.exe 命令即可，参考下图所示：



1.3 建站步骤

本小节谈谈具体的建站过程，最终的目标是让他人能浏览网站。

1.3.1 第一步：需求分析

老套路啦，根据业务需要，做成什么样的网站，这里不再赘述。

1.3.2 第二步：开发网站

在准确把握用户的真实需求后，采用 Axure 工具绘制网站的 UI，经过客户确认后，就可以正式开发了。首先要做好技术选型，一般是后端采用 php/python+MariaDB，前端采用 JavaScript+CSS3+HTML5 或前端框架 vue、react、angular 等；采用自己熟悉的建站工具，比如 photoshop/fireworks 图片处理工具、vscode 代码编辑器、chrome 浏览器、chrome 开发者工具等，完成网站前/后端的开发；一般租用阿里云或亚马逊云上的虚拟主机，采用 LNMP 或 LAMP 套件搭建 Web 服务器。

1.3.3 第三步：注册域名

人们都是通过域名访问网站的，比如百度的域名是 baidu.com。所以注册一个好记又贴近公司业务的域名至关重要，但是目前好的域名基本上都被注册完了。一个好的域名最好满足以下几个

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

要求：好记、与公司名称一致、贴近主营业务、与 logo 契合、尽量短、域名类型符合公司业务类型、用字母和数字。域名的格式如下：

```
名字.类型
```

其中“名字”是你要重点考虑的核心部分，域名好不好也主要是看这个“名字”好不好。“类型”不能自己定义，只能选择，一些主要的一级域名类型及其含义如表 1.1。

表 1.1 域名类型

域名类型	含义
com	商业机构
org	非营利性组织
gov	政府部门
edu	教育部门
net	网络服务机构
me	个人网站或博客
cn	中国国内机构

常见的二级域名类型一般是在一级域名类型后加上国别，比如 com.cn，edu.cn，gov.cn，再比如 com.jp，edu.hk，gov.us 等。如果你的公司是商业机构，显然采用 com 类型最好。一般跨国企业会在各个国家建立当地网站，域名加上国别，比如微软中国的域名就是 microsoft.com.cn。有时为了好记，也会采用不怎么契合公司业务的域名类型，比如“饿了么”公司是一家商业公司，但是注册了 ele.me 域名，好记又跟中文名一致，虽然没有采用 com 类型。

取域名往往是个灵感活动，你特意去想，想到的往往被别人注册了。但是某天早晨刷牙时灵感可能就来了，一下子就想到了一个没被注册的好域名。比如笔者的个人物品分享平台网站，我差不多想了一个月，后来灵感一来，就想到了一个好域名 youduo.me，取名叫“‘有多么’个人物品分享平台”。

在域名注册公司网站上搜索你想到的域名，就可以查询到是否被别人注册了，如果没注册，你就可以注册。常见的域名注册公司有国内的万网（https://wanwang.aliyun.com/），有国外的 godaddy 公司（https://www.godaddy.com/zh）。你成功注册的域名要按年缴纳租金，如果在国内注册，年租金很高，一般都在 300 元以上，在国外注册，年租金要便宜很多，一般都在 200 元以内，而且国内要备案，国外不用备案。笔者的域名现在都放在 godaddy 公司了。

1.3.4 第四步：申请网站空间

开发完的网站如果放在局域网内自己的电脑上，外面的用户是没法访问的（除非做静态 NAT 表穿透），所以必须要把网页文件上传到公网上的某台电脑上才行。接下来看看如何申请公网上的网站空间。作为公司网站，为了安全起见，最好是独占一台机器，物理机或虚拟机都行。当然大的企业比如华为，公司网站完全是架设在公司内部的服务器集群上，然后租用的公网 ip 地址直接配置在集群的负载均衡器上。由于公网 ip 的租金比较贵，很多中小企业负担不起，这时租用云主机是最佳办法。国内阿里云主机比较好，国外有亚马逊云主机、Azure 云主机以及谷歌云主机等。国内云主机政府监管严格，需要备案，国外云主机就没有这些限制。阿里云主机申请网站为 https://cn.aliyun.com/product/ecs，亚马逊云主机申请网站为 https://amazonaws-china.com/cn/，谷歌云主机申请网站为 https://cloud.google.com/。谷歌有一款配置最低的云主机，是免费的，跑一个网站完全没问题，只收流量费，笔者的网站就是放在谷歌免费的云主机上，每个月流量费 30 元左右。云主机中的操作系统最好不要选择 windows，windows 云主机太贵又不稳定，一般都是选择 Linux 操作系统，Linux 本身免费且稳定可靠。一台云主机至少配备了一个公网 ip 地址，有了这个 ip、用户和密码，你就可以进入下一步了。什么？你不懂 Linux？我告诉你，一个不懂 Linux 的网站开发人员不是个好项目经理，可以看我写的《Linux 操作系统基础教程（第三版）》，我搞 Linux 十多年了，太清楚一个菜鸟如何入门 Linux 了。

假设你的云主机的 ip 为 210.13.187.235，用户名 root，密码 A1b2C3，采用 putty 登陆到云主机，然后安装 LAMP 网站系统（一次性安装 apache+MariaDB+Php）。

如果云主机上安装 Ubuntu 18.04 Linux 操作系统，运行以下命令安装 lamp：

```
tasksel  install  lamp-server  -y
```

安装过程中会提示输入 MariaDB 数据库的 root 用户密码，不输入直接回车确认，表示没有密码。如果安装没有报错，此时我们直接浏览 http://210.13.187.235 就可以看到如图 1.17 所示的默认网页了。

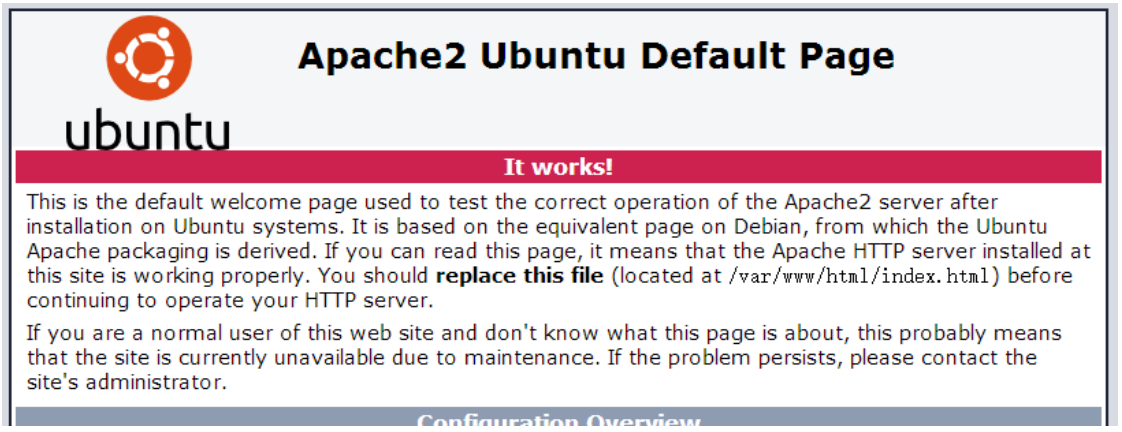


图 1.17　LAMP 安装后的默认网页

默认网站的目录是/var/www/html，静态网页文件是 index.html，在下一步，我们需要把开发的网页文件上传到云主机的/var/www/html 目录中即可。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

1.3.5 第五步：上传网页文件

接下来就可以把你开发好的网页文件上传到云主机，从 windows 上传到 Linux，我们采用传输工具 WinSCP，具体“§1.2.5 传输工具”小节。

一般都存在 index.html 或 index.php 网页文件，所以这时你用浏览器打开网址 http://210.13.187.235 就能看到你的网页了，假设云主机的 Ip 地址为 210.13.187.235。

1.3.6 第六步：给域名做 DNS 解析

如果只做测试，可以直接在本机上的 hosts 文件中添加一行（参考“§1.5.1.1 WEB 服务”的拓展练习 1），为了让世界范围内的人都能通过域名访问你开发的网站，必须要在 DNS 服务器的数据库中添加一条域名与云主机 ip 的对应记录（术语称为 A 记录），假设你在第三步中注册的域名是 abc.com，在第四步中申请的云主机 ip 是 210.13.187.235，则要在 DNS 服务器中添加如下一行：

```
210.13.187.235  abc.com
```

如果你是在 godaddy 公司注册域名，那么就登录 www.godaddy.com，然后参照图 1.18 完成 DNS 解析。

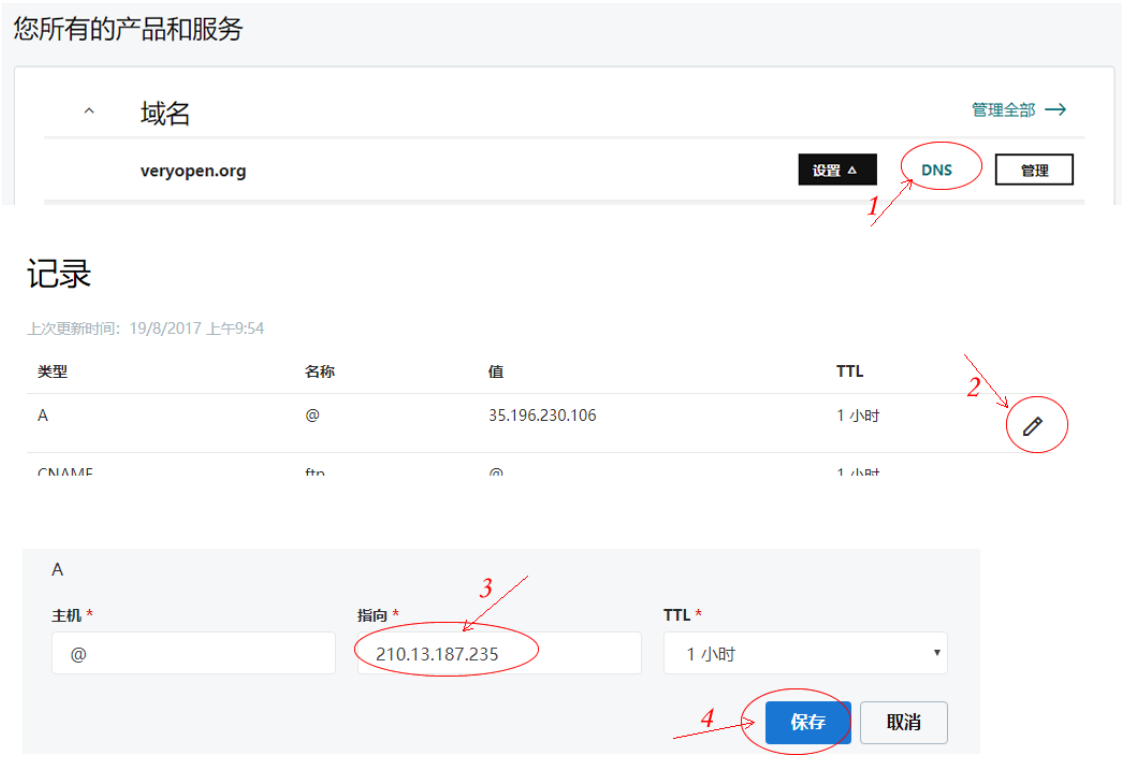


图 1.18 在 godaddy 上做 DNS 解析

我们只要把域名 abc.com 和云主机的 ip 建立对应关系，那么此后 www.abc.com（网站主机）、ftp.abc.com（下载主机）、mx.abc.com（邮件主机）等主机域名都可以用，而且还可以创建更多的同域主机别名，即：

主机名.abc.com

如 mail.abc.com、shop.abc.com、doc.abc.com、webrtc.abc.com 等等。

好了，大概 2 个小时后域名解析就生效了，生效后任何人都可以通过你新注册的“www.域名”访问你的网站了，如 www.abc.com。如果你在自己电脑上的 hosts 文件中做了解析，记得删除 hosts 文件中对应的行，这样就能通过 DNS 解释访问网站了。等等，那别人又怎么知道你的域名呢？这就是第七步的任务：推广网站。

1.3.7 第七步：网站推广与运营

目的是让更多的人知道你的网站，具体方法超出了本书范围，这里就不再赘述了。

1.4 搭建网站运行平台

一般都是在 windows 上做开发，在 Linux 上跑生产。但在国外，也有很多人在 MAC 和 Linux 上做开发，但是生产环境一般都是 Linux 的天下，因为它是免费的、高效的、稳定可靠的，而且没有病毒。

1.4.1 开发环境

1.4.1.1 WEB 服务

在 windows 开发测试时需要用到 web 服务器，因为 HTML5 的一些新增元素如<audio>、<video>、AJAX 以及离线应用等都要通过 http/https 协议做测试，而且 web 离线应用开发只能通过 https 协议测试。在 windows 上搭建一个 web 服务器最简便的方法是在 chrome 浏览器中安装 Web Server for Chrome 插件即可（可从 https://www.crx4chrome.com/crx/2282/下载），不过它不支持动态网页。搭建一个全功能 Web 服务器需要手工安装 nginx、mariadb、php 或 python，或者干脆下载一键安装套件，比如 winNMP、WNMP、WAMP 和 XAMPP 等，前两个套件都是采用 Nginx+MariaDB+Php，winNMP 还集成了 redis 数据库和 LetsEncrypt 证书工具用于实现免费的 https；后两个套件采用 Apache+MariaDB+Php，XAMPP 另外还集成了 Perl 语言。不过对于这些套件中“p”为什么不是 python，我一直耿耿于怀，因为 python 比 php 好的不是一点点，在计算机编程语言排行网站（https://www.tiobe.com/tiobe-index/）上，python 排名第三（截至 2021 年 2 月），俗话说“不懂前端的 python 工程师不是好的产品经理”。这些套件下载的地址如下：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

winNMP：https://sourceforge.net/projects/wtnmp/

WNMP：https://sourceforge.net/projects/wnmp-env/

WAMP：

https://sourceforge.net/projects/wampserver/files/WampServer%203/WampServer%203.0.0/

XAMPP：https://sourceforge.net/projects/xampp/files/XAMPP%20Windows/

本教材采用 WAMP 套件，安装时如果报错“由于找不到 MSVCR120.dll 无法继续执行代码……”，那是操作系统没有安装 vcredist_x64.exe（请从 https://www.microsoft.com/zh-CN/download/details.aspx?id=40784 下载并安装它），然后再次安装 wampserver 即可。强烈建议不要安装在 C 盘（假设你安装在 E:\wamp64 目录中），安装之后点击桌面上的“Wampserver64”图标启动它，就会在屏幕右下角的托盘上出现一个图标：  ，鼠标右击此图标可以设置语言等全局性的配置，鼠标左击图标就会弹出如图 1.19 所示的主菜单。

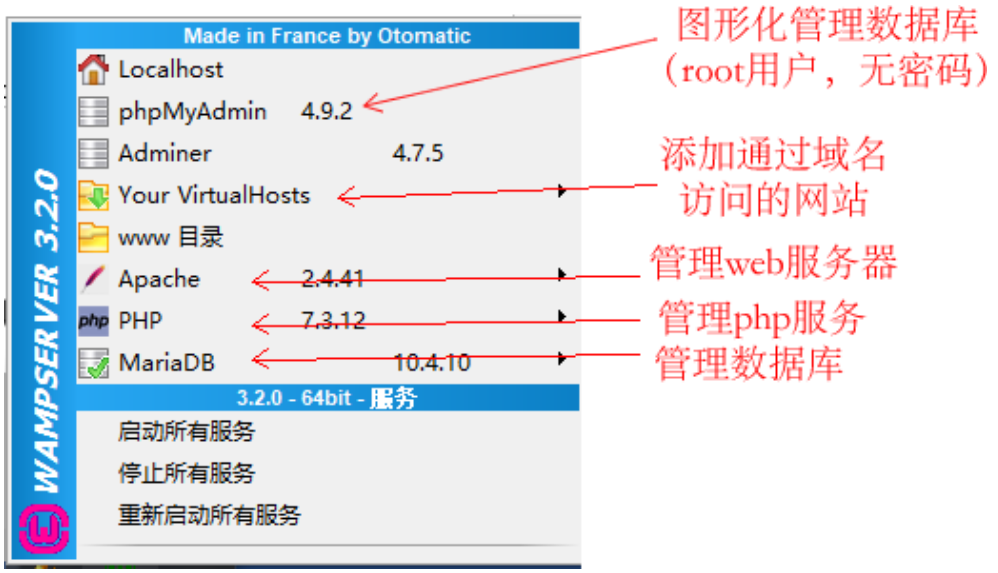


图 1.19 wamp 主菜单

通过图 1.20 所示的方法查看 web 服务程序是不是在运行：



图 1.20 查看 80 端口是否被监听

这时在浏览器中可以打开网址 http://localhost，对应的网站目录为 E:\wamp64\www，我们后续开发的网页文件都放在这个目录中。为了以后学习测试 HTML5 的离线应用，下面我们给

WampServer 增加 https 协议，这样就可以访问 https://localhost，因为很多 HTML5 的高级应用都要求走 https 协议。

1、创建网站证书

由于 https 协议端口是 443，所以要确保你电脑的 443 端口没有被占用，可运行命令 netstat -aon|findstr 443 查看，比如图 1.21 所示的情况表示被占用了。

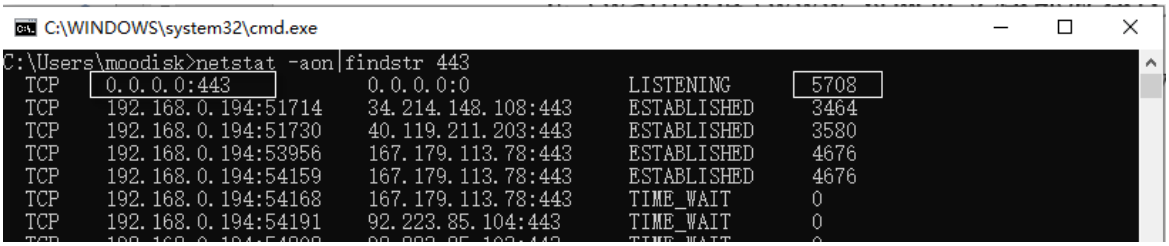


图 1.21 443 端口被占用

这时再用命令 tasklist|findstr 5708 查找 5708 号进程对应的程序名：



最后在任务管理器中结束该进程，或者卸载该软件。如果你电脑上安装了 VMware 虚拟机，443 端口会被虚拟机软件占用，这时可以修改 VMware 占用的 443 端口为其他数字，比如 444，具体修改方法如下：

启动虚拟机，菜单“编辑》首选项》共享虚拟机”，然后点击“更改设置”按钮，把 443 改为 444 或其他数字即可。对于 Vmware WorkStation 16 及更高版本，需要修改文件 C:\ProgramData\VMware\hostd\proxy.xml 中的 443 为 444，然后重启电脑即可。

从网站 https://slproweb.com/products/Win32OpenSSL.html 下载并安装 Win64OpenSSL_Light-1_1_1i.msi，然后把 C:\Program Files\OpenSSL-Win64\bin 加到环境变量 Path 中，右击“此电脑”》属性》高级系统设置》“环境变量”按钮，再参考图 1.22 所示操作。

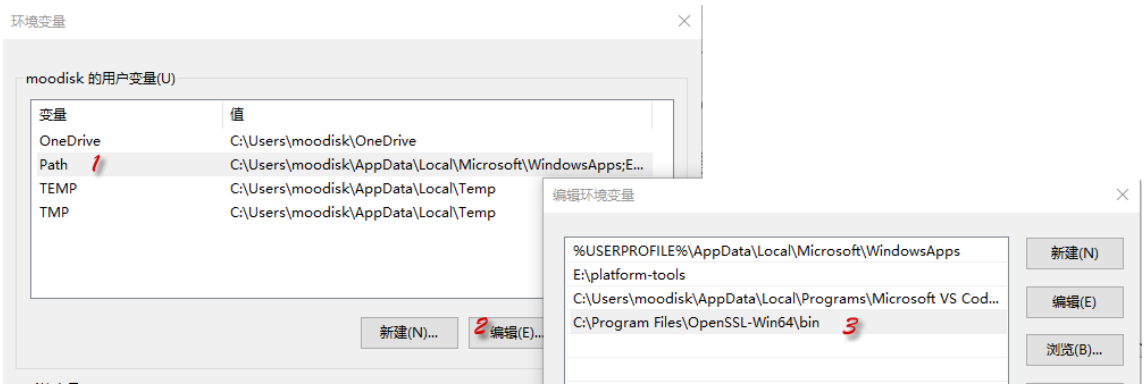


图 1.22 设置环境变量

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

执行 cmd 程序打开命令窗口，首先输入如下命令进入 E:\wamp64\bin\apache\apache2.4.46\conf 目录，然后创建网站证书。

```
e:
cd \wamp64\bin\apache\apache2.4.46\conf
openssl genrsa -aes256 -out private.key 2048
```

会提示输入一个保护密码（Enter pass phrase for private.key:），你随便输入一个即可，要求两次输入一样。

```
openssl rsa  in private key  out server key
```

要求输入上一条命令设置的密码，最后签发一年的证书。

```
openssl req -new -x509 -nodes -sha512 -key server.key -out server.crt -days 365
```

此命令会要求输入一系列的参数，直接回车即可，但是参数“Common Name (e.g. server FQDN or YOUR name) []:”必须输入 localhost。这样我们在 E:\wamp64\bin\apache\apache2.4.46\conf 目录中新建了两个文件 server.crt 和 server.key，前者是证书，后者是密钥。

2、修改 WEB 服务配置

修改文件 E:\wamp64\bin\apache\apache2.4.46\conf\httpd.conf，保证下面三行的行首没有“#”。

```
LoadModule ssl_module modules/mod_ssl.so
Include conf/extra/httpd-ssl.conf
LoadModule socache_shmcb_module modules/mod_socache_shmcb.so
```

修改文件 E:/wamp64/bin/apache/apache2.4.46/conf/extra/httpd-ssl.conf，确保几个参数及其值如下：

```
DocumentRoot "E:/wamp64/www"
ServerName localhost:443
SSLCertificateFile "${SRVROOT}/conf/server.crt"
SSLCertificateKeyFile "${SRVROOT}/conf/server.key"
```

最后重启 WampServer 服务，这时如果能打开网站 https://localhost 说明网站证书配置成功，另外 http://localhost 也能访问。

拓展练习 1

增加一个通过域名 www.abc.com 来访问的网站，首先创建目录 E:\wamp64\www\abccom（假如 wamp 的安装目录为 E:\wamp64），并在此目录中放置一个网页文件 index.html，然后参考图 1.23 操作：依次点击屏幕右下角的 wamp 图标》Your VirtualHosts》VirtualHost Management。

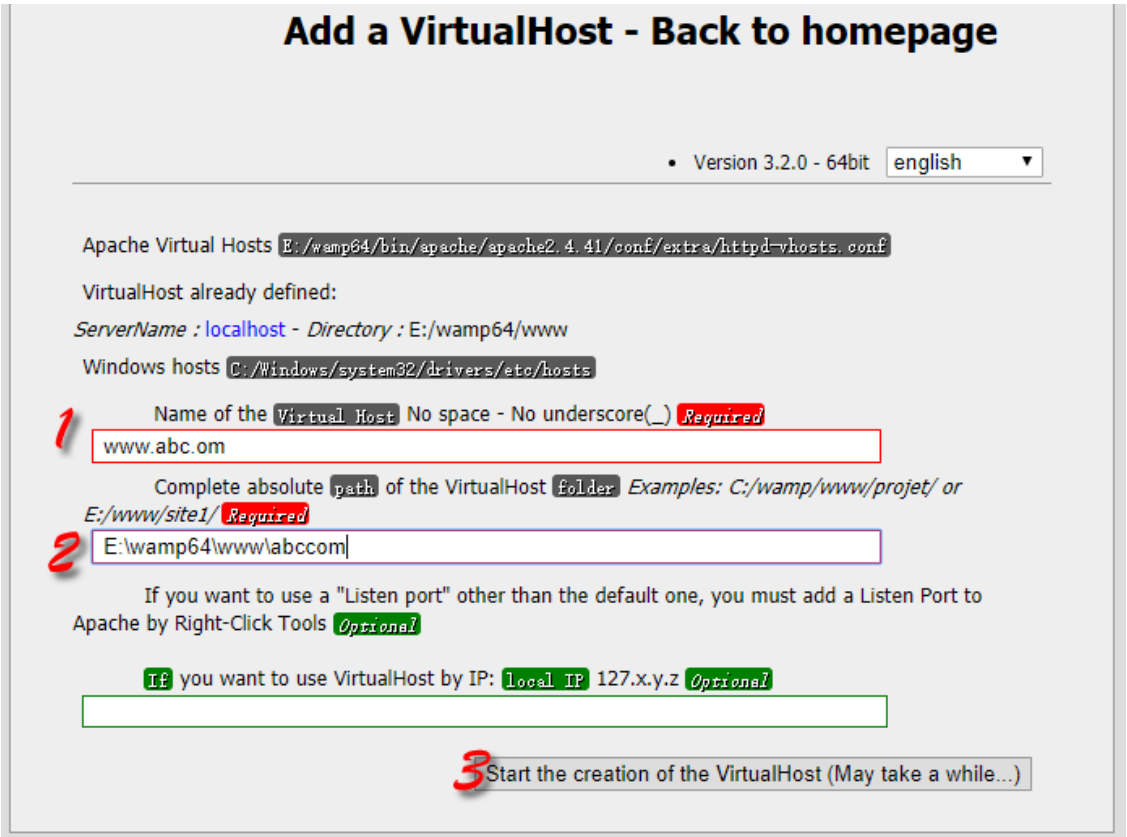


图 1.23 新增一个虚拟网站

实际上就是修改了文件 E:\wamp64\bin\apache\apache2.4.41\conf\extra\httpd-vhosts.conf，添加了如下的内容：

```
<VirtualHost *:80>
ServerName www.abc.om
DocumentRoot "e:/wamp64/www/abccom"
<Directory  "e:/wamp64/www/abccom/">
Options +Indexes +Includes +FollowSymLinks +MultiViews
AllowOverride All
Require local
</Directory>
</VirtualHost>
```

接下来还要修改文件 C:\Windows\System32\drivers\etc\hosts，加入一行：

```
10.7.57.10  www.abc.com
```

其中 10.7.57.10 为你 windows 网卡的 ip 地址，要根据你自己的电脑实际的 ip 填写，然后再重启 wamp 的全部服务（左击 wamp 图标-->重新启动所有服务）。这样就通过 www.abc.com 可以访问你的网页了。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

1.4.1.2 WebSocket 服务

我们在学习 HTML5 的新增功能 WebSocket 时要用到 WebSocket 后端服务。在 Linux 操作系统上可以采用 pywebsocket 项目搭建一个简易的 WebSocket 服务，具体过程如下：

```
git clone https://github.com/google/pywebsocket.git
cd pywebsocket
python setup.py build
python setup.py install
#用下面的命令查看说明文档:
pydoc mod_pywebsocket

#启动服务
cd mod_pywebsocket
nohup python standalone.py -H 10.142.0.2 -p 5905 -w ../example/ &
```

注意：命令中的 ip 地址 10.142.0.2 和端口 5905 要根据你自己的实际情况修改。读者可以直接使用我的 WebSocket 后端，即 ws://www.veryopen.org:5905/echo，这是一个回弹服务，即后端直接弹回前端发送的消息。

1.4.2 生产环境

如果你申请的是 Linux 云主机，可以采用 putty 工具远程登录到云主机。这里介绍搭建最流行的 LNMP 网站套件，我们需要安装 Nginx 网站服务、MariaDB 数据和 php。假设云主机的 ip 为 210.13.187.235，用户名 root，密码 A1b2C3，安装 ubuntu server 18.04 操作系统。首先采用 Putty 登录到云主机，参见图 1.24。

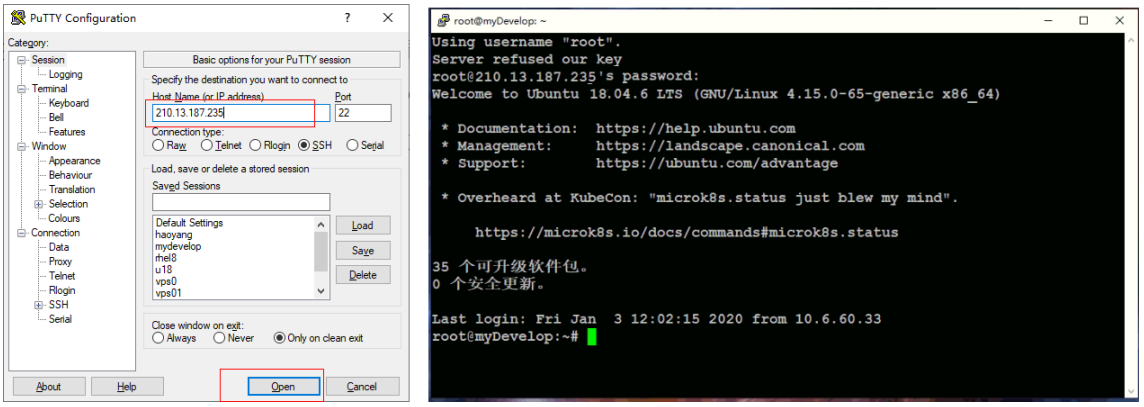


图 1.24 远程登录到云主机

登录之后采用下面的命令安装 Nginx、MariaDB 和 Php：

```
apt install nginx mariadb-server mariadb-client php php-fpm php-mysql -y
```

然后在浏览器中输入 http://210.13.187.235，就可以看到默认的静态网页内容。下面简单介绍下增加一个通过域名访问的动态网站，假设你注册的域名是 www.abc.com，执行下述命令：

命令 1：新建配置文件

```
vim  /etc/nginx/sites-enabled/abccom.conf
```

文件内容为：

```
server {
    listen 80;
    root /var/www/abccom;

    index index.html index.php;
    server_name www.abc.com;

    location ~ \.php$ {
            include snippets/fastcgi-php.conf;
            fastcgi_pass unix:/var/run/php/php7.2-fpm.sock;
    }
}
```

这个网站配置文件定义默认的网页文件名为 index.html 和 index.php，也就是说在网站目录中，如果存在 index.html 文件，当用户在浏览器中只输入 www.abc.com 域名时看到的就是 index.html，如果不存在 index.html 而存在 index.php，那么用户看到的就是 index.php 动态网页执行的结果，当这两个文件都存在时，index.html 优先，当两个文件都不存在时，返回 404 错误。

配置中的“/var/run/php/php7.2-fpm.sock”为 php 解释器创建的套接字文件，Web 服务器 Nginx 把 php 网页文件通过这个套接字文件发给 php 解析执行，执行的结果会返回给 nginx。你可以进入目录/var/run/php，查看真实的套接字文件名，然后把真实的文件名写到网页配置文件中。

命令 2：通知 web 服务器重新加载配置文件

```
systemctl reload nginx
```

注意：每次修改了网站配置文件，都要运行 systemctl reload nginx 使得配置生效，或者干脆重启 web 服务器：systemctl restart nginx。

命令 3：创建网站目录并新建一个 php 网页文件，执行这两条命令：

```
mkdir -p /var/www/abccom
echo -e "<?php\necho 'Hello Php';\n?>" > /var/www/abccom/index.php
```

在你自己的电脑上的 hosts 文件中增加如下一行，做域名解析（对于 windows，此文件在 c:\windows\system32\drivers\etc\目录中，对于 Linux 操作系统，此文件在/etc/目录中）：

```
210.13.187.235 www.abc.com
```

最后你用浏览器打开网站 www.abc.com，如能看到“Hello Php”，说明通过域名访问的动态网站就建好了。

维护 LNMP 的一些常用命令有：

netstat -tnlp （查看被监听的端口号）

必须要能看到下面的两行：

```
tcp   0   0 127.0.0.1:3306  0.0.0.0:*   LISTEN   2072/mysqld
tcp   0   0 0.0.0.0:80      0.0.0.0:*   LISTEN   1421/nginx -g daemo
```

第一行代表 MariaDB 数据库启动正常，正在监听 3306 端口，第二行表示 Nginx 正在监听 80 端口。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

1、针对 Nginx：

```
systemctl  <动作>  nginx
```

例如：

systemctl start nginx（启动 web 服务）、systemctl restart nginx（重启 web 服务）、systemctl stop nginx（关闭 web 服务）、systemctl status nginx（查看 web 服务的运行状态）、systemctl reload nginx（通知 web 服务重新加载网站配置）。

2、针对 MariaDB：

```
systemctl  <动作>  mysql
```

允许的动作有 start, restat, stop, status，意义同上。

mariadb（以 root 身份登录到数据库）、mysql_secure_installation（初始化 root 管理元密码）、mysqldump abc_db > /tmp/abc_db.sql（备份数据库 abc_db 到/tmp/abc_db.sql 文件中）、mariadb -uroot -p abc_db < abc_db.sql（从备份中恢复）

3、针对 php：

```
systemctl <动作> php-fpm
```

允许的<动作>有 start, restat, stop, status，意义同上。php-fpm 启动正常的话一定在目录/var/run/php 中存在套接字文件。

最后如果你的网站要走 https 安全协议，那么就要给网站申请证书，国内申请的证书比较贵，一年大概要六千以上。好在国际上有一些大公司联合推出免费的网站证书工具 LetsEncrypt，目前有近两亿个网站正在使用它，LetsEncrypt 的官方网站是 https://letsencrypt.org/，不过需要每三个月刷新一次证书(在 Linux 上很容易做到，做一个定时任务即可。windows 也能作定时任务)，同时为了减少滥用证书，LetsEncrypt 要求你是域名的实际拥有者（检查域名解析的 ip 是否是你电脑的 ip 地址）。

好了，接下来我们来具体采用 LetsEncrypt 制作证书，并在网站上启用，先要确认 nginx 支持 TLSv1.2 或以上版本（执行 nginx -V 查看）。

安装 LetsEncrypt 软件：

```
apt  install  letsencrypt
```

关闭 nginx 网站服务：

```
systemctl  stop  nginx
```

产生证书（假如域名为 www.abc.com，域名必须能 DNS 解析）：

```
letsencrypt  certonly  --standalone  -d  www.abc.com
```

产生的证书在/etc/letsencrypt/live/www.abc.com 目录下，我们需要 fullchain.pem、privkey.pem 两个文件。然后修改 www.abc.com 网站的配置文件（位于/etc/nginx/sites-enabled/下），增加下面三行：

```
ssl  on;
ssl_certificate  /etc/letsencrypt/live/www.abc.com/fullchain.pem;
ssl_certificate_key  /etc/letsencrypt/live/www.abc.com/privkey.pem;
```

最后让 nginx 重新加载网站配置即可：

```
systemctl  reload  nginx
```

以后每过 90 天执行下面的命令更新证书：

```
letsencrypt renew
```

或者做一个定时任务（运行命令 crontab –e）：

```
0 4 1 */3 * /usr/bin/ letsencrypt renew
```

意思是每隔三个月的 1 号 4:00 点钟运行命令/usr/bin/ letsencrypt renew，对所有的 https 证书进行更新。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

第 2 章

网站开发基础

一看到“基础”二字你就想跳过去，这个习惯可不好，俗话说“磨刀不误砍柴工”，我在这里介绍的屏幕色彩原理、尺寸度量单位、浏览器加载网页的背后机制、网站的三层架构以及前端三层分离和 DOM 模型，哪一项怎一个“基础”二字了得！是的，掌握这些基础，也许你以后不能成为网站开发人才或者天才，但是人才和天才一定了解这些基础。

2.1 屏幕与长度单位

现在都是液晶屏，老式的 CRT 屏几乎被完全淘汰了。所以本小节就专门介绍液晶屏的各种参数。首先请看图 2.1。

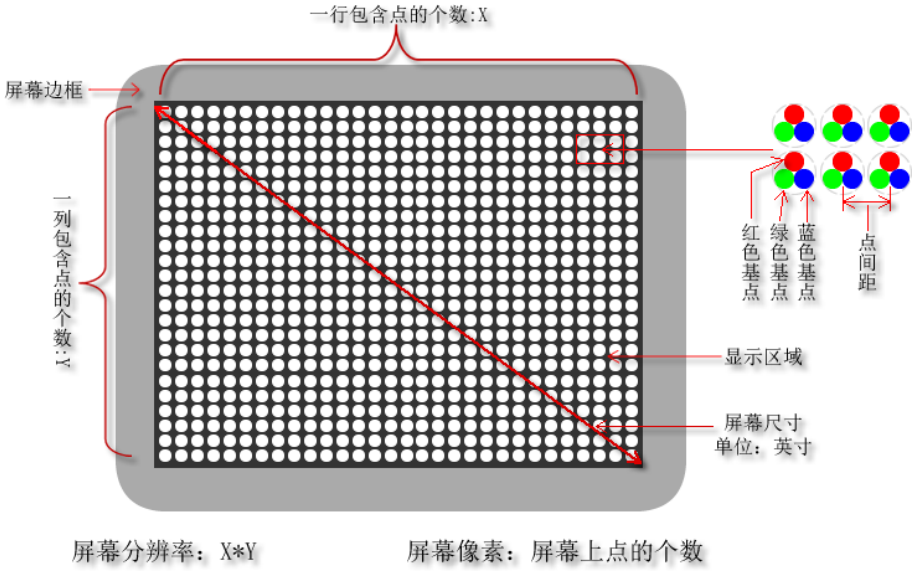


图 2.1 液晶屏面板构成

显示器都有一个外壳，外壳包裹液晶面板，从正面看到的外壳就叫屏幕边框。显示屏上的显示区域的对角线长度就是屏幕的尺寸（单位是英寸），显示区域由很多点组成的，点也称为像素，每个像素又是由三个基点构成的。其中最常用的配色方案是红色基点、绿色基点和蓝色基点，简称为 RGB 配色方案，每个基点只能显示一种颜色，但可以改变亮度。由于三个基点互相挨得非常近，因此这三个基点颜色的混合色就是屏幕上对应像素的颜色。分别控制各个基点的亮度，就可以改变屏幕上对应像素的颜色，比如绿色和蓝色基点最暗，那么屏幕上对应的那个像素就是红色，再比如三个基点都最亮，对应像素就是白色。每个基点都可以在最暗和最亮之间非连续变化，那么对应的像素颜色也会发生非连续变化。为啥称为“非连续”变化呢？那是因为电脑是采用若干位二进制来控制一个基点的亮度，比如现在最流行的 RGB 配色方案，每个基点采用 8 位二进来制控制亮度，因此 00000000 最暗，00000001 要亮一点，00000010 更亮一点，00000011 再更亮一点，……，直到 11111111 达到最亮，从最暗到最亮只有 256 次变化，因而亮度不是连续变化。一个像素由 3 个基点组成，所以一个像素的颜色就是由 24 位二进制控制，24 个二进制能表示的数的个数就是从 24 个 0（最暗：黑色）到 24 个 1（最亮：白色），共有 2^{24} 个，2 的 24 次方等于 16777216，即一个像素的颜色数，这是目前流行的大众显示器的颜色数。这里的 24 就是色深，16777216 就是显示器的色彩数（简称为色数），因此有公式：

$$色数=2^{色深}$$

参考图 2.2。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

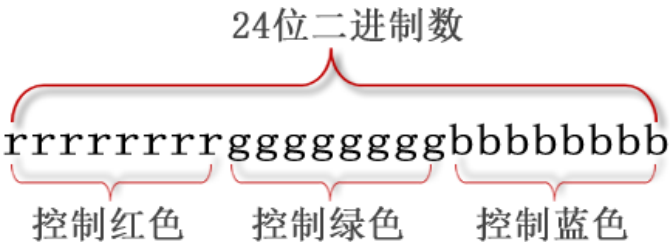


图 2.2 色深位数

我们在使用图形处理软件或者在开发网页时很少采用 24 位二进制表示颜色，而是转换为十六进制，比如鲜红色#FF0000、灰色#666666，如图 2.3 所示。

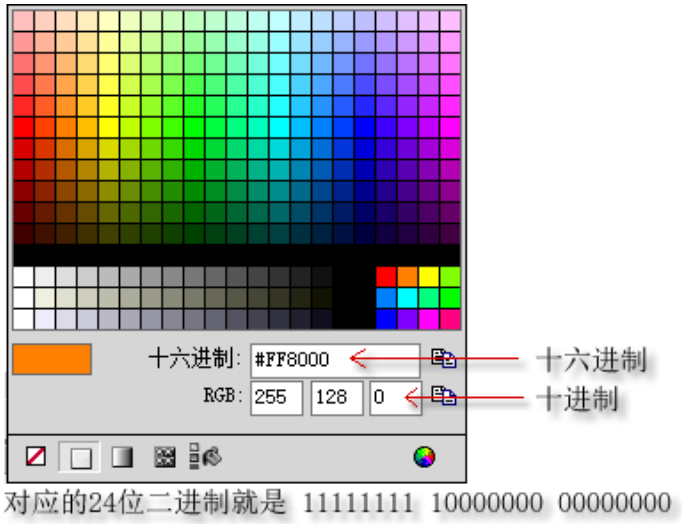


图 2.3 选择颜色

有时也会采用十进制表示颜色，这时的格式为 rgb(r, g, b)，比如上图中的颜色就可以表示为 rgb(255, 128, 0)。rgb(r, g, b)函数中的参数 r/g/b 有两种取值法：一是 0~255 的整数，二是 0%~100%的百分数，有的浏览器不支持百分数，不过能支持百分数的浏览器最终还是转换为整数，即 x%×255，如果实参大于 255，浏览器都会当成 255。现在绝大部分的显示屏用 32 位二进制来控制一个像素的颜色，还是用 8 位控制每个基点亮度，多出来的 8 位定义透明度，称为 rgba 配色方案，字母 a 表示透明度，如图 2.4 所示。

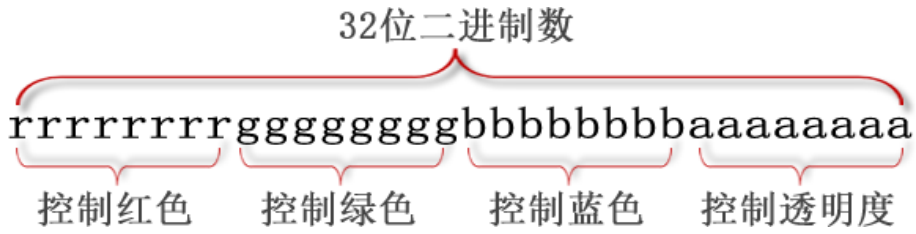


图 2.4 32 位配色方案

控制透明度的 8 位二进制都是 0 时全透明，都是 1 时不透明。因此#FF00007F 表示半透明的红色，用十进制表示就是 rgba(255, 0, 0, 127)。且慢，为了方便开发人员设置透明度，rgba()函数要求表示透明度的第四个参数用小数表示，比如半透明我们很容易知道就是 0.5，80%的透明度就是 0.8，rgba()函数内部再换算成十进制数（乘以 255 即可）。现在的一些高档显示器色数达到 10

亿以上，就是采用 10 位二进制来控制一个基点亮度，因此一个像素的色数就是 2^{30}，即约等于 10 亿。色数达 10 亿或以上的显示器特别贵。

显示器还有一个参数就是像素密度(PPI，即对角线上每英寸多少个像素)，计算公式就是：$ppi = \frac{\sqrt{x^2+y^2}}{z}$，其中 x 就是一行包含的像素个数，y 就是一列包含的像素个数，z 就是屏幕尺寸。

横向像素的个数除以纵向像素的个数(即x/y)就是显示屏的宽高比，4:3 称为正屏，这是大部分老式显示屏的比例，外观上看差不多是正方形，而现在显示屏基本上都是宽屏，主要的比例有 16:9 和 16:10。

最后我们总结显示屏的全部参数如下：

1. 屏幕尺寸：可显示区域对角线的长度，单位是英寸。

2. 分辨率：表示为$x * y$。比如 1920*900。

3. 色深或者颜色位数：控制一个像素的二进制位数（不包含透明度）。

4. 色数或色彩数：一个像素能显示的颜色个数，等于$2^{色深}$。

5. 点距或点间距：相邻两个像素圆心之间的距离。

6. ppi 或像素密度：对角线上每英寸包含的像素个数。点距与 ppi 都体现了像素密度。

色数越大，画面层次感越强，ppi 越大或者点距越小，画面越细腻。手机的 ppi 要远远大于电脑的 ppi，所以同一个图片，在电脑屏幕上显示很大，在手机上显示就非常小，还有现在 4k 屏（比如平果笔记本），ppi 也是非常高的。怎么理解这种现象呢？我们都知道一张图片有固定的分辨率，假设图片宽 800 个像素，高 600 个像素，那么这张图片在 22 寸 1920*1080 的电脑屏幕上显示出来，其宽和高到底多少厘米，下面是具体的计算过程：

根据前面的计算公式 $ppi = \frac{\sqrt{1920^2+1080^2}}{22} \approx 100$，即 1 英寸约 100 个像素。再用图片的宽 800 除以 100 得 8（英寸）约等于 20 厘米，高 600 除以 100 得 6（英寸）约等于 15 厘米。

接下来我们计算在平果 6 手机上显示此图片，其宽和高有多少厘米，平果 6 手机是 5 英寸 1920*1080 分辨率，计算出来的 ppi 约等于 440，再用图片的宽 800 除以 440 得 1.8（英寸）约等于 4.6 厘米，高 600 除以 440 得 1.36（英寸）约等于 3.5 厘米。直观图参考图 2.5。

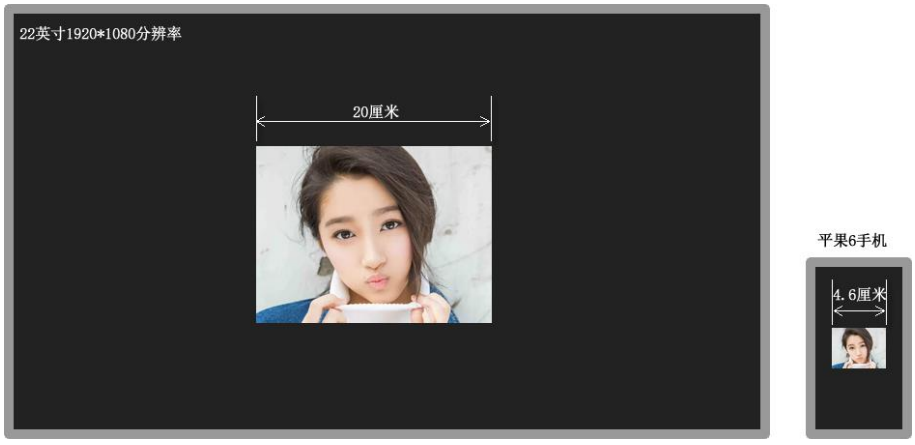


图 2.5 同一幅图在不同屏上的显示尺寸

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

显示图片似乎效果还不太糟糕，但是对于字体、按钮、图标等网页元件，那么在高 ppi 屏上小到几乎看不清了，这无形中给网页开发者带来挑战。可是别担心，硬件设备厂家也在想办法，他们为每一款产品推出了设计分辨率的概念，也就是我们常说的逻辑分辨率。厂家计算逻辑分辨率的方式之一是这样的：css3 定义的 16 号字体（即 font-size:16px）在屏幕上应该呈现的物理尺寸实际上占据的物理像素数假设为 32，那么 32/16=2 就是设备像素比 DPR，最后用物理分辨率除以 DPR 得到逻辑分辨率。逻辑分辨率和 DPR 设备厂家已经确定好了，我们直接在浏览器的开发者工具中打印 screen.width、screen.height 和 window.devicePixelRatio，就知道本设备的逻辑分辨率和设备像素比 DPR 了，如图 2.6 中显示的 DPR 等于 1.2。

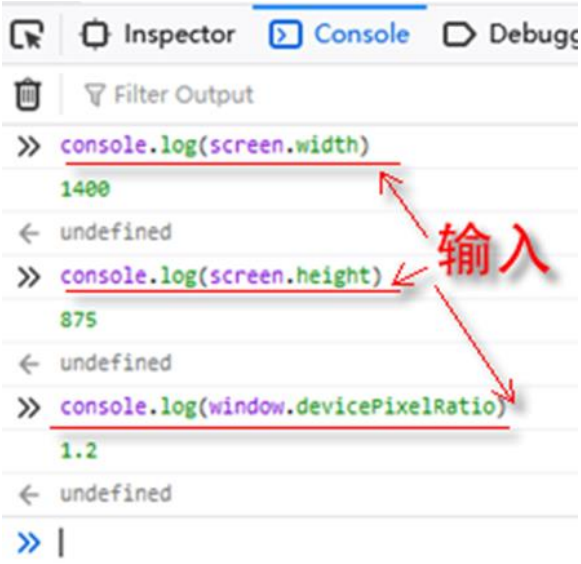
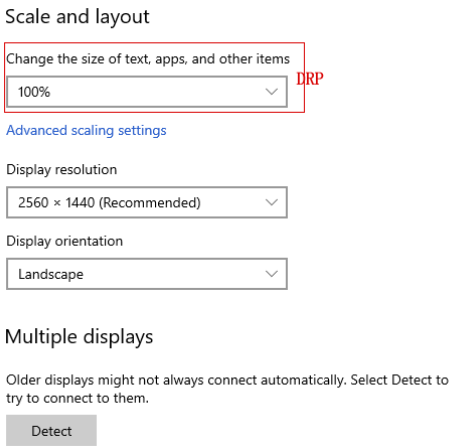


图 2.6 设备像素比 DPR

DPR 只与显示器有关，而与浏览器无关，我们可以改变 DRP，右击屏幕左面空白处，选择“显示器设置”，参考下图：



记住：我们都是在逻辑分辨率的“屏幕”上开发网站的，在 css3 样式表中定义的长度，浏览器在渲染时会先乘以 DPR 换算成物理像素数，最后再显示出来。尽管 ccs3 引入了很多逻辑长度单位（见表 2.1），但最后都要浏览器换算成逻辑像素数，再由显示驱动换算成物理像素数。

表 2.1 逻辑长度单位

px	代表屏幕上的一个像素
em	相对于当前元素内文本的字体尺寸，如 1.5em，2em 等
rem	相对于根元素(即 html 元素)内文本的字体尺寸，即*:root{font-size:npx;}*
n%	父元素相应属性的占比，如 width:10%父元素宽度 10%
vw	相对于视窗的宽度。视窗被均分为 100 单位的 vw
vh	相对于视窗的高度。视窗被均分为 100 单位的 vh
vmin	是当前 vw 和 vh 中较小的值，1vmin=min{1vh，1vw}。
vmax	是当前 vw 和 vh 中较大的值，1vmax=max{1vh，1vw}。
ch	通常是数字“0”的宽度
ex	相对于字符“x”的高度。通常为字体高度的一半
in	英寸
pt	在印刷场合用的比较多

其中 cm、mm、in 直接指定显示出来后的绝对尺寸，比如有一个 div 宽度是 3cm，浏览器会反计算出 3 厘米占多少像素，然后再以计算出来的像素值显示元素，因此不管显示屏的 ppi 多少，最终显示的尺寸都一样。rem 比 em“稳定”，比如一个 div 的宽度设为 10rem，且 html 元素的 font-size 为 20px，最终这个 div 的宽度就是 200px，而不管父元素中如何修改了字体大小。

下面重点讲讲 css3 新增的几个单位：vm, vh, vmin, vmax。

首先要明白什么叫视窗。在电脑上，视窗指的是浏览器的可视区域，会随着浏览器窗口改变而发生变化，与屏幕物理尺寸无关。而在移动端，存在三种视窗，分别是可视视窗（Visual Viewport）、理想视窗（Ideal Viewport）和布局视窗（Layout Viewport）。

可视视窗是浏览器给我们用的、能真正用来显示网页内容的区域，不包括屏幕键盘和缩放外的区域，在 JS 中，可以通过 window.innerWidth 和 window.innerHeight 获取可视视窗的宽度和高度。为了能在移动设备上浏览专门为电脑开发的网页，引入了布局视窗，它是一个虚拟的窗口，比可视视窗要大，加载网页时，直接把网页渲染在这个虚拟的窗口中，这样网页就不会错乱了，在查看的时候，毕竟手机的可视视窗要小，那就只能通过滚动条来看了。打个比方：布局视窗就是一张大白纸，HTML 的内容就写在这个大白纸上，可视视窗就是一个矩形放大镜，可上下左右移动，这样每时每刻只能浏览其中的一部分内容。参考下图 2.7。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 2.7 可视视窗和布局视窗

布局视窗大小可以通过 document.documentElement.clientWidth 和 document.document.clientHeight 获取。

不过现在有很多网页专门针对手机做了设计，比如现在的一些 H5 活动页，设计的尺寸就是在手机上看的。此时如果还是把网页渲染到这个大的布局视窗上，实在是有点不合适了。所以，又引入了理想视窗的概念，它与手机屏幕大小相等，更准确地说，等于可视视窗的大小。把页面渲染到这个理想视窗里面，就能在可视视窗中完美显示。

1vw 等于视窗宽度的 1%，1vh 等于视窗高度的 1%，1vmin=min(1vw,1vh)，1vmax=max(1vw,1vh)。vm, vh, vmin, vmax 单位中涉及的视窗就是网页被渲染进去的那个视窗，因此对于电脑来说，视窗就是浏览器的网页窗口，当手机浏览专门为电脑开发的网页时，视窗就是指布局视窗，当浏览特别为手机开发的网页时，视窗就是指理想视窗（即手机屏幕）。

比如：浏览器窗口高度 950px，宽度为 1920px, 1vh = 950px/100 = 9.5 px，1vw = 1920px/100 =19.2 px。

请参考例子 chapter-2\index-1.html，例子中的样式代码 height:100vh 能否写成 height:100%呢？可以，但效果不同，x%是定义本元素在父元素中的占比，而且当没有元素内容时，元素不会被撑开，浏览的效果见图 2.8。

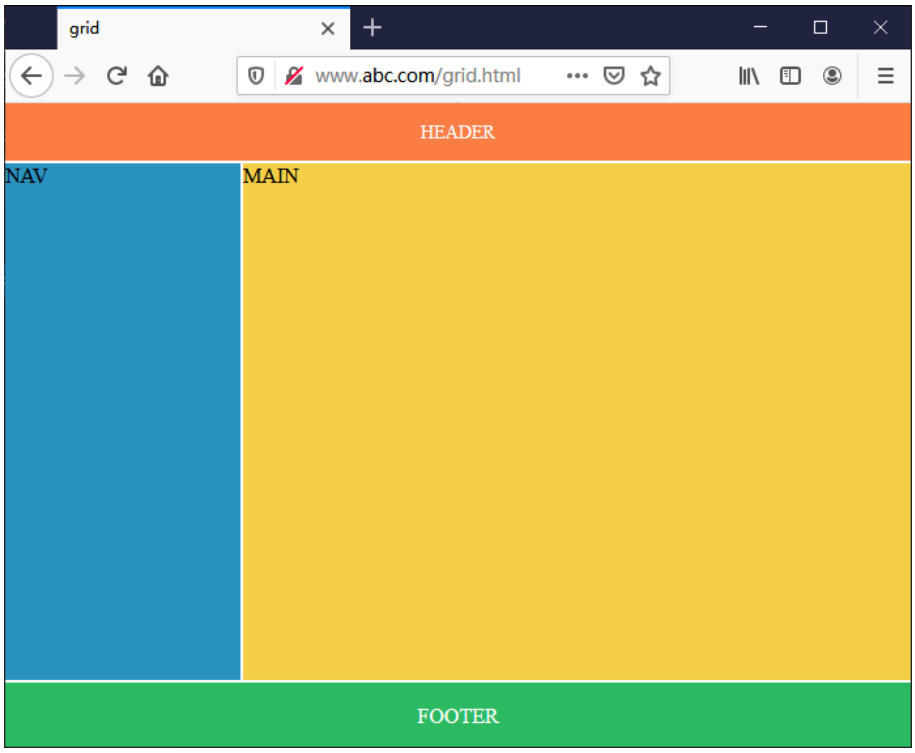


图 2.8 采用 vw 和 vh 布局

在定义距离属性（比如 width）的值时，也可以使用变量和表达式。css3 变量定义语法是：

```
--变量名:值
```

如：

```
:root{
   --bc: red;
   --good-width: 100vw;
   --good-height: calc(100vh - 10px);
}
```

使用变量的语法：

```
var(--变量名[,默认值])
```

即如果“--变量名”没定义，就取默认值。表达式的计算要通过函数 calc()，此函数的语法如下：

```
calc(表达式)
```

但是表达式中的运算符左右至少有一个空格，例如：

```
#container{
    background-color: var(--bc);
    height: var(--good-height, 100vh);
    width: calc(var(--good-width, 100vw) - 10px);
}
```

再比如我最喜欢采用下面的样式定义字号，从而使得不管在什么样的 ppi 屏幕上都能清晰看清楚文字内容。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
body{
   font-size: calc(.5vh + .5vw);
}
```

我们也可以采用 JS 语言在网页根样式中定义类似的单位，如下面的代码就在:root 中定义了--vh 变量：

```
<script>
     let vh = window.innerHeight * 0.01;
     document.documentElement.style.setProperty('--vh', `${vh}px`);
</script>
```

2018 年 CSS3 又引入了三个函数：min()、max()和 clamp()，分别计算最小值、最大值和合适值，比如 clamp(200px, 50vw, 600px)表示 50vw 换算成像素值后如果大于 600px，那么函数返回 600px，如果小于 200px 就返回 200px，否则就返回 50vw。

实操练习：定义一个容器，宽度是比 69vh 小 10px，高度比 30vw 大 20px，元素内容是“广东理工职业学院”，再定义一种字号，以便在各种屏幕上都看得清楚。并在 chrome 浏览器的开发者工具中测试各种大小的屏幕。

2.2 浏览网页的过程

网页分为静态和动态两种，静态网页文件就是 HTML 格式文件，当用户浏览静态网页时，服务器就直接把 html 网页文件发过来，因此用户每次看到的内容都一样，即内容是静态的，是没有变化的。图 2.9 展示了这个浏览静态网页的过程。

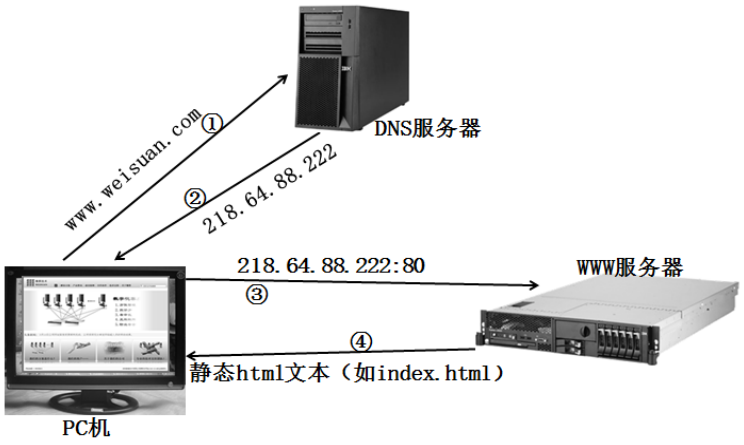


图 2.9 浏览静态网页的过程

我们在浏览器里输入网址（域名）www.bing.com 并回车，马上就看到网页的内容，图 2.9 中的①、②、③、④步是电脑自动完成的。当用户没有指定具体的网页文件时，服务器默认返回

index.html 文件，如果在浏览器中输入 www.weisuan.com/products.html，表示用户想浏览 www.weisuan.com 网站上的 products.html 网页文件，这时 Web 服务器就返回 products.html 文件，如果用户浏览的网页不存在，就返回臭名昭著的错误代码 404。浏览器收到 index.html 文件后就从上至下解析里面的内容，并显示在浏览器的视窗中，解析页面的过程称为页面渲染。使用域名便于人们记忆，但是电脑实际上还是通过 ip 地址访问网站服务器的，当然你也可以在浏览器中直接输入 Ip 地址 163.177.151.109 来访问百度。DNS 服务器的主要功能就是登记域名与 ip 地址的对应关系，当有人通过域名查询 ip 时返回对应的 ip 地址。所以在配置电脑的网卡参数时，DNS 必须采用 ip 地址，如果 DNS 本身采用域名，那么又由谁来解析这个域名呢。

一个动态网页文件其实就是服务器上的一个程序，这个程序可以用 C、bash、php、Java、node.js 或 python 等编程语言编写。在中国，人们更热衷于采用 php 或 java 来编写动态网页程序。当用户浏览某个动态网页程序时，服务器临时去执行这个网页程序，执行的结果就是生成一份完整的 HTML 静态网页文件，然后服务器再把这个临时产生的 HTML 文件发回给用户，只不过此 HTML 文件保存在内存里，而不是硬盘上。由于用户看到的内容是程序执行的结果，而程序可以访问数据库并把数据库里的内容输出，因此用户浏览同一个网页可能每次看到的内容不同，因为数据库中的内容可能被他人修改。图 2.10 展示了用户浏览动态网页的过程。

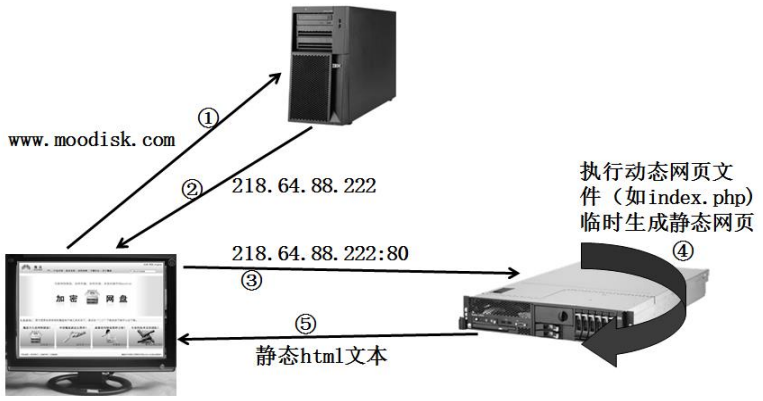


图 2.10 浏览动态网页的过程

我们开发的网页文件需要放在网站系统中，别人才可以通过浏览器浏览。套件 LNMP（Linux + Nginx + MariaDB/MySQL + Php/Python）或 LAMP（Linux + Apache + MariaDB/MySQL + Php/Python）是目前最流行的网站系统，世界上绝大多数的网站都是采用其中之一搭建的。套件中的 Apache 是 web 服务器，Nginx 是 Apache 的精简版，两者是完全开源的，可靠、速度快，俄罗斯访问量最大的网站就是采用 ngnix。MariaDB 是一个短小精悍的开源数据库，它是 MySQL 的后续版本，与 MySQL 完全兼容，MySQL 自从被 Oracle 公司收购后就不再发行新版本了。php 是国内非常流行的用于编写动态网页的脚本语言，其他的比较流行的脚本语言还有 python、perl、node.js 等，facebook、优酷、google 等大型网站和 Odoo 系统都是采用 python 语言开发的，在国外，Python 语言的流行度要远远高于 Php。

采用 LNMP 套件搭建网站服务系统的详细过程请参见我的另一本书《Linux 操作系统基础教程（第三版）》，由清华大学出版社出版。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

诸如 chrome 和 firefox 这样的浏览器与诸如 apache 和 nginx 这样的 Web 服务器之间构成“C/S”关系，即“客户机/服务器”关系。由于基于网页形式的应用占据了半边江山，于是有人进一步把网页应用称为“B/S”应用，即“浏览器/服务器”应用。Web 服务器程序启动之后监听 80 或/和 443 端口，80 端口对应 http 协议，443 端口对应 https 协议，https 协议规定传输的内容必须加密，所以现在的银行网站、网店都是采用 https 协议，比如图 2.11 所示的就是中国银行的网站。

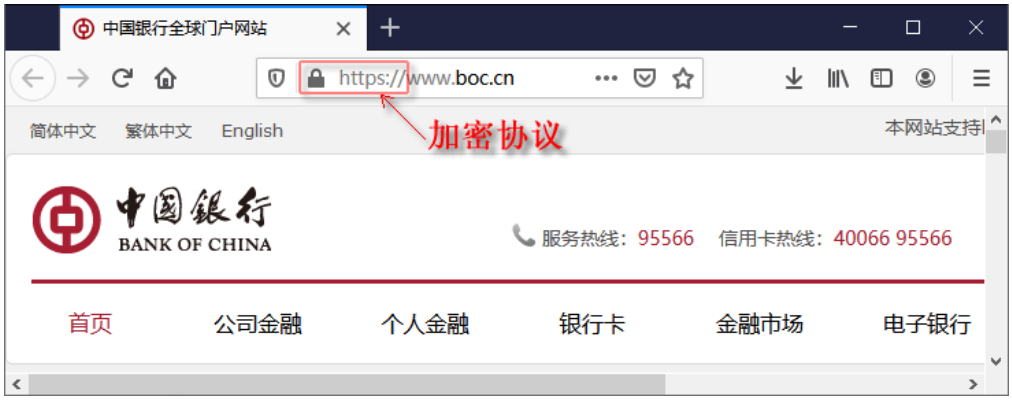


图 2.11 银行网站走 https 协议

浏览器向服务器请求一个网页文件时会发送一个请求报文，为了让 web 服务器能理解浏览器发来的请求，浏览器的请求报文必须符合双方事先约定的格式，格式如图 2.12 所示。

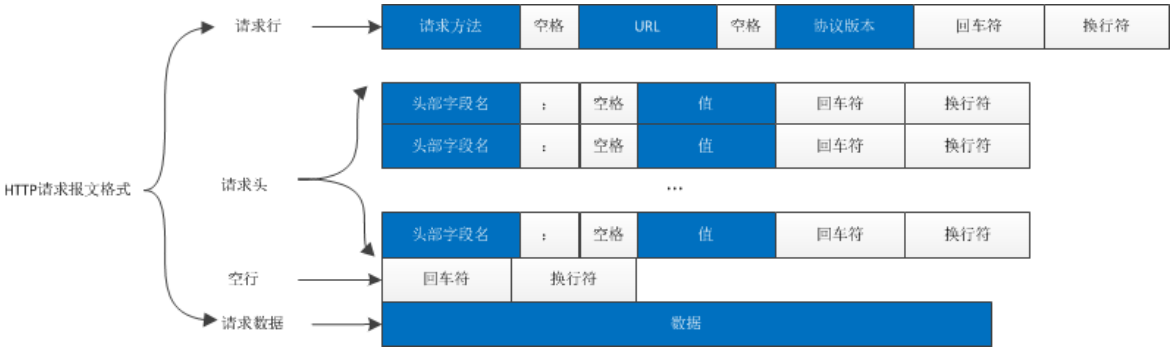


图 2.12 请求报文的格式

其中回车符换行符为“\r\n”。而服务器响应报文格式也是有规定的，如图 2.13 所示。

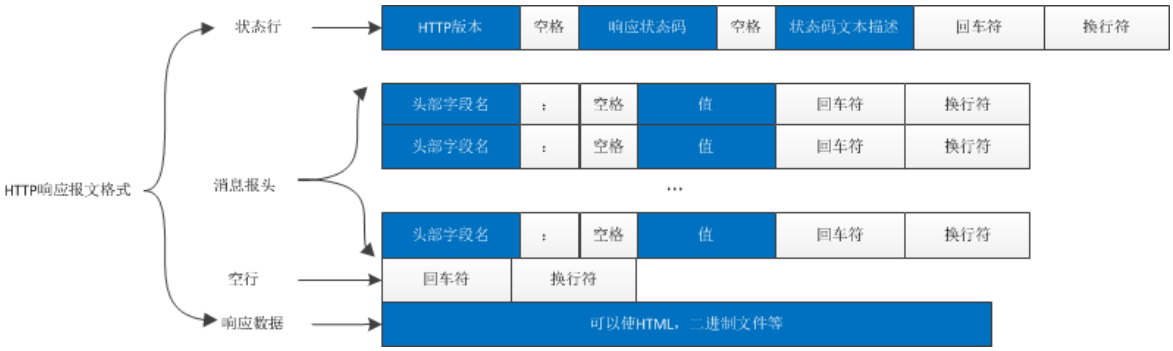


图 2.13 响应报文的格式

我们来看一个实际的例子，参考图 2.14。



图 2.14 网页的请求和响应头部

即在 Linux 操作系统上，采用命令 curl -v http://www.baidu.com 模拟浏览器请求百度网页，上图显示请求和响应的详细过程，大家对照上面讲的报文格式，仔细理解这个例子。

2.3 网站三层架构

前面讲过，基于网页的应用称为“B/S”应用，“B”就是 Browser，即浏览器，当下的 SaaS 云计算产品都是“B/S”应用。诸如*.html、*.JS 和*.css 这样的网页文件构成网站的前端，这些文件由浏览器从 Web 服务器上下载，然后浏览器就渲染或执行它们。诸如*.php、*.py 和*.java 等这样的程序构成网站的后端，这些程序在 web 服务器上被执行。因此这二者之间形成了“前端/后端”关系，前端与后端之间的“通讯”是由浏览器和 Web 服务器代理完成的，因此依然走 http/https 协议，参考图 2.15 所示。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

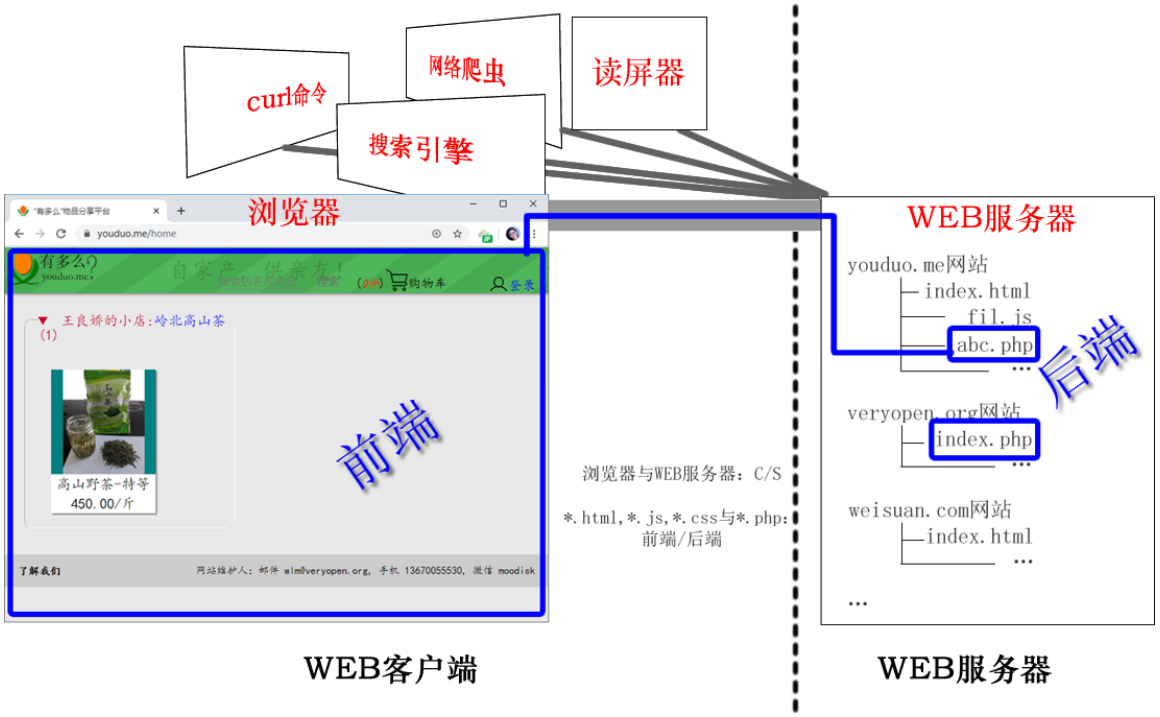


图 2.15 网站的前端和后端

从上图可知，能访问 web 服务器的工具有很多，其中浏览器只是一种，其他的还有搜索引擎、网络爬虫、读屏器、curl 命令等，这些统称为 WEB 客户端。

前端也称为展现层，后端又被划分为应用层和数据层。一个页面就是从数据层中取出数据，通过应用层计算后，在展现层中显示出来。也有人称应用层为业务逻辑层，也就是说：如何“计算”是与具体的业务相关的。应用层本质上就是把数据层中的数据映射到展现层上，到底如何映射，与具体的业务有关。举个例子，数据层中保存了张宇法同学《云计算》这门课的平时分 80 和期末考试分 93，应用层中计算该同学的总评分数方法是平时成绩占 30%，期末成绩占 70%，因此应用层计算的总评分为 89，最后展现层中以白底黑字宋体 30 字号显示“89”，如图 2.16 所示。

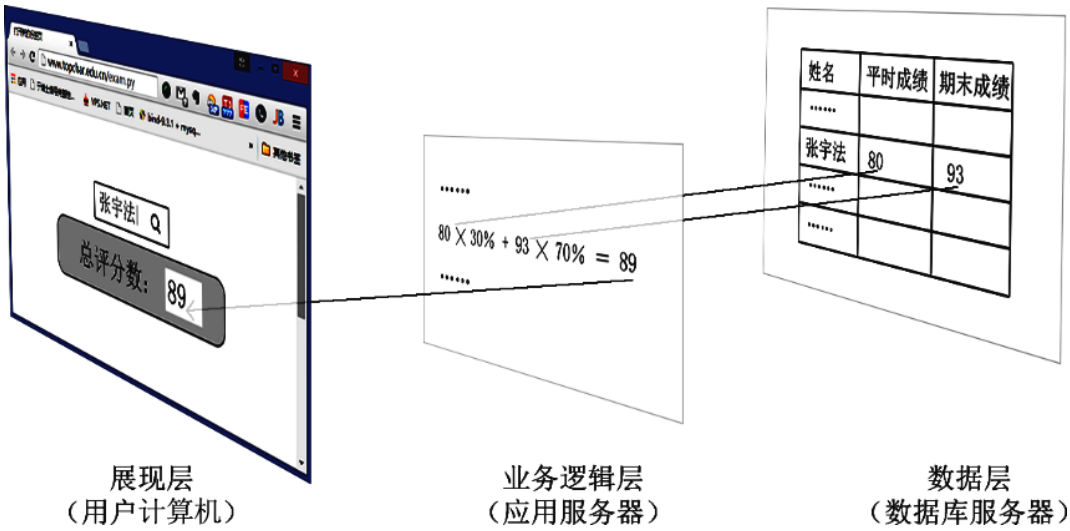


图 2.16 网站三层架构

而学工处发放奖学金时只看期末成绩，所以应用层的计算公式又不同了，因为学工处发放奖学金和任课老师综合评分属于不同的“业务”，所以应用层的“计算”也就不同。

一个简单的例子：index.html 文件为展现层、score.php 为业务层、MariaDB 的 score_db 数据库中的 student_tb 表为数据层。不要把展现层的功能放入业务层，比如不要在 score.php 中定义分数的颜色和字号大小；也不要把本属于业务层中的任务放到数据层，比如不要用数据库中的存储过程完成上面例子中的综合评分计算任务。

2.4 前端三层分离

曾几何时，我只用 HTML 元素为客户搭建网站，丑是丑了一点，但基本能用，后来，给网站加点 CSS，嗯，味道不错，再后来，又撒上几把 JavaScript，啊，色香味俱全！什么？还嫌不够！那只有祭出 Web 组件了，这是后话，放在第六章吧。

除了“展现层、业务层、数据层”这种对 B/S 应用垂直划分成三层结构外，网站前端还存在一个水平划分的三层结构——网页的结构、表现和行为。

1、网页结构

采用 HTML5 元素编写网页的结构和内容，并保存在以 html 或 htm 为后缀的文件中。图 2.17 中的网页只有结构——内容和 HTML5 元素。

```
<!doctype html>
<html>

<head>
    <title>只有结构的网页</title>
</head>

<body>
    <div>
        <p>数学：100分</p>
        <button type="button">确认</button>
    </div>
</body>

</html>
```

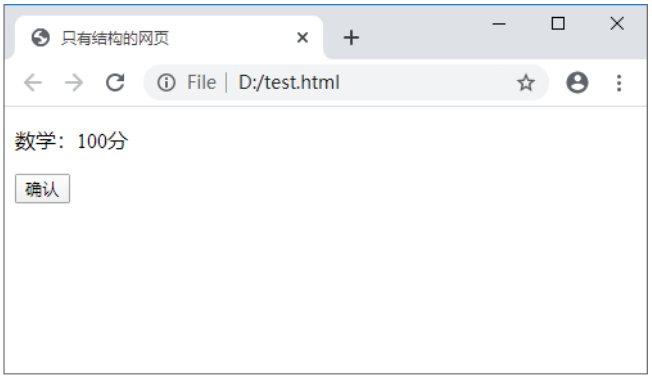


图 2.17 只有结构的网页

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

网页浏览器从 Web 服务器上下载一个 html 网页文件后，从上到下解析里面的各个元素（称为渲染），元素用来标记内容。HTML5 元素本质上就是开发人员与浏览器约定的暗号——告诉浏览器如何处理对应的内容，而暗号本身不会显示在页面上。比如当浏览器遇到“ 数学：100 分 ”时，就知道“数学：100 分”要显示在单独一个段落中，因为 就是段落元素，当遇到“<button type= "button">确认</button>”时，浏览器就画出一个“确认”按钮。本例子中的<html>、<head>、<body>、 等都是 HTML5 的元素。HTML5 提供了 10 大类共 103 个元素，用于标记各种内容：标题、段落、图片、按钮、音频、视频、……，元素的详细介绍请参阅第 4 章。如果只有网页结构，那么最终显示出来的内容的次序就是这些内容在 html 文件中的次序，比如在图 2.17 中右侧，显示出来的“确认”按钮一定出现在“数学：100 分”的后面，因为在 html 文件中就是这样的顺序（见图 2.17 左侧）。但是如果加上网页表现和网页行为，这种顺序可能就被打乱。

2、网页表现

网页表现也称为网页样式，即采用 CSS3 代码定义网页内容的外观（比如位置、颜色、阴影等）。现在我们改造上图中的网页，要求内容水平方向居中对齐，以红色显示“数学：100 分”，参见图 2.18 所示。

```
<!doctype html>
<html>

<head>
    <title>结构和样式</title>
</head>

<body>
    <div style="text-align: center;">
        <p style="color: red;">数学：100分</p>
        <button type="button">确认</button>
    </div>
</body>

</html>
```

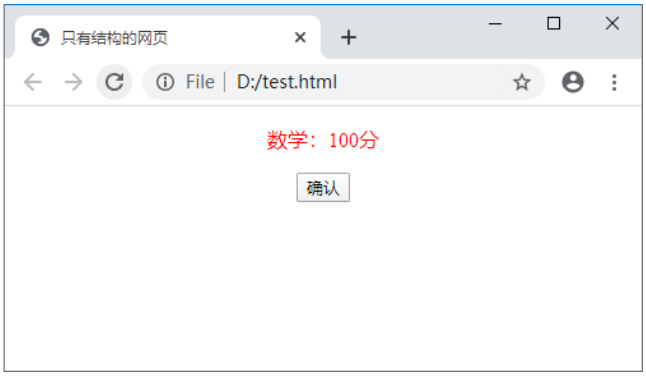


图 2.18 包含结构和样式的网页

网页的结构和内容没变，还是“数学：100 分”和一个“确认”按钮，但是外观变了，即显示的位置和颜色改变了，因为我们给 和 元素添加了 CSS3 样式（见图 2.18 左侧下划线部分），至于如何编写样式代码，请参考第 5 章。如果 CSS3 样式代码过多，建议单独保存在以 css 为后缀的文件中（具体操作见后续内容）。

3、网页行为

网页在渲染过程中或渲染后，可以动态改变网页的结构或/和表现，我们采用 JavaScript 代码来定义网页的行为。现在我们继续在图 2.16 的基础上添加网页行为：当点击按钮时，文字的颜色由红色变成蓝色，见图 2.19 所示。



图 2.19 包含结构、样式和行为的网页

图 2.19 中，虚线框内的代码就是用 JavaScript 语言编写的网页行为，如果 JavaScript 代码很多，最好单独保存在以 js 为后缀的文件中。什么？代码看不懂？看不懂就对了，说明你继续学习后面的内容就有意义。

接下来我们对图 2.19 所示的网页做改造，把结构、样式和行为的代码分别保存在 test.html、css1.css 和 j1.js 文件中，以达到“三层分离”的目的，见图 2.20 所示。



图 2.20 结构、样式和行为分离

三个文件放在同一个目录下面，然后我们浏览 test.html 网页文件，就能看到同样的效果，再点击按钮，“数学：100 分”就变成蓝色了。

样式除了可以嵌入到 HTML5 元素当中（见图 2.18）和单独保存在文件中（见图 2.20）外，还可以作为样式表嵌入在 html 网页文件中，比如对图 2.18 中的样式代码改造成图 2.21 所示。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 2.21 嵌入在网页文件中的样式表

样式是通过元素来影响相应内容的外观的，那么放在样式表中的样式或者保存在单独文件中的样式是如何与元素建立联系呢？答案是采用样式选择器，比如图 2.21 的样式表中的 div 和 p 就是样式选择器，意思是网页中所有 元素标记的内容都居中对齐，所有<p>元素标记的内容都以红色显示。CSS3 提供了各种各样的选择器，利用这些选择器，你就能精准定位到具体的元素了，选择器的详细介绍请参考第 5 章。

网站前端“三层分离”原则有利于维护和更新、并加快网页显示速度，且符合网站内容安全策略（CSP: Content Security Prolicy）的最佳实践要求。当然，对于只有一、二个页面的微型网站，“三层不分离”不会带来严重的维护问题，但是对于大型甚至特大型网站来说，“三层分离”是必须的，否则后续的维护、更新就是一场噩梦。

对于本书中一些简单的例子，为了简化，我们都把 html、css 和 JavaScript 代码写在同一个网页文件中，如果代码量比较大，还是分开来好。

2.5 DOM 模型树

浏览器每渲染一个 html 文件，就会产生一个基于 JavaScript 语言的 window 对象，通过 window 对象，我们能获取浏览器和显示器的一些重要信息，也能操控浏览器的某些行为。window 对象中最主要的一些属性、方法、事件和子对象请参阅《HTML5&CSS3 开发大起底》.xlsx 文件中的“window 对象速查表”工作表。

其中属性 screenX、screenY、innerHeight、innnerWidth、outerHeight、outerWidth、screen.width、screen.height、document.documentElement.clientWidth 和 document.documentElement.clientHeight 的解释参考图 2.22。

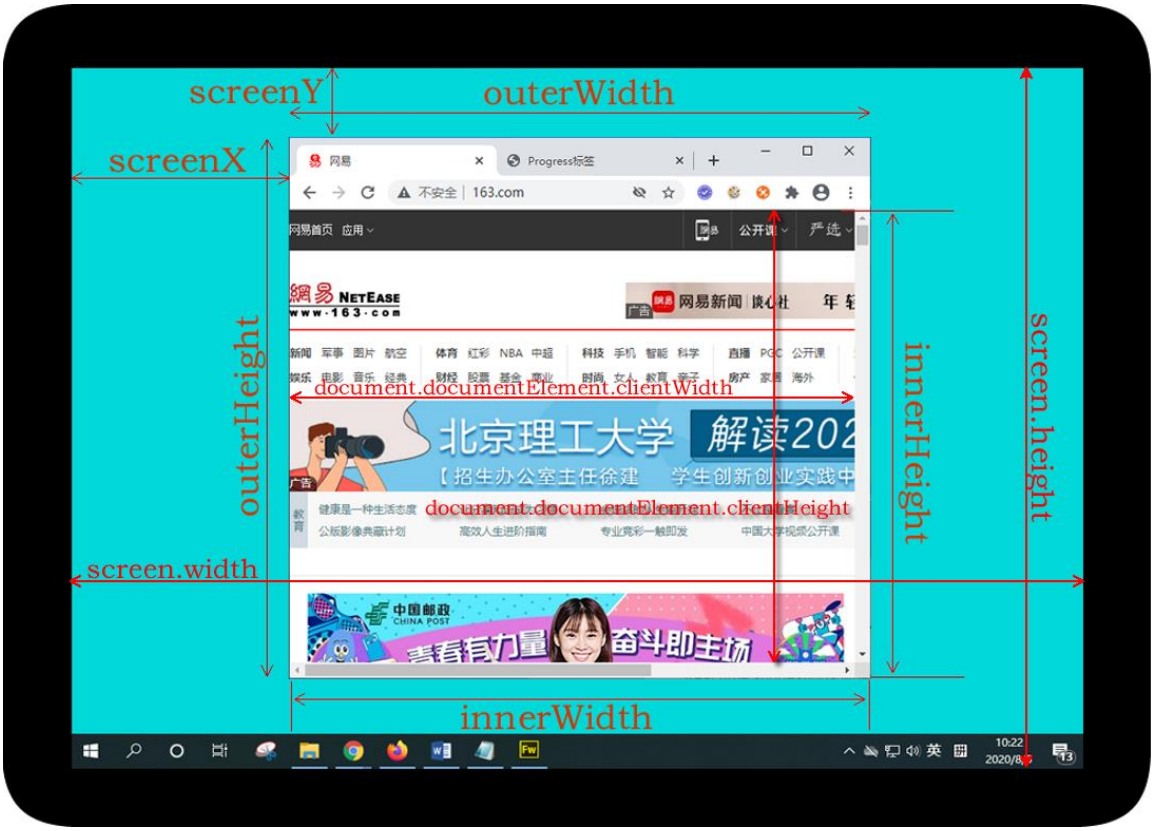


图 2.22 window 对象中的各种宽/高

由于 window 对象是全局的，我们就是在其中写 javascript 代码的，所以可以直接使用 window 的成员，比如 document 等价于 window.document，window.alert("你好啊！")等价于 alert("你好啊！")。

DOM 就是 Document Object Model 三个单词的首字母缩写，意为文档对象模型，即 window.document 对象，这里的 Document 就是指一个 html 网页文件，object 就是 html 中的元素（也称为对象，比如 div 元素）。window.document 对象中主要的属性、方法请请参阅《HTML5&CSS3 开发大起底》.xlsx 文件中的“document 对象速查表”工作表。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

DOM 定义了访问和操作 html 文档的标准方法，它把 html 网页文档表述为一个倒树型结构，各个元素之间是父子或者兄弟关系，见图 2.23。

```
1  <!doctype html>
2  <html>
3  <head>
4  <title>学习JS</title>
5  </head>
6
7  <body>
8  <a herf="js.html">我的连接</a>
9  <h1>任务12 JS基础</h1>
10 </body>
11 </html>
```

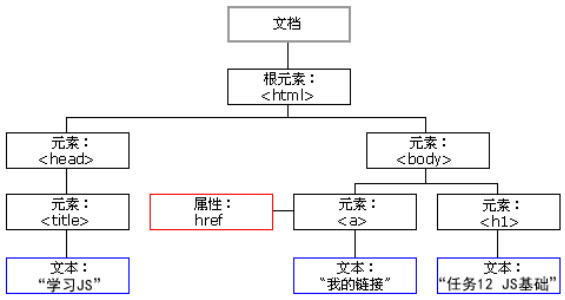


图 2.23 DOM 树型结构

根据 DOM 树型结构，我们很容得出如下的元素关系：

父元素：一个元素的直接外层元素，如上图中，h1 的父元素就是 body。

子元素：元素的直接内层元素，比如 head 的子元素为 title 元素。

后代元素：内层的全部元素，比如 body 的后代有 a 和 h1。

兄弟元素：同一个父亲的元素，比如 a 和 h1 互为兄弟元素。

但是能成为父子关系的元素是有规定的，比如 img 元素是没有子元素的，再比如 head 和 body 只能是 html 元素的子元素。

浏览器在渲染网页的时候同时在内存中生成对应的 DOM 结构，采用 JavaScript 语言编写网页的行为实际上就是操纵当前的 DOM 树，DOM 树一变，网页就立即发生变化。

第 3 章

网页行为：JavaScript 基础

从本章开始我们来简单介绍一下 JS，它的全称是 JavaScript，是一种解析型的脚本语言，虽然里面含了“java”字符串，但是 JavaScript 语言与 java 语言毫无关系。另外在开发实际的网站项目中，人们更喜欢采用各种基于 JavaScript 的库，比如 jQuery、jQueryMobile、Vue、React 或 Angular 等，但是不管怎么样，JavaScript 是基础，学好它是用好其他 JS 库的关键。JS 库是某些人闲得蛋痛，就采用 JavaScript 语言编写各种功能较强的函数，调用这些强功能函数能大大减少代码量，从而提高开发效率。不过各种各样的 JS 库变化实在太快，你往往还没完全学会掌握一个库，新的库又出现了，真正的高手不太愿意去追逐这些时髦的 JS 库，只专注于原生的 JavaScript 或 TypeScript 语言。为了学好本书的“HTML5 高级应用”部分，您最好是具备一定的 JavaScript 语言基础。

学习 JavaScript，必须从三部分入手：ECMAScript、DOM 和 BOM，ECMAScript 定义了 JS 语言的基本语法（如变量定义、循环语句等），DOM 提供了与页面交互的 API（如获取某个元素对象），而 BOM 提供了与浏览器交互的 API（如改变窗口大小）。

JS 语言成为当今非常流行的计算机编程语言，但是存在两个缺陷：一是语法太灵活导致开发大型 Web 项目困难重重，二是性能相比于 c/c++等编译语言要差很多。微软推出的 TypeScript 只克服了第 1 个问题，2018 年提出的 WebAssembly 基本克服了 JS 的两大缺陷，官方推荐采用 c/c++语言编写源代码，敬请各位关注此技术。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

3.1 JavaScript 运行时

JS 代码可以出现在页面文件的行内（参见下面的例子）、嵌入在页面文件中或者保存在单独的文件中。代码可以在浏览器加载页面前准备好，也允许加载页面之后动态通过向 DOM 树添加<script>元素节点的方式动态产生。

```
<input type="button" value="点我" onclick="alert('你点了我啦！')"
<a href="javascript: alert('去哪儿？')">点我</a>
```

把 JS 代码写在 html 文件的<script>元素中，而<script>元素允许出现在 html 文件中任何位置，且在同一个网页文件中允许出现多个<script>元素，“<script>JS 语句</script>”称为一个 JS 代码块。多个 JS 代码块的执行顺序一般就是它们在文件中出现的先后顺序，但是存在另外一种情况，就是网页开发人员给<script>元素添加 async 或 defer 属性，或者特意安排一些代码在发生某些事件时才执行，比如点击按钮时执行。对于<script>元素的详细介绍请参考“§4.4 脚本元素”小节。

浏览器执行一个 JS 代码块的大致过程是这样：首先处理所有的函数声明，然后处理所有的变量声明，最后执行语句，参考图 3.1 所示。

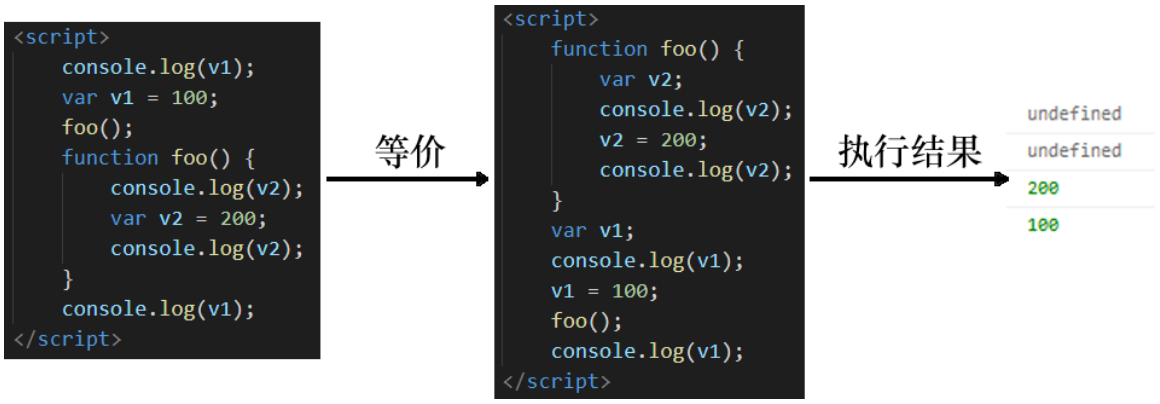


图 3.1 JavaScript 执行顺序

即把函数 foo()的声明提升到最前面，而函数体内的语句“var v2=200;”被拆分为变量声明“var v2;”和赋值“v2=200;”两条语句，然后在函数体内把变量声明语句提升到最前面。同样语句“var v1=100;”也作类似处理。如果用户写的代码如图 3.1 最左侧部分所示，那么浏览器会按照图 3.1 中间那样等价的代码执行。console.log()函数在控制台上打印信息，具体参见“§3.2.1 几个常用的函数”。ES6 新增的 let 关键字定义的变量作用域在当前语句块（即最近的{}内部）或者当前的文档内部（如果外面没有{}），且声明不会被提升，同时不能多次定义同一个变量。搞懂 var 与 let 的区别很重要，建议尽量使用 let。

后执行的 JS 语句块可以使用先执行的 JS 语句块中定义的非局部变量和函数，见图 3.2 所示。

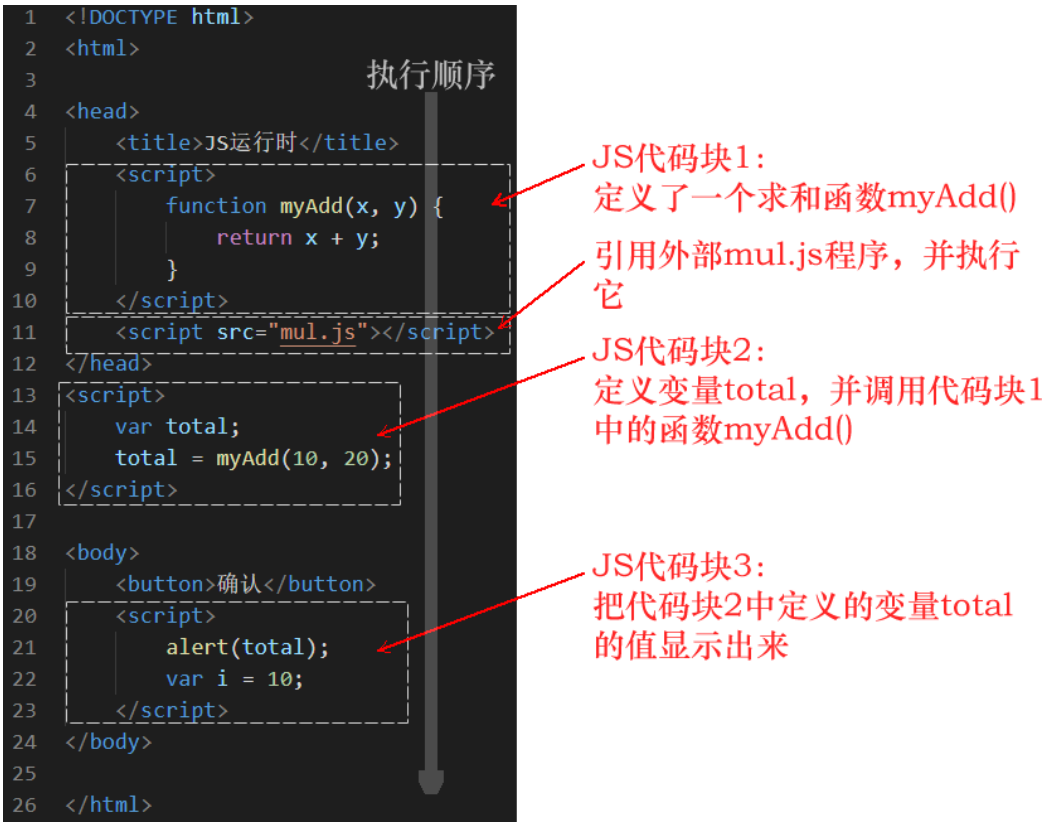


图 3.2 多个 JS 语句块

在图 3.2 中，代码块 1 或者代码块 2 就不能使用代码块 3 中定义的变量 i，因为在执行代码块 1 和 2 时代码块 3 还没有执行，此刻变量 i 根本不存在。另外浏览器在执行完代码快 1 后会把外部的 mul.js 程序读进来并执行里面的语句。

当然，如果 JS 语句比较多，还是建议单独保存在另一个文件中，就像图 2.20 所示的那样。

在更多的情况下，JS 代码是被事件触发执行的，关于事件请继续阅读后续章节。

3.2 JavaScript 基本用法

JS 是一门编程语言，具备大多数编程语言（如 c、java 等）类似的语法和程序结构，比如定义变量、定义函数、调用库函数、分支判断、循环等。但是在 JS 中，变量是没有类型的，这也是它备受诟病的地方，所以微软在 JavaScript 的基础上开发了 TypeScript 语言，它弥补了 JS 的不足，大有替代 JavaScript 的趋势。对于 JavaScript 的详细讲解，请大家参考额外的书籍，我给大家强

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

烈推荐《JavaScript 权威指南（第七版）》。这里只简单介绍一下后续章节需要用到的 JS 知识点：几个常用方法、获取元素对象、操纵元素对象。

3.2.1 几个常用方法

1、alert()方法

调用此方法会弹出一个系统警告对话框，对话框中显示的内容就是函数的实际参数，比如下面几种调用：

```
alert("你好，JS！");   //直接显示字符串“你好，JS！”。
alert(total);          //显示变量 total 的值。
alert(20*9);           //显示 180。
alert("我叫张三\n 今年 20 岁。");   //第一行显示“我叫张三”，第二行显示“今年 20 岁。”
```

上面最后一条语句的效果参见图 3.3。

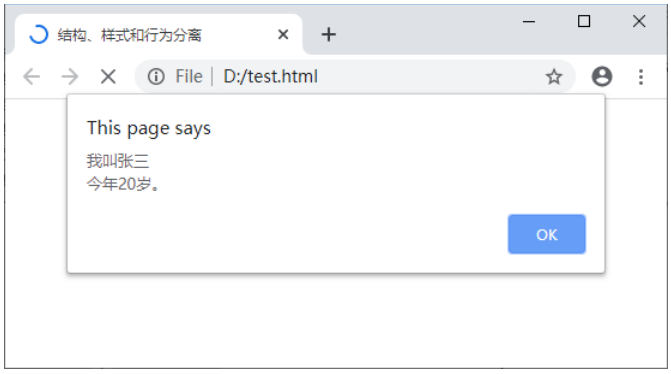


图 3.3 调用 alert()方法

2、console.log()方法

这个方法在浏览器的控制台上显示信息，而不会影响页面内容，在开发网站时，往往采用此函数来显示某些信息，以便观察程序运行的状态。对于 chrome 浏览器，按 F12 或者 Ctrl+Shift+i 快捷键打开开发者工具，其中的 console 就是浏览器的控制台，见图 3.4。

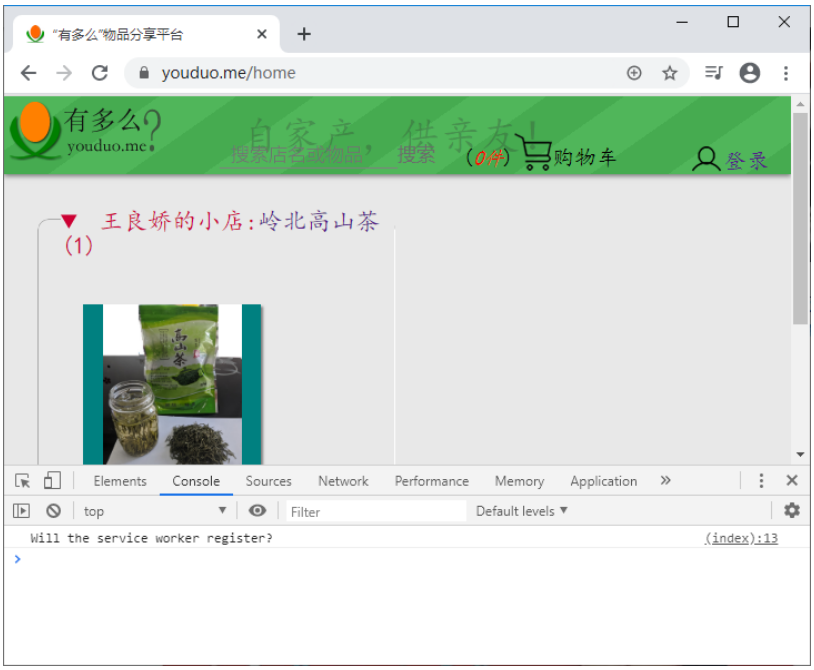


图 3.4 chrome 的控制台

比如直接在控制套中敲入如下的代码：

```
var pass;
pass = 10;
console.log(pass,"号啊");
```

结果参考图 3.5。



图 3.5 控制台输出

3、confirm()函数

这是个简单的输入函数，在弹出的窗口上点击“确认”时输入 true，点击“取消”时输入 false，比如在控制台上敲入如下代码（或写道网页文件中）：

```
var yn;
yn = confirm("确实要删除吗？")
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

变量 yn 保存输入的内容，调用的结果如图 3.6 所示。



图 3.6 confirm()方法

当点击“确认”按钮时，yn 等于 true，否则等于 false，后续语句可以判断变量 yn 的值做不同的处理。

4、prompt()方法

此方法的功能是弹出一个系统输入对话框，用户可以通过键盘输入数据。调用参数为一到两个，见下面几个例子：

```
var verf;
verf = prompt("请输入验证码：");
```

本段代码执行的结果见图 3.7 所示。



图 3.7 带一个参数调用 prompt 函数

点击“取消”按钮，返回 null。再看下面的代码：

```
var pass;
pass = prompt("请输入密码：","123456");
```

运行结果如图 3.8 所示。



图 3.8 带两个参数调用 prompt 函数

3.2.2 获取元素对象

采用 JavaScript 编写网页行为的一般做法是：获取 DOM 中的某个元素对象，然后操纵这个对象。例如图 3.9 中的一条语句：

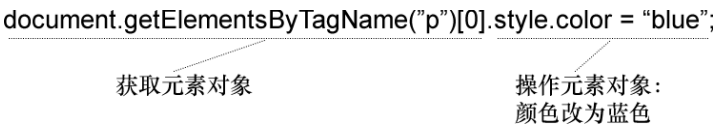


图 3.9 获取元素对象并操纵它

JavaScript 语言有多种获取 DOM 中元素对象的方法，下面介绍最常用的几种方法：

一、绝对方法

就是直接在当前的 DOM 中全局搜索并获取目标元素对象。

1. document.getElementById()方法：

通过元素的 id 号来获取元素对象，此方法返回的结果是唯一一个元素对象。例如在 html 文件中存在一行：

```
<p id="score">数学：100 分</p>
```

然后可以采用下面的语句获取这个 p 元素对象并保存在 pObj 变量中，以便后续对它做进一步地操纵：

```
var pObj = document.getElementById("score");
```

此方法的参数就一个，指明元素的 id 号。

2. document.getElementsByTagName()方法：

通过元素名来获取元素对象，此方法返回的结果是一个元素对象集合（旧版浏览器返回数组），数组下标从 0 开始，数组中下标为 0 的单元保存 html 网页文档中第一个元素对象，类似的，1 号数组单元保存第二个元素对象，以此类推。例如对于下面的 html 片段：

```
<p>18 级应用班各科平均分</p>
<div>
<p id="score">数学：100 分</p>
<p>语文：80 分</p>
<button type="button">确认</button>
<img src="me.png">
</div>
<p>总评：不错</p>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

然后采用下面的语句获取全部的 p 元素对象并保存在 pObJS 变量中，第二条语句把第一个 p 元素的内容的字号改为 30px，即以字号 30 显示“18 级应用班各科平均分”：

```
var pObJS = document.getElementsByTagName("p");
pObJS[0].style.fontSize = "30px";
```

此方法的参数就一个，指明元素的名称。

3. document.getElementsByClassName()方法：

通过元素的类名来获取元素对象，此方法返回的结果是一个元素对象集合（旧版浏览器返回数组）。例如对于下面的 html 片段：

```
<button type="button" class="carefull">删除</button>
<button type="button">确认</button>
<button type="button" class="carefull">修改</button>
```

然后采用下面的语句获取属于 carefull 类的两个 button 元素对象并保存在 btns 变量中：

```
var btns = document.getElementsByClassName("carefull");
```

此方法的参数就一个，指明若干类的名称，如果是多个类名称，中间用空格分隔开来。我们可以采用类似这样的语句把集合转换为数组：

```
arrayBtns = Array.from(btns)
```

这样 arrayBtns[0]对应“删除”按钮，arrayBtns [1]对应“修改”按钮。集合中的值不可能重复，但数组中元素的值允许重复。例如 new Set([1,2,3,4,3,3])实例化的集合中只有 1、2、3、4，重复的 3 被删除。而 new Array(1,2,3,4,3,3)实例化的数组中包含重复的 3。

4. document.getElementsByName()方法：

通过元素的名字来获取元素对象,此方法返回的结果是一个元素对象集合。例如对于下面的 html 片段：

```
<p id="score">数学: 100 分</p>
<p name="math">语文: 80 分</p>
<button type="button">确认</button>
```

采用下面的语句获取“语文”这个 p 元素并修改字号为 40：

```
Array.from(document.getElementsByName("math"))[0].style.fontSize = "40px";
```

5. elementObj.querySelector()方法：

通过 css 样式选择器在 elementObj 内部查找满足条件的第一个元素对象，此方法的参数就是一个 css 选择器。例如对于下面的 html 片段：

```
<h2 class="example">使用 class="example" 的标题</h2>
<p class="example">使用 class="example" 的段落</p>
```

可采用如下语句获取<h2>元素对象：

```
document.querySelector(".example").style.color="blue";
```

再比如<form id="profile">元素内有很多子元素<input>，可以这样做：

```
let formObj = document.getElemenetById("profile");
console.log(formObj.querySelector("input").value);
```

6. elementObj.querySelectorAll()方法：

通过 css 样式选择器来获取所有的元素对象，此方法返回的结果是一个元素对象列表（注意：不是数组）。例如对于上面的 html 片段，可以采用如下语句操纵 p 元素对象：

```
var x = document.querySelectorAll(".example");
x.item[1].style.backgroundColor = "red";
```

或者把列表转换为数据后再遍历数据元素，如：

```
Array.from(document.querySelectorAll(".example")).forEach(e =>{e.style.color="red"});
```

操纵元素对象的方法特别多，可以处理样式、行为和网页结构，这里的“处理”包括添加、删除、修改以及查询。

二、相对方法

利用 DOM 树中的亲属关系（父子、兄弟等）在某个元素对象的亲属中搜索并获取目标元素对象。

node.parentElement 表示父元素节点，node.children 代表全部的子元素节点，node.nextElementSibling 代表下一个兄弟元素节点，node.previousElementSibling 代表上一个兄弟元素节点。

例如对于下面的代码：

```
  <form id="search" novalidate action="search.php">
       <input name="search" type="search" placeholder="搜索...">
       <input id="btn" type="image" src="images/search.png">
  </form>
  …
  let btnObj = document.getElementById("btn");
//btnObj.parentElement 就是<form>元素对象，btcObj.children 为空，
btnObj.nextElementSibing 为空，btnObj.previousElementSibling 为<input name="search">
元素对象。
```

除了元素节点外，还存在一种文本节点，看下面的例子：

```
   <div id="test"><img src="images/like.png"> 99</div>
   ……
   <script>
      let nodeObj = document.getElementById("test");
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    consolg.log(nodeObj.childNodes);
    document.querySelector("img").nextSibling.nodeValue = " 100";
</script>
```

其中 nodeObj.childNodes 为 元素的子节点集合，共有两个子节点，一个是<img>元素节点，另一个是文本节点（文本节点的值为“ 99”），上面的最后一条代码把此文本节点的值从“ 99”改为“ 100”。注意这些的区别：parentElement 和 parentNode、children 和 childNodes、nextElementSibling 和 nextSibling、previousElementSibling 和 previousSibling、firstElementChild 和 firstChild、lastElementChild 和 lastChild，前者专门指元素节点，后者代表元素节点和文本节点。

3.2.3 操纵网页样式

通过 JavaScript 编程我们可以操纵元素的样式，具体的语法如下：

```
元素对象.style.样式名称 = 样式值
```

这里的样式名称大部分与 css 样式名相同，比如 color，但是少部分不一样，由多个单词组成的名字，在 css 中单词之间用“-”连接，而在 JavaScript 中采用驼峰写法（即后面的单词第一个字母用大写），比如 font-size 对应 fontSize，background-image 对应 backgroundImage。如果“样式值”为空（即""），那么相当于对应的样式被恢复成默认值，如：

```
document.querySelectorAll(".example")[0].style.backgroundColor = "";
```

这种直接给单个样式名赋值的方法只能影响一个样式，其他样式保持不变，如果要一次性替换嵌入在元素头部中的样式表，可以采用下面的语法：

```
元素对象.style.cssText = 样式表
```

例如：

```
var vp = document.getElementById("password");
vp.style.cssText = "color: red; background-color: yellow;"
```

由于来源于元素内嵌的样式表优先级最高，所以其他来源的同名样式属性都会被覆盖，但“元素对象.style.cssText”并不影响不同名的样式属性。

3.2.4 操纵网页结构

通过 JavaScript 编程我们还可以操纵网页的结构——直接操纵 html 元素、元素的属性和元素的内容。网页中一个元素的语法如下：

```
<元素名称[ 属性=值[ 属性=值 …]]>[内容][</元素名称>]
```

通过 JavaScript 编程可以在网页中添加和删除 html 元素、增加和删除元素的属性、获取和修改元素的内容。下面我们从最简单的方法讲起。

1. 获取元素的内容

方法： 元素对象.innerHtml 或者 元素对象.textContent

其中前者把元素内容作为普通字符直接输出，后者把元素内容解析后再输出，看下面的例子：

```
<div>
    <p id="score">数学：100 分</p>
</div>
```

下面的语句：

```
alert(document.getElementsByTagName("div")[0].innerHTML);
```

浏览结果参见图 3.10。



图 3.10 通过 innerHtml 属性输出元素内容

再看下面的语句：

```
alert(document.getElementsByTagName("div")[0].textContent);
```

浏览结果参考图 3.11。



图 3.11 通过 textContent 属性输出元素内容

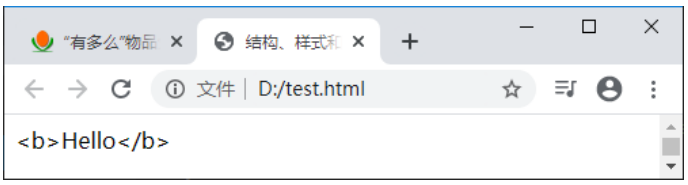
大家看到了区别吗？

2. 修改元素的内容

直接给元素对象的 innerHTML 或 textContent 属性赋值就可以了，但是这个两个属性赋值的效果不同，给 innerHTML 赋值的内容会被浏览器解析后再显示，而给 textContent 赋值的内容直接显示。比如有语句：

```
document.getElementsByTagName("div")[0].textContent="<b>Hello</b>";
```

浏览结果参考图 3.12 所示。



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

图 3.12 给 textContent 属性赋值

再比如语句：

```
document.getElementsByTagName("div")[0].innerHTML="<b>Hello</b>";
```

浏览效果参考图 3.13。

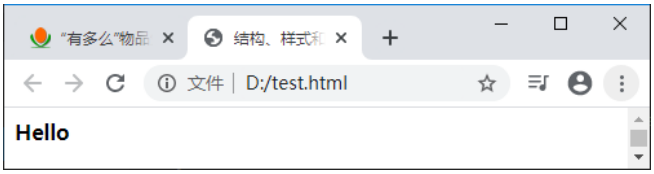


图 3.13 给 innerHtml 属性赋值

如果赋值的内容不含 html 元素，那么这二者是没有什么区别的。如果想给元素添加更多的子元素，一般都是给 innerHTML 赋值，因为用户是不希望在浏览器中看到 html 元素的。追加内容其实也容易，采用类似下面的例子即可：

```
var dem = document.getElementById("demo");
dem.innerHTML = dem.innerHTML + "<br><span>11111</span>";
```

3. 给元素添加属性

方法： 元素对象.setAttribute(属性名称，属性的值)

属性名称一定要是合法的 HTML 元素的属性名。例如对于 html 代码：

```
<p id="score">数学：100 分</p>
```

下面的语句给 id 为 score 的元素添加类属性 class="abc"：

```
document.getElementById("score").setAttribute("class","abc");
```

实际的效果为：

```
<p id="score" class="abc">数学：100 分</p>
```

对于类似 data-*的自定义属性，可以采用更方便的方法，如下面的语句添加自定义属性 data-id="10"：

```
document.getElementById("score").dataset.id = "10";
```

对于 class 属性，也存在更方便的方法，即 classList.add()。如上面增加 class 属性的语句也可改为：

```
document.getElementById("score").clasList.add("abc");
```

4. 删除元素的属性

方法： 元素对象.removeAttribute (属性名称)，对于 class 属性可采用：元素对象.remove (属性值)。

5. 新建和删除元素

第一种方法是给元素对象的 innerHTML 赋值来增加或删除子元素，例如：

```
document.getElementById("box").innerHTML = "<p>我的 dom 文本</p>";
```

另外还可以采用 document.createElement()和<元素对象>.appendChild()的方法创建元素，createElement()函数的参数是合法的 HTML 元素名称，比如 p、button、input 等。appendChild()函数的参数为元素对象，如果此元素对象是当前网页上的某个元素，那么 appendChild()相当于移动元素了，例如下面的例子就是把 id 为 myList 的元素移到 id 为 box 的元素内：

```
document.getElementById("box").appendChild(document.getElementById("myList"));
```

下面的例子创建一个“Click Me”的按钮：

```
var btn = document.createElement("button");
var t = document.createTextNode("Click Me");
btn.appendChild(t);
document.getElementById("box").appendChild(btn);
```

采用<元素对象>.removeChild()函数可以删除子元素，此函数的参数就是被删除的子元素对象。例如：

```
<ul id="myList">
    <li>咖啡</li>
    <li>茶叶</li>
    <li>牛奶</li>
</ul>
    <p id="demo">单击按钮移除列表的第一项</p>
    <button onclick="myFunction()">点我</button>
<script>
    function myFunction() {
        var list = document.getElementById("myList");
        list.removeChild(document.querySelector("#myList li"));
    }
</script>
```

更多的操纵网页结构的方法请参考专门的书籍，这里不再复述了。

3.2.5 操纵 COOKIE

document.cookie 能存储 4k 的字符串数据，关闭浏览器后 cookie 数据被存储在磁盘上，等下次启动浏览器并打开同一个“域”的页面时，浏览器会自动恢复 document.cookie 的值，只要数据没有过期。保存的数据格式一般是：

```
document.cookie="<key>=<value>;…;expires=<日期>"
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

其中“<key>=<value>”为“键/值”对格式的用户数据，“expires=<日期>”为有效期，<日期>如果比当前日期还新，表明还没过期，否则就过期了。过期的 cookie 数据浏览器不会再发往网站后端，没过期的 cookie 就会被浏览器自动发往后端，比如 php 后端就能通过$_COOKIE 数组获取 cookie 数据（如$_COOKIE["user"]）。

一般页面先判断 document.cookie 有无值，如果没有就显示登录页面，然后在用户成功登录之后写 cookie，类似下面的代码：

```
window.addEventListener("load", () => {

   if (!document.cookie) {
      document.getElementById("loginPage").style.display = "block";
      document.getElementById("loginForm").addEventListener("submit", (evt) => {
         evt.preventDefault();
         if       (evt.target.elements["user"].value       ==       "zhangsan"       &&
event.target.elements["pass"].value == "123456") { //登录成功
            document.getElementById("loginPage ").style.display = "none";
            var date = new Date();
            $livetime = 24 * 3600 * 1000; // cookie 生命周期为一天
            date.setTime(date.getTime() + $livetime);
            document.cookie = "user=zhangsan;expires=" + date.toGMTString();
         } else { //登录失败
            …
         }
      });
   }
});
```

3.3 页面事件

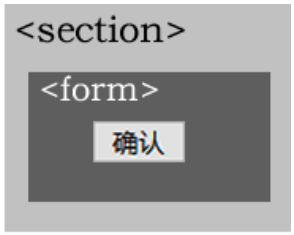
从我们打开一个页面、与页面交互、一直到关闭页面的整个过程中，在浏览器窗口里会发生各种各样的事件，比如页面加载事件、点击鼠标事件、移动鼠标事件、按下键盘事件、改变窗口大小事件、关闭页面事件等等。触发事件的原因有三个：⑴用户行为，如点击鼠标，按下键盘等；⑵异步 API 调用，如 XHR；⑶编程触发，如调用“元素对象.click()”。

事件是浏览器给网站开发人员提供执行 JS 代码的机会，通过事件我们能事先告诉浏览器：“喂，当有人点击这个‘确认’按钮时，记得执行这段 JS 代码呢，谢谢啦。”这句话实际上要给浏览器说清楚三个问题：在哪里发生什么事情时要执行什么代码？“在哪里”就是确定具体的元素，比如“确认”按钮这个元素；“发生什么事情”就是指明具体的事件，比如点击鼠标、鼠标停靠等；“执行什么代码”就是程序员具体编写的事件处理函数了。

“我们告诉浏览器……”实际上是调用元素对象的方法 addEventListener()完成的，执行这个方法就是向浏览器注册一个事件处理程序。每一个事件都有一个默认的事件处理程序（内置在浏览器中），另外允许网站开发人员给同一个元素的同一个事件注册多个处理程序，事件发生时就按注册的先后顺序执行——先注册先执行，默认事件处理程序排最后，见下图所示。另外，

我们都知道，在一个网页文件中，元素都是嵌套的，对应的 DOM 中，元素对象之间形成父子关系。由于一个子元素就在父元素里面，所以当发生在子元素上的一些事件（如点击），同时也发生在父元素上，而<html>元素是所有元素的祖宗，因此也发生在<html>元素上，并最终传导到 document 和 window 对象上。而事件的触发是串行的，因此是先触发父元素还是先触发子元素这个问题，以前不同的浏览器有不同的规定，比如 IE 浏览器规定先触发父元素，而 chrome 浏览器规定先触发子元素，这样从父到子这种链式触发称为事件捕获，从子到父这种链式触发称为事件冒泡，具体参见图 3.14。幸好最近微软 Edge 浏览器也使用谷歌的 Blink 内核，这样各种浏览器基本统一采用事件冒泡机制了。不过并不是所有的事件都会引起链式触发。

```
<section>
    <form>
        <button type="submit">确认</button>
    </form>
</section>
```



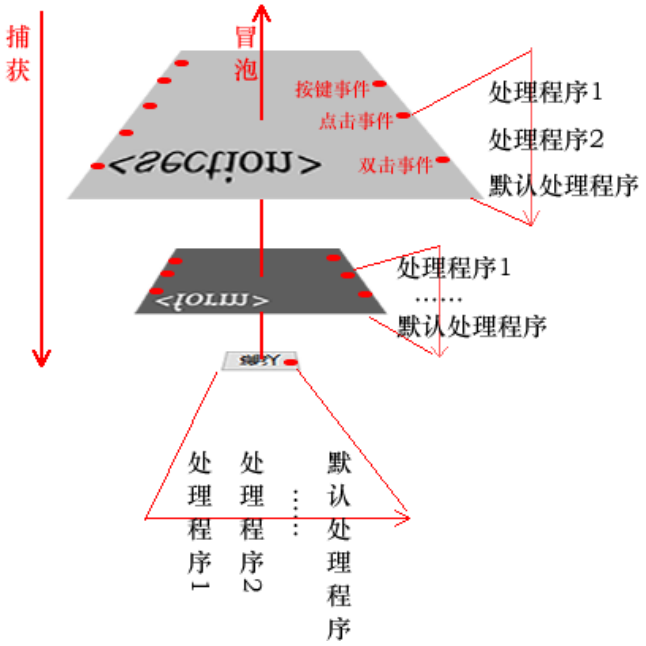


图 3.14 页面事件机制

对于上面图中所示的例子，假如用户点击了“确认”按钮，事件冒泡机制的处理过程是这样：首先触发“确认”按钮的点击事件（执行“处理程序 1”→“处理程序 2”→“……”→“默认事件处理程序”），然后触发<form>元素的点击事件（执行“处理程序 1”→“……”→“默认事件处理程序”），最后触发<section>元素的点击事件（执行“处理程序 1”→“处理程序 2”→“默认事件处理程序”）。当然<section>元素还有父元素，事件会继续向上冒泡，一直要经过<html>元素、document 对象，并最终到达 window 对象。允许在任何元素对象上的任何事件处理函数中采用 stopPropagation()方法或 stopImmediatePropagation()方法中断冒泡，也允许采用 preventDefault()方法阻止执行事件的默认处理程序，比如在上图中的“确认”按钮的点击事件的

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

“处理程序 2”中调用事件对象的 stopPropagation()方法和 preventDefault()方法，这样就不再冒泡到<form>元素了，同时不再执行“确认”按钮上的点击事件的默认处理程序了，参考如下代码：

```
document.querySelector("button").addEventListener("click", (evt)=>{
    //处理程序 2
    evt.stopPropagation();
    evt.preventDefault();
    ……;
});
```

元素对象的 addEventListener()方法有两种使用语法，根据需要选择一种即可，参考图 3.15。



图 3.15 addEventListener()方法的两种用法

语法中的 listener 就是事件发生时执行的处理程序，在具体写代码时，要么直接采用箭头函数或匿名函数，要么采用函数名（在其他地方声明的命名函数），注意千万不要在函数名后加小括号。类似下面的例子：

```
//采用箭头函数:
document.querySelector("button").addEventListener("click", (evt)=>{……});

//采用匿名函数:
document.querySelector("button").addEventListener("click", function(evt){……});

//采用函数名:
document.querySelector("button").addEventListener("click", myHandle);

//其地方声明函数:
function myHandle(event){
……
}
```

在发生事件时浏览器会把事件对象作为实参传递给事件处理函数，事件不同，事件对象的属性和方法也有所不同，比如鼠标点击事件的事件对象中的属性 clientX 和 clientY 就是当前鼠标指针的坐标，再比如键盘事件的事件对象中的属性 key 保存了按下的键，具体参考《HTML5&CSS3 开发大起底》.xlsx 文件中的“事件对象”工作表。例如下面的代码把点击鼠标时的指针在屏幕上的坐标显示出来：

```
document.getElementById("divBox").addEventListener("click", function(event){
    console.log("横坐标="+event.clientX+", 纵坐标="+event.clientY);
});
```

另外这两种语法中的 options 和 useCapture 参数是可以省略的，其中 useCapture 实参省略时代表其默认值为 false（即采用事件冒泡机制），true 就是采用事件捕获机制；options 是一个对象：{capture: true|false, once: true|false, passive: true|false}，对象成员省略时代表其默认值是 false，成员 capture 的意义与 useCapture 相同，成员 once 的值如果为 true，就是告诉浏览器第一次发生事件时执行代码，以后再发生同样的事件时不执行了；成员 passive 的值如果为 true，表示注册的事件处理程序中不能调用事件对象的 preventDefault()方法。下面的代码摘自 chapter-3\index-1.html。

```
<div>
      <button id="sm">确认</button>
</div>
……
function button1() {
    alert("我是‘确认’按钮啊！");
}
function div1(evt) {
    alert("我是‘div’元素啊！");
}
document.getElementById("sm").addEventListener("click", button1);
document.querySelector("div").addEventListener("click", div1);
```

最后两条语句给按钮和按钮的父元素添加点击事件处理函数。大家在电脑上测试这个例子，点击按钮后，仔细观察警告窗口弹出的先后顺序，再想想是冒泡机制还是捕获机制。

对于复杂的页面，可能涉及非常多的元素，而且元素之间嵌套很深，加上各个元素上又注册了不少事件处理程序，那根据图 3.14 我们可知，当一个深度嵌套的元素上发生事件时，事件冒泡的路径会很长，而且还会触发执行祖宗元素上的事件处理程序，这样既耗时又耗计算机资源。此时可以调用事件对象的 stopPropagation()方法或 stopImmediatePropagation()方法，以阻止事件继续向上冒泡，再结合事件对象的 preventDefault()方法，以阻止执行默认的事件处理程序，这样我们就可以精心控制事件的链式反应了。对于已经通过 addEventListener()方法注册的事件处理程序，可以采用 removeEventListener()方法注销，此方法的参数与 addEventListener()方法一样。最后再来看看 addEventListener()方法的参数 type，它用来指定事件名称，也就是回答“发生什么事情？”，最常用的事件请参见参阅《HTML5&CSS3 开发大起底》.xlsx 文件中的“事件”工作表。

要特别提醒：不要采用下面两种方式注册事件处理程序。

1、嵌入在元素中，类似于“<元素名 … onxxx="JS 代码" …>…”格式。

2、给对象的 onxxx 属性赋值，类似于“对象.onxxx=函数对象或匿名函数”格式。

其中 xxx 就是上表中的事件名称。这两种注册方式的缺陷是：一方面导致同一个元素的同一个事件只能注册一个事件处理程序，另一方面没法注销，另外第(1)种方法还导致网页的行为和结构没有分离，给后续维护带来麻烦。看图 3.16 中的代码。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 3.16 如何注册事件处理函数

浏览器调用某些事件处理程序时会传递一个参数，即对应的事件对象，最主要的四个对象是 Event、UIEvent、MouseEvent（鼠标事件对象）、KeyboardEvent（键盘事件对象），它们的继承关系为：MouseEvent→UIEvent→Event、KeyboardEvent→UIEvent→Event。这几个对象的主要成员请参阅《HTML5&CSS3 开发大起底》.xlsx 文件中的“事件对象”工作表。如果要额外传递更多参数怎么办？采用匿名函数即可，如下面的例子：

```
btnObject.addEventListener("click", function(evt){show1(evt,1);});
function show1(e,s) {
      console.log(e);
      console.log(s);
}
```

对于鼠标点击事件，会传递 MouseEvent 类型的参数，即对应上面代码中的形参 evt。在事件处理函数中，this 等于 event.currentTarget，尽量不要使用外部定义的变量是一个好习惯，由于 this 代表当前的元素对象，因此尽量采用相对方法获取“亲缘”的其他元素对象（参见§3.2.2 小节），例如采用下面的代码获取前一个兄弟元素的最后一个子元素。

```
let imgEmt = this.previousElementSibling.lastElementChild
```

3.4 定时器

在开发网页时经常会用到定时器，用于实现动画或者计算任务，JavaScript 提供的两种定时器 setTimeout()和 setInterval()可能会出现掉帧或卡顿现象，这是因为浏览器的固有刷新周期与定时周期不一致导致的。为此，引入了 requestAnimationFrame(回调函数)方法，就是在屏幕每次刷新时调用这个“回调函数”。

setTimeout(要执行的代码,毫秒数)为单次定时器，即再过多少毫秒执行某代码，具体的语法如下：

```
setTimeout(handler[, delay[, arg1, arg2, ...]]);
```

即过 delay 毫秒后调用函数 handler(arg1, arg2, ...)，delay、arg1、arg2 都可以省略，如果省略 delay，就是立即调用函数。setTimeout()函数返回一个表示定时器编号的正整数，这个值可以传递给 clearTimeout()来取消该定时器。看下面的例子：

```
setTimeout(function(){alert("时间到")}, 2000);
```

即 2 秒中后执行那个匿名函数。再看例子：

```
var timeoutID;
timeoutID = setTimeout(slowAlert, 5000);

function slowAlert() {
  alert('太慢了!');
}
```

setInterval()函数定义循环定时器，即每隔多少毫秒执行某些代码，它的语法与 setTimeout()一样。看例子：

```
<script>
    var total = 0;
    function cba(){
        total ++;
        console.log("total=",total);
    }
    setInterval(cba,2000);
</script>
```

每隔 2 秒就在控制台上打印变量 total 的值（按 F12 打开浏览器控制台），循环是无限的，直到关闭浏览器，但是用 clearInterval(参数)函数可以结束循环定时器，参数就是定时器句柄（由 setInterval()函数返回），下面的代码片段摘自 chapter-3\index-2.html。

```
    <p>倒计时：100</p>
    <button type="button" onclick="terminal()">终止倒计时</button>

    <script>
        var total = 100;
        var oTimer = setInterval(function () {
            total --;
            document.querySelector("p").textContent = "倒计时：" + total;
        }, 1000);

        function terminal() {
            clearInterval(oTimer);
        }
    </script>
```

从 100 开始倒计时，当点击“终止倒计时”按钮时结束倒计时，就是采用了 clearInterval()方法终止计数器，浏览的效果见图 3.17。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

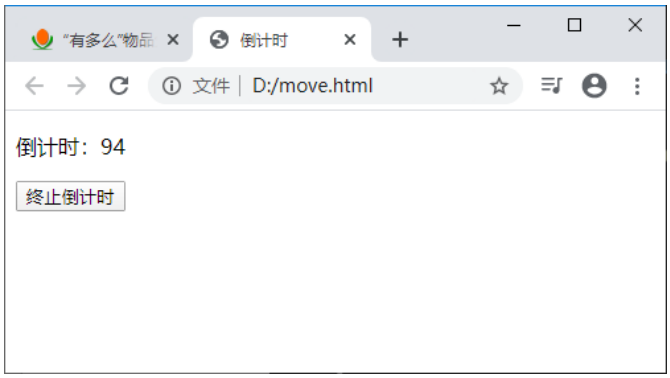


图 3.17 启动、终止定时器

在游戏开发中，设计动画时最常用的定时器是 window 对象的 requestAnimationFrame()方法，此方法的定义原型如下：

```
long window.requestAnimationFrame(callback);
```

参数 callback 是回调函数名，你需要额外编写这个回调函数，requestAnimationFrame()告诉浏览器下次重绘页面之前调用指定的回调函数，浏览器重绘频率一般是每秒 60 次。所以在回调函数中一般会继续调用 requestAnimationFrame()方法。与 setInterval()循环定时器相比，它具备两大优势：一是不会掉帧，二是节约计算机资源（当浏览器窗口不是焦点程序或焦点窗口时，动画停止）。requestAnimationFrame() 函数返回一个长整型数据，你可以把它传给 window.cancelAnimationFrame() 以取消回调函数

下面这段代码摘自 chapter-3\index-3.html，浏览此页面能看到一个徐徐升起的红太阳。注意 abc()函数里再次调用 requestAnimationFrame(abc)，告诉浏览器下次刷新页面时记得调用 abc()函数，这样就导致浏览器每次刷新页面就会调用 abc()函数的局面。

```
    var x=0;
    function abc(){
         document.getElementsByTagName("div")[0].style.bottom = x+"px";
         x ++;
         if(x==600)x=0;
         window.requestAnimationFrame(abc);
     }
      window.requestAnimationFrame(abc);
```

另一个例子 chapter-3\index-4.html 实现了红色的球从左往右移动。

3.5 Ajax 编程

AJAX = Asynchronous JavaScript and XML（异步的 JavaScript 和 XML），在不重新加载整个页面的情况下，可以与服务器交换数据并更新部分网页内容。AJAX 不需要任何浏览器插件，但需要用户允许 JavaScript 在浏览器中执行。可以采用 XMLHttpRequest 类或 fetch()方法具体实现

AJAX，前者较后者更复杂、更灵活且被浏览器支持得更好，且能实现所有的应用场景，而 fetch() 使用更简单，从 2020 年开始所有浏览器都支持它了。先看一个例子：

```
1  <!DOCTYPE html>
2  <html>
3
4  <head>
5      <meta charset="utf-8">
6      <script>
7          function loadXMLDoc() {
8              var xmlhttp;
9              if (window.XMLHttpRequest) {
10                 //  IE7+, Firefox, Chrome, Opera, Safari 浏览器执行代码
11                 xmlhttp = new XMLHttpRequest();
12             } else {
13                 // IE6, IE5 浏览器执行代码
14                 xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
15             }
16             xmlhttp.onreadystatechange = function () {
17                 if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
18                     document.getElementById("myDiv").innerHTML = xmlhttp.responseText;
19                 }
20             }
21             xmlhttp.open("GET", "/proxy.pac", true);
22             xmlhttp.send();
23         }
24     </script>
25 </head>
26
27 <body>
28
29     <div id="myDiv">
30         <h2>使用 AJAX 修改该文本内容</h2>
31     </div>
32     <button type="button" onclick="loadXMLDoc()">修改内容</button>
33
34 </body>
35
36 </html>
```

浏览的效果见图 3.18。



图 3.18 ajax 请求资源

点击“修改内容”按钮后会把服务器上的文件/proxy.pac 的内容替换 div 元素的内容。采用 JavaScript 编写 ajax 的步骤如下：

创建 XMLHttpRequest 对象（参考上图中 11 和 14 行代码）→给此对象的 onreadystatechange 事件定义需要执行的代码（也就是从服务器获取到数据后该怎么做，参见第 16 至 20 行代码）→调用对象的 open 方法来定义如何从服务器上获取数据（参见第 21 行代码：以异步方式(true) GET 服务器上的/proxy.pac 文件。当然还可以同步方式(false) POST 服务器上的文件）→调用对象的 send 方法正式向服务器发请求（参见第 22 行代码）。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

XMLHttpRequest 对象的几个常用的属性、方法和事件：

responseText 属性：保存了后端返回的数据（字符串类型），一般在 load 事件的处理函数中访问此属性，比如 console.log(this.responseText)。

status 属性：以整数表示的后端的反馈状态，比如 200 表示成功。

timeout 属性：请求的超时毫秒数，在正式请求之前设置，当请求超时时触发 timeout 事件。

upload 属性：返回一个 XMLHttpRequestUpload 对象，此对象几个有用的事件：loadstart(开始上传时触发)、progress(上传过程中周期性触发，可以提实现上传进度)、load(上传完成时触发)。

abort()方法：取消已经发出的请求。

open(method, url[, async, user, password])方法：初始化一个请求（还没发出请求）。async、user 和 password 参数是可选的。参数 method 定义请求的方法，可取值"GET"、"POST"、"PUT"和"DELETE"，参数 url 定义请求的后端，参数 async 是布尔类型，默认为 true，表示异步请求，这样 send()方法会立即返回。如 open("POST", "php/login.php")。

send([body])方法：正式向后端发起请求。可选参数 body 存放提交的数据，如 send("user=Alice&pass=123456")。

setRequestHeader(header, value)方法：设置请求头部的值，如 setRequestHeader("Content-Type", "application/x-www-form-urlencoded")。本方法必须在 open()之后 send()之前调用。

load 事件：请求成功完成时触发，通常就在本事件的处理函数中处理反馈的数据，比如显示在页面上。

timeout 事件：超过规定的时间（由触发 timeout 属性定义）时触发。

error 事件：请求出错时触发。

下面看一个比较完整的例子：

```
const xhr = new XMLHttpRequest();
xhr.timeout = 6000;
xhr.addEventListener('timeout', () => {
    xhr.abort();
    console.log("请求超时");
});
xhr.addEventListener("load", function () {
    //请求返回的数据
    console.log(this.responseText);
});

xhr.upload.addEventListener("progress",(evt)=>{
    //处理进度
    if (evt.lengthComputable) {
        console.log(evt.loaded / evt.total * 100) + "%";
    }
});


xhr.open('POST', 'php/login.php');
xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
xhr.send('user=Alice&password=123456');
```

fetch()方法的调用语法请参阅“§6.7.4 ServiceWorkerGlobalScope”小节。下面的例子采用 fetch()方法实现上面例子的功能：

```
fetch('/proxy.pac')
    .then(response => { return response.json(); })
    .then(myJson => { document.getElementById("myDiv").innerHTML = myJson; })
    .catch(error => console.log('获取资源出错: ', error));
```

下面的例子发送 JSON 数据：

```
var url = 'https://example.com/profile';
var data = { username: 'example' };

fetch(url, {
    method: 'POST', // 或 'PUT'
    body: JSON.stringify(data),
    headers: new Headers({
        'Content-Type': 'application/json'
    })
}).then(res => res.json())
    .then(response => console.log('成功:', response))
    .catch(error => console.error('失败:', error));
```

下面的代码上传一个文件：

```
var fileField = document.querySelector("input[type='file']");

var formData = new FormData();
formData.append('username', 'abc123');
formData.append('avatar', fileField.files[0]);

fetch('https://example.com/profile/avatar', {
    method: 'PUT',
    body: formData
})
    .then(response => response.json())
    .catch(error => console.error('失败:', error))
    .then(response => console.log('成功:', response));
```

下面的代码上传多个文件：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>上传多个文件</title>
</head>

<body>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    <input type="file" multiple>
    <button>上传</button>
    <script>
        document.querySelector("button").addEventListener("click", (event) => {
            var photos = document.querySelector("input[type='file'][multiple]");

            var formData = new FormData();
            // formData 只接受文件、Blob 或字符串，不能直接传递数组，所以必须循环嵌入
            for (let i = 0; i < photos.files.length; i++) {
                formData.append('files[]', photos.files[i]);
            }

            fetch('upload.php', {
                method: 'POST',
                body: formData
            })
                .then(response => response.json())
                .then(response => console.log('成功上传的文件数:', response.files))
                .catch(error => console.error('失败:', error));
        });
    </script>

</body>

</html>
```

后端 upload.php 的代码如下：

```
<?php

$countfiles = count($_FILES['files']["name"]);
$upload_location = "uploads/";

$count = 0;
$valid_ext = array("pdf", "doc", "docx", "jpg", "png", "jpeg");
for ($i = 0; $i < $countfiles; $i++) {
    $filename = $_FILES['files']['name'][$i];
    $path = $upload_location . $filename;
    $file_extension = pathinfo($path, PATHINFO_EXTENSION);
    $file_extension = strtolower($file_extension);

    if (in_array($file_extension, $valid_ext)) {
        if (move_uploaded_file($_FILES['files']['tmp_name'][$i], $path)) {
            $count += 1;
        }
    }
}

echo json_encode(array('files'=>$count));
?>
```

第 4 章

网页结构：HTML5

HTML5 的元素就是网站开发人员告诉浏览器如何对待相应的内容，比如我要在页面上显示“数学：100 分”，就告诉浏览器：“麻烦你在单独的段落中显示它，谢谢哦！”，可是浏览器听不懂人话呢，此时只能用段落元素 了——把“ 数学：100 分 ”代码写入网页文件即可。等一下，我都考了 100 分了，“你要突出显示‘100 分’才对得起我付出的努力！”。行，把刚才的代码改改，对，改成“ 数学：100 分 ”就可以了。

由于网页内容涉及文字、图片、音频、视频、表格、按钮、输入框等等众多类型，为此 HTML5 引入 10 大类共 112 个元素（截止到 2021 年 2 月），另外引入了诸如拖放、web 存储、推送通知等很多高级应用。而且开发人员可以采用 JS 语言的 customElements.define()方法自定义新的元素。学习网页结构，就是学习如何使用 HTML5 的这 112 个元素和众多的高级应用。

读者们，准备方便面，我们要开始派雪山过草地了。哈，这啃方便面的峥嵘岁月啊，我要赞美你！

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

4.1 元素简介

HTML5 提供了 10 大类共 112 个元素，具体请参阅 “《HTML5&CSS3 开发大起底》.xlsx” 文件中的“HTML5 元素速查表”工作表，这些元素都遵循如图 4.1 所示的语法：

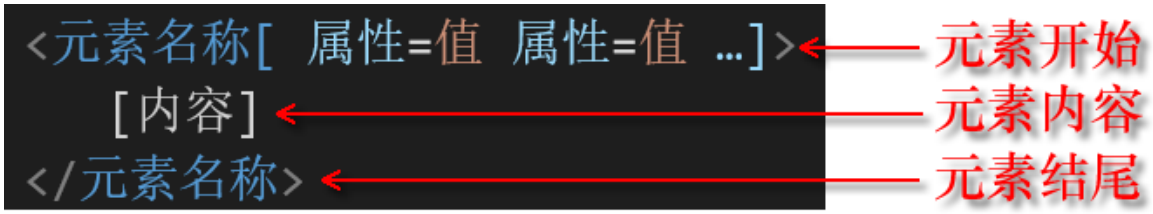


图 4.1 HTML5 元素语法

语法中的三行也可以写在一行中。元素内容中还可以包含更多的元素，嵌套元素形成父子关系。不过有的元素没有中括号部分，有的元素没有“元素内容”部分，有的元素没有“元素结尾”部分。先看如图 4.2 所示的例子，权当热热身。

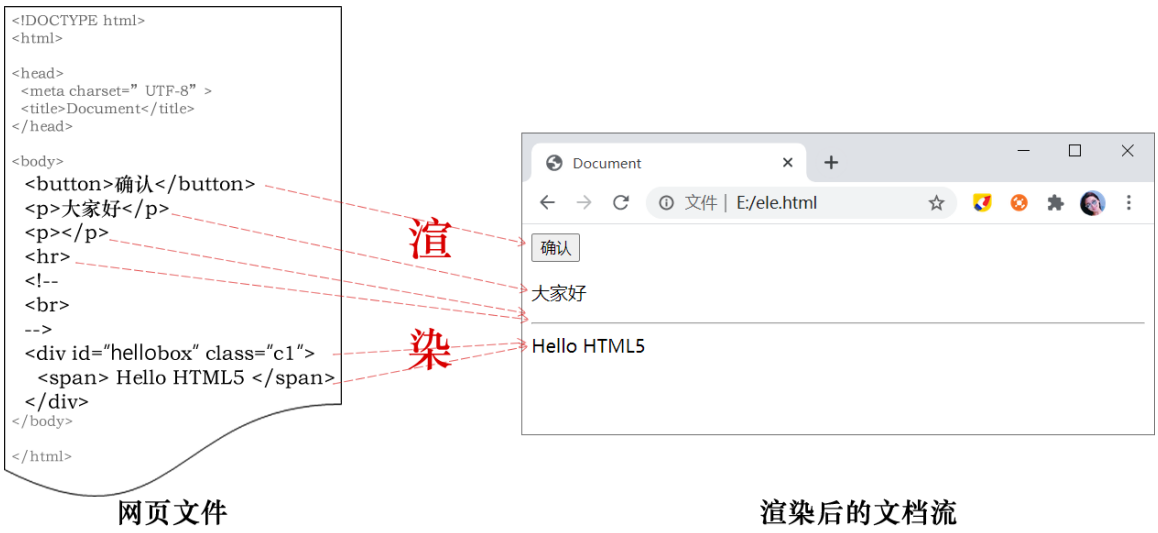


图 4.2 浏览器渲染页面

如图 4.2 左侧所示的网页文件代码，浏览器从上至下处理，遇到<button>元素就画出一个叫“确认”的按钮，遇到第一个<p>元素，就在下一行中显示“大家好”，接下来遇到第二个<p>元素就空一行（行高为 0），遇到<hr>就在新行内绘制一条水平线，忽略<!--和-->之间的内容，因为它是注释，遇到 的元素就开辟新的一行，然后处理元素，因为它是 的子元素，因此就在当前行内显示“ Hello HTML5 ”。由此这般，浏览器依照网页文件中的元素绘制页面的过程称为渲染，渲染后的页面见图 4.2 右侧。其实渲染还包括执行 JS 代码和应用 CSS3 样式。

在如图 4.1 所示的元素语法的“元素开始”中可以添加若干个属性，所有元素都支持的属性称为通用属性，否则称为专用属性。如在上面的例子中， 元素添加了两个通用属性：id 和 class，等号右侧部分是属性的值。表 4.1 列出了目前浏览器都支持的通用属性。

表 4.1 元素的通用属性

属性名	解析	举例
accesskey	定义访问元素的快捷键	<button accesskey="o">OK</button>，按 Alt+o 相当点击 ok 按钮
autocapitalize	输入字母时是否自动转换为大写字母，off | none - 不转换为大写，on | sentences - 句子的首字母转为大写，words - 单词的首字母转为大写，characters - 所有字母转为大写本属性对于物理键盘无效。	<input autocapitalize="words">
class	定义类。便于样式选择器通过类来选择元素	我的简历……
contenteditable	定义用户是否能修改元素内容，TRUE - 能修改，FALSE - 不能修改。修改的内容并没有保存到 Web 服务器上。开启整个页面编辑模式：document.designMode="on"	来，修改我啊！
data-*	自定义属性，用于存放用户的数据，在 JS 中采用 dataset.*访问属性的值。data-*中允许出现‘-’，但在 dataset.*中采用驼峰写法，如 data-birth-day 对应 dataset.birthDay。	我的年龄 你好
dir	定义文字书写方向。ltr - 左到右，rtl - 右到左，auto - 由元素内容决定。	Hello HTML5!
draggable	定义元素能否被拖动(按住鼠标左键并移动鼠标)，TRUE - 能，FALSE - 不能，auto - 采用浏览器的默认配置。	拖动我试试
hidden	隐藏元素，不显示。用法是：hidden 或者 hidden="hidden"	<div hidden>看不到我 我也不显示
id	给元素定义唯一的标识符，用于连接、样式和 JS 定位元素。	<button id='ok2'>确认</button>

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

lang	定义语言。en - 英语、zh - 简体和繁体中文、fr - 法语、ja - 日语、ru - 俄语、es - 西班牙语、ar - 阿拉伯语	<form lang="zh"> <input type="range" name="rate1"> </form>
spellcheck	是否作拼写或语法检查，空|TRUE - 作检查、FALSE - 不检查	<p spellcheck>是否对元素内容进行拼写检查
style	内嵌样式表	新产品
tabindex	定义元素的 tab 键控制序号，值可为：0,1,2,3,4,…	吃饭 <input tabindex="5">
title	定义鼠标停靠时的提示信息	<abbr title="世界卫生组织">WHO</abbr>
translate	当对页面做本地化翻译时,定义是否翻译元素的属性值和内容，空|yes - 翻译，no - 不翻译	<p translate>我是流氓我怕谁 给你颜色看看

HTML5 新增的通用属性 contenteditable 有点意思，就是我们能在页面上直接修改元素的内容，现在很多所见即所得的在线网站设计都是采用了此属性，比如 facebook 网站上的评论输入框。

至于元素的专用属性，我们以后在具体介绍各个元素时再来讲解。

这 112 个元素分为三大类，即块元素（block）、内联元素（inline）和内联块元素（inline-block），这主要根据元素在父元素中是否占据整行来作区分。

块元素（block）

块元素内容在渲染后的文档流中要占据一行，尽管元素内容本身宽度可能很小，这时左/右留下空白，也不管在网页文件中块元素是否与其他元素在同一行内，比如图 4.2 所示的代码，把<hr>移到上一行，并不影响渲染后的页面效果。“一行”的意思就是横跨父元素的宽度。 元素是最常用的块元素，其他的还有<h1>、<h2>、<h3>、<h4>、<h5>、<h6>、 、<form>、<header>、<footer>、<nav>、<section>和<article>元素等等。一个块元素可以通过添加 display:inline 样式强行转化为内联元素，或者通过 display:inline-block 样式强制转化为内联块元素。块元素可以使用 CSS3 样式设置宽度（width）和高度（height）值，尽管 width 可能设置得非常小，但还是要占据一整行，如图 4.3 所示。

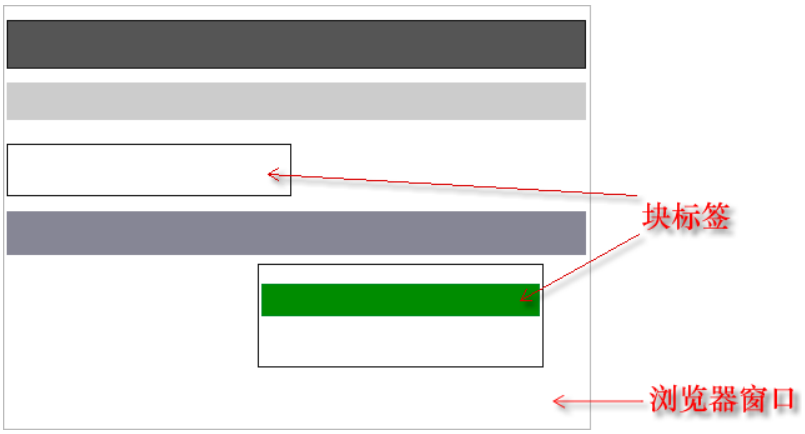


图 4.3 块元素占据整行

下面的代码把 元素转化为内联元素：

```
<div style="display:inline">我现在是内联元素了</div>
```

内联元素（inline）

按照元素内容实际宽度占用页面空间，所以在页面的同一行内可以排列多个内联元素。最典型的内联元素有、<a>、<img>、<button>、<input>等。对内联元素添加 width 和 height 样式无效。一个内联元素可以通过添加 display:block 样式强行转化为块元素，或转化为内联块元素。参考图 4.4。

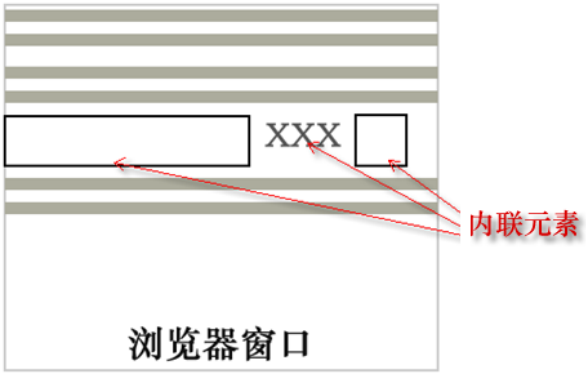


图 4.4 内联元素

内联块元素（inline-block）

由元素内容的实际宽度确定，或者由 width 样式定义的宽度占据一行内的部分空间，同一行内只要有足够多的空间，则允许排列更多的非块元素。给内联块元素添加 width 和 height 样式有效。参考图 4.5。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

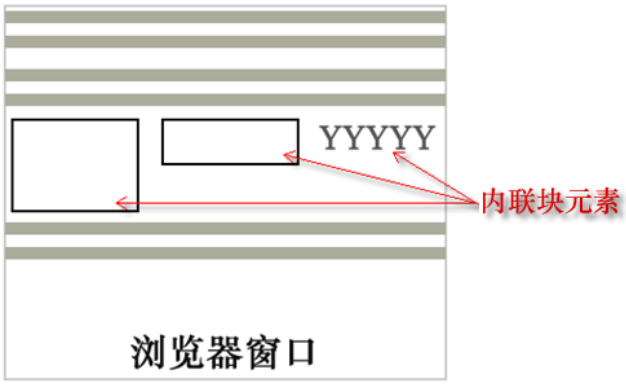


图 4.5 内联块元素

内联快元素与内联元素最主要的区别是后者不能设置宽度和高度（设置了也无效）。后续小节按照元素的 10 大类，详细介绍各个元素的用法。

备好几箱方便面，吃苦两个月，前途好得不要不要的。

4.2 页面结构元素

“《HTML5&CSS3 开发大起底》.xlsx”文件中的“HTML5 元素速查表”工作表中的“网页结构”分类列出的四个元素用于构建网页文件的框架。为方便起见，以后统一写成“HTML5 元素速查表 • 网页结构”的形式。

本节介绍这些元素：<!doctype html>、<html>、<head>、<body>。

图 4.6 就是一个 HTML5 页面文件的结构，在一个页面文件中，这四个元素可以省略，但至多只允许出现一次，而且<head>和<body>必须是<html>的子元素。

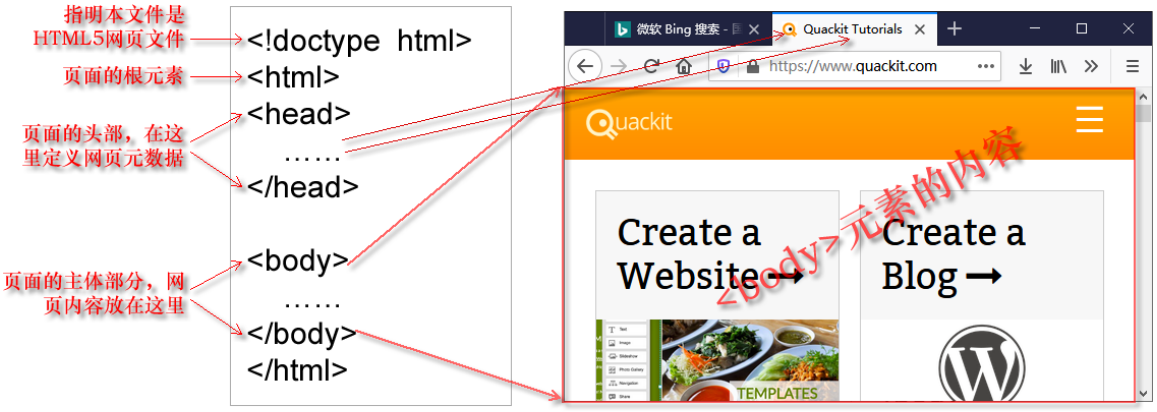


图 4.6 HTML5 网页结构

<html>元素有一个可选的专用属性 manifest，指明一个资源清单文件，清单文件中列出了需要在本地缓存的网站资源清单，例如<html manifest="shop.appcache">，不过这种本地缓存技术已经淘汰了，取而代之的是服务线程技术，具体参考“§6.7 服务线程”小节。<html>的通用属性

lang 定义页面的语言，浏览器、在线翻译和搜索引擎根据此属性来判断页面的语言，如果与本地电脑的语言不同，在线翻译就会提示要不要翻译页面内容，强烈建议给<html>元素的 lang 属性设置成页面内容的真实语言，这样其他语种国家的人浏览网站时就会提醒翻译页面内容，见图 4.7 所示。不过浏览器在渲染页面时会先参考<meta>元素的属性 charset 定义的语言，如果没有，再参考<html>元素的 lang 属性，如果还没有，就采用浏览器的默认语言。

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>这是英文页面</title>
</head>
<body>
    <p>How are you?</p>
    <p>早上好</p>

</body>
</html>
```

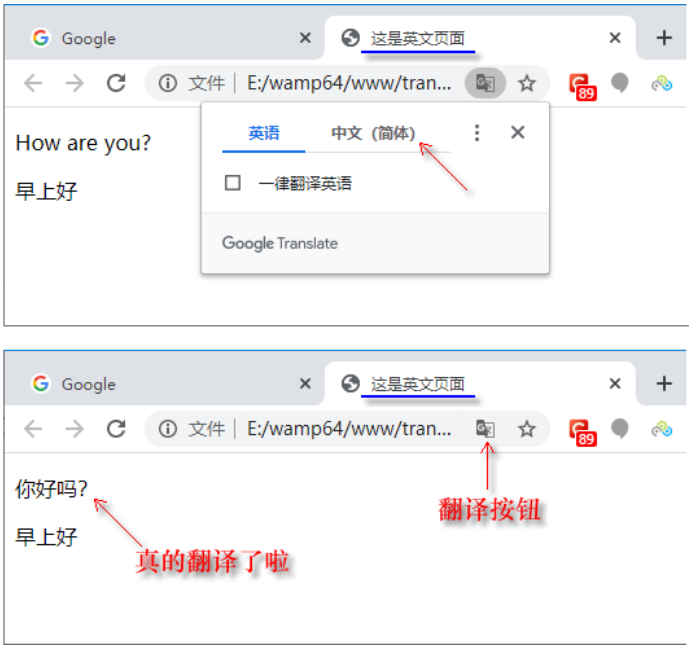


图 4.7 浏览器提醒翻译页面内容

<head>元素主要用于定义页面的元数据，比如语言、页面标题、基础域、样式表等，这里的内容不会显示在浏览器的视口中，但是会影响<body>元素内容的显示效果。

<body>元素就是页面的主体部分，元素的内容会显示在浏览器中。我们开发网页，大部分精力都花在设计<body>的元素内容上。除了通用属性外，<body>不再使用专用属性。但是有一些涉及全局范围内的事件需要注意，这些常用的事件见表 4.3。

表 4.3 常用的事件

事件名称	说明
onafterprint	页面被打印完毕时触发
onbeforeprint	用户请求打印页面时触发
onbeforeunload	离开页面时触发，要先于 onunload 事件发生
onblur	页面失去焦点时触发
onerror	页面不能正确加载时触发

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

onfocus	页面获得焦点时触发
onhashchange	当前URL的片段标识部分(以'#'开始的部分)发生改变时触发。如http://x.y/a.html#1变为 http://x.y/a.html#2
onload	页面加载完毕时触发
onmessage	页面收到消息时触发
onoffline	网络断线时触发
ononline	网络从断线中恢复时触发
onpopstate	window.history 改变时触发，如点击浏览器的回退按钮
onredo	页面执行重做时触发
onresize	视窗尺寸改变时触发。如果拖动鼠标调整浏览器窗口尺寸，会连续触发此事件
onstorage	存储内容（localStorage / sessionStorage）发生改变时触发。只有非本页面改变存储才会触发，比如手工在开发者工具中修改，或者其他页面修改，才会触发
onundo	页面执行撤销时触发
onunload	离开页面时触发

最常用的事件有 onload 和 onunload，比如在页面加载完后做一些工作，这时相应 onload 事件即可，在比如在关闭页面时保存一些数据，我们响应 onunload 事件即可。看例子：

```
<body onload='ok()' onunload='save()'>
    <script>
        function ok() {
            console.log('页面加载完毕! ');
        }

        function save() {
            localStorage.time = (new Date()).toLocaleString();
        }
    </script>
</body>
```

网页加载完之后调用 ok()函数，关闭网页时调用 save()函数。语句 (new Date()).toLocaleString() 获取系统时间，至于如何在本地保存数据，请参考“§6.4 Web 存储”小节。

4.3 文档元数据

文档元数据就是告诉浏览器关于本页面的一些描述信息，如网页标题、页面语言、样式表、网页图标等，正如你作自我介绍那样：“我叫二狗子，来自旮旯县，……。”你介绍的内容其实就是你的元数据。“HTML5 元素速查表·文档元数据”分类列举了全部的文档元数据元素，这些元素一般都放在页面的头部，即<head>元素内，作为其子元素。

本节介绍这些元素：<base>、<link>、<meta>、<style>、<title>。

<base>元素

定义网站的基础 URL 和默认的打开页面链接的方式，它的语法：

```
<base href="xxx" || target="yyy">
```

语法当中的“||”表示左右部分选一个或全选，但不能都不选。<base>通过其属性 href 定义网站的基础 URL，以后打开页面中的相对 URL 链接时就会在左边拼接基础 URL，从而形成一个绝对 URL。通过 target 属性定义默认打开新页面的方式，可能的取值有：

_self：在浏览器的当前视窗中打开新的链接，从而覆盖了旧的页面内容。这是默认值。

_blank：在新的浏览器视窗中加载链接，例如新打开一个标签页，并在其显示页面内容。

_parent：在父页面所在的视窗中加载链接，如果没有父页面，就等同于_self。如果当前页面在<iframe>中，那么<iframe>元素所在页面就是父页面。

_top：在最顶端的页面的视窗中加载链接。没有父页面的页面就是最顶端的页面，显然如果当前页面没有父页面，则_top 等同于_self。

看例子，参见图 4.8。

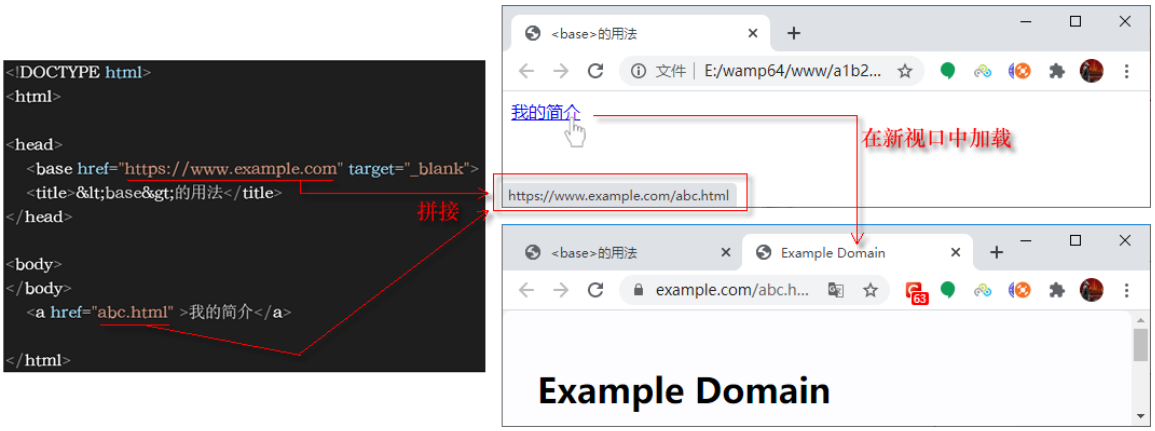


图 4.8 <base>元素的作用

<base>元素的 href 属性的值也可以是相对 URL（最后一个字符为“/”），实际上就是一个目录，如'musics/'、'/books/'、'news/china/'，如果第一个字符为“/”，则表示绝对目录，否则就是相对目录。如果是一个绝对目录，那么就这样拼接：location.origin+<base>元素 href 属性的值

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

+相对链接，如果是相对目录，就这样拼接：document.baseURI+<base>元素 href 属性的值+相对链接。举个例子，当前页面的 URL 为 https://www.abc.com/abc/cba.html，cba.html 文件当中有如下代码：

```
…
<base href="/musics/" target="_blank">
…
<a href="b3.html" >周杰伦</a>
…
```

此时的 location.origin 属性的值等于 https://www.abc.com，而 document.baseURI 等于 https://www.abc.com/abc/，所以“周杰伦”这个链接的绝对 URL 为 https://www.abc.com/musics/b3.html。

实操练习：假设上面代码中<base>的 href='musics/'，此时“周杰伦”的绝对 URL 又是什么？

<link>元素

链接外部资源，如外部样式表、网站图标等，它的语法如下：

```
<link rel= href= media= title= crossorigin= type= … >
```

先看看几个常用的例子。

链接外部样式表文件：<link rel="stylesheet" href="styles/main.css">

指定网页图标：<link rel="icon" href="favicon.ico">，网页图标文件名默认为 favicon.ico，参考图 4.9。



图 4.9 网页图标和标题

网页图标的做法：浏览网站 https://www.bitbug.net/，上传图片，然后转换为 ico 文件，下载后放在与 index.html 文件所在的目录中，并改名为 favicon.ico 即可。

下面的例子在加载网页时不加载样式表文件，以后如果通过 JS 语句把 disabled 属性的值改为 false 或者干脆删除此属性，浏览器就会马上加载样式表文件：

```
<link rel="stylesheet" href="duome.css" disabled=true>
```

下面的例子是针对手机和电脑，加载不同的样式表：

```
<link rel="stylesheet" href="mobile.css" media="screen and (max-width: 600px)">
```

```
<link rel="stylesheet" href="computer.css" media="screen and (min-
width: 900px)">
```

media 属性的值要遵循@media 的语法，具体参见“§ 5.6.9 媒体查询”小节。

下面的例子更有意思：

```
<link href="common.css" rel="stylesheet">

<link href="default.css" rel="stylesheet" title="Default Style">
<link href="simple.css" rel="alternate stylesheet" title="Simple">
<link href="insane.css" rel="alternate stylesheet" title="Insane">
```

本例子表明一定会用 common.css 文件中的样式，然后再选择 default.css（默认），用户可以在 default.css、simple.css 和 insane.css 中挑选一个，相当于更换网站页面皮肤。<link>元素的 title 属性只能集合 rel 属性的 stylesheet 值使用，如果 rel 属性取其他值，title 属性无效。在火狐浏览器上，参考图 4.10 选择不同的样式。

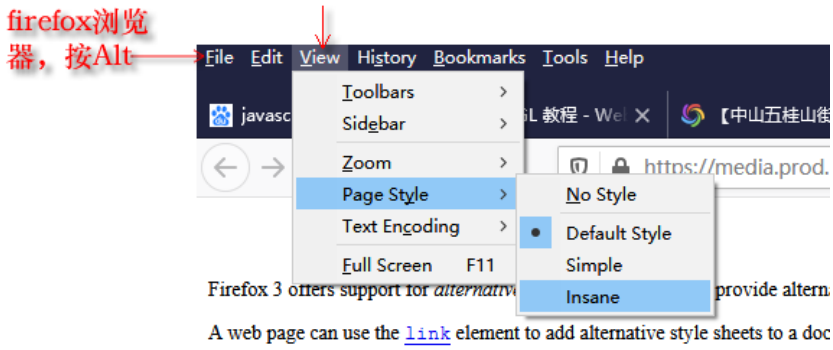


图 4.10 加载不同的样式

备选样式所在的<link>元素中，要给 title 属性一个值，同时必须给 rel 属性的值加上“alternate”。默认的样式，其 title 属性值只能是“Default Style”。

rel 属性的值是以空格隔开的若干链接类型名字，除了上面已经介绍的 alternate、stylesheet、icon 外，其他的一些常用的链接类型见下表。

表 4.5 <link>元素 rel 属性的取值

链接类型	说明	例子
author	指向介绍作者的页面链接，这个作者就是当前页面文档的作者。搜索引擎或者网络爬虫会根据作者信息找到本页面文档。	<link rel='author' href='zhangsan.html' >
help	链接的页面对本页面做进一步的说明	<link rel='help' href='more.html' >
license	链接指向描述许可信息的文档，关于当前页面文档的许可信息	<link rel="license" href="copyright.html">

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

prefetch	建议浏览器预取资源，这资源用户很可能在后续导航的页面上会用到。可结合 as 属性一起用	<link rel='prefetch' href='details.html' type='text/html' >
preload	告诉浏览器预加载当前页上的一个资源,必须结合 as 属性。preload 的优先级要高于 prefetch	<link rel='preload' href='bigphoto.png' type='image/png' as='image' >，如果此后用户点击某个按钮要用到这个大图片，他就不用等待很久了
search	指到一个搜索链接，专门搜索当前网站内的页面，JS 的 AddSearchProvider()方法提供类似的功能。	<link rel="search" href="movies-search.html">

rel 属性的 preload 值必须和 as 属性一起使用，as 属性定义预加载资源的类型，它的值主要有：audio - 音频、document - 页面文档、script - 脚本、font - 字体、image - 图片、video - 视频等，比如：

```
<link rel="preload" as="audio" href="beijingbeijing.mp3">
```

<link>元素的 crossorigin 属性定义跨域资源，它取值只能是 anonymous 或 use-credentials，两者的区别是：请求跨域资源时后者要求提供身份凭据（如 cookie），前者不提供。

```
<link rel="icon" href="http://www.others.com/news.png" crossorigin="anonymous">
```

跨域资源机制给我们提供了一个手段，可以把样式表、JS 代码、音视频、图片等通过 CDN 就近存放，这样能让用户体验更快的加载速度。

type 属性定义资源的 MIME 类型，比如 type= "image/png"。最常见的 MIME 类型有 audio/mpeg（MP3 音频）、video/mpeg（MPEG 视频）、text/javascript（JS 代码）、image/jpeg（jepg 图片）、image/png（PNG 图片）、application/pdf（PDF 文件）、font/ttf（TrueType 字体）、text/xml（xml 代码）、application/json（JSON 数据）、text/html 请（HTML 文件）。其他的 MIME 类型请参考网站 https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types。

<meta>元素

定义其他元数据元素（<base>、<link>、<script>、<style>和<title>）无法定义的元数据，如文档字符集。它的语法如下：

```
<meta [http-equiv= | name=] && content=  charset= >
```

charset 属性定义文档的字符编码格式，它的值一般都是 utf-8。http-equiv 和 name 属性不能同时存在，且必须搭配 content 属性。name 搭配 content 定义“键-值”对格式的元数据，name 属性定义键名，content 属性定义值，如<meta name= 'author' content='Joe Bidd'>定义了元数据 author='Joe Bidd'。键名虽然可以任意取，但存在一些约定俗成的标准键名，见表 4.6。

表 4.6 标准元数据之键/值对

键名	说明	例子

application-name	给 web app 取一个名字，此名一般是<title>定义的页面标题的前缀。	<meta name='application-name' content='优酷' >，而一个页面的标题可能是“优酷-卢旺达饭店-404”
author	页面文档的作者	<meta name='author' content='李四' >
description	页面简介。浏览器一般把它作为书签描述	<meta name='description' content='良心茶，放心喝' >
generator	产生本页面文档的工具名称，手工编码的页面不要使用此元数据	<meta name="generator" content="Discuz!6.1.0">
keywords	本页面的关键字，关键字之间用逗号分开	<meta name="keywords" content="诸暨,深山野茶,自制">
viewport	定义初始视窗尺寸（对移动设备有效），此时 content 的值是若干个用逗号分隔的“键/值”对（表 4.7）。视窗就是显示页面内容的区域	<meta name="viewport" content="width=device-width, initial-scale=1.0">

表 4.7 <meta>元素的 content 属性的键/值对

键	**值**	**描述**
width	正整数或者 device-width	以像素数表示的视窗宽度
height	正整数或者 device-height	以像素数表示的视窗高度，但没有一款浏览器会使用此参数
initial-scale	取值范围是[0.0,10.0]	定义设备宽度（纵向模式下的设备宽度或横向模式下的设备高度）与视窗大小之间的比率
maximum-scale	取值范围是[0.0,10.0]	定义最大缩放比率
minimum-scale	取值范围是[0.0,10.0]	定义最小缩放比率。 minimum-scale ≤ maximum-scale
user-scalable	yes 或 no	用户是否可以缩放页面
viewport-fit	auto, contain 或 cover	auto - 不会影响初始布局视窗，整个网页都是可见的。 content - 视窗尺寸调整到显示屏的最大矩形尺寸，比如“刘海”两侧的空间被排除在外。 cover - 视窗覆盖整个显示屏。 强烈建议使用安全区域插入变量，以确保重要内容不会出现在显示屏之外（如被“刘海”遮盖）

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

<meta>元素的最后一个属性 http-equiv 结合 content 属性用于定义 http 响应的头部，它允许的取值如下：

content-security-policy - 允许网站开发者为当前页面定义内容安全策略，以防止跨域攻击。此时的 content 属性的值就是以逗号分隔的策略规则，如 content="img-src 'http://*.example.com', media-src '*.media.com' 'self' "。关于 CSP 更详细的介绍请参考 https://developer.mozilla.org/zh-CN/。

content-type - 定义资源类型。此时 content 属性的值只能是"text/html; charset=utf-8"。

default-style - 定义默认样式表的名字。样式表的名字参见<link>元素的 titil 属性。

refresh - 定义页面重新加载或者重定向到其他页面的秒数，显然重新加载页面是周期性的。具体多少秒由 content 属性定义。定义刷新和重定向的<meta>不能同时在一个页面文件中。

看下面的几个例子：

```
<meta http-equiv="content-security-policy" content="img-src 'http://*.news.com'
'self'">
<meta http-equiv="default-style" content="Insane">
<meta http-equiv="refresh" content="500"><!--每隔 500 秒刷新一次-->
```

下面的例子是 100 秒后重定向到 orders/settle.html 页面：

```
<meta http-equiv="refresh" content="100; url=orders/settle.html">
```

<style>元素

嵌入 CSS 样式表，它的语法如下：

```
<style[ media= nonce= title= ]>样式代码</style>
```

<style>元素有三个可选属性，其中 media 属性定义样式适用的媒介，它的值要遵循@media 的语法，具体介绍参见“§5.6.9 媒体查询”小节，比如 media="screen and (min-width: 900px)"表示本样式表只在宽度至少 900px 的显示屏上使用。title 属性的作用与<link>的 title 属性相同。nonce 意为只使用一次的数字，也就是说每次刷新页面，nonce 的值都不同。网站内容安全策略（CSP）要求样式表必须单独保存在外部文件中，但是我就要对着干，偏要嵌入在在页面文件中，这时只好启用<style>元素的 nonce 属性，每次用户请求页面文件时，后台临时产生一个随机字符串，比如"uG2bsk6JIH923"，把随机字符串赋给 nonce 属性，同时给随机字符串附加前缀“nonce-”后加到 style-src 安全策略指令中，例如：

```
   <style nonce="uG2bsk6JIH923">
      样式表;
   </style>
```

安全策略：

```
Content-Security-Policy: script-src 'nonce-uG2bsk6JIH923'
```

不建议在静态网站中使用 nonce，因为没法做到“只使用一次”，只有动态网站才行。下面的例子表明：在手机上，段落文本为红色。

```
  <style media="screen and (max-width: 540px)">
        p {
            color: red;
        };
```

```
  </style>
```

<title>元素

定义网页的标题，标题被显示在浏览器的标题栏或元素页上，参考“图 4.9 网页图标和标题”，它的语法如下：

```
<title>网页标题</title>
```

<title>元素没有专用属性。

4.4 脚本元素

嵌入 JS 代码块需要用到脚本元素，一共有四个脚本元素，请参阅“HTML5 元素速查表 • 脚本元素”分类，其中<template>和<slot>两个脚本元素的介绍放在“§6.12.3 HTML 模板”小节。

本节介绍这些元素：<script>、<noscript>、<template>、<slot>。

<script>元素

嵌入数据或者可执行的代码，大部分情况下是嵌入 JS 代码。此元素允许出现在 HTML 文件的任何地方，出现的次数也没有限制。它的语法是：

```
<script[ async
defer crossorigin= integrity= nomodule nonce= referrerpolicy= src= type=]>[……]
</script>
```

各个属性介绍如下：

async - 凡是具有此属性的代码块会被并行加载(同时渲染 HTML)，一旦加载完毕就立即执行，这样在文档中排在后面的代码有可能先执行。注意：执行 js 和渲染 HTML 不能并行。

defer - 并行加载，但是代码被推迟到页面渲染后且触发 window 对象的 load 事件前执行。执行的顺序就是带有 defer 属性的<script>元素在文档中的排列顺序。

crossorigin - 参照<link>元素的同名属性。

integrity - 为资源的 hash 值，用于验证资源的完整性。浏览器加载资源后再做一次 hash，然后比较两个 hash 值，如果相等，表明资源没有被恶意篡改，否则浏览器拒绝执行代码。假如有一个 JS 文件 example.js，首先对它执行一次 hash 运算得到一个 HASH 值：

```
cat example.js | openssl dgst -sha384 -binary | openssl base64 -A
```

如果此命令返回的结果是 sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K，那么<script>元素写成：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
<script  src="example.js"  integrity="sha384-oqVuAfXRKap7fdgcCY5uykM6+R9GqQ8K">
</script>
```

如果 example.js 被黑客篡改，那么浏览器下载它之后再对它做 hash 运算，得到的 HASH 值完全不同，这时浏览器拒绝执行它。

nomodule - 指示支持 ES2015 模块的浏览器不要执行代码。一些老旧的浏览器不支持 JS 模块机制，这时就会执行<script nomodule>中的代码。

nonce - 请参考<style>元素的同名属性。

src - 资源的 URI。如果存在 src 属性，那么嵌入在<script>元素内部的代码被忽略。

referrerpolicy - 浏览器加载资源时的请求头部中有两个参数 Referrer Policy 和 Referer，<script>元素的 referrerpolicy 属性的值被赋给参数 Referrer Policy，同时控制头部参数 Referer 的值。本属性可取值：no-referrer：头部无 Referer；no-referrer-when-downgrade：当 https 页面请求 http 资源时头部无 Referer；origin：Referer 等于 location.origin；origin-when-cross-origin：请求跨域资源时 Referfer 等于 location.origin，否则等于 document.URL；same-origin：同域请求时 Referer 等于 location.origin，跨域时无 Referer；strict-origin：如果当前页面协议为 https，而请求资源的协议为 http，那么就没有 Referer 参数，否则 Referer 等于 location.origin；strict-origin-when-cross-origin：请求同域资源时 Referer 等于 document.URL，请求跨域资源时又分两种情况：如果当前页面协议为 https，而请求资源的协议为 http，那么就没有 Referer 参数，否则 Referer 等于 location.origin；unsafe-url：Referer 的值等于 location.origin+location.pathname，这是非常不安全的。看例子：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>script 元素的用法</title>
    <script src="main.js" referrerpolicy="origin"></script>
</head>

<body>
    <div>
        <p>Hello VScode!</p>
        <a href="#123">aaaa</a>
    </div>
</body>

</html>
```

浏览该网页后，进入开发者工具的 Network 的标签页，然后点击页面上的 aaa，效果见图 4.11。

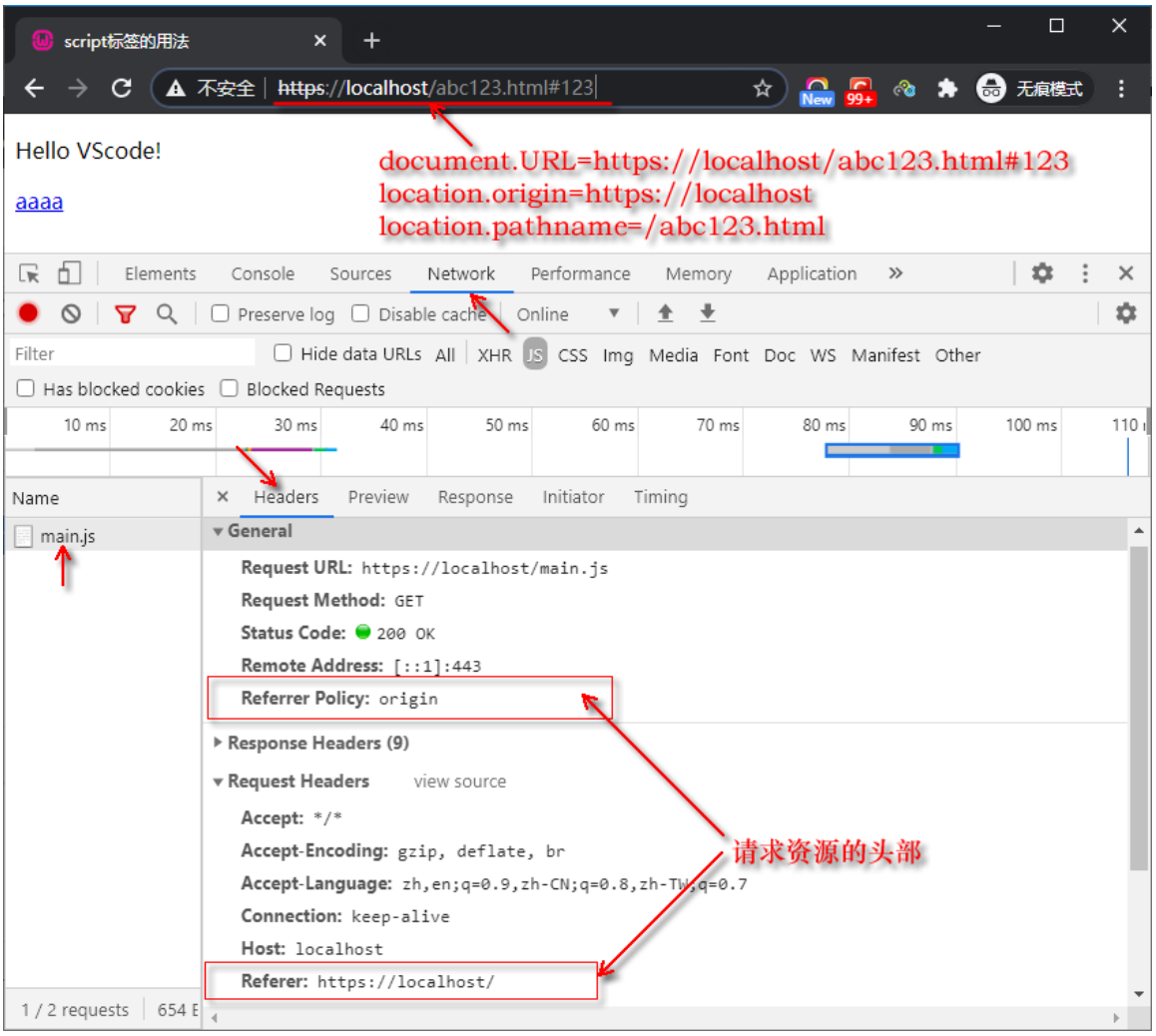


图 4.11 请求同域 js 资源

type - 定义引用的脚本类型，可取值：Javascript MIME 类型或者干脆省略 type 属性（普通 javascript 脚本）、module（JavaScript 模块）、其他值（数据块，浏览器不去处理）。当 type="module"时隐式包含了 defer 属性。

最后再来看一个例子：

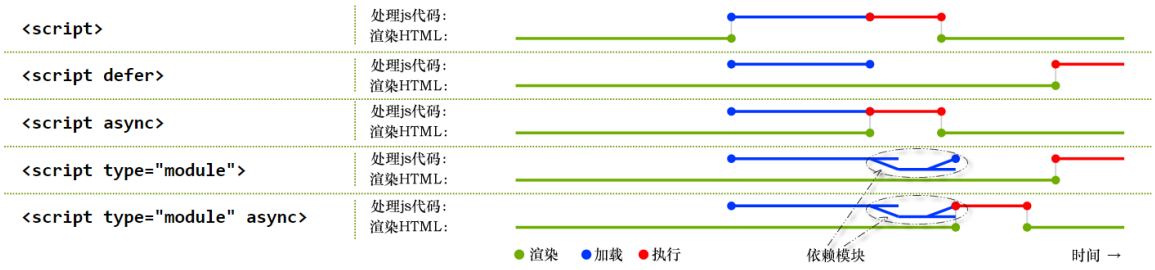
```
<script src="javascript.js" async></script>
……
<script>
    alert("Hello World!");
    console.log(window);
</script>
<script nomodule src="fallback.js"></script>
……
<script id="data" type="application/json">
    {
        "userId": 1234,
        "userName": "张三",
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
        "memberSince": "2020-10-01 08:20:00.000"
    }
</script>
……
<script defer>
    const userInfo = JSON.parse(document.getElementById("data").textContent);
    console.log("用户信息: %o", userInfo);
</script>
……
```

这个例子中，还用<script>元素嵌入数据，并且使用嵌入的数据。

至于 defer、async 和 type="module"的区别，请仔细参考下图：



<noscript>元素

如果浏览器不支持脚本语言（老版本或者禁止执行脚本），那么就渲染<noscript>的元素内容。例如：

```
<body>
    <noscript>
        <p>啊，糟糕，不能执行 JS! </p>
    </noscript>
    <p>Hello VScode!</p>
</body>
```

4.5 章节编排

针对文章章节编排和页面布局，HTML5 专门增加了几个语义元素。语义元素就是有意义的元素，即元素本身代表某种意义和作用，比如元素<header>包含“头部”的意思、元素<article>含有“文章”的意思。相反，无语义的元素本身不隐含任何意义和作用，比如 和元素都是无语义元素。HTML5 新增的部分语义元素参见“HTML5 元素速查表 • 章节”分类，这些语义元素都没有专用的属性。

本节介绍这些元素：<address>、<article>、<aside>、<footer>、<h1>,<h2>,<h3>,<h4>,<h5>,<h6>、<header>、<main>、<nav>、<section>。

我们在开发网页时要尽量根据内容来选择合理的语义元素，而不是简单地做全文替换，比如把文件中的全部 替换为<section>就不对。使用语义元素有几个好处：一是浏览器在渲染网页时效率会更高，二是增强了代码的可读性，三是有利于搜索引擎和爬虫检索，四是便于读屏器拾取内容。图 4.12 是采用语义元素做的一个页面布局。

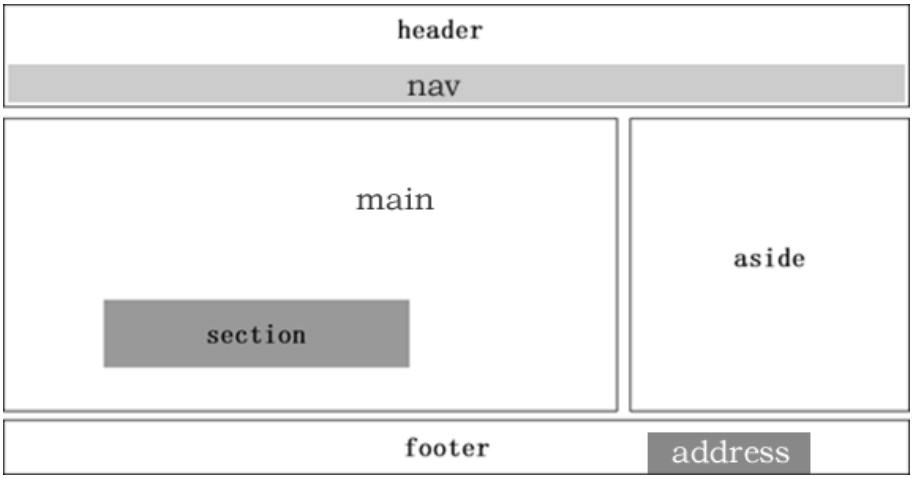


图 4.12 采用语义元素布局

图 4.13 所示的文章编排尽量采用了语义元素，完整的网页代码参见 chapter-4\index-1.html。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 4.13 采用语义元素编排文章

从这个例子可以看出，<article>中一般包含<header>、多个<section>、<footer>，而<section>中包含标题、嵌套更多的<section>和<article>等，<aside>的内容与<article>内容相关，里面包含标题和列表，但是一般不再嵌套<aside>。

<address>元素

通常用于显示联系方法，比如地址、电话、邮件等，它的语法如下：

```
<address>联系方式</address>
```

请参考图 4.14 所示的例子。

```
<address>
    你可以浏览我的网站 <a href="http://www.veryopen.org/">
        www.veryopen.org</a><br>
    或者给我发邮件 <a href="mailto:wlm@veryopen.org">
        wlm@veryopen.org</a>.<br>
    或者直接拜访我：<br>光明大队留几根村<br>
</address>
```



图 4.14 <address>元素举例

<article>元素

代表独立的内容，比如一篇文章、一条评论、一个新闻等，“独立”意味着它可以独立存在，不会因为周边的内容被删而影响其完整性。在语义上，<article>元素的内容要有一个标题，比如文章的题目，新闻的题目等。允许嵌套，但里外的<article>内容在语义上具有相关性，比如一篇博文的评论可以作为一个子<article>存在。语法如下：

```
<article>内容</article>
```

例子请参考图 4.13 和 chapter-4\index-1.html。

<aside>元素

代表与页面的主要内容相关的内容，其存在与否不会影响页面主要内容的完整性，比如文章作者的信息、相关内容的链接、参考数目、尾注、广告以及二级导航栏等。但是文章的署名、发布日期、版权声明以及“上一篇”、“下一篇”这种前后导航不适合使用<aside>元素。其语法如下：

```
<aside>内容</aside>
```

例子请参考图 4.13 和 chapter-4\index-1.html。

<footer>元素

代表页面、文章或者章节的尾部，相应地，<footer>一般就是<body>、<article>或<section>元素的最后一个子元素，显然它的内容一般与父元素的内容紧密相关，比如版权信息、联系方式、作者信息等。其语法如下：

```
<footer>内容</footer>
```

例子请参考图 4.13 和 chapter-4\index-1.html。

<h1>~<h6>元素

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

代表六级标题，主要用于给页面、文章、章节添加标题。在使用过程中需要注意几点：不要使用这些元素调整普通文本的字号，调整字号应采用样式属性 font-size；标题级别要连续，比如<h1>的下一级标题用<h2>，而不是使用 3 级或之后的级别；一个页面或一篇文章只用一个<h1>。标题元素和对应的字号参考图 4.15。

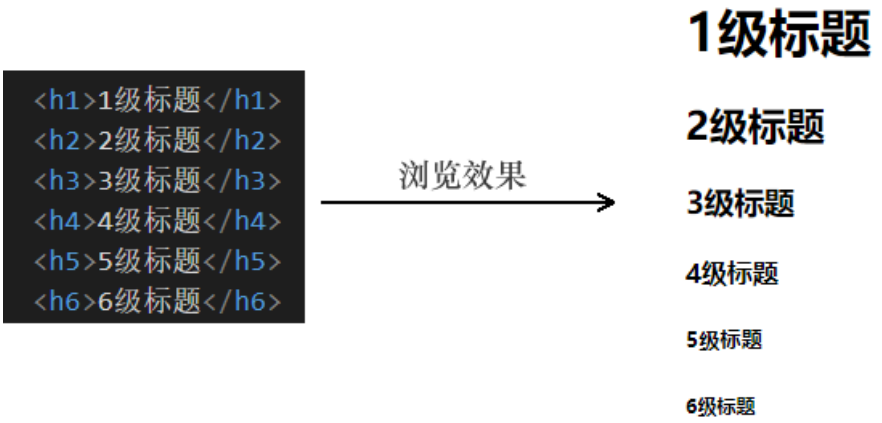


图 4.15 各级标题元素与对应的字号

它们的语法如下：

```
<h*>标题</h*>
```

例子请参考图 4.13 和 chapter-4\index-1.html。

<header>元素

与<footer>元素相对应，为页面、文章或章节的头部，相应地，<header>一般就是<body>、<article>或<section>元素的第一个子元素，元素内容是主要内容的引导部分，比如页面的导航、logo、公司名称等，再比如文章的首标题、作者、写作日期、摘要等。作为网站页面的头部，一般带有“公共性”——网站的全部页面的头部都一样，<footer>元素也具备“公共性”。

它的语法如下：

```
<header>内容</header>
```

例子请参考图 4.13 和 chapter-4\index-1.html。

<main>元素

代表与页面中心主题密切相关的主要内容，一般为<body>元素的子元素，且占据页面的大部分区域。页面间的共性内容不要放在<main>元素内，比如页面的头部、尾部、导航等。在一个页面文件中，不要出现多个<main>元素。它的语法如下：

```
<main>内容</main>
```

例子请参考图 4.13 和 chapter-4\index-1.html。

<nav>元素

标识页面上重要的导航区，如一级导航、二级导航。次要的导航一般不用<nav>，比如<footer>内的链接就不要用<nav>了。具体的导航菜单需要采用诸如无序列表实现。它的语法如下：

```
<nav>内容</nav>
```

看例子：

```
    <header>
        <nav>
            <ul>
                <li><a href='#'>首页</a></li>
                <li><a href='#'>产品</a></li>
                <li><a href='#'>服务</a></li>
            </ul>
        </nav>
    </header>
```

<section>元素

Section 意为章节、部分的意思，一般用在<article>、<header>、<footer>和<nav>元素中，<section>中一般采用<h1>~<h6>定义一个标题，通过标题我们能称呼一个<section>，一个元素（如<article>）下的全部<section>的标题组成一个大纲。没有标题的区域在文章中就是段落，用<p>元素即可；在页面中就是一块无语义的空间，用 元素即可。例如图 4.16 就是某个网站的页尾。

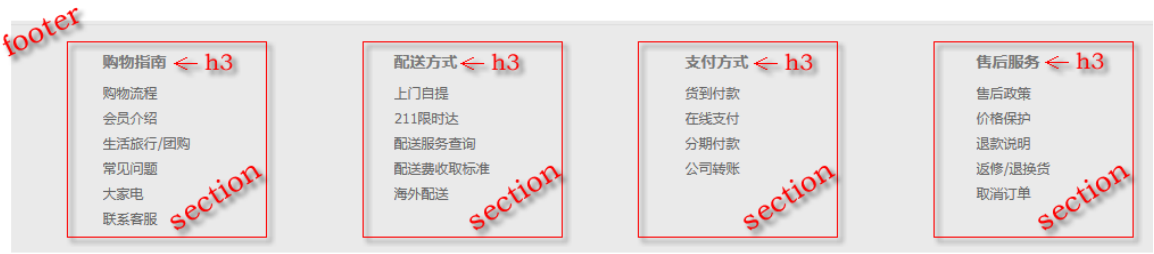


图 4.16 页尾中的节

<section>元素的语法如下：

```
<section>内容</section>
```

4.6 组织内容

用于组织和编排内容的元素请参阅“HTML5 元素速查表 • 组织内容”分类。

本节介绍这些元素：<dl>、<dt>、<dd>、 、<figure>、<figcaption>、<hr>、<ol>、<ul>、<li>、<p>、<pre>。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

<dd>、<dl>、<dt>元素

这三个元素定义一组术语，每个术语可能还存在解释，解释会缩进显示。<dl>包裹在最外面，<dt>包裹术语本身，<dd>包裹术语的解释，它们不一定是一一对应关系，比如有的术语没有解释，有时多个术语对应一个解释。看一个例子：

```
    <dl>
        <dt>术语 1</dt>
        <dd>解释……</dd>
        <dt>术语 2</dt><dt>术语 3</dt>
        <dd>解释……</dd>
        <dt>术语 4</dt>
        <dd>解释……</dd><dd>解释……</dd>
    </dl>
```

浏览效果图 4.17。

术语1
　　解释……
术语2
术语3
　　解释……
术语4
　　解释……
　　解释……

图 4.17 定义术语的元素

元素

这是个无语义的通用容器，可以包裹任何内容（包括嵌套更多的 元素），除非附加样式，否则 本身不会显示出来，当然里面的内容还是会显示出来的。如果能用<article>、<header>、<footer>、<nav>、<section>等语义元素就不要用 。它的语法很简单：

```
<div>任何内容</div>
```

<figure>、<figcation>元素

<figure>包裹页面中的插图、图表或其他通常带有标题的可视化内容，它与主题（父元素）紧密相关，又自成一体。<figcation>为<figure>的子元素，用来包裹插图的标题，<figure>中允许出现多个<figcation>元素，但只有第一个起作用。看例子：

```
    <article>
        <h1>有白色乌鸦吗？</h1>
        <p>你从未看过白色乌鸦，也不能断言乌鸦全是黑的。我们的经验不能穷尽所有的对象，所以经验主义不能从经验出发得出全盘肯定或全盘否定的结论。</p>
        <figure>
            <img src="crow.jpg">
            <figcaption>图 1. 白乌鸦</figcaption>
        </figure>
```

```
        <p>那这样不就是不可知论了？非业，理性主义就可以作全盘判断。</p>
    </article>
```

浏览的效果如下：



图 4.18 <figure>元素用法

<hr>元素

定义一根分隔线以分开两部分内容，被分开的内容一般没有标题，如果有标题就建议不要使用分隔线，因为标题本身就有分割的效果。分隔线默认长度为父元素的宽度，而且是水平方向的，但是可以通过样式改为其他方向，如垂直或者斜线等。其语法如下：

```
<hr>
```

看例子：

```
<p>做为一个网站开发人员，你要照顾很多“人”的感受：浏览器渲染速度快不快，搜索引擎能否轻易检索到你的网站，网络爬虫是否乐于光顾，别人看你的代码舒服吗，接手人愿意继续升级代码吗？</p>
<hr>
<p>我做了几十年的 IT 工作，现在我认为当技术达到某种程度后，提高个人品德更重要，你在写代码时不要忽视他人的感受。“做好人，做好事！”，你理解了这句话吗？</p>
```

浏览效果见图 4.19。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

做为一个网站开发人员，你要照顾很多“人”的感受：浏览器渲染速度快不快，搜索引擎能否轻易检索到你的网站，网络爬虫是否乐于光顾，别人看你的代码舒服吗，接手人愿意继续升级代码吗？

我做了几十年的IT工作，现在我认为当技术达到某种程度后，提高个人品德更重要，你在写代码时不要忽视他人的感受。“做好人，做好事！”，你理解了这句话吗？

图 4.19 分割线元素<hr>

、<ul>、<li>元素

定义有序列表（列表项前自动添加序号，序号默认是 1、2、3、…），<ul>定义无序列表（列表项前默认自动添加符号●），<li>作为<ol>和<ul>的子元素，用来包裹列表项。<ol>和<li>结合使用的语法如下：

```
<ol[ reversed start="数字" type="a|A|i|I|1"]>
    <li[ value="数字"]>列表 1</li>
    <li[ value="数字"]>列表 2</li>
    ……
</ol>
```

元素有三个可选专用属性，其中 reversed 表示列表项倒序编号，比如 5、4、3、2、1；type 属性定义编号的类型，可取 5 个值中的一个：

a - 用小写字幕编号，比如 a,b,c,d,……。

A - 用大写字幕编号，比如 H,I,J,K,……。

i - 用小写罗马数字编号，比如 ⅰ , ⅱ , ⅲ , ⅳ , ⅴ ,……。

I - 用大写罗马数字编号，比如Ⅲ,Ⅳ,Ⅴ,Ⅵ,Ⅶ,……。

1 - 用阿拉伯数字编号，比如 1,2,3,4,……，这是默认的编号类型。

元素的 start 属性取值只能是自然数，指明第一个列表项取第几个编号，比如“start="3" type="a"”，那么第一个列表项取第 3 个编号，即 c，第二个列表项就是 d 了，依次类推。

元素的可选属性 value 取值只能是自然数，指明本列表项采用第几个编号。如果省略此属性，就是采用前一个列表项的编号的下一个编号，比如前一个编号是 9，那么本列表项的编号就是 10。看几个例子（参见图 4.20）。

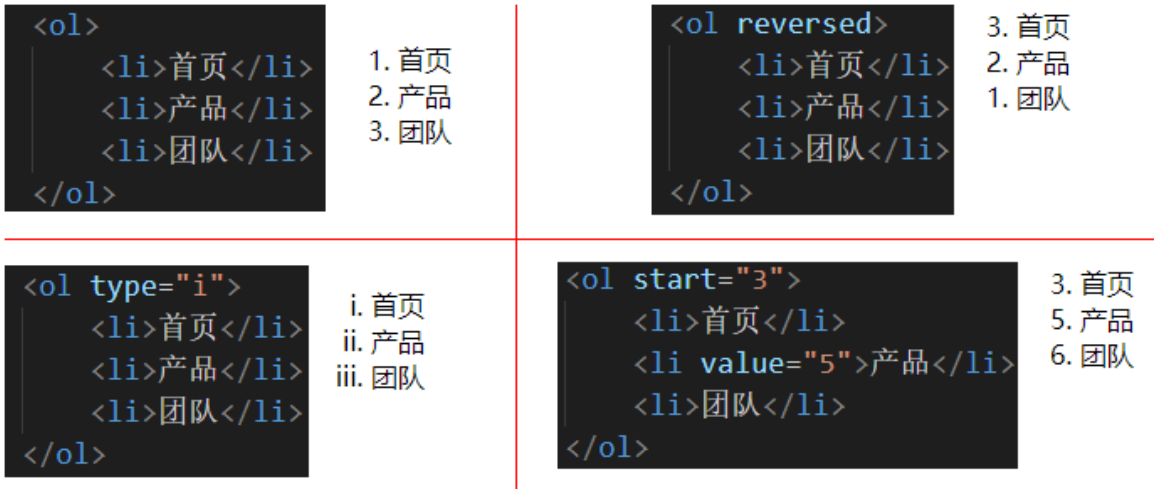


图 4.20 有序列表

注意思考上图中右下角的例子，现在假设 start="6"，其它代码不变，这时浏览的效果会是什么？编号为 6、5、6 了！就是说<ol>的 start 属性与<li>的 value 属性毫无关系。

元素没有专用属性，只是把若干<li>元素包裹起来，形成项目编号。<ol>、<ul>和<li>之间允许互相嵌套，形成多层级列表，类似图 4.21 所示。

```
<ul>
    <li>概述</li>
    <li>网页结构
        <ol>
            <li>HTML5基本标签</li>
            <li>HTML5高级应用
                <ul>
                    <li>拖放</li>
                    <li>canvas</li>
                    <li>离线应用</li>
                </ul>
            </li>
            <li>HTML5综合例子</li>
        </ol>
    </li>
    <li>网页样式</li>
</ul>
```

浏览效果

- 概述
- 网页结构
  1. HTML5基本标签
  2. HTML5高级应用
     - 拖放
     - canvas
     - 离线应用
  3. HTML5综合例子
- 网页样式

图 4.21 多层级列表

<p>元素

包裹一个段落，段落不一定是纯文字，还可以是图片和表单，段落末尾会自动换行。这个元素无语义，如果有更好的语义元素可选择，就尽量不要用<p>元素，比如包裹联系方式的段落用<address>、包裹主题末尾附加的段落用<footer>。其语法是：

```
<p>段落内容</p>
```

段落内容还可以是更多的 HTML5 元素，比如<p>、<img>等等。

<pre>元素

包裹的内容会原封不动地显示出来，空白字符和换行符都保留，文字内容默认采用等宽字体渲染。这个元素常常用于：显示由字符绘制的图形（结合<figure>、<figcation>是个好习惯），或者显示源代码（结合<code>元素是个好习惯），或者诗歌。参考图 4.22 所示的例子。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
<figure>
    <pre>
     _____________________________
    < 我是田里的专家。 >
     -----------------------------
            \   ^__^
             \  (oo)\_______
                (__)\       )\/\
                    ||----w |
                    ||     ||
    </pre>
    <figcaption id="cow-caption">
      一头牛说： "我是田里的专家。"
    </figcaption>
</figure>
```

浏览效果

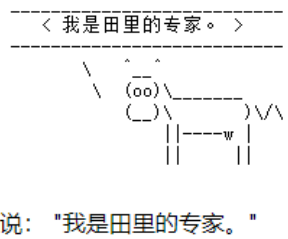


图 4.22 <pre>元素内容保留格式

再看显示源代码的例子（参见图 4.23）。

```
<code>
function fibo(n){
    if(n<3)
        return(n-1);
    return fibo(n-1) + fibo(n-2);
}
</code>
```

```
function fibo(n){ if(n<2) return(n); else return fibo(n-1) + fibo(n-2); }
```

```
<pre>
    <code>
function fibo(n){
    if(n<3)
        return(n-1);
    return fibo(n-1) + fibo(n-2);
}
    </code>
</pre>
```

```
function fibo(n){
    if(n<3)
        return(n-1);
    return fibo(n-1) + fibo(n-2);
}
```

图 4.23 嵌入源代码

直接用<pre>包裹源代码而不用<code>行不行？行啊，人类看到的效果一样，可是诸如搜索引擎、网络爬虫等这些非人类呢？它们不知道是诗歌、还是字符图形、抑或程序代码。加上<code>元素，它们就懂了，因为<code>是语义元素。关于<code>元素的详细介绍，请参考下一小节。

4.7 文字形式

文档《HTML5&CSS3 开发大起底》.xlsx 中的“HTML5 元素速查表 • 文字形式”分类列举的元素，专门用于对文本作更精细化的编排。

**本节介绍这些元素：<a>、<abbr>、<dfn>、<b>、<bdi>、<bdo>、<blockquote>、
、<cite>、<code>、<data>、、<i>、<kbd>、<mark>、<q>、<rb>、<rp>、<rt>、<ruby>、<s>、<samp>、<small>、、、<sub>,<sup>、<time>、<u>、<var>、<wbr>。**

<a>元素

定义一个指到其他资源的超链接，这些资源包括页面、文件、邮件地址、电话号码、当前页面中的锚点等，用户点击超链接后就加载对应的资源。其语法是：

```
<a href=""[ download="" rel="" target="" type="" ping="" referrerpolicy=""]>内容
</a>
```

中括号内的属性是可选的。当用户点击“内容”时，浏览器就“打开”href 属性定义的 URL。由于 URL 中的协议不同，以及其他属性不同的值，“打开”操作包括：打开网页、下载文件、发送邮件、拨打电话等。下面我们来看看各个属性的意义。

href - 资源的 URL。发邮件的格式为 href="mailto:邮件地址"，拨打电话的格式为 href="tel:电话号码"，到达当前页面某个锚点的格式为 href="#[id]"，id 就是页面上某个元素的 id 号(例如 href="#product123"), 如果省略 id 就是回到页面顶部（即 href="#"），打开页面的格式为 href="<页面的 url>"。看下面的例子：

```
    <a href="#">回到本页面顶部</a>
    <a href="products.html">产品介绍</a>
    <a href="http://www.abc.com">去 abc 网站看看</a>
    <a href="mailto:wlm@veryopen.org">给我发邮件</a>
    <a href="tel:07689843323">电话 07689843323</a>
    <a href="contact.html#us">联系我们</a>
```

例子中的最后一条代码表示打开 contact.html 页面并定位到页面中 id 为 us 的那个元素。

download - 下载资源。本属性定义资源保存的文件名，如果是空值，下载时浏览器会提示用户输入文件名。URL 中的协议如果是 mailto、tel、file 就不会下载。看例子：

```
    <a href="abc1234.html" download>产品介绍</a>
    <a href="contact.html#us" download="contactus.html">联系我们</a>
```

这样用户点击“产品介绍”或这“联系我们”，就是下载文件了，而不是渲染在浏览器的窗口中。

rel、type - 请参考前面介绍的<link>元素的同名属性。

referrerpolicy - 请参考前面介绍的<script>元素的同名属性

target - 请参考前面介绍的<base>元素的同名属性。

不能采用 disabled="true"属性来禁用<a>元素，只能采用样式 pointer-events: none 实现禁用的目的。

<abbr>元素

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

代表省略或缩写 ，其完整内容由元素的 title 属性定义，当鼠标停靠时显示完整内容，缩写内容会加虚线下划线。它的语法如下：

```
<abbr [ title="" ]>缩写内容</abbr>
```

title 是通用属性，<abbr>无专用属性。看例子：

```
<p>学好 <abbr title="HyperText Markup Language">HTML5</abbr>，就业门路广！ </p>
```

浏览效果见图 4.24。



图 4.24 <abbr>元素举例

<b>、、<mark>、元素

让标注的内容与周边文字区别开来，从而吸引用户的注意力，表示这部分内容要重点注意。先看一个例子：

```
<p>周边<b>b 美国</b>周边<em>em 美国</em>周边<mark>mark 美国</mark>周边<strong>strong 美国</strong>周边</p>
```

浏览效果见图 4.25。



图 4.25 标准特殊的内容

从外观上看，使内容为斜体，<mark>给内容加上黄色背景色，而<b>和都使内容加粗。这几个元素主要是语义上的区别，语义上表示重要性请采用元素，语义上要强调一些内容，请采用元素，存在相关性的内容请使用<mark>元素（比如搜索结果中的关键字，参见图 4.26 左侧部分，在线编辑器中的错误标注等），而<b>只是与上下文作区分（比如文章的第一段加粗，因为第一段往往是摘要）。换句话说，实在没有更好的元素可选择，那就用<b>吧。但要注意一点：如果纯粹为了加粗文本而没有语义上的要求，那就采用 CSS3 的样式“font-weight: bold”。看例子：

```
<p>新发布的<b>iMac</b> 苹果电脑配置不错，我喜欢。</p>
<p><strong>警告</strong>：这里有鲨鱼出没，请游泳爱好者小心！ </p>
<p><q>不管你<em>信</em>还是不<em>信</em>，反正我<em>信</em>了！ </q> - <cite>王勇平</cite></p>
```

浏览效果见图 4.26 右侧部分。

你搜索 html5 返回10万条结果:

HTML5 Codes

Free copy/paste HTML5 codes for your website. Simply copy then paste the HTML5 code to your own website.

HTML5 Codes, Editors, and Generators

Your one-stop HTML5 resource. Free HTML5 editor. Free copy/paste HTML5 codes for your website. HTML tutorials. HTML generators.

...

新发布的**iMac** 苹果电脑配置不错，我喜欢。

警告：这里有鲨鱼出没，请游泳爱好者小心！

"不管你*信*还是不*信*，反正我*信了*！" - *王勇平*

<dfn>元素

把术语定义文字当中的这个术语包裹起来，使得浏览器或网络爬虫明白附近的文本就是术语的解释，“附近”一般是指<dfn>元素的父元素，比如<p>、<dt>或者<section>，常见的语法为：

```
<p>这是术语<dfn>术语</dfn>的解释</p>
```

如果术语本身又是几个单词的首字母组合而成的，这时就让<abbr>成为<dfn>的唯一子元素，让<abbr>的 title 属性指定那几个单词。例如：

```
<p><dfn><abbr title="Hubble Space Telescope">HST</abbr></dfn>就是指哈勃望远镜，是研究太空最好的科学设备。</p>
```

<bdi>元素

告诉浏览器这部分内容的书写方向不同于上下文，需要隔离出来，这样内外的文本书写方向互不影响。当插入一些不明方向性的文本时，就采用这个元素。例如图 4.26 所示的例子是比赛前两名的名单。

```
<ul>
  <li><span class="name">ٱلْأَعْشَى</span> - 1st place</li><br>
  <li><span class="name">Jerry Cruncher</span> - 2nd place</li>
</ul>
<ul>
  <li><bdi class="name">ٱلْأَعْشَى</bdi> - 1st place</li><br>
  <li><bdi class="name">Jerry Cruncher</bdi> - 2nd place</li>
</ul>
```

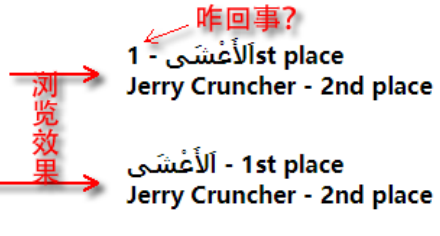


图 4.26 改变文字书写方向

由于阿拉伯语言书写方向是从右到左，而英语是从左到右，所以上面的例子中，包裹在元素中的阿拉伯文字连同后面的“- 1”都被浏览器当成阿拉伯文字，因而一起从右到左被渲染出来，因为阿拉巴语中也有“-”、“1”符号。

<bdo>元素

给被包裹的文字重新定义书写方向，它的语法是：

```
<bdo dir="ltr|rtl">内容</dbo>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

属性 dir 有两个值，dir="ltr"从左到右，dir="rtl"从右到左。例如：

```
<p>上海就是<bdo dir="rtl">上海</bdo></p>
```

浏览时发现后面的“上海”变成“海上”了。

、<wbr>元素

添加换行符。<wbr>是建议换行，当文本超过父元素宽度时将会在此处添加换行符，否则不换行，在没有空白字符的长文本中间添加<wbr>有助于保持排版美观。下面的例子中两段文字相同，但是第二段中添加了若干<wbr>。

```
<p>You will find the file you need by going to directory/subdirectory/deeper/and/deeper/down/the/rabbit/hole/hereitis.</p>
<p>You will find the file you need by going to directory<wbr>/subdirectory<wbr>/deeper<wbr>/and<wbr>/deeper<wbr>/down<wbr>/the<wbr>/rabbit<wbr>/hole<wbr>/hereitis.</p>
```

浏览效果见图 4.27。

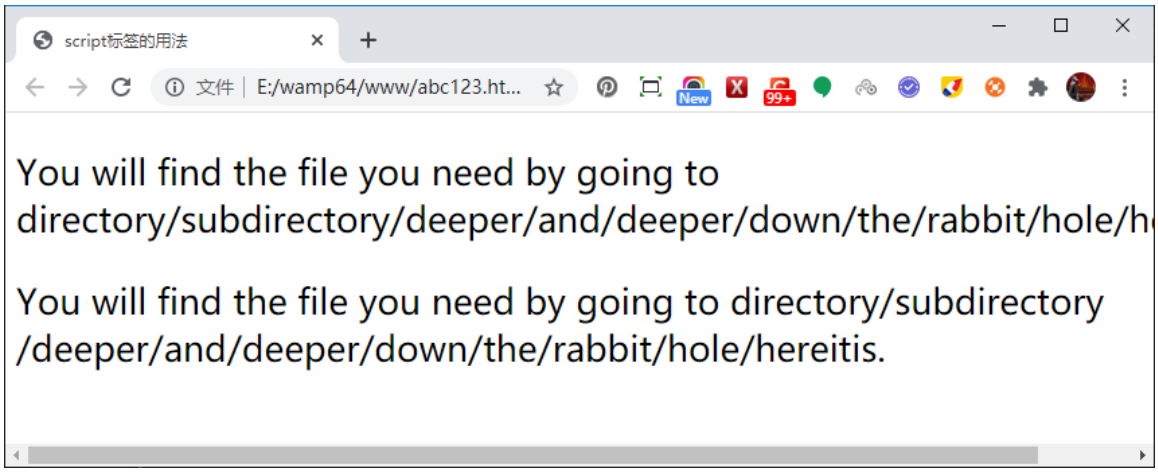


图 4.27 建议换行元素<wbr>

是强制换行，而不管父元素的宽度如何。

<blockquote>、<cite>、<q>元素

这三个元素都是引用其他地方的内容。一个完整的“引用”由两部分组成，一是引用的内容，二是引用的出处，比如：“现在是最好的时代，也是最坏的时代。”——《双城记》，“——”左侧部分是引用的内容，右侧是出处；再比如：康德说过“自由是道德的来源。”“康德”是出处，“自由是道德的来源。”是引用的内容。<cite>元素包裹引用的出处，而<blockquote>和<q>元素包裹引用的内容，只不过前者包裹的内容比较多，往往是整段文字，浏览器将其渲染为缩进，而<q>只引用一句话，浏览器渲染时会加上引号。<blockquote>和<q>元素都有一个专用属性 cite，指明引用内容来自什么网站。看一个综合例子：

```
<p>Linus Torvalds 宣布:<q cite="https://itsfoss.com/linux-kernel-5/">Linux 内核 5.0 是一个划时代的版本.</q></p>
<blockquote>
```

```
    <p>基于网页的应用称为“B/S”应用，“B”就是 Browser，即浏览器，当下的 SaaS 云计算产品都是“B/S”
应用。诸如*.html、*.JS 和*.css 这样的网页文件构成网站的前端，这些文件由浏览器从 Web 服务器上下
载，然后浏览器就渲染或执行它们。
    </p>
    <p>——<cite>《HTML5 建站全过程大起底》</cite></p>
</blockquote>
<p>More information can be found in <cite>[ISO-0000]</cite>.</p>
```

浏览效果如图 4.28 所示。

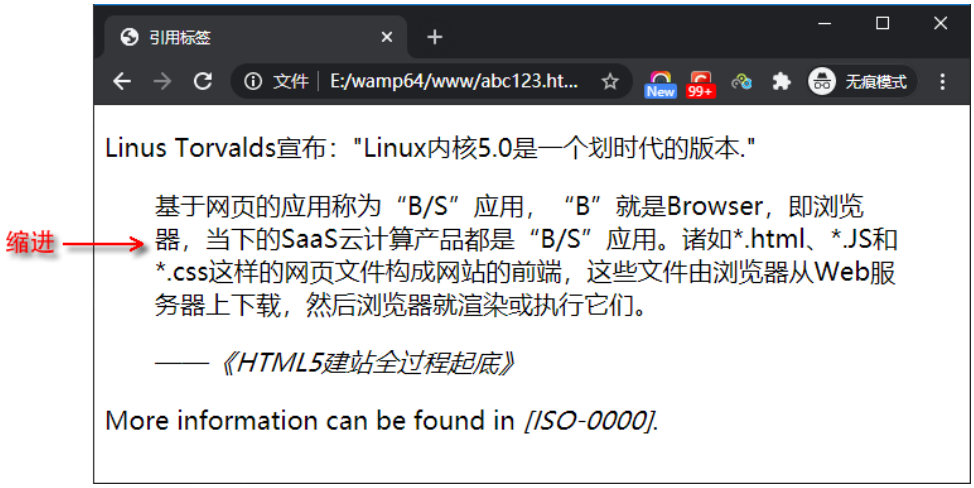


图 4.28 编排“引用”内容

<code>元素

包裹一段代码，比如 C 语言代码、一段 HTML5 代码等，浏览器默认用等宽字体（如 Courier）渲染它。<code>是语义元素，它告诉搜索引擎、爬网程序、社交媒体应用程序、RSS 阅读器以及其他互操作的 Web 服务：“嗨，这是计算机程序源代码。”如果涉及多行源代码，请再用<pre>元素包裹<code>元素。例如：

```
<p>你能看懂下面这条语句吗？<br><code>a=++i++;</code></p>
```

<data>元素

在便于人类阅读的信息和便于机器处理的数据之间建立关联，比如网店上的每件商品有一个名称和产品编码，名称显示出来以便用户选购，而产品编码保存在数据库中以便计算机处理。<data>元素的使用语法如下：

```
<data value="便于机器处理的数据">便于人类阅读的文字</data>
```

属性 value 的值不会显示出来，我们可以采用 JS 语句读取它的值，并作进一步的处理。看例子：

```
    <p>热销商品：</p>
    <ul>
        <li><data value="398">人性论(精装)</data></li>
        <li><data value="789">联想 ThinkPad X13 (02CD)</data></li>
        <li><data value="100">云计算通俗讲义（第 3 版）</data></li>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    </ul>
```

浏览效果参见图 4.29。

热销商品：

- 人性论(精装)
- 联想ThinkPad X13 (02CD)
- 云计算通俗讲义（第3版）

图 4.29 <data>元素用法举例

<i>元素

被包裹的文字与周边内容作区分，它不是语义元素，这点不同于，默认显示为斜体。例如：

```
<p>这些拉丁词语<i>Veni, vidi, vici</i> 经常出现在文学、艺术和音乐作品中。</p>
```

<kbd>、<samp>元素

<kbd>元素包裹的内容表示需要用户在输入设备上作某种操作，看一个例子：

```
<p>执行命令 <kbd>systemctl restart nginx</kbd> 重新启动 Web 服务器。</p>
<p>同时按键 <kbd>Ctrl</kbd> + <kbd>Shift</kbd> + <kbd>r</kbd> 刷新页面。</p>
```

“systemctl restart nginx”是要用户从键盘上输入的文本，“Ctrl”、“Shift”和“r”是要同时按下的三个键。浏览效果键图 4.30 左侧部分。

执行命令 systemctl restart nginx 重新启动Web服务器。

同时按键 Ctrl + Shift + r 刷新页面。

当我重启电脑时，显示错误：

OS Not Found
Press F1 to continue

图 4.30 <kbd>和<samp>用法举例

<samp>元素包裹的内容代表是计算机程序输出来的，比如：

```
  <p>当我重启电脑时，显示错误：</p>
  <p><samp>OS Not Found <br>Press F1 to continue</samp></p>
```

浏览效果见图 4.30 右侧部分。如果页面是讲解一个人机交互过程，很可能同时涉及输入和输出，这时我们可以在<samp>中嵌套<kbd>，或者<kbd>中嵌套<samp>，看例子：

```
    <pre>
        <samp>moodisk@localhost:~$ <kbd>md5 -s "Hello world"</kbd>
        MD5 ("Hello world") = 3e25960a79dbc69b674cd4ec67a72c62

        moodisk@localhost:~$ </samp>
    </pre>
```

浏览效果见图 4.31。

```
moodisk@localhost:~$ md5 -s "Hello world"
MD5 ("Hello world") = 3e25960a79dbc69b674cd4ec67a72c62

moodisk@localhost:~$
```

图 4.31 <kbd>和<samp>嵌套举例

再看一个例子：

```
<p>When you click the <kbd>Delete</kbd> button, you will be asked <samp>Are you
 sure you want to delete 9 to 5 job?</samp> Click <kbd>Yes</kbd> to continue.</
p>
```

<rp>、<rt>、<ruby>元素

利用这几个元素给中文加上拼音，比如“中国”二字加拼音，在支持这些元素的浏览器中显示“中(zhōng)国(guó)”，不支持这些元素的浏览器中显示“中(zhōng)国(guó)”。先看一个例子：

```
    <p>“我
        <ruby>
            中<rp>(</rp><rt>chè</rt><rp>)</rp>
            艸<rp>(</rp><rt>cǎo</rt><rp>)</rp>
            灨<rp>(</rp><rt>huì</rt><rp>)</rp>
            爿<rp>(</rp><rt>mǎng</rt><rp>)</rp>
        </ruby>
        ”是网络用语，为“我擦”的意思。
    </p>
```

浏览的效果如下：

chè cǎo huì mǎng

“我 中艸灨爿 ”是网络用语，为“我擦”的意思。

从这个例子可以看出：要把汉字及其拼音包裹在<ruby>元素内，拼音包裹在<rt>元素内，在不支持注音元素的浏览器中额外要显示的内容包裹在<rp>元素内，如“中(chè)艸(cǎo)灨(huì)爿(mǎng)”中的圆括号。由于目前的浏览器几乎全部支持注音，所以可以省略<rp>元素，直接写成：

```
<ruby>中<rt>chè</rt>艸<rt>cǎo</rt>灨<rt>huì</rt>爿<rt>mǎng</rt></ruby>
```

<s>元素

给文本添加中划线，语义上表示不存在或不相关了。而在线编辑页面上往往加上中划线表示删除一些文字，此时不要用<s>，而用<del>元素。如果无语义要求，只是给文字加中划线，建议采用样式属性 text-decoration-line。看例子：

```
<p>特级咖啡降价促销 <s>￥500</s> ￥200!</p>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

<small>元素

对主体内容作附加说明的次要的文字应包裹在<small>元素内，比如注释、免责声明、版权信息等。 例如：

```
<p>developer.mozilla.org 是一个学习 WEB 编程的好平台，这里的信息更新很快，常有很多骨灰级的大牛出没，如果你运气好，就会遇到他们！ </p>
<hr>
<p><small>内容受到 GPL 2.0 版权的保护，不要抄袭哦。2020-10-01</small></p>
```

浏览效果参见图 4.32。

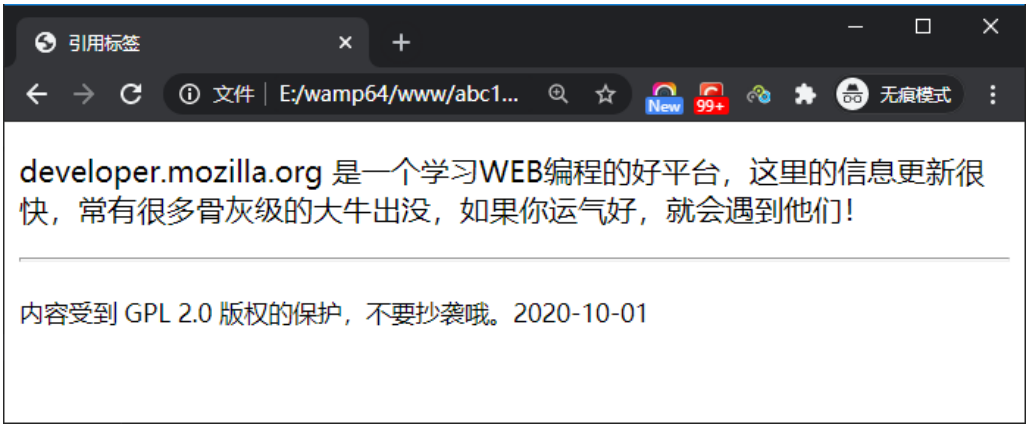


图 4.32 <small>元素用法举例

<sub>、<sup>元素

标注下标和上标，如图 4.33 左侧部分。例如：

```
<p>水分子是：H<sub>2</sub>O</p>
<p>勾股定理：a<sup>2</sup> + b<sup>2</sup> = c<sup>2</sup></p>
```

浏览效果参见图 4.33 右侧部分。

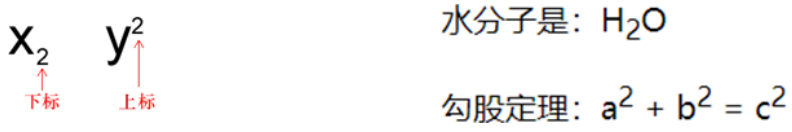


图 4.33 上标和下标

<time>元素

类似<data>元素，把便于人类阅读的时间和便于机器处理的时间联系在一起，其语法是：

```
<time datetime="便于机器处理的时间">便于人类阅读的时间</time>
```

用 JS 语句读取 datetime 属性的值，然后作进一步处理。看例子：

```
<p>音乐会在<time datetime="2020-10-28">下下周三</time>举行。</p>
```

<u>元素

给文字添加下划线，一般在线编辑页面上标注拼写有误的文字。

```
<p>This paragraph includes a<u>wrnogly</u>spelled word.</p>
```

<var>元素

包裹数学表达式或编程上下文中的变量名称，默认显示为斜体。例如：

```
<p> 一 个 盒 子 的 体 积 等 于 <var>l</var> × <var>w</var> × <var>h</var>， 其中 <var>l</var> 表示长度，<var>w</var> 表示宽度 <var>h</var> 表示高度。</p>
```

浏览的效果：

一个盒子的体积等于 $l \times w \times h$，其中 l 表示长度，w 表示宽度 h 表示高度。

4.8 编辑元素

<del>和<ins>两个元素用于在线编辑页面中。

<ins>元素

表示新增加的内容，默认会给内容加下划线。它的语法如下：

```
<ins[ cite="" datetime=""]>新增加的内容</ins>
```

有两个可选属性，cite 属性指到一个 URI 链接，该链接解释为何要增加这些内容，datatime 属性定义这些内容是什么时候增加的。看例子：

```
<ins>成为一个网站全栈开发工程师，你就牛逼了！</ins>
<ins cite="../howtohtml5.html" datetime="2020-10-18">
    <p>学好 HTML5，走偏天下都不怕。</p>
</ins>
```

浏览效果见图 4.34。

成为一个网站全栈开发工程师，你就牛逼了！

学好HTML5，走偏天下都不怕。

图 4.34 <ins>元素用法举例

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

<del>元素

表示被删除的内容，默认会给内容加中划线。它的语法如下：

```
<del[ cite="" datetime=""]>被删除的内容</del>
```

两个可选属性意义与<ins>元素相同。

4.9 表单元素

新生报到处的老师说：“这位同学，请填一下这个表格。”你结果表格一看，发现要写上姓名、身份证、电话、家庭地址和性别等信息。对，表格也叫表单，表单就是用来接受用户的输入，然后提交到处理页面，例如图 4.35 所示的一个简单表单，输入姓名和电话，然后点击提交，就会把输入的信息提交给处理页面。

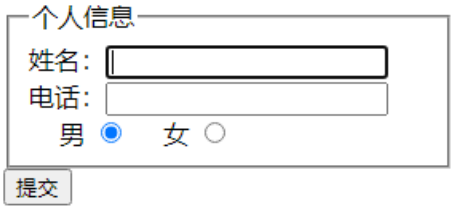


图 4.35 一个简单的输入表单

“HTML5 元素速查表 • 表单”分类列举了全部的表单元素，用这些元素，我们能设计出任形状的输入界面。

本节介绍这些元素：<form>、<fieldset>、<legend>、<label>、<input>、<datalist>、<button>、<select>、<optgroup>、<option>、<progress>、<meter>、<textarea>、<output>。

4.9.1 <datalist>/<option>/<label>

<option>元素

作为<select>、<datalist>、<optgroup>元素的子元素，用于定义具体的条目，其语法如下：

```
<option value="" disabled label="" selected>显示内容</option>
```

用户根据“显示内容”选中相应条目，最终把选中的条目的 value 属性的值提交给处理页面。属性 disabled 不让选择（显示为灰色），属性 selected 表明本条目默认被选中，属性 label 定义条目的标题给用户看，它会覆盖“显示内容”。看一个例子：

```
选择省份:
<select name="sel">
    <option value="jx" label="jiangxi">江西</option>
    <option value="gd" selected>广东</option>
    <option value="gx" disabled>广西</option>
    <option value="hn">湖南</option>
</select>
```

浏览效果见如图 4.36 所示。



图 4.36 <option>元素用法举例

广西不让选择，因为它被 disabled 了，默认选中“广东”。

<datalist>元素

包含若干<option>子元素，为其他交互元素提供输入域，其他交互元素通过 list 属性关联对应的<datalist>元素的 id。比如：

```
成绩: <input list="eval" name="score">
<datalist id="eval">
    <option value="50">不及格</option>
    <option value="60">及格</option>
    <option value="70" selected>良好</option>
    <option value="80">优良</option>
    <option value="90">优秀</option>
    <option value="100" disabled>牛逼</option>
</datalist>
```

双击输入框后浏览的效果参见图 4.37 所示。

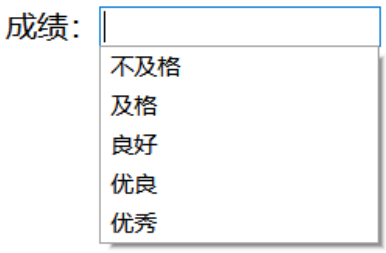


图 4.37 <datalist>用法举例

<label>元素

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

为交互元素提供标题或说明，点击标题就相当于点击了交互元素，它们被捆绑在一起了。它有两种使用方法：

```
    <label>标题<其他交互元素></label>     <!--第 1 种用法-->
    <label for="其他交互元素的 id">标题</label>     <!--第 2 种用法-->
```

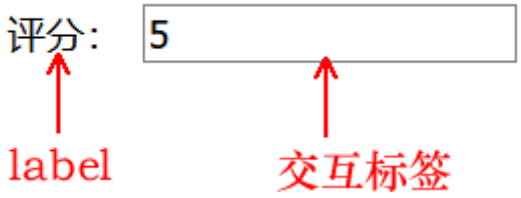


图 4.38 <label>元素用法举例

例如对于图 4.38，我们用两种方法实现：

```
<label>评分：<input name="score"></label>
```

第 2 种方法如下：

```
<label for="sco">评分：</label>
<input name="score" id="sco">
```

当你点击“评分”相当于点击了后面的输入框，不信你试试！

4.9.2 <form>元素

<form>元素为表单容器，代表页面上的一个区域，里面包含各种交互控件，共同完成提交信息的任务。它的使用语法如下：

```
<form [accept-
charset="" autocomplete="" rel="" action="" enctype="" method="" novalidate tar
get=""]>各种交互元素</form>
```

各个属性介绍如下：

accept-charset 属性 - 定义后端能接受的字符编码集，值是用空格分开的若干字符编码集名称，如 accept-charset="UTF-8 GBK GB18030"。默认是当前页面的字符编码集。

autocomplete 属性 - 设置是否联想输入，如果开启，则历史上递交过的具有相同 name 的控件的所有输入值作为本次输入的联想输入域，这时在输入框中点击鼠标会以下拉或上拉框方式显示全部的历史输入值供你选择，一旦开始输入就会实时搜索联想输入域。autocomplete 取值为"on"、"off"或没有值，如果省略 autocomplete 属性或者省略属性的值，那么相当于 autocomplete="on"。参见图 4.39。



图 4.39 <form>元素的 autocomplete 属性

演示代码如下：

```
<form autocomplete>
    <fieldset>
    <legend>选择性别</legend>
        姓名<input name="name"><br>
        <label>男 <input type="radio" name="sex" checked></label><br>
        <label>女 <input type="radio" name="sex"></label>
    </fieldset>
    <input type="submit" value="提交">
</form>
```

对于本例<form>元素下的子元素，先不管它，后面会详细介绍。

所以给交互元素取名（属性 name="xxx"）时尽量不要与其他网站的输入元素同名，以避免联想输入产生“串联”现象，比如上图中的“wlm”是我登录 qq 邮箱时输入过的用户名，现在“串联”到 www.youduo.me 网站的登录页面了。为了保护隐私，一般网银或者股票操作网站的表单都会设置 autocomplete="off"，这样就不会记录历史账户信息，当然用户自己也可以在浏览器中设置不记录隐私信息来达到同样的目标，chrome 和 firefox 浏览器还提供了无痕模式，chrome 浏览器上直接按 Ctrl+Shift+N 即进入无痕模式，见图 4.40。



图 4.40 chrome 浏览器的无痕模式

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

novalidate 属性 - 存在此属性代表 true，省略代表 false。表单递交时默认都要经过 html5 的验证机制，novalidate 属性声明不需要验证，直接递交到后端。提交按钮（类型为 image 或 submit 的<input>元素）的属性 formnovalidate 会覆盖<form>元素的 novalidate 属性，formovalidate 也是布尔类型。参考图 4.41。

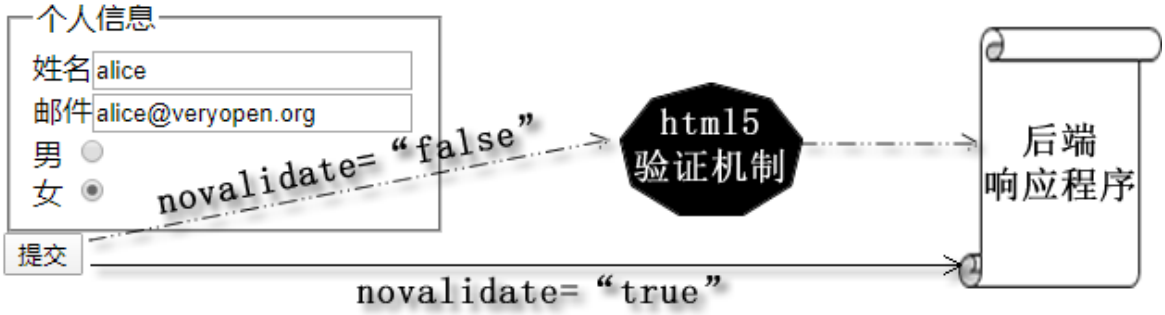


图 4.41 <form>元素的 novalidate 属性

欲了解 HTML5 的验证机制，请参考 https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/HTML5/Constraint_validation。HTML5 的验证机制只是做最基本的验证操作，比如邮件地址中是否包含“@”符号，如果要进一步做业务逻辑上的验证，比如考试分数不能超过 100 分，这时可以给<form>元素的“submit”事件编程，类似代码如下：

```
<form action="saveScore.php" id="login-form" name="login">
    <label>考试成绩: <input name="score"></label>
    ……
</form>
<script>
   var loginForm = document.getElementById("login-form");
   //或者 var loginForm = document.forms["login"];
   loginForm.addEventListener("submit", (event) => {
      if (parseInt(loginForm.elements["score"].value) > 100){
         loginForm.elements["score"].style.border = "1px solid red";
         event.preventDefault();//阻止提交
      }
      //or loginForm.submit();
    });
</script>
```

action 属性 - 表单提交后，定义处理页面的 URL。如 action="process.html"。提交按钮的 formaction 属性会覆盖本属性。

enctype 属性 - 当 method 属性的值为"post"时，enctype 定义表单提交的 MIME 类型。可取值："application/x-www-form-urlencoded"（默认值），"multipart/form-data"（当<input>子元素存在属性 type="file"时）、"text/plain"（用于 debug，这是 HTML5 新增的值）。提交按钮的 formenctype 属性会覆盖本属性。

method 属性 - 定义提交方式，可能的取值有："post"（表单数据作为请求的 body 被提交）、"dialog"（表单为<dialog>元素的子元素，提交后同时关闭对话框）、"get"（表单数据附加在 action 属性值的后面被提交，用“?”分隔），比如 action="process.html"，表单数据 age=18，那么提交时的 URL 为 process.html?age=18。默认值为"get"。提交按钮的 formmethod 属性会覆盖本属性。

rel 属性参考<link>元素的同名属性，**target 属性**参考<base>元素的同名属性。

看例子：

```
<form autocomplete action="process.html">
    邮件<input name="mail" type="email"><br>
    <input type="submit" value="验证提交">
    <input type="submit" value="不验证提交" formnovalidate>
</form>
```

例子中<form>的最后两个<input>子元素是两个提交按钮，由于最后一个提交按钮加了 formnovalidate 属性，所以点击它提交表单不会验证邮件地址的合法性。浏览效果见图 4.42。

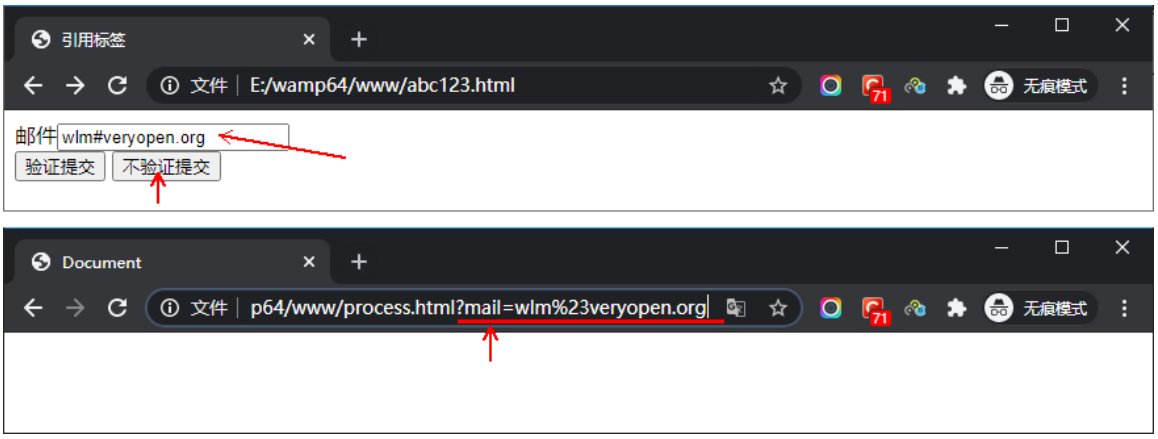


图 4.42 GET 提交表单举例

给例子中的<form>加上属性 method="post"，打开开发者工具的 Network 页，然后点击“不验证提交”按钮，看看此时的数据如何提交，参看图 4.43。

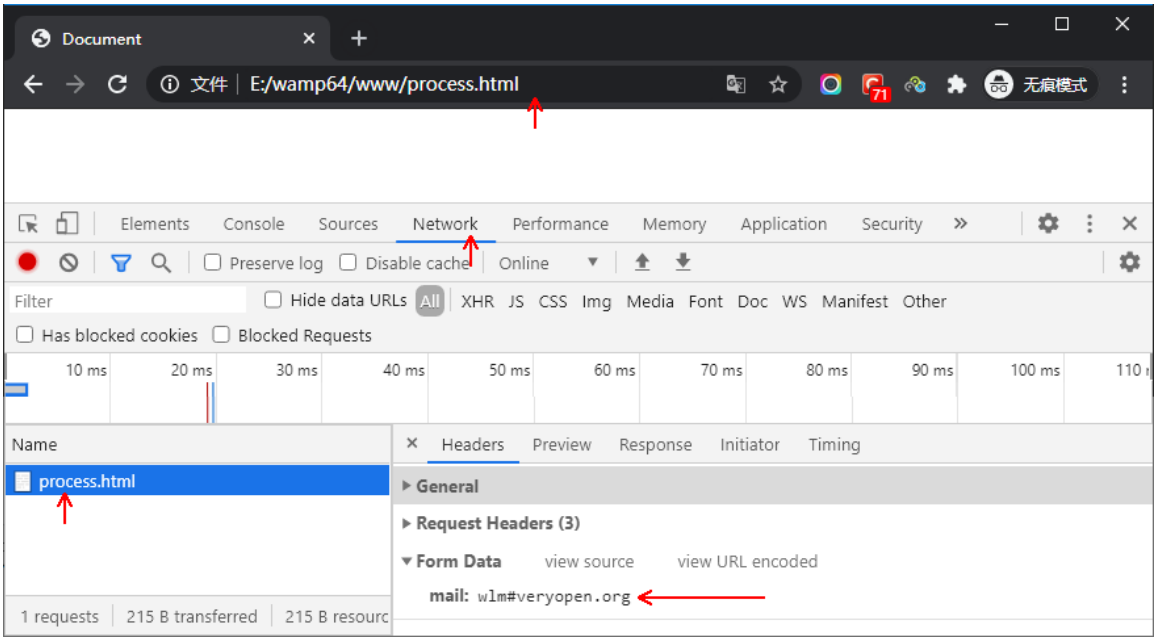


图 4.42 POST 提交表单举例

点击“验证提交”按钮呢？

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

接下来的小节介绍表单中的各种交互元素。

4.9.3 <input>元素

这是表单里最复杂最常用的交互控件，能接受键盘输入、鼠标点击、上传文件、选取颜色、选取日期等等。它的使用语法如下：

```
<input [type="" 其他属性]>
```

type 属性定义交互类型和默认的显示形状，可取的值参见表 4.14 首行，默认值为"text"，表示可输入任何文本。type 的值不同，能搭配的其他属性也不同，比如 type="button"，此时根据下表可知能搭配的其他属性只能是 autocomplete、autofocus、disabled、form、name、value。

表 4.13 <input>元素的 type 类型与其他属性的搭配

其他属性 \ type属性	button	checkbox	color	date	datetime-local	email	file	hidden	image	month	number	password	radio	range	reset	search	submit	tel	text	time	url	week	
accept							√																文件类型
alt								√															代替文本
autocomplete			√	√	√	√		√		√	√		√		√		√	√	√	√	√	√	自动填充策略
autofocus	√	√	√	√	√	√	√		√	√	√	√	√	√	√	√	√	√	√	√	√	√	自动获取焦点
capture							√																媒体捕获规范
checked		√											√										被选中
dirname															√				√				书写方向
disabled	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	禁用
form	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	关联的表单
formaction								√								√							受理页面的URL
formenctype								√								√							表单的MIME类型
formmethod								√								√							提交方法
formnovalidate								√								√							不验证
formtarget								√								√							打开页面的方法
height								√															图片高度
list			√	√	√	√			√	√			√		√		√	√	√	√	√		指明datalist的id
max				√	√				√	√			√						√			√	最大值
maxlength						√					√				√		√	√		√			最大字符数
min				√	√				√	√			√						√			√	最小值
minlength						√					√				√		√	√		√			最小字符数
multiple						√	√																多值(逗号分隔)
name	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	取名
pattern											√						√	√					匹配正则表达式
placeholder											√				√		√	√		√			占位符
readonly				√	√	√			√	√	√				√		√	√	√	√	√		只读
required		√		√	√	√	√		√	√	√	√			√		√	√	√	√	√		必填项
size						√					√						√	√					显示的字符数
src								√															图片的URL
step				√						√			√						√				步进
value	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	√	输入的值
width					□			√														□	图片的宽度

下面对于其他属性中的一些较通用的属性做个简单介绍。

autofocus 属性 - 页面加载后自动获得焦点，一个页面上只允许一个控件带有此属性。

disabled 属性 - 元素被禁用，显示为灰色，比如按钮被禁用后不能点击。

form 属性 - 指定关联的<form>元素的 id，如果省略本属性，则关联最近的父辈<form>元素。注意：一个<input>元素只能关联一个表单元素。

list 属性 - 指定当前页面中的一个<datalist>元素的 id 号，<datalist>定义好了输入范围，参考图 4.17 所示的例子。

min、max 属性 - 定义输入域的最小值和最大值。注意：对于周期域（比如 date 或 time 等），max 可能小于 min，例如时钟范围 min=10:00，max=4:00 表示从今天上午 10 点到明天早晨 4 点。

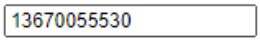
name 属性 - 取一个名字，在一个表单中，名字允许相同。name 和 value 属性的值组成“键/值”对被提交到处理页面，处理页面通过“键”获取“值”。比如<input name="age">，用户输入 18，最终提交 age=18。

placeholder 属性 - 指定占位字字符串，一般是一些提示性的文本。如 placeholder="输入 6 位数字的密码"，显示为 

。一旦输入了第一个字符，占位信息就消失了。

required 属性 - 必须要输入内容，否则提交表单时会警告，并且不让提交。

size 属性 - 当类型为 email、password、tel、text 时，表示输入框能显示的字符数目（能输入的字符最大数目由 maxlength 属性决定），因此也间接确定了输入框的宽度。

value 属性 – 存储了输入的值。如果给它赋值就表示初始值，比如<input type="tel" value="13670055530">，显示为 

，接下来可以修改它。

下面利用<input>元素产生一些常见的应用。

1、 按钮

类似图 4.43 所示的一些按钮。



图 4.43 常见的一些按钮

通用按钮：<input type="button" value="确定">，显示为 

，value 属性给按钮取名。点击按钮时默认没有行为，需要给按钮的诸如 click/onclick 事件编程才行。

提交按钮：<input type="submit"[其他属性]>或者<input type="image" src="" [其他属性]>，src 属性指名一个小图片的 URL，此图片显示在按钮上。最简单的形式就是省略其他属性，这时显示为 

或者图形按钮，也可以给 value 属性赋值来修改按钮的名字，图形按钮没法修改名字。点击提交按钮后会向<form>的 action 属性定义的处理页面 URL 提交表单，如果本身带有 formaction 属性，此属性会覆盖<form>的 action 属性。

重置按钮：<input type="reset"[其他属性]>，最简单的形式就是省略其他属性，这时显示为 

，给 value 属性赋值修改按钮的名字。点击重置按钮后表单被初始化——清空或值为默认值。

在一个表单中允许存在多个“提交”和“重置”按钮。

2、 复选框

类似图 4.44 的复选框。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

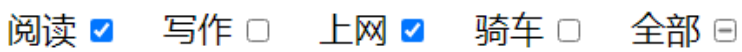


图 4.44 复选框

复选框显示一个空白小正方形，允许在里面打勾选中。其语法如下：

```
<input type="checkbox" name="" [ checked value=""]>
```

如果提供了 value 属性，就以 name=value 形式提交“键/值”对，否则“值”就是"on"。如果没勾选，就不提交本复选框任何数据。checked 属性表明默认被选中。在一个表单中，同一组的复选框 name 属性的值必须相同，提交的数据中可能存在多个同键的“键/值”对，处理页面要注意这种情况。复选框除了“选中”和“未选中”两种状态外，还有一种状态“不确定”，比如图 4.44 中的“全部”复选框就是“不确定”状态，这是通过 JS 语句给<input type="checkbox">元素对象添加了 indeterminate=true 属性所致。在图 4.44 中，当用户选择了四个爱好时，“全部”复选框就被“选中”，如果没有选择任何爱好，“全部”复选框就“未选中”，如果只选择了部分爱好，“全部”复选框就“不确定”。“未选中”和“不确定”的复选框不会向处理页面提交任何数据。看例子：

```
<form>
    <label>阅读<input type="checkbox" name="hoby" value="reading"></label>
    <label>写作<input type="checkbox" name="hoby" value="writting"></label>
    <label>上网
    <input type="checkbox" name="hoby" checked value="navgating"></label>
    <label>骑车<input type="checkbox" name="hoby" value="ridding"></label>
    <br>
    <input type="submit" autofocus> <input type="reset">
</form>
```

名字相同的<input type="checkbox" …>组成一组复选框。

3、 单选框

参考示意图 4.45。

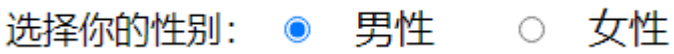


图 4.45 单选框示意图

不同于复选框，在一组单选框种每次最多只能选择一个，就有相同 name 属性的单选框组成一组。语法如下：

```
<input type="radio" name="" value="" [ checked ]>
```

一组单选框中的<input>元素具有相同值的 name 属性，同一组单选框中，最多只有一个<input>元素存在 checked 属性（表示默认被选中），被选中元素的 value 值和 name 值组成“键/值”对被提交到处理页面。看例子：

```
<form action="#">
    <p>选择你最喜欢的联系方式：</p>
    <label><input type="radio" name="contact" value="email" checked>邮件</label>
    <label><input type="radio" name="contact" value="phone">电话</label>
    <label><input type="radio" name="contact" value="wsapp" >whatsapp</label>
    <br><input type="submit" value="提交">
```

```
</form>
```

如果你选中了“电话”，那么提交的数据就是“contact= 'phone'”。

4、 输入颜色

参考图 4.46。

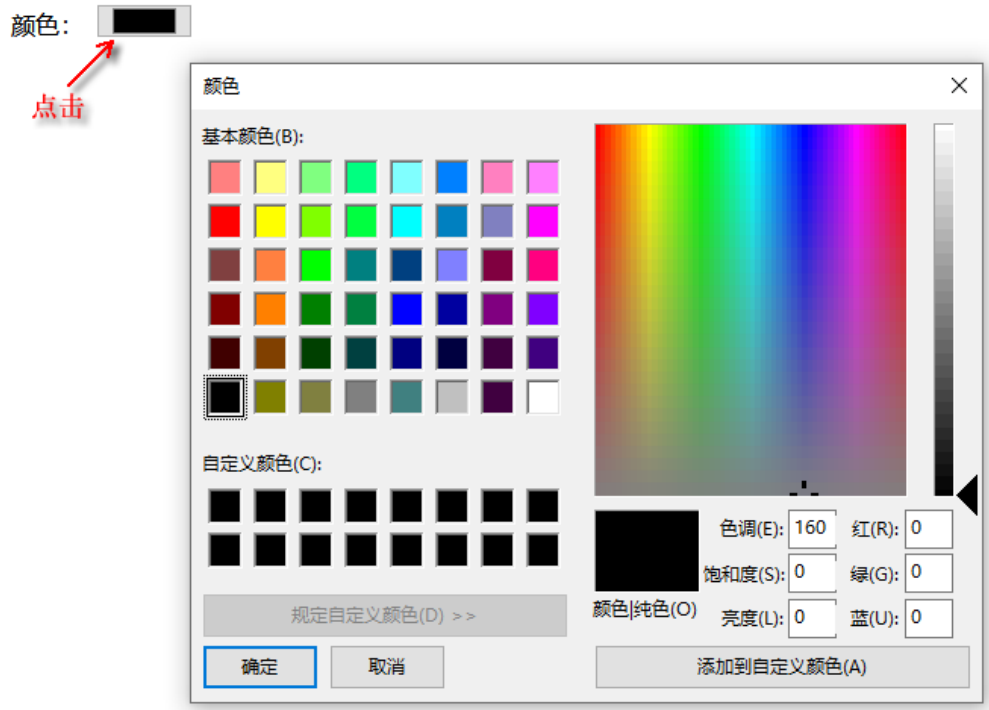


图 4.46 输入颜色

语法如下：

```
<input type="color" name="" [ value="" list="" …]>
```

属性 value 的值为十六进制表示的颜色，比如红色为"#FF0000"，list 属性指定<datalist>元素的 id，这时颜色只能从<datalist>定义的颜色中选择。看例子：

```
<label>颜色 1: <input type="color" name="colr1" value="#FF0000"></label><br>
<label>颜色 2: <input type="color" name="colr2" list="cols"></label>
<datalist id="cols">
    <option value="#FF0000">红色</option>
    <option value="#00FF00">绿色</option>
    <option value="#0000FF">蓝色</option>
</datalist>
```

第一个颜色输入框定义了初始颜色为红色，第二个输入框只能从红、绿和蓝色中选择一个（因为<datalist>元素中只提供这三种颜色）。浏览效果参见图 4.47。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

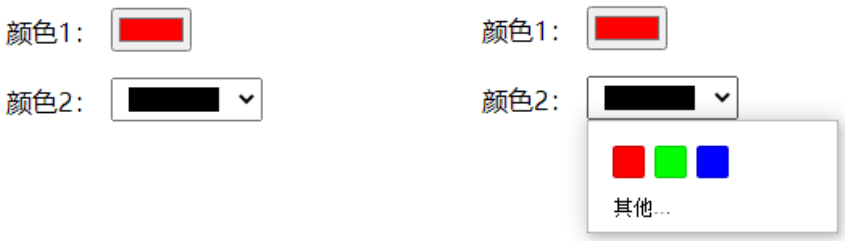


图 4.47 <input>元素输入颜色举例

注意：如果不需要提交，那么交互元素允许不与<form>元素关联。

5、 输入日期和时间

参考示意图 4.48。

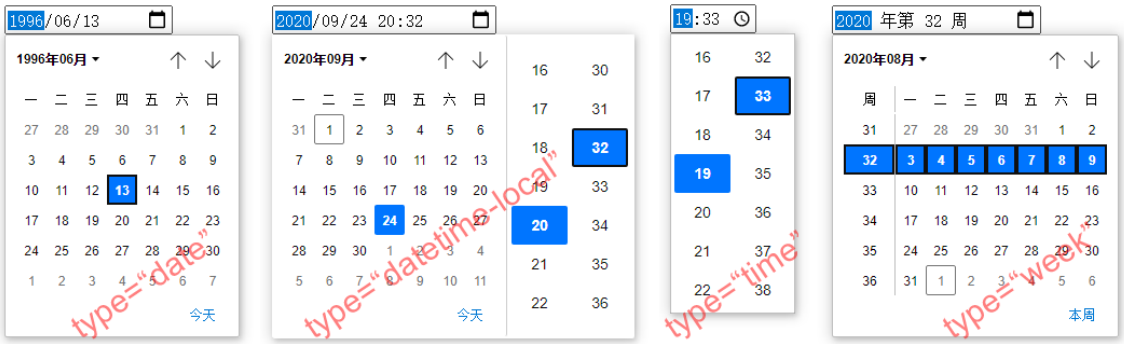


图 4.48 日期输入

允许输入日期、星期、月份、时间等，语法如下：

```
<input type="date" name="" [ value="" max="" min="" step=""]>
<input type="datetime-local" name="" [ value="" max="" min="" step=""]>
<input type="time" name="" [ value="" list="" max="" min="" step="" readonly]>
<input type="week" name="" [ value="" max="" min="" step="" readonly]>
<input type="month" name="" [ value="" max="" min="" step=""]>
```

看例子：

```
<input type="date" name="birth" value="2010-01-01" min="2010-01-01" max="2020-
12-31" step="3">
```

由于步进是 3（天），最小日期是 2010-01-01，所以只能选择以 3 天为步进的日期，即：2010-01-01 号、04 号、07 号、……，其他日期不能选择，见图 4.49。

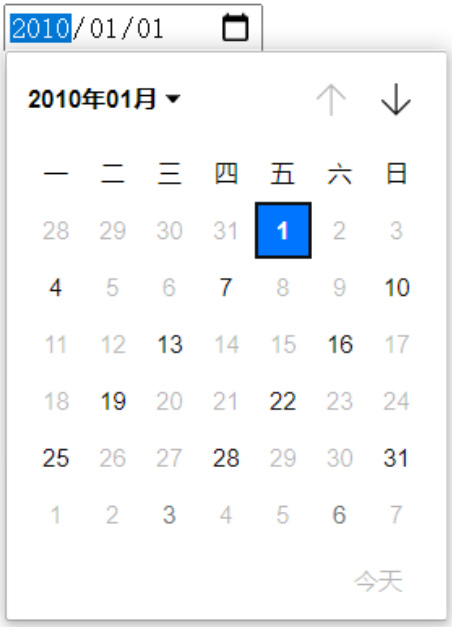


图 4.49 步进为 3 输入日期

6、 上传文件

参考示意图 4.50。

上传照片： 选择文件 未选择任何文件

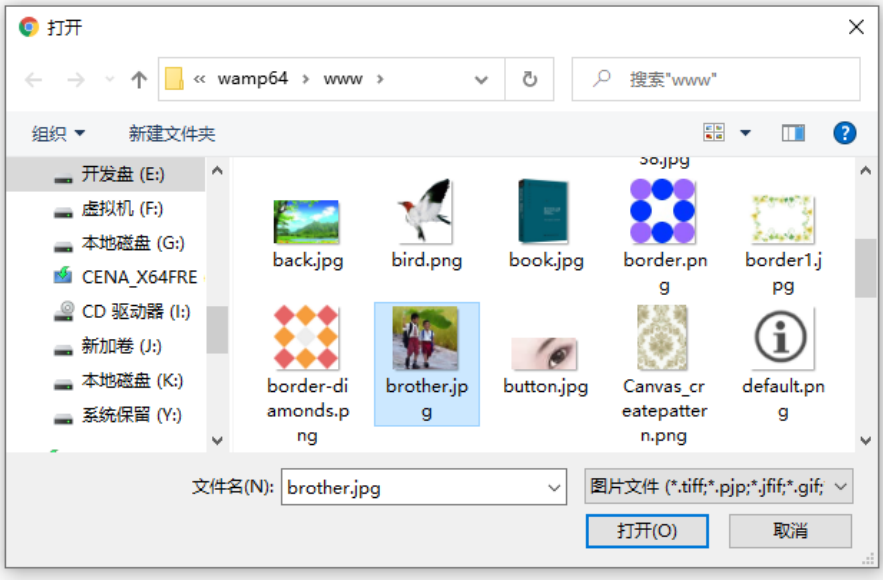


图 4.50 选择磁盘文件

使用语法如下：

```
<input type="file" name="" [ accept="" capture="" multiple ]>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

accept 属性指定后端能接受的文件类型，其值是用逗号分隔的若干文件类型："audio/*"-任何音频文件、"video/*"-任何视频文件、"image/*"-任何图片文件、".pdf"-pdf 文件、".docx"-word 文件、".jpeg"-jpg 图片文件、等等。例如 accept= ".pdf,image/*,.pptx"。

capture 属性主要用于移动设备上，表示从摄像头捕捉照片或录像，需要配合 accept="image/*, video/*"。

multiple 属性允许上传多个文件，否则只能上传一个文件。

还有一个属性 files 的值只能用 JS 读取，它是 FileList 对象数组，数组中的每个元素代表一个文件。FileList 对象主要属性有：name - 文件名（不含目录），size - 文件大小，type - 文件类型，lastModified - 最近修改时间。如：

```
<form>
    上传照片：<input type="file" name="birth" multiple accept="image/*"><br>
    <input type="button" onclick="abc()" value="确认">
</form>
<script>
    function abc() {
        alert(document.querySelector("input").files[0].name);
    }
</script>
```

例子中的代码 document.querySelector("input").files 就是获取<input>元素的 files 属性，它的值是个数组，数组的下标从 0 开始，所以 files[0]代表第一个文件、files[1]代表第二个文件、以此类推。

完整的上传文件任务，还需要后端的配合。具体参考“6.2 访问剪贴板”小节中的例子。下面的例子上传一个音乐文件，并且隐藏“选择文件”按钮。

```
<style>
    input{
          border-style: none;
          border-bottom: 1px solid black;
    }
    input[type="file"]::-webkit-file-upload-button {
          display: none;
    }
</style>
……
<body>
    <label>上传音乐文件：<input type="file" accept="audio/*" name="file"></label>
</body>
```

浏览的效果如下：



一般在上传图片前先要预览，可在<input type=”file” ...>的 change 事件中添加下面的语句：

```
imgEmt.setAttribute("src",URL.createObjectURL(this.files[0]));
```

7、 隐藏信息

有一些信息需要提交给处理页面，但是又不能让用户看见，比如修改用户信息时需要提交的用户的 id 号（一般是数据库自动产生的），这时就要用到<input>元素的“hidden”类型，配合 POST 提交方式，用法如下：

```
<input type="hidden" name="" value="">
```

这样提交时组合 name 和 value 属性的值成“键/值”对。在浏览器视窗中，用户看不到也不能修改，但是能在右键菜单“查看网页源代码”中查看或者在开发者工具查看和修改，所以不要使用“hidden”类型的<input>元素提交敏感信息。看例子：

```
<form action="#">
    <input type="hidden" name="hid" value="108731">
    <button autofocus type="submit" formmethod="POST">确认</button>
</form>
```

8、 滑动输入

点过鼠标上下或左右滑动来输入数字，参考图 4.51 所示。



图 4.51 滑动输入数字

语法如下：

```
<input type="number" name="" [list="" max="" min="" placeholder="" readonly
step="" value=""]>
<input type="range" name="" [list="" max="" min="" step="" value=""]>
```

可选属性 list 指定了提供输入域的<datalist>元素的 id，max 属性定义了允许输入的最大值，min 是允许输入的最小值，要求 max≥min，step 是步进（默认为 1），当步进为小数时允许输入实数，比如 step="0.01"，value 属性定义初始值，此后可以用 JavaScript 语句修改它的值，从而控制滑块的位置。看例子：

```
<form action="#">
    <label>评分:
       <input type="number" name="score" value="5" max="10" min="0"></label><br>
    <label for="tem" >体温: </label>
    <input type="range" name="temp" max="43" min="33" step="0.1" value="37.5"
id="tem">
    <br><input type="submit" value="提交">
</form>
```

浏览的效果如图 4.52 所示。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

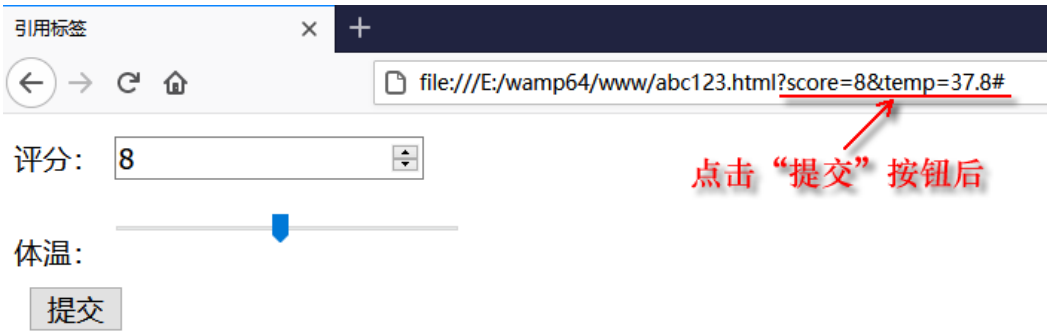


图 4.52 滑动输入数字

通过样式 width 可以改变轨道的长度，通过样式 height 可以改变轨道的样子，也可以深度定制规定和滑块，比如利用图 4.53 左侧的两幅小图片产生右侧所示的滑动标尺。

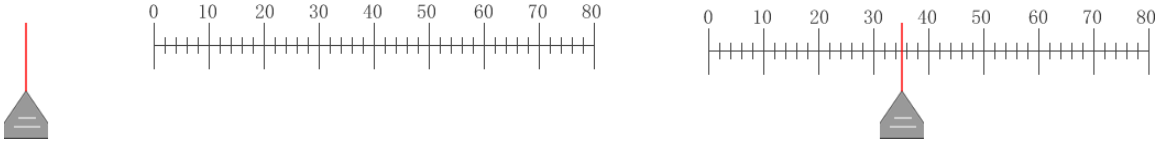


图 4.53 定制滑块和轨道

先准备图 4.53 左侧所示的滑块图片文件 slider.png（尺寸为 40*104）和轨道图片文件 track.png（图中间部分，尺寸为 440*67），然后应用如下样式即可：

```
<head>
    <style>
        input[type=range] {
            -webkit-appearance: none;
            width: 440px;
        }

        input[type=range]:focus {
            outline: none;
        }

        input[type=range]::-webkit-slider-runnable-track {
            width: 440px;
            height: 67px;
            background-origin: content-box;
            background-image: url("track.png");
        }

        input[type=range]::-webkit-slider-thumb {
            -webkit-appearance: none;
            height: 104px;
            width: 40px;
            margin-top: 22px;
            background-image: url("slider.png");
            background-origin: border-box;
        }
    </style>
</head>
```

```
<body>
    <input type="range">
</body>
```

关于样式价绍请参考第 5 章。

实操练习：利用 JS 的循环定时器慢慢把滑块从左侧移动右侧（提醒：不断增大 value 属性的值）。

9、 输入密码

密码输入参考如 4.54。



图 4.54 输入密码

采用 password 类型，其语法如下：

```
<input type="password" name="" [ 其他可选属性 ]>
```

其他属性包括 maxlength、minlength、required、placeholder、readonly、size、autocomplete、inputmode、pattern 等。其中属性 autocomplete 可取的值除了之前讲过的"on"、"off"外，还增加了"current-password"、"new-password"，分别提醒浏览器或密码管理器要输入以前的密码或新密码，如果用户曾经输入过密码，autocomplete="current-password"会自动填充，而 autocomplete="new-password"一般在新建用户或修改密码时使用。

属性 inputmode 主要用在移动设备上，选择什么类型的虚拟键盘，可取的值包括 none、text、tel、url、email、numeric、decimal 和 search。想必大家对图 4.55 所示的几种虚拟键盘很熟悉吧。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

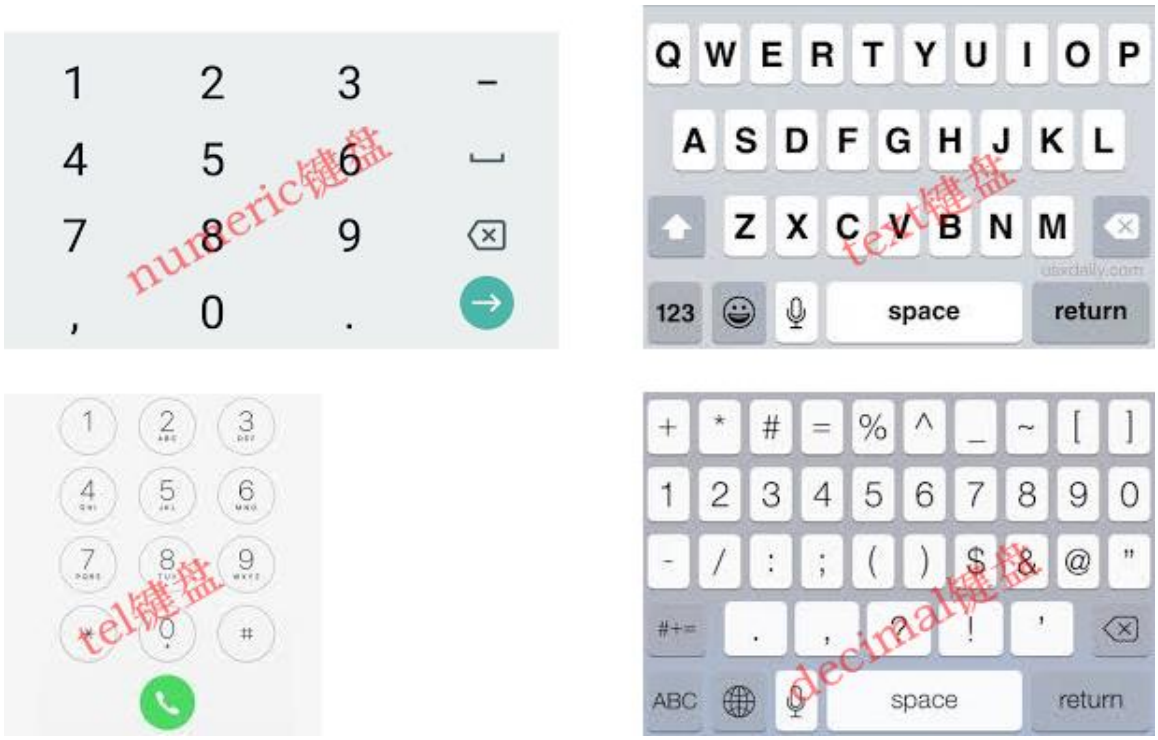


图 4.55 移动设备上的虚拟键盘

属性 pattern 定义输入密码必须符合的规则——正则表达式，比如包含数字、字母，长度大于 8 等，否则提交时会提示警告，类似于图 4.56 所示。



图 4.55 密码输入规则举例

比如至少 6 位数字：pattern="\d{6,}"，再比如必须包含字母、数字且字个数至少 8 个，最大不能超过 16 个：pattern="^(?=.*[a-zA-Z])(?=.*\d)[a-zA-Z\d]{8,16}$"。大家看看下面的例子输入的密码必须满足什么要求：

```
<form action="#">
    <label>请输入密码: <input type="password" name="pass" pattern="\w{9,}"></label>
    <br><input type="submit" value="提交">
</form>
```

一些重要的正则表达式元字符参见表 4.14。

表 4.14 常用的正则表达式元字符

元字符	说明

\	将下一个字符标记为一个特殊字符、或一个原义字符、或一个 向后引用、或一个八进制转义符。例如，'n' 匹配字符 "n"。'\n' 匹配一个换行符。序列 '\\' 匹配 "\" 而 "\(" 则匹配 "("。
^	匹配输入字符串的开始位置。如果设置了 RegExp 对象的 Multiline 属性，^ 也匹配 '\n' 或 '\r' 之后的位置。
$	匹配输入字符串的结束位置。如果设置了 RegExp 对象的 Multiline 属性，$ 也匹配 '\n' 或 '\r' 之前的位置。
*	匹配前面的子表达式零次或多次。例如，zo* 能匹配 "z" 以及 "zoo"。* 等价于{0,}。
+	匹配前面的子表达式一次或多次。例如，'zo+' 能匹配 "zo" 以及 "zoo"，但不能匹配 "z"。+ 等价于 {1,}。
?	匹配前面的子表达式零次或一次。例如，"do(es)?" 可以匹配 "do" 或 "does" 。? 等价于 {0,1}。
{n}	n 是一个非负整数。匹配确定的 n 次。例如，'o{2}' 不能匹配 "Bob" 中的 'o'，但是能匹配 "food" 中的两个 o。
{n,}	n 是一个非负整数。至少匹配 n 次。例如，'o{2,}' 不能匹配 "Bob" 中的 'o'，但能匹配 "foooood" 中的所有 o。'o{1,}' 等价于 'o+'。'o{0,}' 则等价于 'o*'。
{n,m}	m 和 n 均为非负整数，其中 n <= m。最少匹配 n 次且最多匹配 m 次。例如，"o{1,3}" 将匹配 "fooooood" 中的前三个 o。'o{0,1}' 等价于 'o?'。请注意在逗号和两个数之间不能有空格。
\b	匹配一个单词边界，也就是指单词和空格间的位置。例如， 'er\b' 可以匹配"never" 中的 'er'，但不能匹配 "verb" 中的 'er'。
\B	匹配非单词边界。'er\B' 能匹配 "verb" 中的 'er'，但不能匹配 "never" 中的 'er'。
\d	匹配一个数字字符。等价于 [0-9]。
\D	匹配一个非数字字符。等价于 [^0-9]。
\n	匹配一个换行符。等价于 \x0a 和 \cJ。
\s	匹配任何空白字符，包括空格、制表符、换页符等等。等价于 [\f\n\r\t\v]。
\S	匹配任何非空白字符。等价于 [^ \f\n\r\t\v]。
\t	匹配一个制表符。等价于 \x09 和 \cI。
\w	匹配字母、数字、下划线。等价于'[A-Za-z0-9_]'。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

\W	匹配非字母、数字、下划线。等价于 '[^A-Za-z0-9_]'。

实操练习：实现如下图所示的密码输入框（提示：在睁眼/闭眼小图片的点击事件中，更改<img>元素对象的 src 属性的值——在 openeye.png 和 closeeye.png 两图片中切换，同时修改<input>元素的 type 属性的值——在 text 和 password 中切换，采用语句 this.setAttribute("src","openeye.png")。



10、 搜索输入框

参考示意图 4.56。

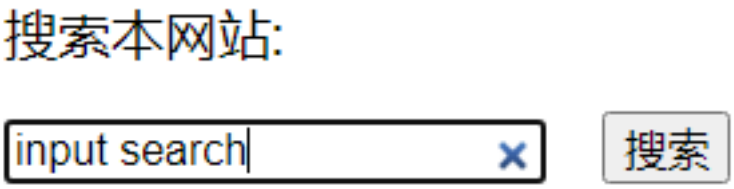


图 4.56 搜索输入框

<input>元素的 search 类型与 text 类型差不多，但是不同的浏览器渲染效果不同，上图是 chrome 浏览器的渲染效果。语法如下：

```
<input type="search" name="" [ 其他可选属性 ]>
```

其他属性包括 list、maxlength、minlength、placeholder、readonly、size、pattern、spellcheck、aria-label 等。一般在页面上加一个搜索框，不会单独增加标题（如上图中的“搜索本网站”），因为谁都知道就是搜索站内信息，但这时读屏器就没法读出关于搜索框的信息，因此属性 aria-label 就派上用场了，它的值就是标题，但是不会显示在屏幕上。属性 spellcheck 激活拼写检查。看例子：

```
<input type="search" name="sear" spellcheck="true" placeholder="输入关键字" aria-label="搜索站内信息">
```

11、 其他输入框

<input type="tel" …> - 输入电话号码，没有默认的检查模式，需要根据实际情况设计正则表达式并赋给 pattern 属性。

<input type="url" …> - 输入网络资源 URL，默认检查是必须以 http://或 https://开头。

<input type="email" …> - 输入邮件地址，默认检查是必须包含“@”字符。

<input type="text" ···> - 这是<input>元素的默认类型，为通用的输入框，可以输入任何字符串。

12、个性化输入框

大家应该对于谷歌的登录画面感到惊奇，参见下图所示：



可用下面的代码实现这个效果。

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>新式输入框</title>
    <style>
        * {
            left: 0;
            margin: 0;
            font-family: STZhongsong, 华文中宋, 华文行楷, serif;
        }

        div {
            position: relative;
            top: 100px;
            left: 100px;
        }

        span {
            font-size: 18px;
            color: gray;
            position: absolute;
            transition: all 0.5s;
            top: 2px;
        }

        input {
            position: absolute;
            background-color: transparent;
            border-style: none;
            border-bottom: 1px solid;
            outline: none;
            font-size: 20px;
        }
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    </style>
    <script>
        window.onload = function () {
            document.querySelector("input").addEventListener("focus", (evt) => {
                evt.target.previousElementSibling.style.cssText  =  "color:  blue;
font-size: 14px; top: -1.2em;";
                evt.target.style.borderColor = "blue";
            });
            document.querySelector("input").addEventListener("blur", (evt) => {
                if (evt.target.value.trim() == "") {
                    evt.target.previousElementSibling.style.cssText = "color: gray;
font-size: 18px; top: 2px;";
                    evt.target.value = "";
                } else {
                    evt.target.previousElementSibling.style.color = "gray";
                }
                evt.target.style.borderColor = "black";
            });
        }
    </script>
</head>

<body>
    <div>
        <span>用户名</span>
        <input>
    </div>
</body>

</html>
```

4.9.4 按钮

相比<input type= "button|submit|image" …>所生成的按钮，<button>元素能生成更复杂更漂亮的按钮，它的使用语法如下：

```
<button autofocus disabled name="" type="" value="" form="" formaction="" forme
nctype="" formmethod="" formnovalidate formtarget="">元素内容</button>
```

属性 type 可取值："button"（默认值）- 普通按钮、"submit"- 表单提交按钮、"reset"- 表单复位按钮，如果作为表单提交，那么属性 name 和 value 的值组成“键/值”对提交给处理页面。其他的属性与<input>元素的按钮类型属性一样，不过只有 type="submit"时，formaction、formenctype、formmethod、formnovalidate 和 formtarget 才有意义。<button>的元素内容允许包含更多的子签标，从而可以设计漂亮复杂的按钮。看例子：

```
<button autofocus name="prettyButton" >
    <svg aria-hidden="true" viewBox="0 0 10 10">
        <path d="M7 9L5 8 3 9V6L1 4h3l1-3 1 3h3L7 6z"/>
    </svg>
     加入心愿单
</button>
```

<svg>元素参见“§4.5.5 SVG 矢量图”小节，生成的按钮效果如图 4.57。



图 4.57 漂亮的按钮

4.9.5 多行编辑

<textarea>元素代表多行编辑框，之前讲述的<input>是单行的。其语法如下：

```
<textarea autocomplete autofocus cols="" rows="" disabled form="" maxlength=""
name="" placeholder="" readonly required spellcheck="" wrap=""> 编辑的内容
</textarea>
```

没有 value 属性，编辑的内容就是<textarea>元素的内容。属性 cols 定义一行内可见的字符数，默认 20，属性 rows 定义可见的行数，如果<textarea>元素本身的宽度和高度不足以显示，就会自动产生滚动条。属性 wrap 定义自动换行方式，可能取值有："hard" - 一行字符数等于 cols 时自动插入换行符，"soft"（默认值） - 不插入换行符，可能会折行显示。看例子：

```
<form id="res" ></form>
<textarea autocomplete cols="20" rows="5" form="res" name="resume" placeholder=
"输入你的个人介绍" wrap=""></textarea>
<br>
<button type="submit" form="res" formaction="#" formmethod="GET">提交</button>
```

产生一个 5 行 20 列的输入框，当实际输入的行数超过 5 行时会出现垂直滚动条，浏览效果如图 4.58。

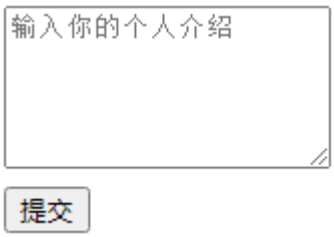
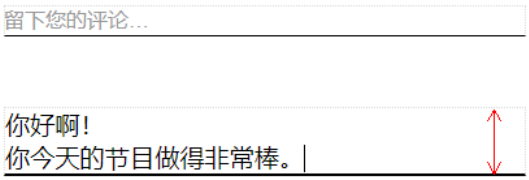


图 4.58 多行输入框举例

通过给元素添加属性“contenteditable="TRUE"”，实现多行输入的方法有很多，比如采用 元素实现多行编辑，类似 facebook 的评论输入框，自动适应内容的高度，用户体验非常好，参考下图所示：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



下面的代码供你参考：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>自适应高度的多行输入</title>
    <style>
        #newTexterea {
            border-style: dotted dotted solid;
            border-color: #DDD #DDD #000;
            border-width: 1px;
            width: 400px;
            outline-style: none;
            font-size: 1.2rem;
        }

        #newTextereaPlaceholder {
            position: absolute;
            color: #999;
            top: 0px;
            z-index: -100;
        }

        hr {
            position: absolute;
            border: 1px solid;
            margin-top: -2px;
            width: 1px;
            left: 0px;
            transition: width 1s;
        }
    </style>
</head>

<body>
    <div style="position: relative;">
        <div id="newTexterea" contenteditable="TRUE"></div>
        <span id="newTextereaPlaceholder">留下您的评论…</span>
        <hr>
    </div>
    <script>
        window.addEventListener("load", () => {
            let textObj = document.getElementById("newTexterea");
            textObj.addEventListener("keyup", (evt) => {
                evt.target.nextElementSibling.style.display = evt.target.innerText == "" ?
"block" : "none";
            });
```

```
        textObj.addEventListener("focus", (evt) => {
            evt.target.nextElementSibling.nextElementSibling.style.width = "401px";
        });
        textObj.addEventListener("blur", (evt) => {
            evt.target.nextElementSibling.nextElementSibling.style.width = "0px";
            console.log(evt.target.innerText); //显式输入的内容。
        })
    })
  </script>
</body>

</html>
```

4.9.6 可视化计量

<meter>元素对已知范围的计量数据生成可视化的效果，比如当前电量、硬盘空闲容量、温度等，如图 4.59 所示。



图 4.59 可视化计量举例

<meter>元素的使用语法如下：

```
<meter [form="" min="" max="" low="" high="" optimum="" value=""]>您的浏览器不支持meter 元素</meter>
```

一些老旧的浏览器不支持这个 HTML5 新增的<meter>，这时就显示元素的内容，即“您的浏览器不支持 meter 元素”。<meter>元素的属性之间的关系参考图 4.60。

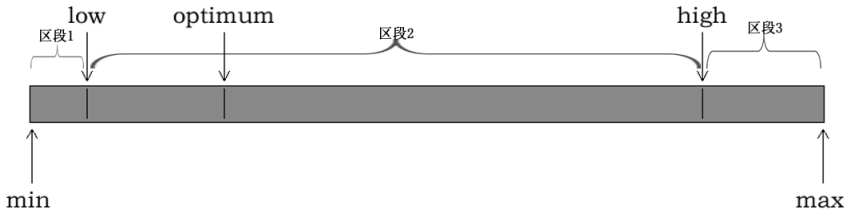


图 4.60 <meter>元素各个属性之间的关系

value 的取值范围是[min,max]，optimum 表示最佳值，比如充电电池长期保存的最佳电量为 75%，最佳体温是 37.5° 等，form 属性指明本<meter>元素归属于哪个表单（值为<form>的 id），如果用了 form 属性还需加上 name 属性，这样提交表单时后端才可以获得<meter>元素的 value 属性的值，通过 form 属性，我们可以把 meter 元素放在<form>表单之外。

当省略 min, max, low, high, optimum, value 时，默认值 min=low=value=0, max=high=1, optimum=0.5，也就是 value 的取值范围为[0,1]。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

属性 min、low、high 和 max 的值必须满足条件：min≤low≤high≤max，这四个属性把 value 的取值范围划分出四个区段（参见上图），当 value 和 optimum 属性的值落在同一个区段时显示为绿色，如： ，落在相邻的区段时显式为黄色，如： ，否则显式为红色，如： 。

看例子：

```
<meter min=0  max=100 low=10  high=90 optimum=60 value=70 style="transform:rotate(-90deg);width:200px;">不支持 meter 元素</meter>
```

浏览效果参见图 4.61，例子中采用了逆时针旋转 90 度的样式，关于样式介绍，请参考第 5 章。



图 4.62 <meter>元素举例

可通过 JS 编程修改 value 的值，就能动态改变计量状态，看例子：

```
<meter min="0" max="100" low="20" high="80" optimum="60" value="40" style="width:300px;">您的浏览器不支持 meter 元素</meter><br>
<input type="button" value="增加" onclick="incre()"/>
……
<script>
    function incre(){
        var  me = document.getElementsByTagName("meter")[0];
        me.value = me.value + 10;
}
</script>
```

浏览效果见图 4.63。



图 4.63 动态改变<meter>的计量值

点击“增加”按钮就会看到绿色往右扩展。

4.9.7 进度条

采用<progress>元素产生进度条，进度条用于指示某项操作（比如下载）的完成进度，例如生成如图 4.64 所示的进度条，采用 JS 代码通过设置元素的 value 的值来控制进度。



图 4.64 进度条

此元素的语法如下：

```
<progress[ max="" value="" ]>您的浏览器不支持 progress 元素</progress>
```

如果浏览器版本太低，可能不支持这个新增元素，这样就会在页面上显示“你的浏览器不支持 progress 元素”。如果省略 max，默认为 1,如果省略 value，默认为 0。看例子：

```
<progress value="20"  max="100">您的浏览器不支持 progress 元素</progress>
```

<progress>元素对象还有一个属性 position，表示当前的进度，它等于 value/max，因此介于 0 到 1 之间，这个属性只能通过 JS 代码读取。例子 chapter-4\index-3.html 演示一个既能前进又能回滚的进度条，通过定时器不断增大 value 属性的值，进度条就前进，反之不断减小 value 的值，进度条就回滚，本例子浏览的效果见图 4.65。



图 4.65 能前进/后退的进度条

4.9.8 下拉框

采用<select>、<optgroup>和<option>元素生成下拉框，如图 4.66 所示。



图 4.66 下拉框示意图

它们组合的基本用法如下：

```
<select autocomplete="" autofocus disabled form="" multiple name="" required si
ze="">
    <optgroup disabled label="">
        <option value="" disabled label="" selected>显示内容</option>
        <option value="" disabled label="">显示内容</option>
        ……
    </optgroup>
    <optgroup disabled label="">
        <option value="" disabled label="" selected>显示内容</option>
        <option value="" disabled label="">显示内容</option>
        ……
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    </optgroup>
    ……
</select>
```

<option>元素请参考“§4.9.1”小节。<optgroup>元素对条目分组，比如上图中的“广东省”和“江西省”就是分组，组名由 label 属性定义，如果存在属性 disabled，那么本组内的所有条目没法选择。<select>元素中的属性 multiple 表示能选择多个条目（按 Ctrl【windows】/Command【mac】并点击鼠标），否则只能选择一个，属性 size 定义下拉框能显示的行数，默认 1 行，其他属性与<input>元素的同名属性一样。看例子：

```
<form action="#">
    <label for="cit">选城市:</label>
    <select id="cit" name="city" multiple>
        <optgroup label="广东省">
            <option value="gz">广州市</option>
            <option value="sz">深圳市</option>
        </optgroup>
        <optgroup label="江西省">
            <option value="nc">南昌市</option>
            <option disabled value="ja">吉安市</option>
            <option value="jj">九江市</option>
        </optgroup>
    </select>
    <input type="submit">
</form>
```

浏览效果如图 4.67。

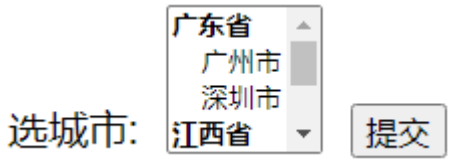


图 4.67 通过下拉框输入

按住 Ctrl 键，并用鼠标点击多个条目（比如深圳和九江），最后点击“提交”按钮，会发现浏览器的地址栏中的网址后面追加了“?city=sz&city=jj#”，大家思考下，为什么会这样？

4.9.9 输出结果

<output>是一个容器元素，页面或应用程序可以在其中注入计算结果或用户操作的结果，其使用语法是：

```
<output[ form="" for="" name="" ]>计算结果</output>
```

form 属性指明关联的<form>元素的 id，格式是 form="form_id1 form_id2 ..."，如果用了 form 属性还需加上 name 属性（表单提交时需要使用到 name 属性），for 属性指定计算中用到的值来

自哪些元素（引用其 id），中间用空格分开。所以<output>元素允许位于 form 元素之外，且一个<output>元素可以关联多个<form>元素。

看例子：

```
<label for="rate1">利率: </label>
<input type="range" min="0" max="100" id="rate1" value="10" oninput="thisRate.v
alue = rate1.value">
<output id="thisRate" for="rate1">10</output>%
```

用到了<input>元素的 oninput 事件，即只要有输入就触发执行“thisRate.value = rate1.value”。浏览效果见图 4.68。



图 4.68 <output>元素举例

再看例子：

```
<form oninput="x.value=parseInt(a.value) + parseInt(b.value)">
    <input type="range" id="a" value="50">
   +<input type="number" id="b" value="50">
   =<output name="x" for="a b">100</output>
</form>
```

只要<form>元素中的任何子元素有输入，就会触发执行“x.value=parseInt(a.value) + parseInt(b.value)”，里面的 parseInt()为 JS 函数，它把字符串转换为整数。浏览效果见图 4.69。

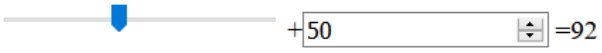


图 4.69 利用<output>元素作加法

实操练习：完成如下的一个计算公式，输入三个数，自动显示计算结果



4.9.10 控件分组

对于一个复杂的表单，往往需要对输入的信息分组，比如基本信息、教育培训、工作经验等。参考示意图 4.70。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

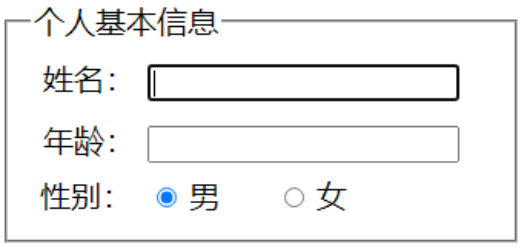


图 4.70 控件分组示意图

采用<fieldset>和<legend>元素实现交互元素分组，它们使用的语法如下：

```
<fieldset disabled form="" name="">
    <legend>分组标题</legend>
    交互元素……
</fieldset>
```

<fieldset>的属性与<input>元素的同名属性意义相同，<legend>定义分组的标题，它可以出现在<fieldset>元素内部任何地方。看下面的例子：

```
<fieldset>
    <legend>个人基本信息</legend>
    <label>姓名: <input name="xm" autofocus></label><br>
    <label>年龄: <input name="age"></label><br>
    <label>性别:
        <label><input type="radio" name="sex" checked>男</label>  
        <label><input type="radio" name="sex">女</label>
    </label>
</fieldset>
```

浏览的效果参见图 4.70。

4.9.11 综合例子

设计一个注册新账号的网站，后端采用 php+MariaDB 数据库。我们在“§1.5.1.1 WEB 服务”小节安装的 WampServer 满足要求，现在启动 WampServer。

1、设计数据库表

我们把注册信息保存在 user_db 数据库的 user_tb 表中，首先参照图 4.71 所示登陆到 MariaDB 数据库管理系统中。

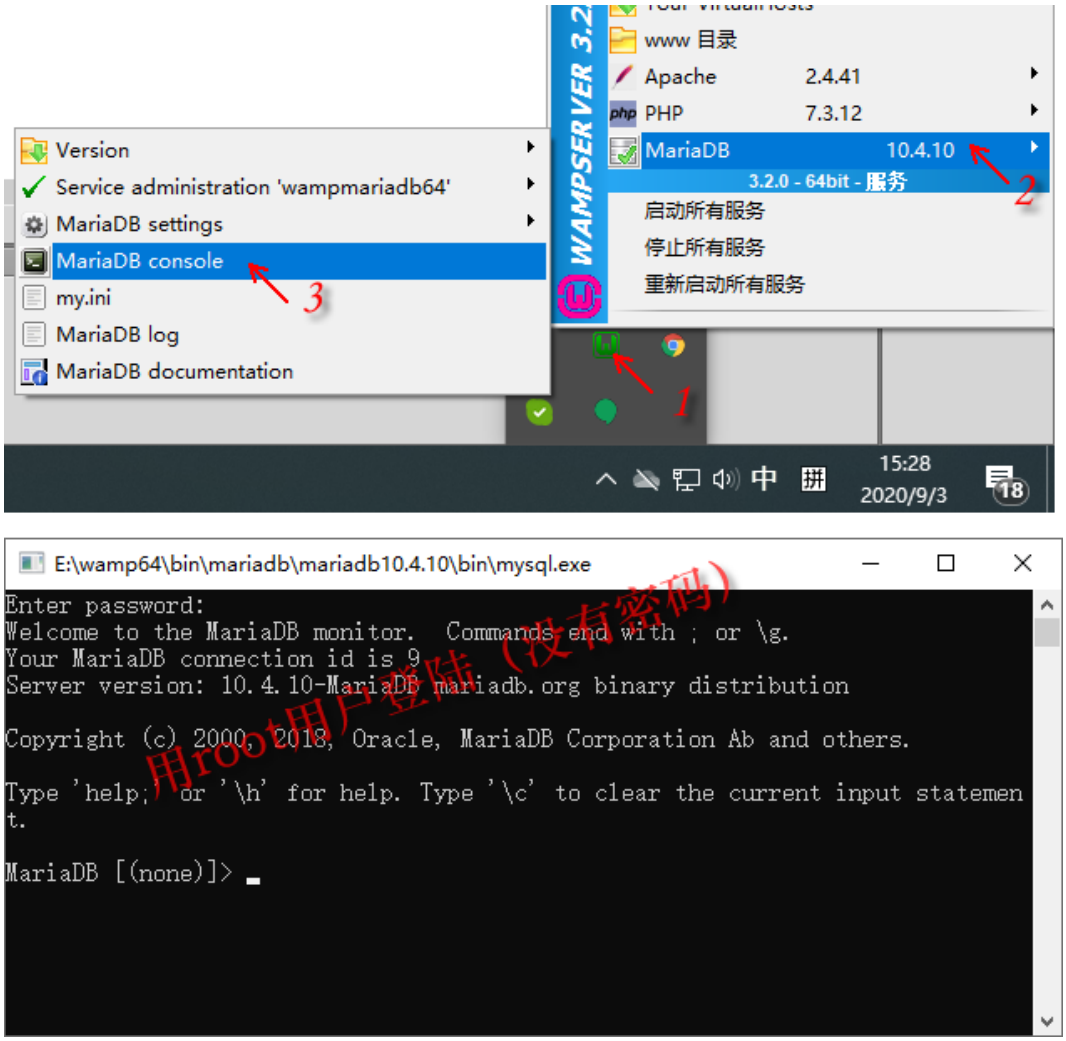


图 4.71 登录到 MariaDB 数据库

然后使用下面的语句建库、建表：

```
CREATE DATABASE user_db CHARACTER SET 'utf8';
USE user_db;
CREATE OR REPLACE TABLE user_tb(
    user CHAR(10) NOT NULL PRIMARY KEY,
    password CHAR(32) NOT NULL,
    name CHAR(12) NULL,
    phone CHAR(11) NOT NULL,
    sex CHAR(6) NOT NULL DEFAULT '男'
);
```

2、页面美工

前端注册页面美工如图 4.72 所示，可以用 photoshop 或者 fireworks 绘制。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 4.72 注册页面美工

3、设计前端 html 文件

前端代码参见 chapter-4\register.html 文件，注意账号和密码输入有要求，采用<input>元素的 pattern 属性定义正则表达式，分别是 pattern="[a-z]\w{2,}"、pattern="^(?=.*[a-zA-Z])(?=.*\d)[a-zA-Z\d]{8,16}$"，大家参照“§4.9.3 <input>元素”小节中的“表 4.14 常用的正则表达式元字符”，不难理解这两个正则表达式。

4、设计后端 php 程序

后端代码保存在 chapter-4\register.php 文件中，并和前端页面文件 register.html 都拷贝到 WampServer 安装目录中的 www 目录下，然后浏览网址 http://localhost/register.html 即可。

前端页面以 GET 方式提交过来的数据放在 php 程序的$_GET 数组中，以 POST 方式提交过来的数据放在$_POST 数组中——$_POST["name"]保存了输入的用户名，$_POST["password"]存储了密码，依次类推。本例前端采用 POST 方法提交表单，后端把信息插入到 MariaDB 的 user_tb 表即可。

4.10 嵌入内容

“HTML5 元素速查表 • 嵌入内容”分类列举的元素用于在页面中嵌入对象，比如图片、音频、视频、画布等。嵌入内容的元素都是可替换的元素——元素的内容由外部资源替换。

本节介绍这些元素：<area>、<map>、<audio>、<canvas>、<embed>、<iframe>、<img>、<object>、<param>、<picture>、<source>、<svg>、<track>、<video>。

4.10.1 自适应图片

让图片自动适应各种尺寸的屏幕，是目前网站开发人员梦寐以求的事情。HTML5 采用<img>和<picture>并结合<source>元素来完成自适应显示图片，是的，就这么简单！

<img>元素

在页面中嵌入图片，它的语法如下：

```
<img src="" [ alt="" crossorigin="" decoding="" height="" ismap loading="" referrerpolicy="" sizes="" srcset="" width="" usemap=""]>
```

这个元素的属性比较多，下面对它们一一讲解。

alt 属性 - 当图片不可用时显示这个属性的值，非可视化的工具（比如读屏器）也会使用这个值，一般为图片的描述。虽然这个属性是可选的，但是建议不要省略它。

srcset 和 sizes 属性 - 为不同的屏幕定义不同的图片，以满足响应式页面的需要。srcset 属性的值有两种格式：

（1）srcset="图片 1 的 URL　图片 1 的宽度, 图片 2 的 URL　图片 2 的宽度, 图片 3 的 URL　图片 3 的宽度, ……"，宽度单位固定为“w”。

（2）srcset="图片 1 的 URL　DPR1, 图片 2 的 URL　DPR2, 图片 3 的 URL　DPR3, ……", DPR 就是设备像素比率，在这里要加上单位“x”，不能出现多个相同的 DPR。关于 DPR，请参见“§5.2 长度单位”小节。

如果采用格式（1），必须结合 sizes 属性才行，sizes 属性的值的格式为：sizes="媒体查询 1　图片显示宽度 1, 媒体查询 2　图片显示宽度 2, ……，图片显示宽度 n"，其中“图片显示宽度”单位为 css3 的长度单位（参见“§5.2 长度单位”小节）。浏览器通过“媒体查询”得到相应的“图片显示宽度”，然后再在 srcset 的格式（1）中找到不比“图片显示宽度”小的最小宽度的那个图片，最后显示这个图片。看例子：

```
<img
    src="default.jpg"
    srcset="fairys.jpg 480w, fairym.jpg 700w, fairyl.jpg 1280w"
    sizes="(max-width: 600px) 480px, (600px<width<970px) 600px, 1200px"
    alt="Alice goes to the fair.">
```

假如我的手机屏幕的宽度为 650px，查询媒体后得到“图片显示宽度”为 600px，再在 srcset 属性的值中找宽度不比 600 小的最小图片，就是 700w 对应的 fairym.jpg，最后显示这个图片。

srcset 属性的值如果用格式（2），就不要 sizes 属性，这时浏览器根据设备像素比率 DPR 选择合适的图片。看例子：

```
<img
    alt="特等深山野茶"
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    srcset="
        http://youduo.me/500.jpg 1x,
        http://youduo.me/850.jpg 2x,
        http://youduo.me/1280.jpg 3x,
        http://youduo.me/2300.jpg 5x"
    src="http://youduo.me/default.jpg">
```

如果手机的设备像素比例为 2，那么就显示 850.jpg。

decoding 属性 – 给浏览器提供解码线索，可取值为："sync"同步解码，以便与其他内容捆绑显示；"async"异步解码，以便尽快显示其他内容，这时你会先看到页面上的文字，然后才逐渐看到各个图片；"auto"（默认值），由浏览器决定最佳的解码方法。

width 和 height 属性 - 定义图片的原始宽度和高度，为整数，不要带单位，默认为 px。这样在加载图片之前浏览器就预留了图片显示空间，从而不会导致图片加载过程中不断调整周边内容的位置。如：<img alt="15 岁的我" width="1024" height="700" src="15old.jpg">。

ismap 属性 - 如果存在此属性，就表明图片是后端地图的一部分，这样用户点击位置的坐标会发回到后端。

loading 属性 - 指示浏览器加载图片的方法。可取值为："eager"（默认值）立即加载图片，即使图片在文档流的位置不在视窗内；"lazy"推迟加载图片，直到图片在文档流中的位置快进入视窗，比如页面很长，需要长时间翻页才显示的图片最好采用 loading="lazy"属性，因为很多用户还没有翻页到最后就关闭了网页，这样后面的图片根本不会加载，从而节省了流量。

usemap 属性 - 为页面另一个<map>元素的 id，从而建立<img>与<map>的关联，具体参见下一节“创建图片热点”。

crossorigin 和 referrerpolicy 属性参考前面<link>元素和<script>元素的同名属性。

<picture>、<source>元素

<picture>结合<source>、<img>提供了一个更精细化的自适应显示图片的方法。语法如下：

```
<picture>
    <source media="" sizes="" srcset="" type="">
    <source media="" sizes="" srcset="" type="">
    ……
    <source media="" sizes="" srcset="" type="">
    <img alt="" src="">
</picture>
```

<picture>元素没有专用属性，子元素中必须要有一个<img>元素和若干个可选的<source>元素。浏览器从第一个<source>开始匹配，不成功的话继续匹配第二个，一旦匹配成功就不再匹配剩下的<source>了。最后在匹配成功的<source>中选择合适的图片，并生成<img>子元素的属性值，从而通过<img>元素显示图片。所以一定要记住：只能在<img>子元素上添加样式，而不能在<picture>附加任何样式，比如要给图片添加边框，只能这样做：img {border:…}。<picture>元素的<img>子元素一般只用 alt 和 src 属性即可，备选图片通过<source>元素定义。

<source>元素的 media 属性的值遵循媒体查询的语法（参见“§ 5.6.9 媒体查询”）。srcset 和 sizes 属性与<img>元素的同名属性一样，当 srcset 中采用 DPR 时，就不要 sizes 属性，否则不能缺少 sizes 属性。type 属性指明图片类型，可取的值有："image/apng"(APNG 图片）、"image/bmp"（bmp

图片）、"image/gif"（gif 图片）、"image/x-icon"（icon 图片）、"image/jpeg"（jpeg 图片）、"image/png"（png 图片）、"image/svg+xml"（svg 图片）、"image/tiff"（tiff 图片）、"image/webp"（webp 图片）。看例子：

```
<picture>
    <source media="(min-width: 36em)"
srcset="large.jpg 1024w, medium.jpg 640w, small.jpg 320w" sizes="33.3vw"/>
    <source srcset="crop-large.jpg 2x, crop-small.jpg 1x"  type="image/jepg"/>
    <img src="small.jpg" alt="一只来自北方的狼" />
</picture>
```

如果屏幕宽度大于 36em，就匹配成功第一个<source>子元素，这样就把 srcset 和 sizes 属性添加到<img>元素中，此后的过程就参考前面讲过的<img>。需要注意的是第一个<source>元素的属性的值 sizes="33.3vw"，表示图片显示出来的宽度是视窗宽度的三分之一，因此视窗宽度不同，三分之一宽度对应的像素数也不同，浏览器由此就能选择合适的图片了，选择原则依然是在不比显示宽度小的图片中选择那个最小的图片。

4.10.2 创建图片热点

鼠标在一幅图片上停靠或点击不同的区域就能激活相应的链接，图片上这样的一个区域就称为一个图片热点，例如在如图 4.73 所示的地图上，鼠标停靠江西时弹出提示，点击时打开一个网页。

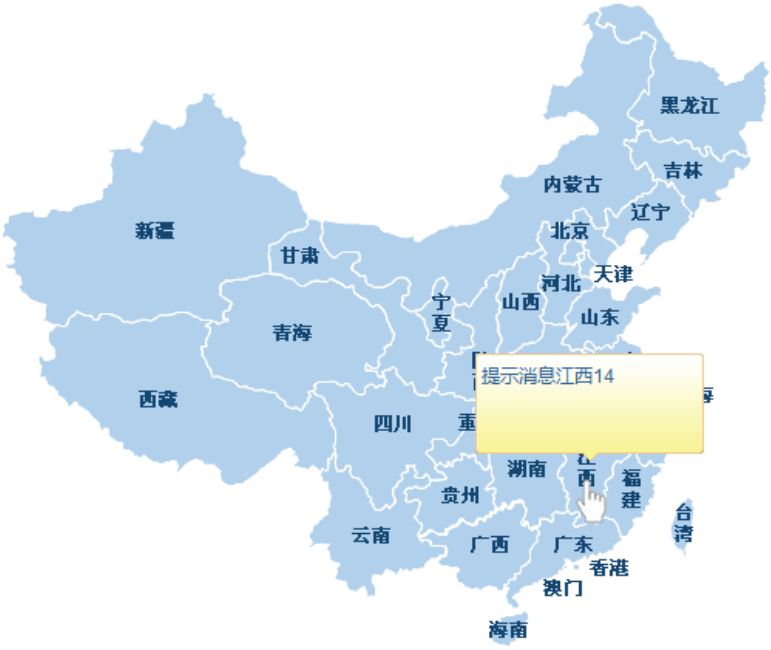


图 4.73 图片热点示意图

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

创建图片热点主要使用<map>和<area>元素并结合<img>元素，而且<area>必须作为<map>的子元素存在。它们配合使用的语法如下：

```
<map name="唯一名字">
  <area alt="" coords="" download href="" ping="" referrerpolicy="" rel="" shape="" target="" />
  ……
</map>
<img usemap="#map的唯一名字" ……>
```

由<img>元素显示图片，<map>和<area>元素定义热点。<map>元素可以放在页面文件中的任意位置，通过<img>元素的 usermap 属性指定<map>元素的名字，来建立两者的关联。<map>元素中的一个子元素<area>定义一个热点——具体由它的属性定义。

alt 属性 - 如果浏览器不能显示图片，就显示 alt 属性的值。如果存在 href 属性，alt 属性必须存在。

coords、shape 属性 - 定义热点区域的形状和相应的坐标。由 shape 属性定义形状，可取的值有："rect"矩形、"circle"圆形、"poly"多边形、"default"图片上剩下的全部区域。形状确定后就用 coords 属性定义坐标了，它的值是用逗号分隔的一系列整数，整数的个数和各自的意思与区域形状有关：

shape="rect" coords="左上角坐标,右下角坐标"，比如 coords="19,28,222,228"。

shape="circle" coords="圆心坐标,半径"，比如 coords="100,50,30"。

shape="poly" coords="各个顶点的坐标"，比如 coords= "3,47,45,12,105,7,140,60,120,125,12,90"。

注意坐标的格式是 x,y,是相对于图片左上角而言的,也就是顶点离图片左上角的水平距离为 x，垂直距离为 y。下面看一个例子，在一幅美女图片上创建三个热点，见图 4.74 所示。

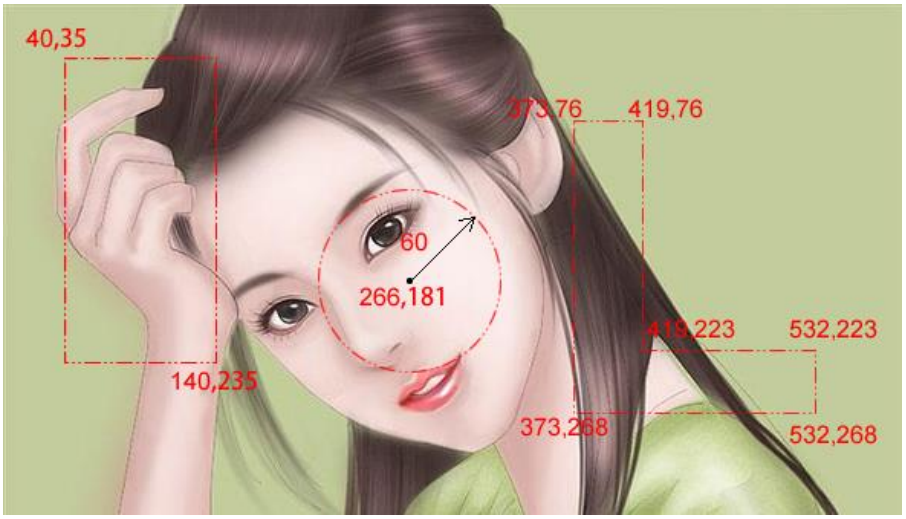


图 4.74 图片热点举例

红色虚线和坐标值是为了演示用的，浏览时是没有的。

```
<img src="spot.jpg" usemap="#spot-me">
<map name="spot-me">
```

```
    <area alt="矩形热点" shape="rect" coords="40,35,140,235" title="矩形热点" href="#">
    <area shape="circle" coords="266,181,60" title="圆形热点" href="product.html">
    <area shape="poly" coords="373,76,419,76,419,223,532,223,532,268,373,268" href="#">
</map>
```

浏览时在图片上移动鼠标，一旦进入热点，鼠标指针变为手指形状，点击就能打开相应的页面。

<area>元素的其他属性请参考<a>元素的同名属性。

4.10.3 多媒体元素

与多媒体有关的 HTML5 元素有<audio>、<video>、<source>、<embed>和<track>五个。

4.10.3.1 <audio>

<audio>元素用于在网页中播放音频内容，音频资源可以通过元素的 src 属性指定，或由<source>元素定义。此元素的语法如下：

```
<audio autoplay controls crossorigin="" loop muted preload="" src="">
    [<source src="" type="">
     <source src="" type="">
     <source src="" type="">
     ……
     你的浏览器不支持 audio 元素]
</audio>
```

如果浏览器不支持<audio>元素，就不会播放音频，并且在网页上显示“你的浏览器不支持 audio 元素”（可以采用元素来给提示的信息附加样式，比如“你的浏览器不支持 audio 元素”），以便提醒用户。<audio>的全部属性和子元素<source>都是可选的。<audio>元素的属性解释如下：

loop 属性 - 存在此属性表示循环播放。

controls 属性 - 显示音频控件，如播放、暂停按钮等。如果缺少本属性，页面上没有任何显示。

autoplay 属性 - 自动播放。不过现在很多浏览器默认不支持了，需要手工开启浏览器的自动播放功能才行。

muted 属性 – 存在此属性就表示静音。

preload 属性 - 加载音频的方式，可取值为："none"不加载，"meta"加载音频元数据（如文件大小、播放时长等），"auto"由浏览器决定是否加载。如果省略此属性，默认是"auto"。

src 属性 - 指定播放音频的 URL。如 src="欢乐颂.mp3"。

先看一个最简单的例子，在网页文件的<body>元素下增加代码：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
<audio controls src="钢琴曲-命运.mp3">你的浏览器不支持 audio 元素</audio>
```

把“钢琴曲-命运.mp3”音频文件放在网页文件所在的同一个目录中，在 chrome 浏览器里看到的效果如图 4.75。



图 4.75 chrome 的<audio>元素

<audio>元素中的每个子元素<source>指定一个音频的 URL，<source>元素的 type 属性定义音频格式，常见的格式有："audio/mp3"-mp3 音频、"audio/wav"-wav 音频、"audio/ogg"-ogg 音频、"audio/webm" - webm 音频、等等。播放时，从最前面的<source>元素查找浏览器支持的音频，当找到一个有效音频时就播放它，剩下的<source>元素指定的音频就再也不找了。换句话说一个<audio>元素最多播放一个音频，尽管你使用了多个<source>元素。所以强烈建议你把同一个音频转换成多个格式文件，比如 mp3 格式、ogg 格式、wav 格式等，然后通过<source>元素分别指定这些格式文件，这样就能够在尽可能多的设备上和尽可能多的浏览器上播放，因为不同的浏览器能播放的音频格式可能不同。

再看例子：

```
<audio controls loop>
    <source src="123.mp3" type="audio/mp3">
    <source src="123.ogg" type="audio/ogg">
    <source src="123.wav" type="audio/wav">
    你的浏览器不支持 audio 元素
</audio>
```

浏览效果与上面的一样，浏览器会依次寻找可播放的文件，先试 123.mp3，如果行就不再查找剩下的文件了，如果不行再试 123.ogg，如果还不行再试 123.wav，如果都不能播放就报错。

<audio>元素对象（JS 代码中使用）的一些主要事件见表 4.16。

表 4.16 <audio>元素对象的事件

事件	描述
canplay	当音频文件就绪（可以开始播放）时触发（缓冲已足够开始时）。
ended	播放结束时触发（可发送类似“感谢观看”之类的消息）。
error	当在文件加载期间发生错误时触发。
loadeddata	当媒介数据已加载时触发。
loadedmetadata	当元数据（比如分辨率和时长）被加载时触发。
pause	当媒介被用户或程序暂停时触发。

play	当用户或程序启动播放时触发。
playing	当媒介已开始播放时触发。
readystatechange	每当就绪状态改变时触发（就绪状态监测媒介数据的状态）。
timeupdate	当属性 currentTime 的值发生改变时触发（可用于歌词跟踪）。
volumechange	每当音量改变时（包括将音量设置为静音）时触发。

在实际开发网页时，最常用的事件是 oncanplay、onended 和 onerror，比如在 oncanplay 事件中激活播放按钮，从而允许用户点击“播放”，在 onended 事件中加载下一首歌曲等。

<audio>元素对象的一些主要的属性有（在 JS 编程时经常用到这些属性）见表 4.17。

表 4.17 <audio>元素对象的属性

属性	描述
controller	返回表示音频当前媒体控制器的 MediaController 对象。
controls	设置或返回音频是否应该显示控件（比如播放/暂停等）。
crossOrigin	设置或返回音频的 CORS 设置。
currentSrc	返回当前音频的 URL。
currentTime	设置或返回音频中的当前播放位置（以秒计）。
duration	返回音频的长度（以秒计）。
ended	返回音频的播放是否已结束。
error	返回表示音频错误状态的 MediaError 对象。
loop	设置或返回音频是否应在结束时再次播放。
muted	设置或返回是否关闭声音。
paused	设置或返回音频是否暂停。
playbackRate	设置或返回音频播放的速度。1.0 正常，2.0 2 倍速，0.5 半速，-1.0 后退
played	返回表示音频已播放部分的 TimeRanges 对象。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

preload	设置或返回音频的 preload 属性的值。
readyState	返回音频当前的就绪状态。
src	设置或返回音频的 src 属性的值。
volume	设置或返回音频的音量，取值范围为[0, 1]，默认值为 0.5，0 就是静音。

<audio>元素对象的主要方法见表 4.18。

图 4.18 <audio>元素对象的方法

方法	描述
load()	重新加载音频元素。
play()	开始播放音频。
pause()	暂停当前播放的音频。

利用这些属性、事件和方法，采用 JS 语言很容易开发网页版播放器，我们接下来实现如图 4.76 所示的音频播放器。



图 4.76 一个简易音频播放器

不用<audio>元素提供的播放控件，我们采用 JS 编程来控制音乐的播放和暂停，代码参见随书配套资源中的 chapter-4\audio.html 文件。

代码中用一个循环定时器来移动进度条，进度条采用 range 类型的<input>元素模拟，这些内容都在前面的章节中介绍过。刚开始时把<audio>对象的 duration 属性（音频播放总秒数）的值赋给<input>元素的 max 属性，然后每隔一秒就给<input>元素的 value 属性加 1，这样滑块就不断右移，当移到最右边时，音频刚好播放完毕。如果用鼠标拖动滑块，就把<input>元素的 value 属性的值赋给<audio>元素的 currentTime 属性，这样就实现了快进或后退功能。

<audio>元素只能对音频播放做诸如播放、暂停、停止等简单的控制，如要做复杂的操作，比如声道的合并与分割、混响、音调、声相控制和音频振幅压缩等等，就要用到 AudioContext 对象，详见“§6.9 媒体流”章节。

实操练习 1：在 chapter-4\audio.html 的基础上增加音量控制，类似图 4.78 所示。

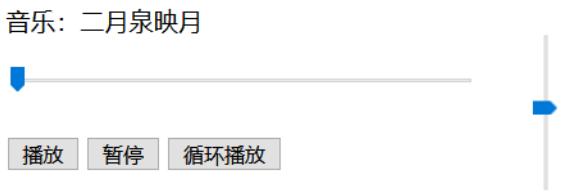


图 4.78 增加音量控制

实操练习 2：把“播放”和“暂停”两个按钮的功能合在一起，类似图 4.79。



图 4.79 合并播放和暂停按钮

4.10.3.2 <video>

<video>元素用于播放视频，其语法如下：

```
<video autoplay controls crossorigin="" loop muted preload="" playsinline=""
poster="" src="" width="" height="" >
    [<source src="" type="">
     <source src="" type="">
     <source src="" type="">
     ……
     你的浏览器不支持 video 元素]
</video>
```

大部分属性与<audio>元素一样，不过<video>元素允许指定视频播放的窗口大小，属性 poster 指定正在加载视频时显示在播放窗口上的一幅图片，一般为视频的介绍之类的。属性 playsinline 启用内联播放，主要应用于移动设备，内联播放就是播放画面局限在<video>元素范围内，而不是全屏。

看下面的例子：

```
<video src="tobymanley.ogg" controls>浏览器不支持 video 元素</video>
```

浏览效果见图 4.80。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 4.80 <video>默认效果图

默认的视频控件功能比较齐全，可播放、暂停、快进和回退、控制音量、全屏播放等。再来看一个例子:

```
<video width="320" height="240" controls>
    <source src="movie.mp4" type="video/mp4">
    <source src="movie.ogg" type="video/ogg">
    <h1>您的浏览器不支持 video 元素。</h1>
</video>
```

指定了播放窗口的大小为 320*240，同时指定了两种视频格式文件，如果浏览器能支持 mp4 格式，那么就播放 movie.mp4 文件，否则如果支持 ogg 格式，就播放 movie.ogg 文件，如果都不支持，就报错。

<video>元素的常用事件和方法都与<audio>元素对象一样,<video>元素对象的属性除了 height、width、poster 外其他的与<audio>元素一样，具体参考本书前面的<audio>元素价绍。

chrome 浏览器从 70 版开始支持<video>元素的“画中画”功能，这里的“画中画”不是传统意义上的视频中套视频的画中画，只是播放画面能游离与页面之外，“漂浮”在屏幕上方，这样我们可以一边看视频一般工作。类似图 4.81。

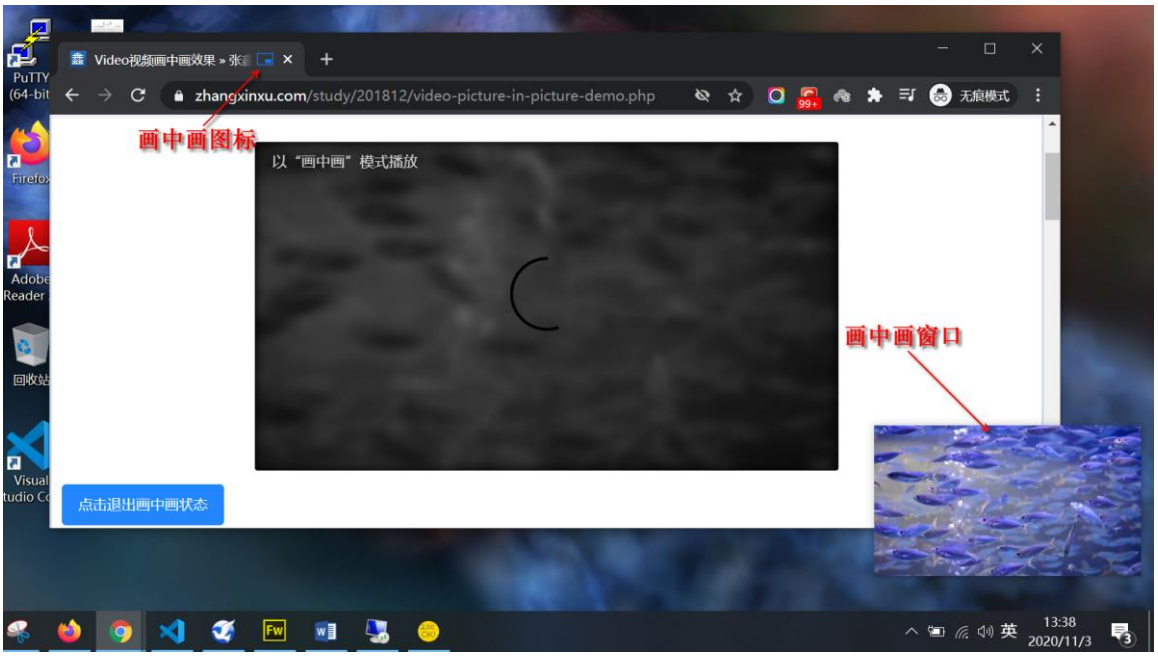


图 4.81 画中画

实现的一般方法是：增加一个按钮“进入画中画”，用户点击按钮后调用 videoElem.requestPictureInPicture()方法，同时按钮变成“退出画中画”，当用户点击“退出画中画”按钮时调用 document.exitPictureInPicture() 方法。另外<video>元素对象增加了两个事件 enterpictureinpicture（进入画中画时触发）和 leavepictureinpicture（退出画中画时触发），在 enterpictureinpicture 事件的处理函数的实参 event 中增加了画中画窗口对象 pictureInPictureWindow，此对象的两个只读属性 width 和 height 保存了画中画的宽度和高度，resize 事件改变窗口大小时触发。看例子：

```
<body>
    <video id="video" controls playsinline loop src="movie.mp4">
        <h1>您的浏览器不支持 video 元素。</h1>
    </video><br>
    <input id="togglePipButton" type="button" value="进入画中画">
    <script>
        const videoElem = document.getElementById("video");
        const togglePipBtn = document.getElementById("togglePipButton");

        togglePipBtn.hidden = !document.pictureInPictureEnabled || videoElem.di
sablePictureInPicture;

        togglePipBtn.addEventListener("click", async() => {
            try {
                if (!document.pictureInPictureElement) {
                    await videoElem.requestPictureInPicture();
                } else {
                    await document.exitPictureInPicture();
                }
            } catch (err) {
                throw err;
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
            }
        });

        function enterPinPHandler(event) {
            var pinpWindow = event.pictureInPictureWindow;
            pinpWindow.addEventListener("resize", () => console.log("改变了尺寸
"));
            console.log("      浮      动      窗      口      大      小      :
", pinpWindow.width, "*", pinpWindow.height);
        }

        videoElem.addEventListener("enterpictureinpicture", enterPinPHandler);
        videoElem.addEventListener("leavepictureinpicture", () => console.log("
退出画中画"));
    </script>
</body>
```

“=>”为箭头函数，() => console.log("改变了尺寸")等价于 function(){console.log("改变了尺寸");}，关于 JavaScript 语言的箭头函数请从网上或其他书籍了解。

实操练习 1：

参照前面的<audio>例子，设计一个视频播放器，效果图如图 4.82。



图 4.82 简易视频播放器

实操练习 2：

在实操练习 1 的基础上把“播放”和“暂停”按钮合并，如图 4.83 所示。



图 4.83 合并播放和暂停按钮

4.10.3.3 <track>

<track>元素允许您将字幕、解说、屏幕阅读器说明和章节添加到视频和音频中，实际上是添加了文本轨道，一个视频的格式见图 4.84 所示。

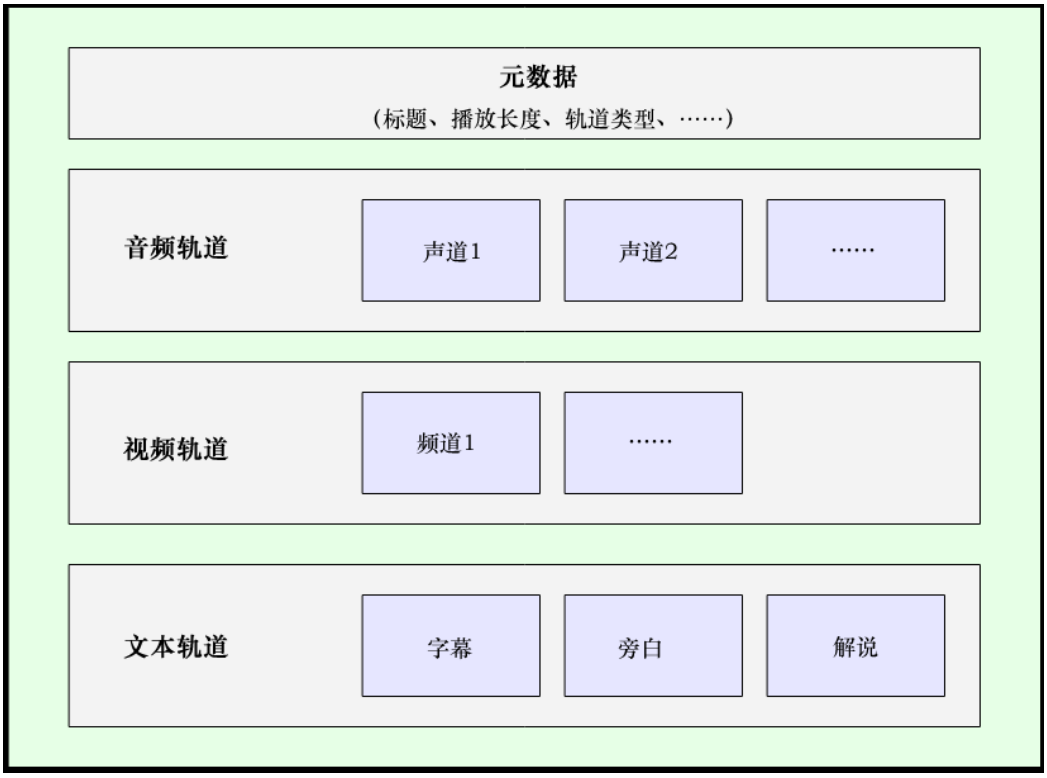


图 4.84 视频由轨道组成

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

<track>元素只能作为<audio>或<video>元素的子元素，且只能排在<source>元素的后面。在同一个<audio>或<video>元素中，<track>允许出现多次，用于指定不同语言的文本轨道，让用户选择。此元素的语法如下：

```
<track [ label="" kind="" srclang="" default ] src="">
```

label 属性 - 定义文本轨道的名字，让用户通过名字来选择不同的文本轨道，如图 4.85 所示。



图 4.85 选择不同的文本轨道

如果出现多个<track>元素，最多只允许一个<track>中出现 default（默认文本轨道），如果都没有 default，就不显示文本轨道内容。

kind 属性 - 定义文本轨道的类型，它的值可为：

subtitles：字幕，把人声转录或翻译成不同语言的文字并显示在最前面。这是默认值。

captions：解说，主要针对耳听困难的场合（如耳聋、媒体静音播放或周围噪声太大），这是与 subtitles 的主要区别所在。

descriptions：一个视频或剧中某个情景的描述性文本，通过屏幕阅读器再次读出这些文本，以帮助观看困难的人(比如失明、正在驾驶、或屏幕分辨率太低看不清)。

chapter：章节标题，旨在允许用户浏览媒体资源以便快速定位播放。播放器会提供一个章节列表供用户选择。

srclang 属性 - 定义文本轨道上的语言，不同的语言有不同的代码，比如常见语言的代码有：en（英语）、zh（简体和繁体中文）、fr（法语）、ja（日语）、ru（俄语）、es（西班牙语）、ar（阿拉伯语）等。具体可搜索“ISO 语言代码”。注意：当 kind="subtitles"时，srclang 属性不能省略。

下面我们来看一个例子：

```
<video controls>
    <source src="movie.mp4" type="video/mp4">
    <track label="中文" src="movie1.vtt"  kind="subtitles" srclang="zh" default>
    <track label="English" src="movie2.vtt"  kind="subtitles" srclang="en">
</video>
```

在这个例子中，提供了英文和中文两种字幕，供用户自行选择，默认是中文。把字幕文件 movie1.vtt、movie2.vtt 和视频文件 movie.mp4 都放在网页文件所在的那个目录中（是的，你现在还没有字幕文件，请参照后面的内容自己制作），然后在浏览器里输入网址 http://localhost 或者类似 http://www.abc.com 这样的域名访问（参考第 1 章中“开发测试“小节，搭建一个通过域名访问的本地 web 服务器），会看到如图 4.86 所示的效果。

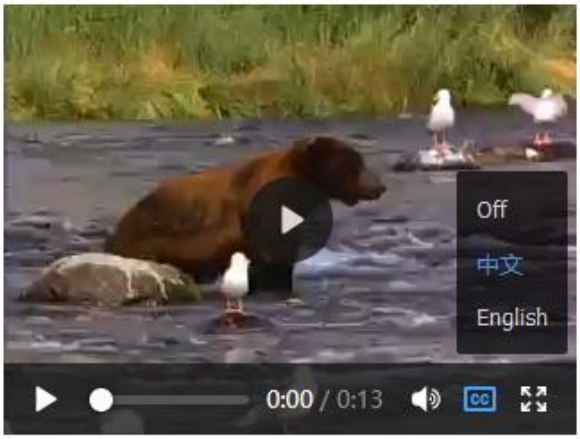


图 4.86 两种语言的字幕

点击CC图标弹出字幕语言选择框，可以看到默认是中文。点击播放后，就会在视频的下方显示字幕，参考图 4.87。



图 4.87 选择中文字幕

使用<track>元素时要注意以下三点：

1. 只能通过 http://或者 https://协议访问，如果通过 file://协议，则<track>元素无效。

2. nginx 或 apache 的配置文件 mime.types 中添加一行（一般都已经有了）：

```
    text/vtt                                vtt;
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

3. 字幕文件以.vtt 为扩展名，且是 utf-8 编码保存，第一行必须是 WEBVTT，然后是以空行开头的第 1 节（字幕文件中可以包含任意多个节）、以空行开头的第 2 节、……。描述内容可以出现在任何地方，并且次数不限。

下面我们再来看看如何制作一个字幕文件。一个字幕文件的完整格式如图 4.88 所示。

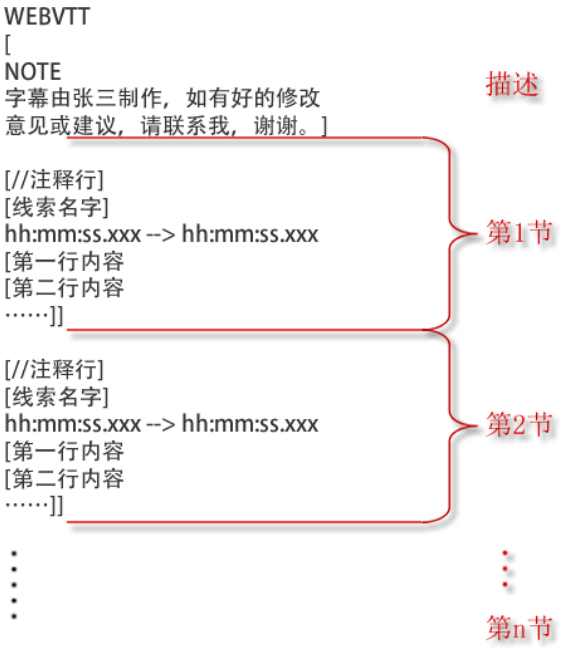


图 4.88 字幕文件格式

由中括号括起来的部分是可选的。时间格式 hh:mm:ss.xxx 表示小时、分钟、秒和微秒，前三者都要用两位数表示，微秒用三位数表示。每一节中“-->”左右两侧的时间点表示播放到这个时间段就显示本节中的内容，一节中的内容可以包含多行。我们来看一个具体的字幕文件例子：

```
WEBVTT

//张三的台词
Zhangsan
00:00:01.000 --> 00:00:05.000
I don’t think so. You?

//李四的台词
Lisi
00:00:07.167 --> 00:00:12.042
I’m Ok!
Where are we going?
```

此例子告诉浏览器，当视频播放到 00:00:01.000 至 00:00:05.000 时段时，显示字幕“I don’t think so. You?”，当播放到 00:00:07.167 至 00:00:12.042 时又显示另外两句话。把上面的内容输入到记事本中，然后以 utf-8 保存为 move.vtt 文件，具体操作参考图 4.89。

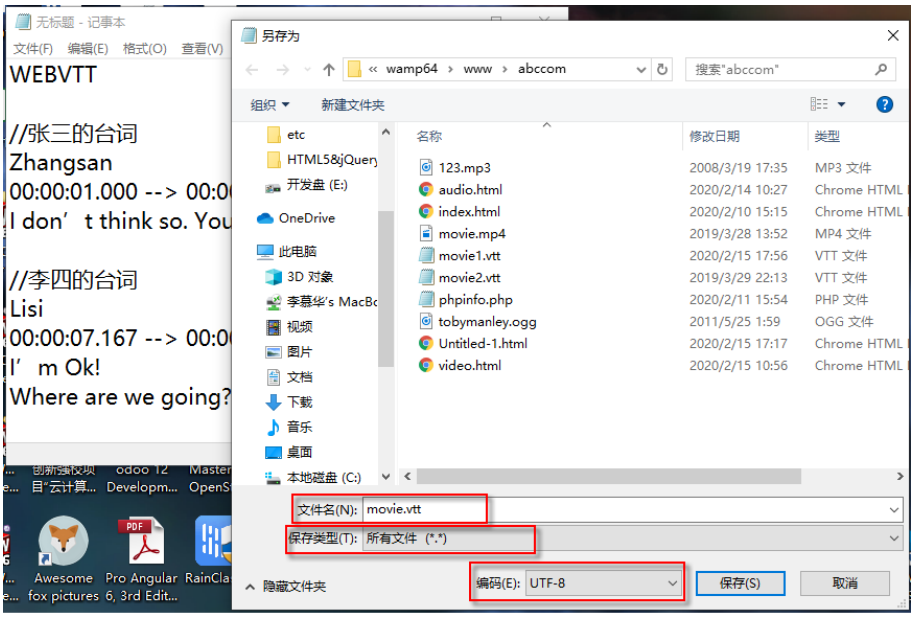


图 4.89 用记事本编辑字幕文件

我们还可以为字幕指定 css 样式，这些样式允许放在三个地方的任何一处：

1. 网页文件的样式表中。采用::cue 伪类，比如：

```
<style>
    video::cue{
        background-color: transparent;
    }
    …
</style>
```

2. 字幕文件的样式表中。采用 STYLE 引导，比如：

```
WEBVTT

STYLE
::cue {
  background-color: rgba(100, 100, 100, 0.4);
  color: white;
}

STYLE
::cue(b) {
  color: red;
}

00:00:00.000 --> 00:00:10.000
Hello <b>world</b>.
```

3. 字幕的每一节中。直接写在时间段的后面，但是样式名不同于 css 的样式名，详细的介绍参考 https://developer.mozilla.org/en-US/docs/Web/API/WebVTT_API。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

最后我们看一个完整的例子，把上面制作的字幕文件 movie.vtt 加到一部英语电影中：

```
<!DOCTYPE html>
<html>

<head>
    <title>中文字幕</title>
    <style>
        video::cue{
            background-color: rgba(100, 100, 100, 0.5);
            color: white;
        }
    </style>
</head>

<body>
    <video src="tobymanley.ogg" controls>
        <track label="中文字幕" src="movie.vtt" kind="subtitles" srclang="zh"
default>
    </video>
</body>

</html>
```

白色的字幕文字显示在半透明的灰色背景上，把字幕文件 movie.vtt 和视频文件 tobymanley.ogg 都放在网页文件所在的目录中，然后通过 http 协议浏览网页，就可以看到字幕效果了。

4.10.4 canvas 画布元素

要说 html5 最大的亮点是什么？那非<canvas>元素莫属，在此元素出现之前，你想在网页上画画、做动画、搞 3D，是非常困难的。现在好了，只要一个<canvas>元素，你就能绘制任何东西出来，只要你有这个“绘画”能力。先让你看看民间高手的作品——图 4.90、图 4.91、图 4.92、图 4.93、图 4.94 就是采用<canvas>元素创作出来的。



图 4.90 动画——慢慢长高的花朵

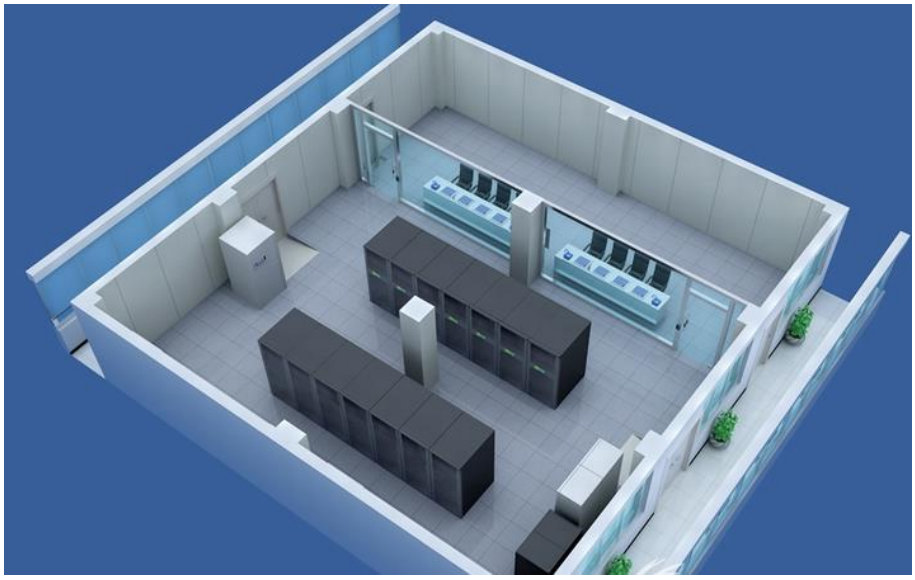


图 4.91 3D 机房效果图（可以旋转哦）

上图摘自网站 http://demo.servasoft.com/room/

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

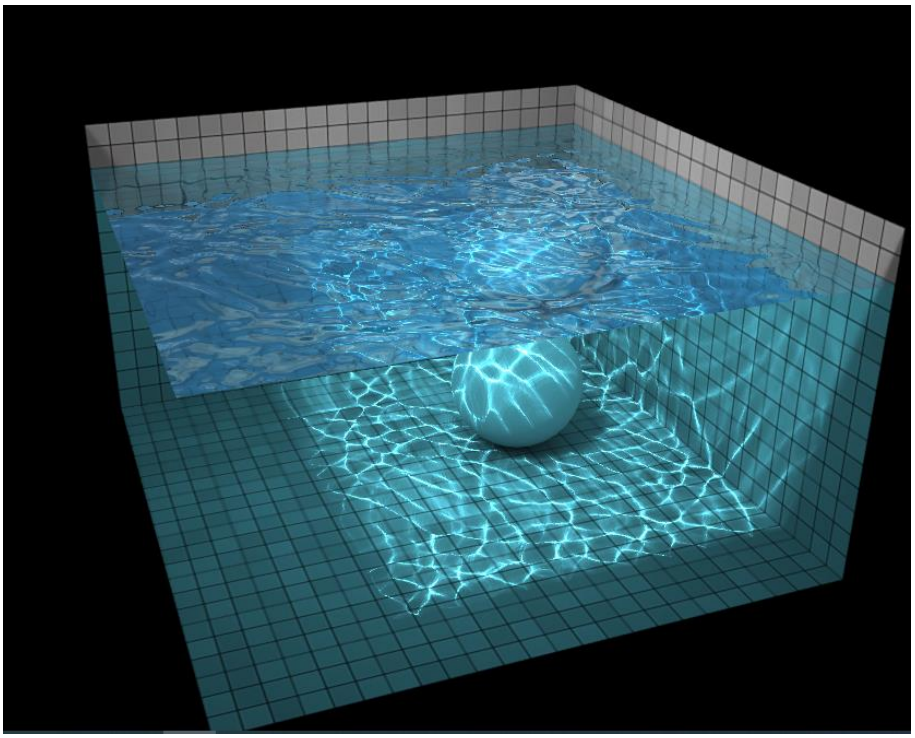


图 4.92 水池里的球



图 4.93 这也能做出来？

还有这个网页版 photoshop： https://www.photopea.com/

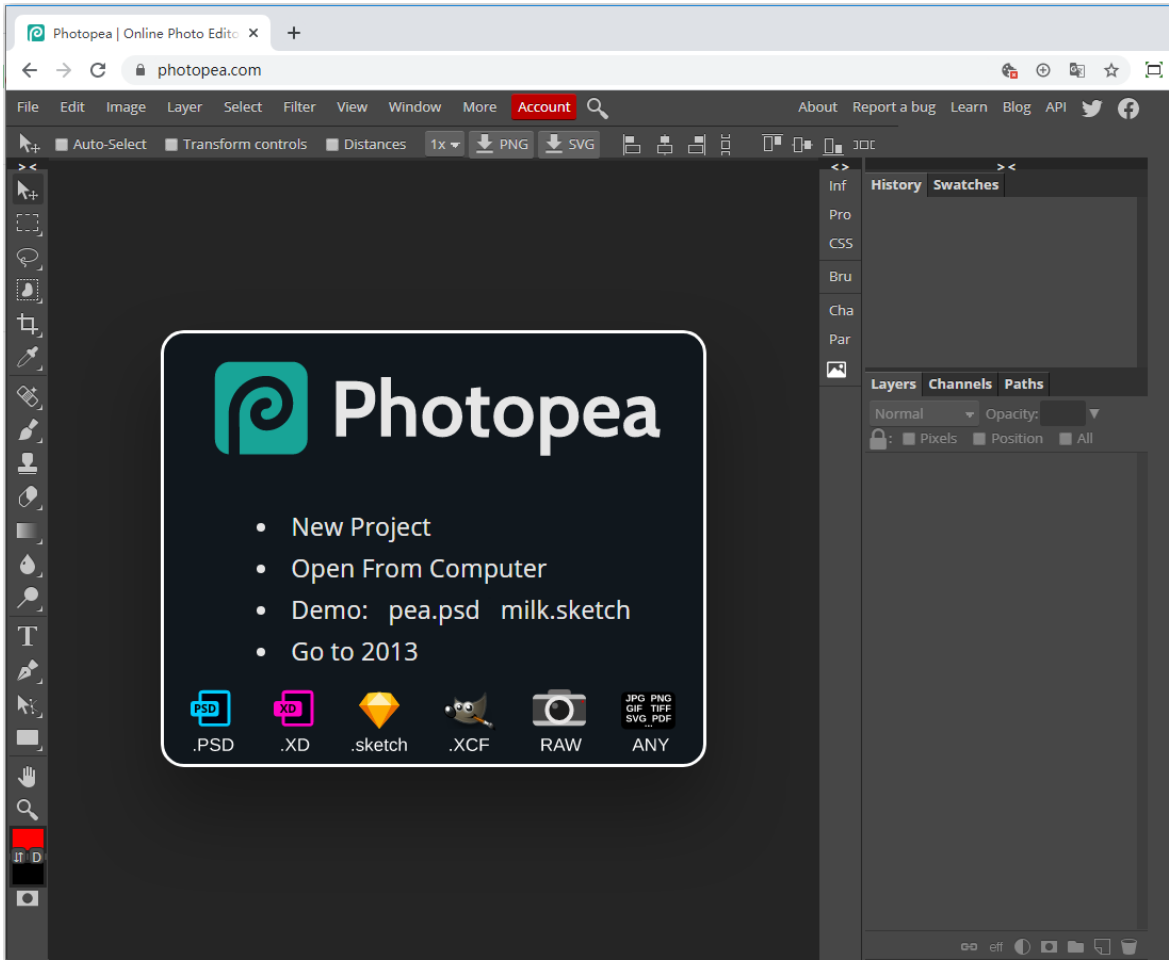


图 4.94 网页版的图形处理工具

<canvas>元素原生支持 WebGL 图形库(JavaScript API)，比如上面的那个 3D 机房就是直接采用 webgl 库搞出来的，也存在一些优秀的第三方 3D 插件库，这些库进一步封装 webgl 库，从而易用性得到增强，比如 three.JS、Babylon.JS（最好的 JS 3D 游戏引擎，它能创建专业级三维游戏）、layCanvas(一个基于 WebGL 游戏引擎的企业级开源 JavaScript 框架)。不过由于苹果不支持 WebGL，这限制了 WebGL 的普及，此后提出的 WebGPU 被几乎全部浏览器支持，所以可以肯定 WebGPU 会淘汰掉 WebGL，如果大家想在 Web 图形方面深入，建议学习 WebGPU。

4.10.4.1 canvas 概览

在 HTML5 网页文件中，<canvas>元素的使用语法如下：

```
<canvas[ id="myCan" width= height= ]>你的浏览器不支持 canvas 元素</canvas>
```

其中属性 width、height 定义画布的尺寸，它们是可选的。如果浏览器不支持 canvas 元素，就会在页面上显示“你的浏览器不支持 canvas 元素”。使用<canvas>元素的基本网页框架如下：

```
<!DOCTYPE html>
<html>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
<head>
    <title>CANVAS 元素</title>
    <style>
        #myCan {
            border: 1px solid grey;
        }
    </style>
</head>

<body>
    <canvas id="myCan" width="800px" height="600px">不支持 canvas</canvas>
    <script>
        var c = document.getElementById("myCan");
        var ctx = c.getContext("2d");

        [定义绘画上下文]      <-- 类似画家选择不同的笔
        执行绘画动作          <-- 允许出现很多的绘画动作
        显示在画布上

        [定义绘画上下文]      <-- 选择不同的笔
        执行绘画动作
        显示在画布上

        ……
    </script>
</body>

</html>
```

<canvas>元素本身没有绘画能力，必须借助于 JavaScript 语句才行。

你想想一下一个画家绘画的过程：首先他要一张画布、几支笔和一个调色板，然后就不断地更换笔、调色和绘制。canvas 元素就相当于画家的画布，上面代码中的 ctx 变量中包含了笔和调色板（计算机术语叫做绘画上下文），其实 ctx 中还包含了绘制动作（各种绘制函数），如绘制直线函数 ctx.lineTo()、画圆函数 ctx.circle()、画矩形函数 ctx.rect()等等。但是采用 canvas 绘画与画家绘画不同的地方是：canvas 画布不但支持 2D 平面绘画，还支持 3D 立体绘画，另外“执行绘画动作”本身不会显示在画布上，一定要明确执行“显示在画布上”的语句才会显示出来，正如画家绘制动作其实是把脑袋里早已创作好的画画在画布上而已。下面的例子是画一个半径为 150px 的红色圆，先让大家有个感性认识。

```
<!DOCTYPE html>
<html>

<head>
    <title>CANVAS 元素</title>
    <style>
        #myCan {
            border: 1px solid grey;
        }
    </style>
</head>

<body>
    <canvas id="myCan" width="800px" height="600px">不支持 canvas</canvas>
```

```
    <script>
        var c = document.getElementById("myCan");
        var ctx = c.getContext("2d");

        ctx.strokeStyle = "#FF0000"; /*调为红色*/
        ctx.arc(200,200,150,0,Math.PI*2,true); /*创作一个圆*/
        ctx.stroke(); /*把创作的圆绘制在画布上*/
    </script>
</body>

</html>
```

浏览的效果如图 4.95。

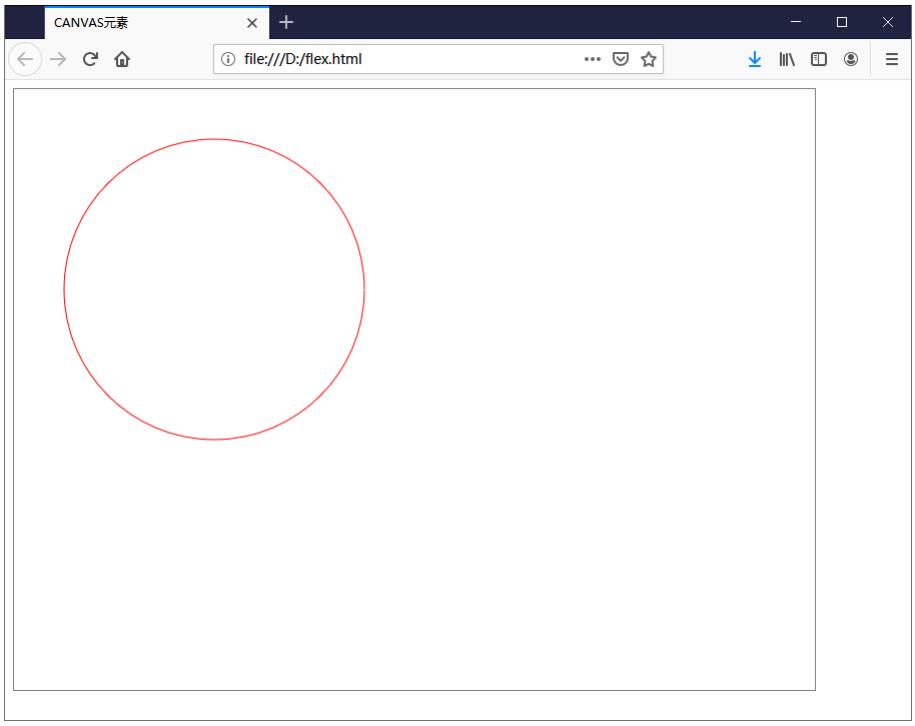


图 4.95 画一个圆——小试牛刀

本例子只是让你感受一下在 canvas 上绘图的基本网页框架，至于具体的各种绘图函数，后面有详细的介绍。下面对画布、绘画上下文（调色板和笔）以及绘画动作逐一讲解。

4.10.4.2 画布坐标

画布的坐标如图 4.96 所示。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

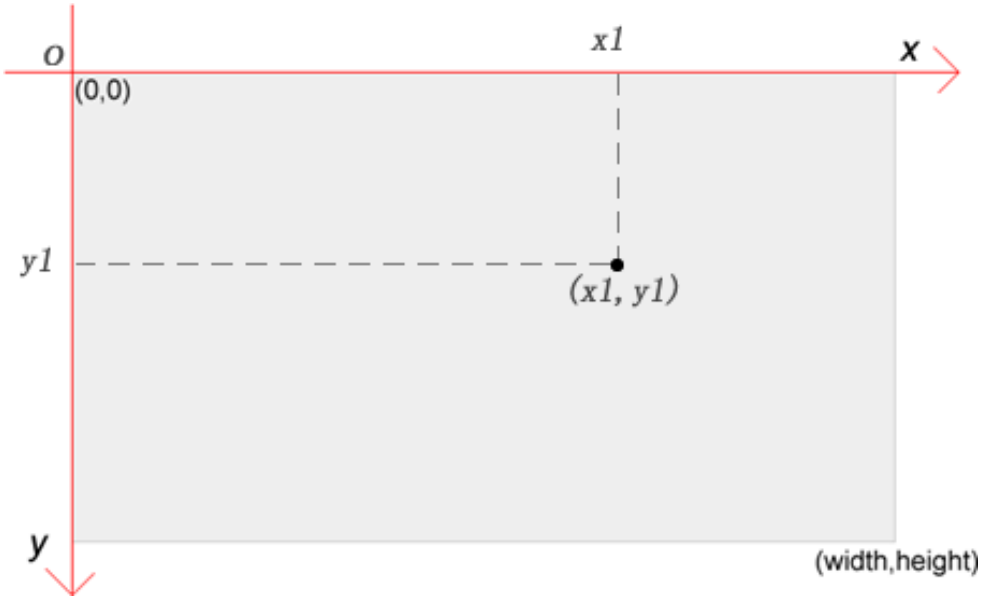


图 4.96 canvas 画布坐标系

画布的默认坐标原点在左上角，即(0,0)，右下角为(width,height)，width 和 height 是 canvas 元素的属性。当然坐标原点的位置可以自定义，具体参见后续章节“变换”。记住这个画布坐标很重要，很多绘画动作都要用到坐标。

4.10.4.3 绘画上下文

绘画上下文就相当于画家的调色板和各种笔，参考图 4.97，用语句“var ctx = document.getElementById("myCan").getContext("2d");”可获取绘画上下文对象（对象类型为 CanvasRenderingContext2D），并保存在 ctx 变量中。如果要绘制 3D，则用参数 webgl 或 webgl2 去调用 getContext()方法，webgl2 是第二版本，比如 getContext("webgl2")，关于在<canvas>元素上用 WebGL 作画，请参考 https://developer.mozilla.org/zh-CN/docs/Web/API/WebGL_API/Tutorial。本教材只讲解 2D 绘画。

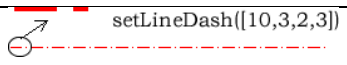


图 4.97 画家的调色板

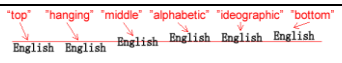
绘画上下文主要是设置线型、描边颜色、填充颜色、文本样式等，绘画上下文存在默认值，所以即使你不设置，也能绘制图形，而且每次也不要求设置全部的上下文参数，比如接下来线条颜色由黑色改为红色，那么只要重新设置描边颜色就可以了。

绘画上下文由一些属性和方法来设置，可以获取属性的当前值，也可以直接给属性赋新值，这些属性参见表 4.18。

表 4.18 绘画上下文

属性/方法	属性值/方法参数	说明
lineWidth	正实数，默认值为 1.0	线条宽度，如 ctx.lineWidth=2.0
lineCap	 "butt","round","square"	线条末端形状。默认为"butt"。square 增加的高度就是线宽的一半。
lineJoin	 'round'，'bevel'，'miter'	线条连接点形状。默认为'miter'。
miterLimit	 正数。默认值为 10.0	当 lineJoin='miter'时，限制斜接长度。当斜接长度大于 miterLimit 时，连接点变为'bevel'类型。
setLineDash(array)	 setLineDash([ ])定义实线	设置线型，如实线、虚线等。参数为偶数个元素的数组，元素依次指明实线、空白、实线、空白、……的长度。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

lineDashOffset	 实数，默认为 0。	设置虚线的偏移量。正值表示线条右移动，否则左移动，参看下面的蚂蚁线例子。
font	字符串	设置字体样式。与 css 的 font 属性语法相同。如 ctx.font = "25px serif";
drection	"ltr"(左往右),"rtl"(右往左)	设置文本书写方向。
textAlign	"left","right","center","start","end"	设置文本对齐方式。其中 start 与书写起点对齐，end 与书写终点对齐，具体与 drection 属性有关。。
textBaseline		设置垂直方向的对齐方式。默认值为 alphabetic
strokeStyle	<color>(颜色),<gradient>(渐变),<pattern>(图片)。如 ctx.strokeStyle="#FF0000"	设置描边样式：颜色、渐变或者图片。<color>与 css 相同,<gradient>参考后续的渐变介绍,<pattern>参考 canvas 对象的 createPattern()方法。
fillStyle	<color>(颜色),<gradient>(渐变),<pattern>(图片)。	设置填充样式，默认是黑色。用法与 strokeStyle 相同。
shadowColor	<color>	设置阴影颜色，默认为透明色。
globalAlpha	[0,1]内的实数	设置全局透明度，0 全投明，1 不透明
imageSmoothingEnabled	true 或 false，默认为 true	对缩放是否作平滑处理。
globalCompositeOperation	字符串类型，共有 12 种合成方式，默认是"source-over"	叠加绘制时的合成方式。详细结束见后续章节

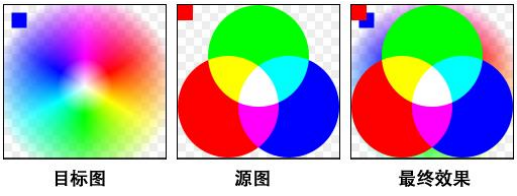
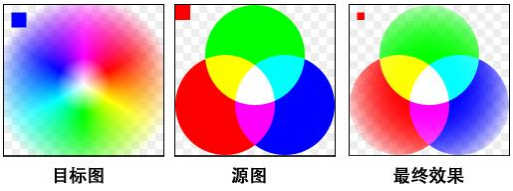
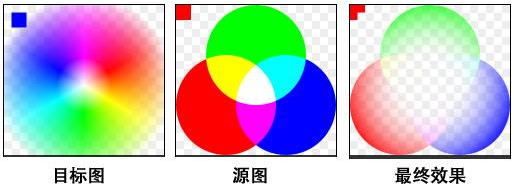
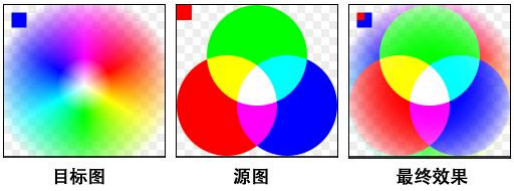
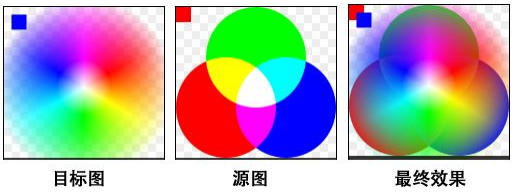
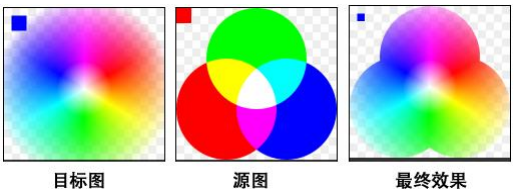
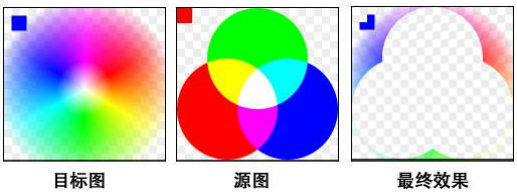
对于 textBaseline 的理解，请参考图 4.98。



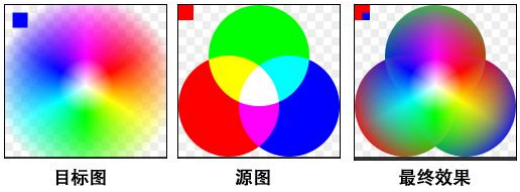
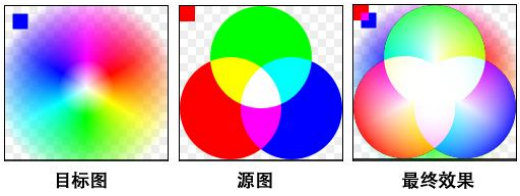
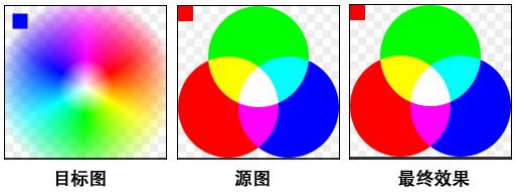
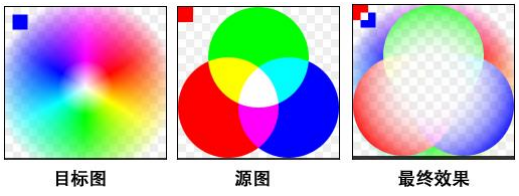
图 4.98 textBaseline 属性说明

对于 globalCompositeOperation 属性，一共有几十种值，表 4.19 解释一些主要的值（于目标图上再绘制原图的效果）。

表 4.19 合成绘画

值	说明	例子
source-over	这是默认值，直接把原图绘制再目标图上，原图会覆盖部分目标图。	
source-in	只绘制制源图与目标图重叠的部分，目标图被删除。	
source-out	只绘制源图与目标图不重叠的部分，目标图被删除。	
source-atop	只绘制源图与目标图重叠的部分，目标图被保留。	
destination-over	源图绘制在目标图下面。	
destination-in	只绘制目标图与源图重叠的部分，源图被删除。	
destination-out	只绘制目标图与源图不重叠的部分，源图被删除。	

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

destination-atop	把目标图与源图重叠的部分绘制在上面，源图绘制在下面。	
lighter	重叠部分的颜色值相加，最大不超过#FFFFFF（白色）	
copy	原图覆盖目标图	
xor	重叠部分为透明	
其他的值还有：multiply、screen、overlay、darken、lighten、color-dodge、color-burn、hard-light、soft-light、difference、exclusion、hue、saturation、color、luminosity，具体参考其他资料。		

看一个设置上下文的例子：

```
<script>
    var canvas = document.getElementById("myCan");
    var ctx = canvas.getContext("2d");

    ctx.lineWidth = 1.5;
    ctx.lineCap = "round";
    ctx.setLineDash([8,3]);
    ctx.font = "30px arial";
    ctx.strokeStyle = 'rgba(100,0,150, 0.4)';
</script>
```

实操练习：思考上面这个例子设计了什么样的虚线，用笔画画看。

再看一个合成图像的例子：

```
    <script>
        var canvas = document.getElementById("myCan");
        var ctx = canvas.getContext("2d");

     //设置合成绘画上下文
```

```
        ctx.globalCompositeOperation = "lighter";

    //绘制红色圆
        ctx.fillStyle = "rgba(255,0,0,1)";
        ctx.arc(100, 200, 100, Math.PI * 2, 0, false);
        ctx.fill()

    //绘制蓝色圆
        ctx.beginPath();
        ctx.fillStyle = "rgba(0,0,255,1)";
        ctx.arc(220, 200, 100, Math.PI * 2, 0, false);
        ctx.fill()

    //绘制绿色圆
        ctx.beginPath();
        ctx.fillStyle = "rgba(0,255,0,1)";
        ctx.arc(160, 100, 100, Math.PI * 2, 0, false);
        ctx.fill();
    </script>
```

浏览的效果如图 4.99。

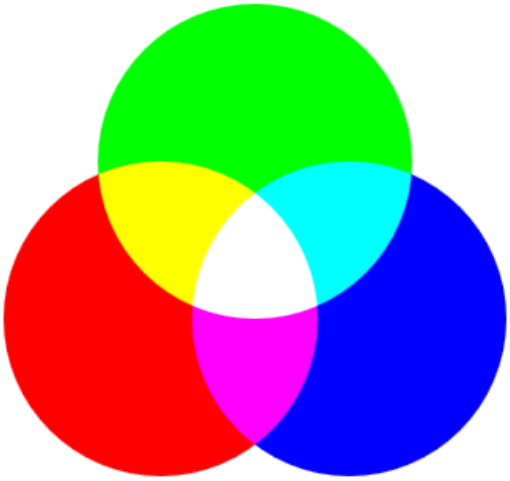


图 4.99 合成绘画

使用下面的语句可以打印全部的上下文参数和各自的默认值：

```
    <script>
        var canvas = document.getElementById("myCan");
        var ctx = canvas.getContext("2d");
        console.log(ctx);
    </script>
```

进入浏览器的开发者模式（按 F12 或者 Ctrl+Shift+i），见图 4.100 所示。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

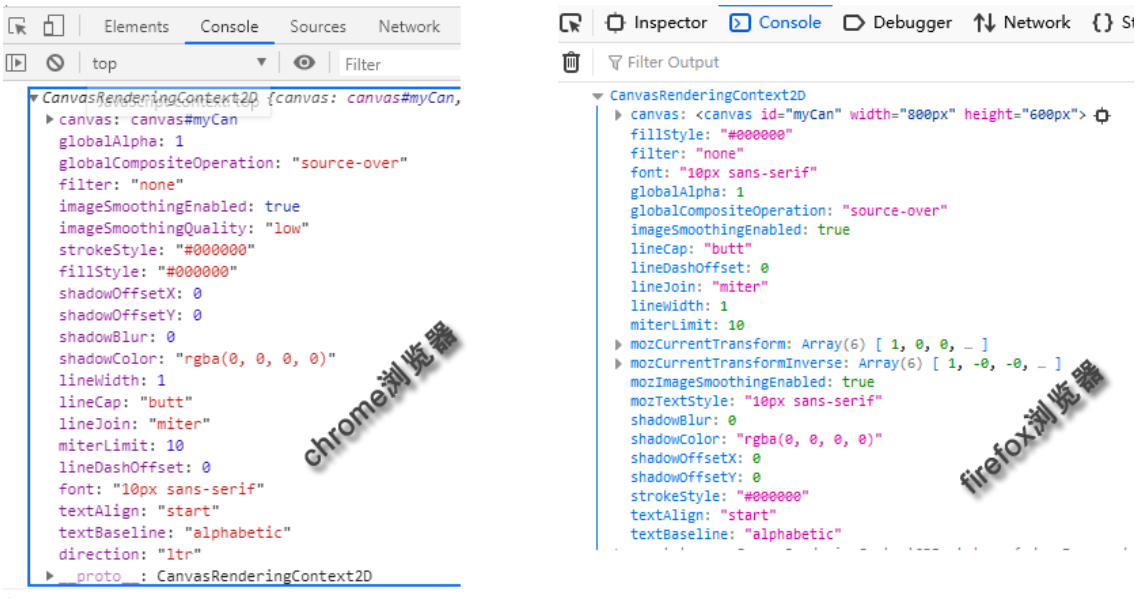


图 4.100 默认的绘画上下文

我们在绘画过程中会经常要修改上下文参数，就像画家那样经常更换笔和颜色，画家好不容易调好的色如果后面还要继续用，他就不会轻易破坏这种颜色，而是放在一边等下次再用。在用<canvas>元素绘画时，也可以采用方法 save()把绘画上下文保存到栈顶，另一个方法 restore()弹出栈顶的绘画上下文并设置成当前绘画上下文。绘画上下文存储栈就像把碗叠在一起，每一次放碗都放在最上面，每一次取碗也是从最上面开始，参考图 4.101。

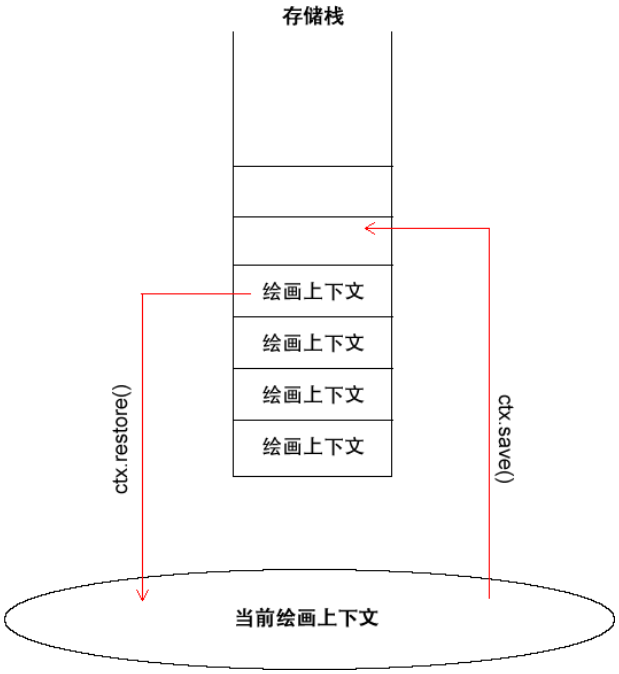


图 4.101 绘画上下文的存储栈

请看一个例子：

```
<script>
      var can = document.getElementById("myCan");
      var ctx = can.getContext("2d");
```

```
        ctx.save();
        ctx.lineWidth = 10;
        ctx.strokeRect(10, 10, 200, 100);

        ctx.restore();
        ctx.strokeRect(10, 120, 200, 100);
    </script>
```

浏览的效果如图 4.102。

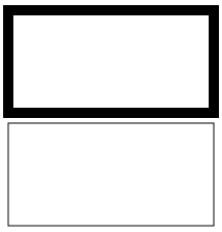


图 4.102 利用保存/回复上下文绘制

默认地，线宽属性 lineWidth 等于 1。例子中，我们在修改此属性前先保存了绘画上下文（其中包含了线宽为 1 的 lineWidth 属性），然后修改线宽为 10 并画了一个粗线条矩形，接下来恢复绘画上下文，这时线宽又为 1 了，因此画的第二个矩形的线宽就是 1。可以多次保存绘画上下文，当然如果自上次保存以来没有修改过绘画上下文，就没别要再保存了。也可以多次恢复绘画上下文，直到最初保存的那个绘画上下文为止。

4.10.4.4 路径

在 canvas 上绘画分为两个阶段：首先创建路径，然后绘制路径。其实这跟画家画画一个道理：画家首先要在脑袋里创作，然后再绘制出来。创建路径只是在内存中定义好一条路径，并没有显示出来，绘制路径（即采用绘画上下文描边）后才会显示出来，绘制路径采用方法 stroke()。一条路径由若干条子路径组成，而且路径允许不连续。参见图 4.103。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

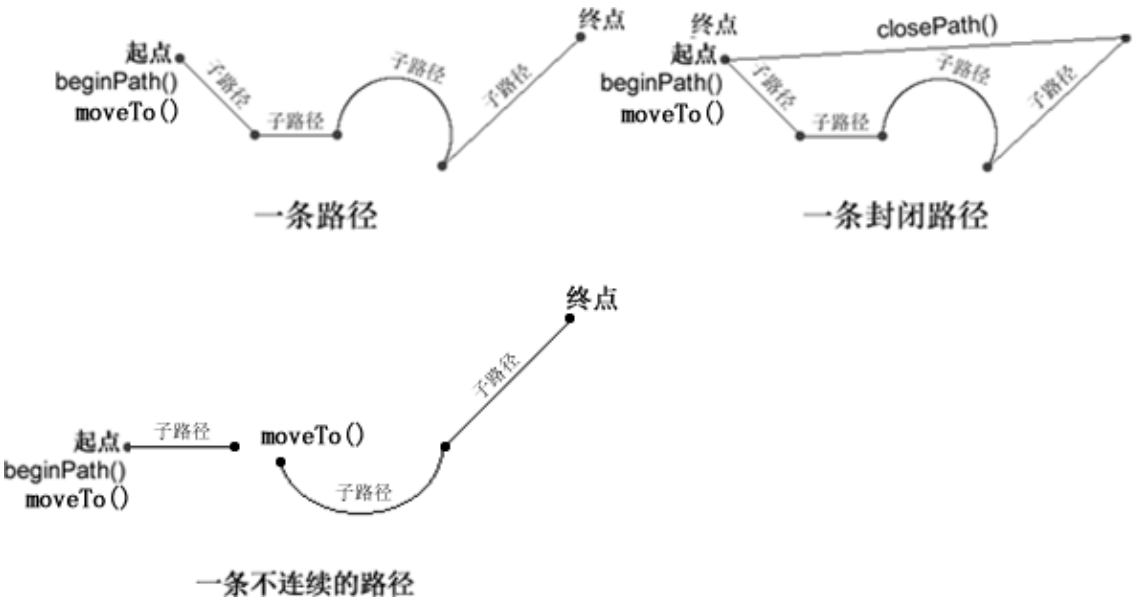


图 4.103 canvas 路径

调用方法 beginPath()开始创建一条新路径，在画布上创建第一条路径时可以省略 beginPath()，此后如果要创建更多的路径，那么创建每条路径前必须先调用 beginPath()方法。调用 closePath()方法创建一条闭合子路径（从终点到起点画一根线），本方法不是关闭一条路径，所以闭合之后还可以创建更多的子路径。创建路径就是定义每一条子路径的起/止端点和形状，绘制路径才涉及线宽、颜色、线型等上下文参数。请看图 4.104。

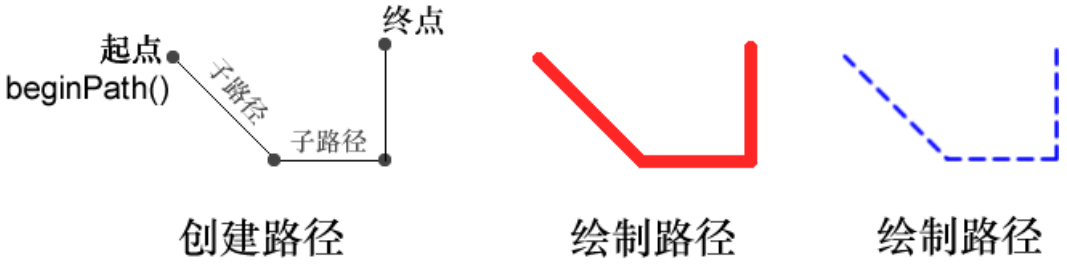


图 4.104 创建和绘制路径

对于创建的同一条路径，通过修改绘画上下文可以绘制出不同的外观，见图 4.104 所示。

接下来介绍各种创建路径的方法。

直线路径

使用 lineTo(x,y)函数从当前坐标创建一条直线到（x,y）坐标，对于一条新路径来说，必须采用函数 moveTo(x,y)明确定义当前坐标，此后每条子路径的终点坐标就是下一条子路径的当前坐标，不过此后创建子路径前也可以使用 moveTo()函数改变当前坐标，因此可能存在不连续的路径，见图 4.105 所示。

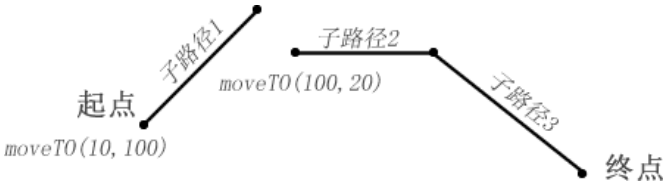


图 4.105 一条不连续路径

看一个具体的例子，参考图 4.106 中的代码。

```
<!DOCTYPE html>
<html>

<head>
  <meta charset=" utf-8" >
  <title>画线</title>
  <style>
    #myCan {
      border: 1px solid grey;
    }
  </style>
</head>

<body>
  <canvas id=" myCan" width=" 800px" height=" 600px" >不支持canvas</canvas>
  <script>
    var canvas = document.getElementById(" myCan" );
    var ctx = canvas.getContext(" 2d" );

    ctx.moveTo(10, 10);
    ctx.lineTo(100, 200);
    ctx.stroke();
  </script>
</body>

</html>
```



图 4.106 创建一条直线路径

这个例子的路径中只有一条子路径，起/止坐标为(10,10)、(100,200)。下面的例子创建包含两条子路径的折线（在上面例子中的两条创建路径语句下面再增加一条）：

```
ctx.moveTo(10, 10);
ctx.lineTo(100, 200);
ctx.lineTo(300, 200);
ctx.stroke();
```

浏览的效果见图 4.107。

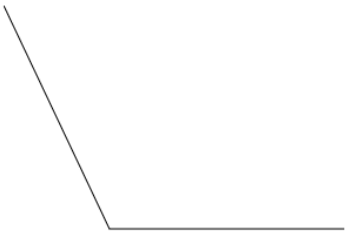


图 4.107 折线路径

第一条子路径的两个端点坐标为(10,10)、(100,200)，第二条子路径的两个端点坐标为(100,200)、(300,200)，显然第一条子路径的终点是第二条子路径的起点，第一条子路径的起点就是整条路径

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

的起点，第二条子路径的终点就是整条路径的终点。如果继续增加如下所示的一条语句，则创建一条闭合路径：

```
ctx.moveTo(10, 10);
ctx.lineTo(100, 200);
ctx.lineTo(300,200);
ctx.closePath();
ctx.stroke();
```

浏览效果如图 4.108。

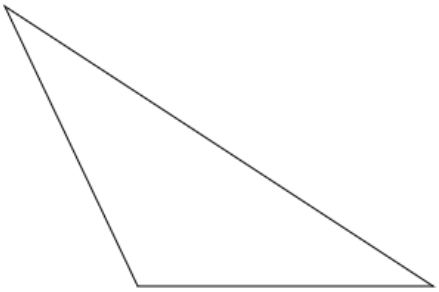


图 4.108 一条闭合路径

此时整条路径包含三条子路径。特别注意：一条闭合路径的起/止点重合在一起了。再增加一条语句，大家思考一下最终显示的效果是什么？

```
ctx.moveTo(10, 10);
ctx.lineTo(100, 200);
ctx.lineTo(300,200);
ctx.closePath();
ctx.lineTo(300,10);
ctx.stroke();
```

到现在为止，我们都是采用默认的绘画上下文，下面来画一条水平线，要求红色，线宽为 6，看代码：

```
    <script>
        var canvas = document.getElementById("myCan");
        var ctx = canvas.getContext("2d");

        ctx.moveTo(20, 50);
        ctx.lineTo(300,50);
        ctx.lineWidth = 6;
        ctx.strokeStyle = "red";
        ctx.stroke();
    </script>
```

浏览效果为图 4.109。

图 4.109 一条红色粗线

注意，为了节约篇幅，以后的例子中就省略上图中灰色部分的代码，读者们实操时要补上这些代码。用 moveTo()方法可以改变子路径的起点，下面的代码就是画两条垂直平行线（但还是一条路径，只不过此路径不连续）：

```
    ctx.moveTo(50, 20);
    ctx.lineTo(50, 200);
    ctx.moveTo(80, 20);
    ctx.lineTo(80, 200);
    ctx.stroke();
```

下面我们来画两条路径，这下就要用到 beginPath()方法了，看代码：

```
    ctx.moveTo(20, 200);
    ctx.lineTo(100,20);
    ctx.lineTo(300,20);
    ctx.strokeStyle = "blue";
    ctx.stroke();

    ctx.beginPath();
    ctx.moveTo(40, 220);
    ctx.lineTo(150,220);
    ctx.lineTo(230,50);
    ctx.lineTo(280,200);
    ctx.lineWidth = 4;
    ctx.strokeStyle = "#0F0";
    ctx.stroke();
```

浏览效果如图 4.110。

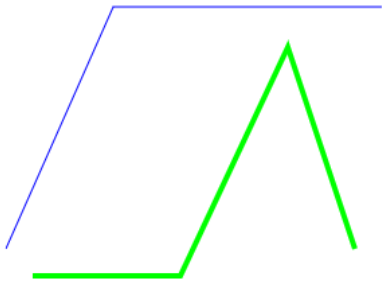


图 4.110 绘制两条路径

语句 ctx.beginPath()开始创建第二条路径，如果还要继续创建第三条路径，同样要再次调用 beginPath()方法，前面已经讲过，第一条路径可以省略 beginPath()语句。在绘制每一条路径前（即调用 ctx.stroke()方法前）可以修改绘画上下文，从而可以画出各种线型的路径。下面再来画一条成 45° 角的虚线：

```
    ctx.moveTo(20, 20);
    ctx.lineTo(200,200);
    ctx.setLineDash([8,3,2,3]);
    ctx.stroke();
```

浏览效果如图 4.111。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

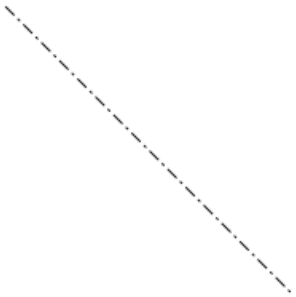


图 4.111 斜的虚线

语句 ctx.setLineDash([8,3,2,3])定义了虚线的最小循环体，即“  ”。再看图 4.112 绘制等腰三角形的例子。

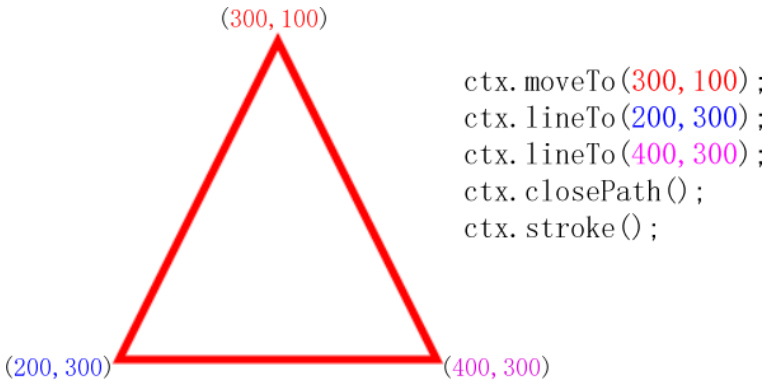


图 4.112 绘制等腰三角形

下面的代码画一个五角星：

```
ctx.beginPath();
ctx.moveTo(250,50);
ctx.lineTo(150,350);
ctx.lineTo(400,150);
ctx.lineTo(100,150);
ctx.lineTo(350,350);
ctx.closePath();
ctx.lineWidth = 5;
ctx.strokeStyle = "red";
ctx.stroke();
```

浏览效果如图 4.113。



图 4.113 绘制五角星

实操训练 1：画一根红色的垂直粗线和一条蓝色的细 45° 斜线，类似图 4.114。

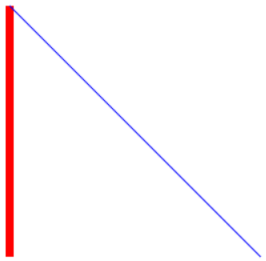
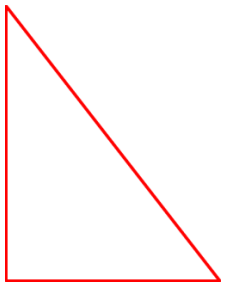


图 4.114 两条颜色和粗度不同的路径

实操训练 2：画一个直角三角形，类似图 4.115。



4.115 绘制三角形

实操训练 3：画一个正方形。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

方形路径

绘制矩形采用 rect()方法，此方法的定义原型为：

```
ctx.rect(x, y, width, height);
```

其中(x,y)为起点坐标（一般是矩形的左上角），*width* 为矩形的宽度，*height* 为矩形的高度。如果 *width* 和 *height* 都为负数，那么起点坐标为矩形的右下角，如果一正一负，起点坐标又是哪个角呢？参见图 4.116。

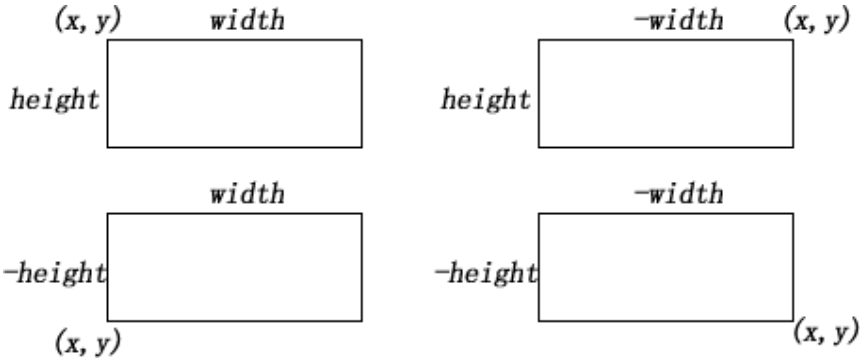


图 4.116 矩形的起点坐标

例如下面的代码画一个矩形（左上角坐标(10,120)，宽度 100，高度 150）：

```
ctx.rect(10, 120, 100, 150);
ctx.stroke();
```

注意：矩形路径的起/止点都是左上角，请看下面的代码：

```
ctx.rect(10, 120, 100, 150);
ctx.lineTo(200,40);
ctx.stroke();
```

浏览的效果见图 4.117。

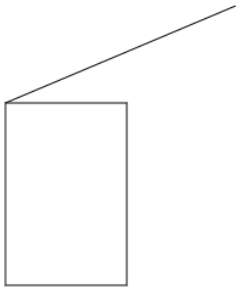


图 4.117 在矩形上增加一条子路径

注意那条斜线子路径的起点就是矩形的左上角。另外一个方法 strokeRect()创建矩形路径的同时完成描边，不需要再额外使用 stroke()方法显示图形了。看例子：

```
    ctx.strokeStyle = 'green';
    ctx.strokeRect(20, 10, 160, 100);
//  ctx.stroke(); /*这条语句不再需要了*/
```

实操练习：画两个如图 4.118 所示的立方体（提醒：第二个立方体存在两条路径）。

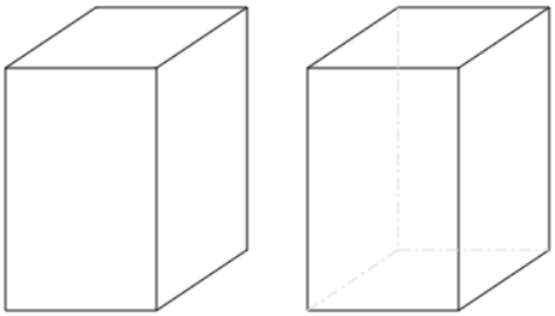


图 4.118 绘制两个立方体

圆弧路径

用 arc()方法画圆弧，此方法的定义原型为：

```
ctx.arc(x, y, radius, startAngle, endAngle[, anticlockwise]);
```

其中 *x,y* 为圆心坐标，*radius* 为半径，*startAngle* 为圆弧起点（单位为弧度），*endAngle* 为圆弧终点，*anticlockwise* 为圆弧方向，true 为逆时针方向，false 顺时针方向，如果 *anticlockwise* 参数省略，默认为顺时针。参考图 4.119 就更容易理解这些参数。

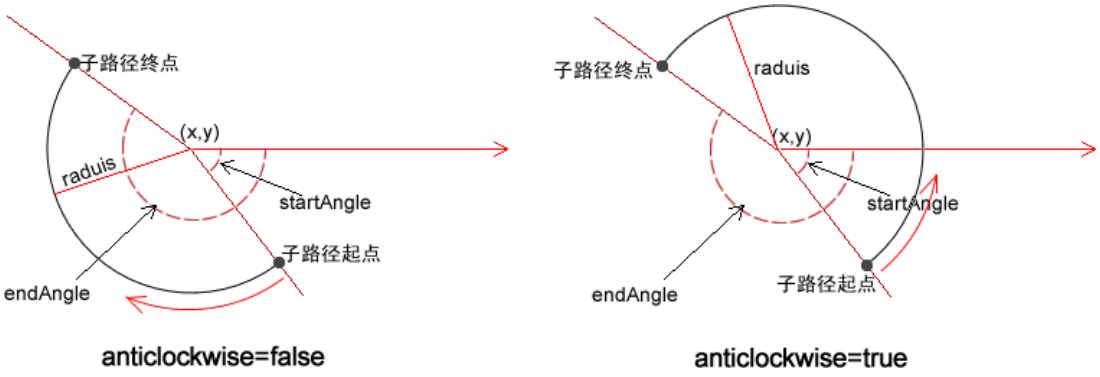


图 4.119 圆弧路径的参数

下面的例子画一个半圆弧：

```
ctx.arc(200, 200, 100, 0, 1*Math.PI);
ctx.stroke();
```

浏览效果见图 4.120。



图 4.120 绘制半圆弧路径

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

大家思考一下下面的例子浏览效果：

```
    ctx.arc(200, 200, 100, 0, 1 * Math.PI, true);
    ctx.stroke();
```

下面的代码画一个完整的圆：

```
    ctx.arc(200, 200, 100, 0, 2 * Math.PI, false);
    ctx.lineWidth = 4;
    ctx.stroke();
```

浏览的效果参见图 4.121。

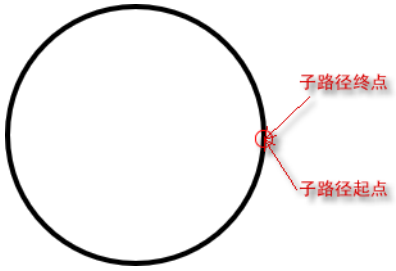


图 4.121 绘制圆形路径

再看下面的例子：

```
    ctx.arc(200, 200, 100, 0, 1.3 * Math.PI, false);
    ctx.closePath();
    ctx.stroke();
```

浏览的效果如图 4.122。

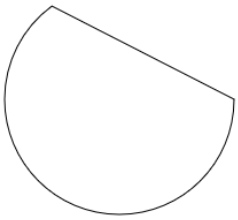


图 4.122 封闭圆弧

实操练习：绘制一个红色的四分之一圆饼图，如图 4.123。

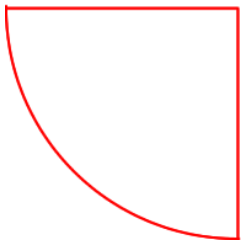


图 4.123 绘制圆饼

另一个绘制带圆弧的子路径的方法是 arcTo()，它从起点出发先创建一段直线，然后再用给定的半径画一段圆弧，此圆弧同时与两条交叉线相切。此方法的定义原型为：

```
ctx.arcTo(x1, y1, x2, y2, radius);
```

坐标(x_1,y_1)为两条线的交叉点，坐标(x_2,y_2)为一条线上的一个点，*radius* 为圆弧的半径，另一条线上的一个点就是本子路径的起点（比如用 moveTo(*x*,*y*)方法确定）。请参考图 4.124 进一步理解各个参数。

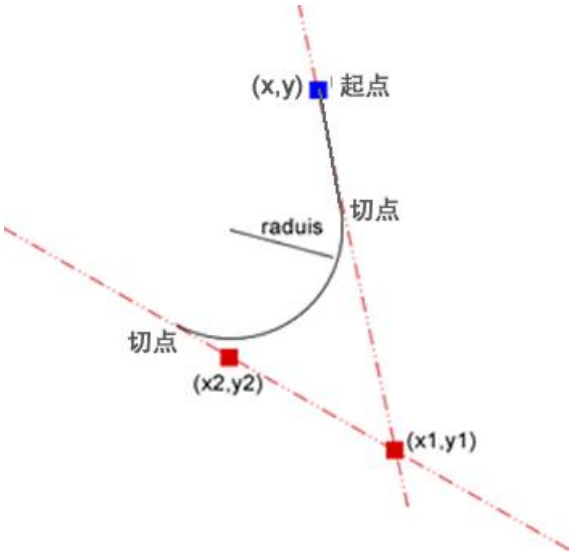


图 4.124 arcTo()方法的参数

在两条交叉直线确定的情况下，需要绘制一段同时与两线相切的圆弧，使用 arcTo()方法是再好不过的了，比如设计两条告诉路之间的过渡弯道。

请看下面的例子：

```
    ctx.moveTo(200, 60);
    ctx.arcTo(300, 300, 50, 150, 50);
    ctx.stroke();
```

浏览的效果如图 4.125。



图 4.125 绘制鱼钩

下面的例子绘制一个圆角：

```
ctx.moveTo(10, 10);
ctx.arcTo(120, 10, 120, 50, 20);
ctx.arcTo(120, 140, 80, 140, 20);
ctx.lineTo(10, 140);
ctx.stroke();
```

浏览的效果见图 4.126。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 4.126 绘制圆角

实操练习：绘制一个虚线圆角矩形（见图 4.127 左侧）（提醒：先确定方角矩形的四个顶点坐标）

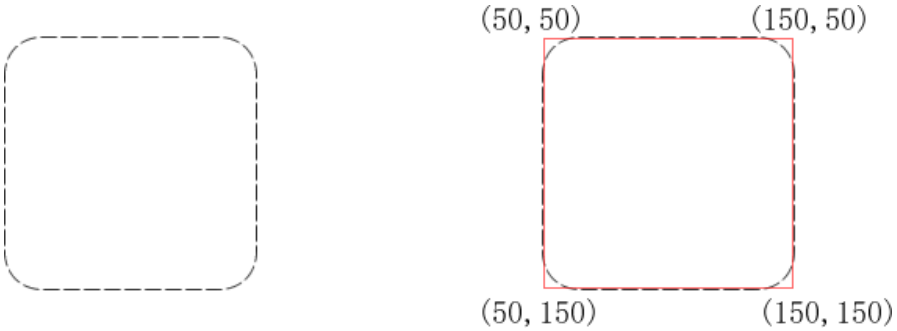


图 4.127 绘制圆角正方形

椭圆路径

采用方法 ellipse()方法绘制椭圆路径，此方法的定义原型为：

```
ctx.ellipse(x, y, radiusX, radiusY, rotation, startAngle, endAngle[, anticlockwise])
```

参数(x,y)为椭圆圆心坐标，*radiusX* 横向半径（x 坐标轴），*radiusY* 纵向半径（y 坐标轴），rotation 椭圆旋转角度（单位弧度），*startAngle* 起点角度，*endAngle* 终点角度，可选参数 anticlockwise 定义绘制方向，true 逆时针，false 顺时针（默认）。请参见图 4.128 进一步理解各种参数。

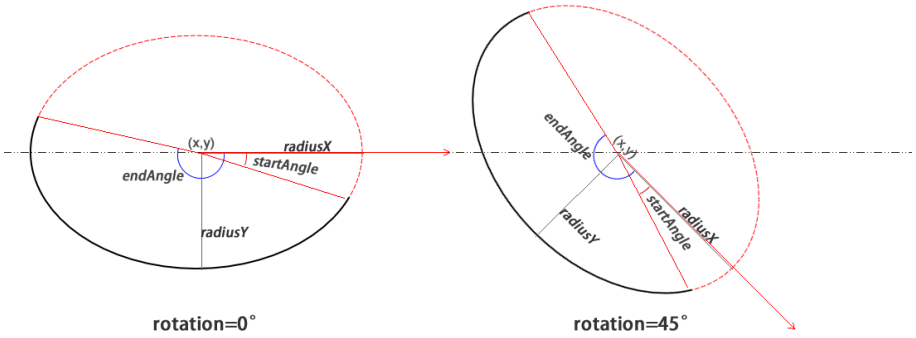


图 4.128 绘制椭圆路径的参数

可以这样理解，先绘制椭圆弧线，然后再旋转 rotation 角度。

下面的例子画一个完整的椭圆形路径，并且旋转 30°：

```
    ctx.strokeStyle = "red";
    ctx.setLineDash([]); /*设置为实线*/
    ctx.lineWidth = 4;
```

```
    ctx.ellipse(200, 200, 100, 60, 30 * Math.PI / 180, 0, 2 * Math.PI);
    ctx.stroke();
```

浏览效果如图 4.129：



图 4.129 绘制椭圆

实操训练：绘制如图 4.130 所示的途径。

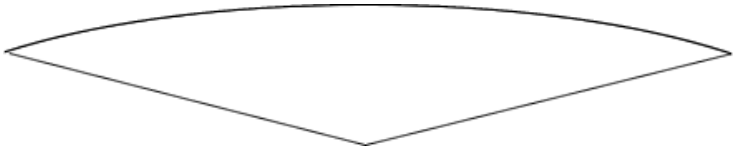


图 4.130 绘制扇形

剪切路径

方法 clip()将当前创建的路径设置为当前剪切路径，剪切路径包围的区域就是接下来的绘画区域，超出部分不会显示。请看例子：

```
    ctx.arc(100, 100, 75, 0, Math.PI * 2, false);
    ctx.clip();
    ctx.fillRect(0, 0, 130, 180);
```

第一条语句创建一条圆形路径，第二条语句把这个圆形路径设置成当前剪切路径，第三条语句绘制一个矩形，这个矩形部分绘在剪切路径之外，因而不会显示，浏览效果见图 4.131 中左侧部分，图右侧解释了剪切绘画的原理即矩形落在剪切区域内的才会显示。

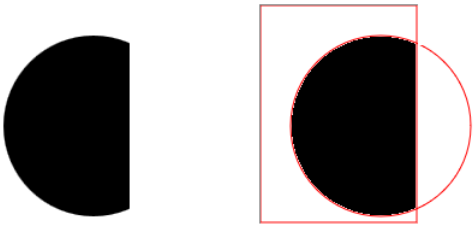


图 4.131 剪切路径绘画

实操练习：绘制如图 4.132 所示的图形。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

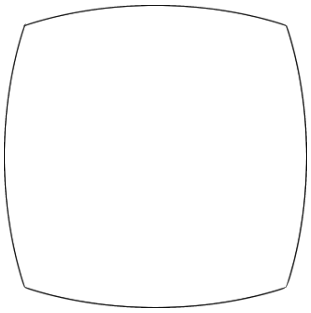


图 4.132 以椭圆为剪切路径

绘制三次贝赛尔曲线路径的方法bezierCurveTo()和二次贝塞尔曲线路径的方法quadraticCurveTo()请参阅其他资料，这里不做介绍。

4.10.4.5 填充

方法 fill()填充当前路径并完成绘制，至于以什么颜色来填充，请参照前面的填充样式定义 fillStyle。如果当前路径不封闭，fill()方法会先从结束点到开始点之间添加一条直线子路径，以封闭该路径，然后再填充该路径的封闭区域。看例子：

```
ctx.rect(20, 10, 160, 100);
ctx.fill();
```

浏览效果见图 4.133。



图 4.133 填充一个矩形

再看例子：

```
ctx.moveTo(10, 20);
ctx.lineTo(200, 20);
ctx.arc(200, 50, 30, -0.5 * Math.PI, 0.5 * Math.PI);
ctx.lineTo(10, 80);
ctx.fillStyle = 'green';
ctx.fill();
```

浏览效果见图 4.134。



图 4.134 填充一条不规则路径

实操练习：绘制一个蓝色的椭圆（填充色为蓝色），参考图 4.135。

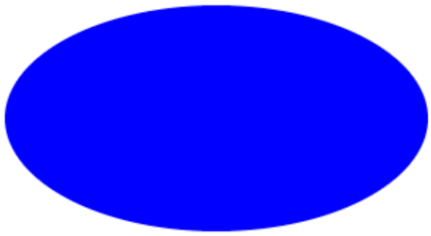


图 4.135 以蓝色填充蓝色路径

方法 fillRect()直接绘制一个实心矩形，此方法的语法如下：

```
ctx.fillRect(x, y, width, height);
```

参数与前面讲过的 rect()方法一样。看例子：

```
ctx.fillStyle = "red"
ctx.fillRect(20, 10, 160, 100);
```

浏览效果见图 4.136。



图 4.136 绘制矩形

方法 clearRect()清除画布上的一个矩形区域，此方法的定义原型如下：

```
ctx.clearRect(x, y, width, height);
```

参数与前面讲过的 rect()方法一样。看下面的例子：

```
    ctx.moveTo(20, 20);
    ctx.lineTo(200, 20);
    ctx.lineTo(120, 120);
    ctx.closePath(); // draws last line of the triangle
    ctx.fill();
```

浏览效果见图 4.137 所示的左边的黑色三角形。

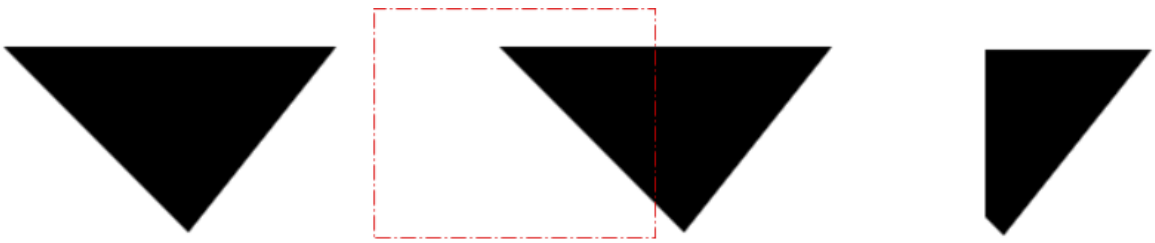


图 4.137 清除部分图形

好了，我们再增加一条清除语句，如下代码：

```
    ctx.moveTo(20, 20);
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    ctx.lineTo(200, 20);
    ctx.lineTo(120, 120);
    ctx.closePath(); // draws last line of the triangle
    ctx.fill();
    ctx.clearRect(10, 10, 100, 150);
```

这样看到的结果如图 4.138 中的右侧部分，中间的图形说明了如何清除一个矩形区域。

结合 fillRect()和 clearRect()两个方法，我们可以轻易实现简单的动画，下面的例子显示一个正方形从左侧移动到右侧。

```
    var x = 10;
    function move(){
        ctx.clearRect(x-2, 48, 44, 44);
        x = (x+2)%(canvas.clientWidth-40);
        ctx.fillRect(x,50,40,40);
    }
    setInterval(move,30);
```

实操训练：结合 arc()、fill()、clearRect()和循环定时器，绘制一个冉冉升起的太阳（提醒：ctx.beginPath();语句要放在 arc(…)语句的前面）。参考图 4.138 所示的提示。

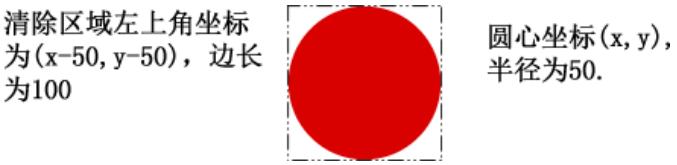


图 4.138 操作提示

4.10.4.6 文本

采用 fillText()方法绘制实心文本，此方法的定义原型如下：

```
ctx.fillText(text, x, y [, maxWidth]);
```

各个参数参见图 4.139 所示。

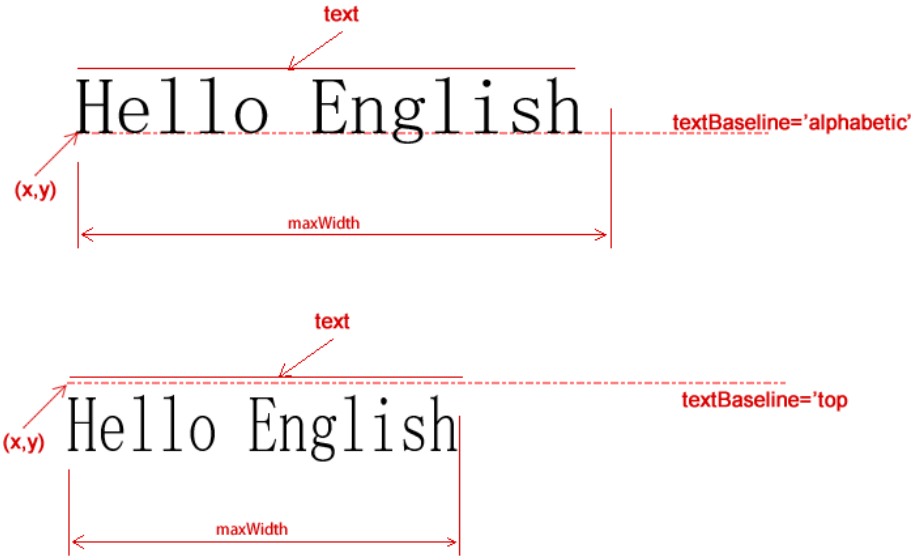


图 4.139 illText()方法的参数

坐标（x,y）相对于文本内容的位置与绘画上下文参数 textBaseline 的值有关。如果 *maxWidth* 小于文本的原始宽度，会压缩文本以满足 *maxWidth* 规定的宽度，看上图的下半部分。看下面的例子：

```
    ctx.font = "60px serif";
    ctx.textBaseline='bottom';
    ctx.fillText("Hello English", 50, 100,300);
    ctx.textBaseline='top';
    ctx.fillStyle = 'red';
    ctx.fillText("Hello English", 50, 100,150);
```

浏览效果参见图 4.140。



图 4.140 绘制实心文本

采用 strokeText()方法绘制空心文本，此方法的定义原型如下：

```
ctx.strokeText(text, x, y [, maxWidth]);
```

各个参数的意思与 fillText()方法一样。看例子：

```
    ctx.font = "60px serif";
    ctx.strokeStyle='green';
    ctx.strokeText("Hello English", 50, 100,300);
```

浏览效果如图 4.141。



图 4.141 绘制空心文本

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

至于如何在各个角度上(比如再 45° 方向)绘制文本,这要结合坐标变换,具体参考后续章节。

4.10.4.7 渐变

渐变就是颜色从一种色过渡到另一种色。有两种渐变方法:一种是线性渐变,即多种颜色沿着直线方向发生渐变,另一种是径向渐变,即多种颜色呈放射状由里向外发生渐变。参见图 4.142,左侧为线性渐变,右侧为径向渐变。

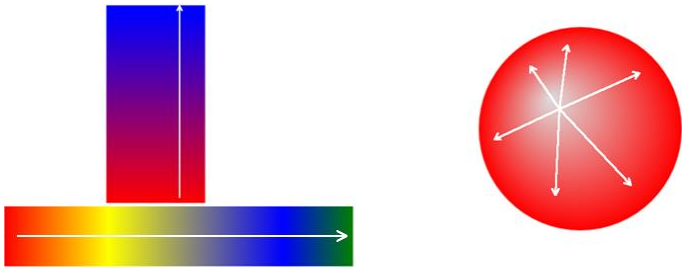


图 4.142 渐变示意图

渐变对象本身不会显示在画布上,它可以作为路径的填充色或描边色使用,即把渐变对象赋给会话上下文的属性 fillStyle 或者属性 strokeStyle。

线性渐变

使用 createLinearGradient()方法创建线型渐变,此方法的定义原型如下:

```
CanvasGradient ctx.createLinearGradient(x0, y0, x1, y1);
```

(*x0,y0*)浅变起点坐标,(*x1,y1*)浅变终点坐标,当 x0==x1 时垂直浅变,y0==y1 时水平浅变,否则斜线浅变。此方法返回一个 CanvasGradient 类型的对象,接下来就要利用此对象的 addColorStop()方法添加颜色。此方法的定义原型如下:

```
void gradient.addColorStop(offset, color);
```

offset 参数取值范围是[0,1],代表离渐变起点的距离占比,因此 0 就落在起点,0.5 落在起/止点的中间,1 落在终点。color 参数取值与 css3 的 color 属性一样。对于这个两个方法的参数进一步理解,请参考图 4.143。

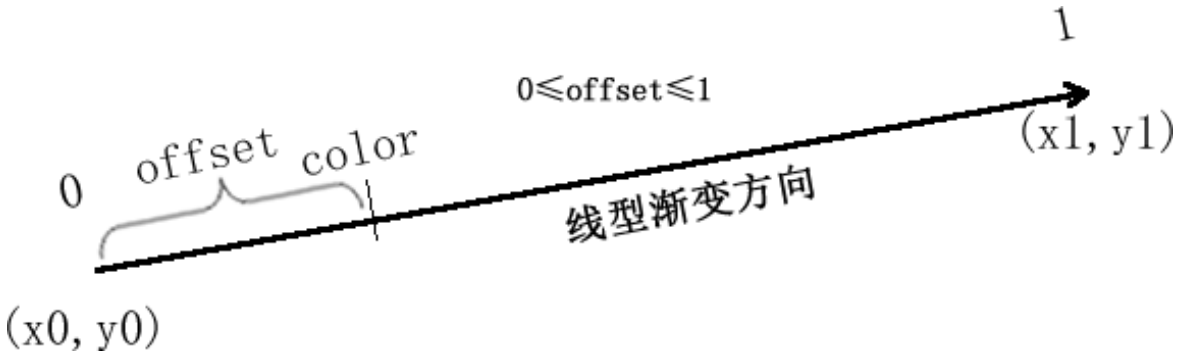


图 4.143 addColorStop()方法的参数

我们先看一个例子:

```
20          var linegra = ctx.createLinearGradient(10, 0, 200, 0);
21          linegra.addColorStop(0, "red");
22          linegra.addColorStop(1, "blue");
23          ctx.fillStyle = linegra;
24          ctx.fillRect(10,10,200,100);
```

序号为 20 的语句创建一个水平的线性渐变对象，并保存在 linegra 变量中，21 号语句设置起点颜色为红色，22 号语句定义终点颜色为蓝色，起点与终点之间产生红到蓝的过渡渐变，23 号语句把渐变对象赋给填充样式，24 号语句绘制一个矩形并用刚刚创建的渐变填充。浏览效果如图 4.144 所示。



图 4.144 绘制渐变矩形

需要特别注意的是，线性渐变存在起/止坐标，如果用线性渐变对象填充的一个矩形超出了渐变的起止坐标，那么超出部分就是纯色，请看下面的例子：

```
        var linegra = ctx.createLinearGradient(50, 0, 400, 0);
        linegra.addColorStop(0, "red");
        linegra.addColorStop(0.3,"yellow");
        linegra.addColorStop(0.8,"blue");
        linegra.addColorStop(1, "green");
        ctx.fillStyle = linegra;
        ctx.fillRect(50,10,350,60);
        ctx.fillRect(2,80,540,60);
        ctx.fillRect(50,150,140,60);
        ctx.fillRect(240,150,160,60);
        ctx.fillRect(130,220,170,60);
        ctx.strokeStyle = linegra;
        ctx.moveTo(0,300);
        ctx.lineTo(canvas.clientWidth,300);
        ctx.stroke();
```

浏览的效果如图 4.145（为了解释，我特意加了很多标注）。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

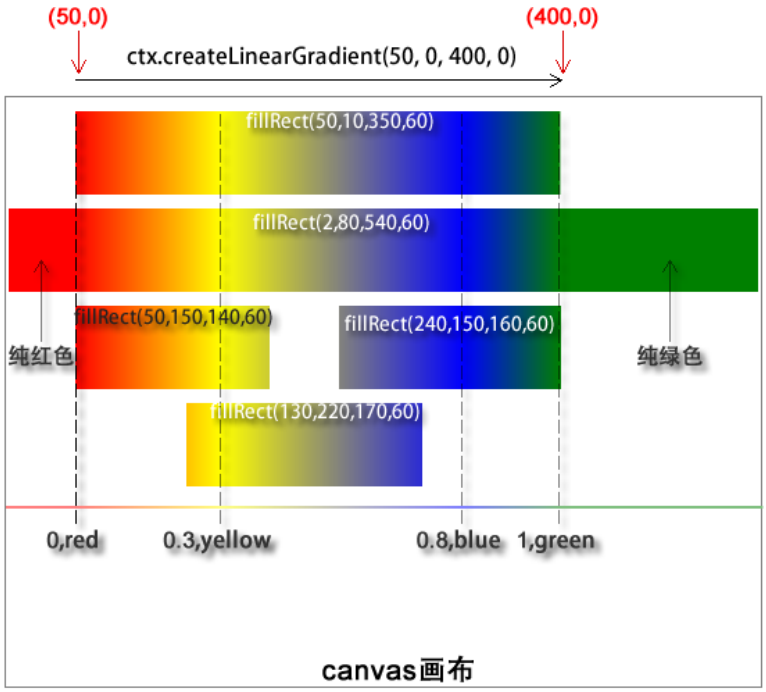


图 4.145 超出渐变起/止位置部分为纯色

本例子用同一个线性渐变填充了五个矩形，并绘制了一根水平线条，但是矩形的坐标和宽度都不同，所以各个矩形中填充的效果是不同的。其实可以这样理解：一个渐变对象针对整块画布，画布上的一个采用此渐变对象的矩形相当于开了一扇窗户，我们透过这个窗户看到了画布上的渐变颜色，移动窗户后看到的渐变颜色是不同的，见图 4.146 所示。

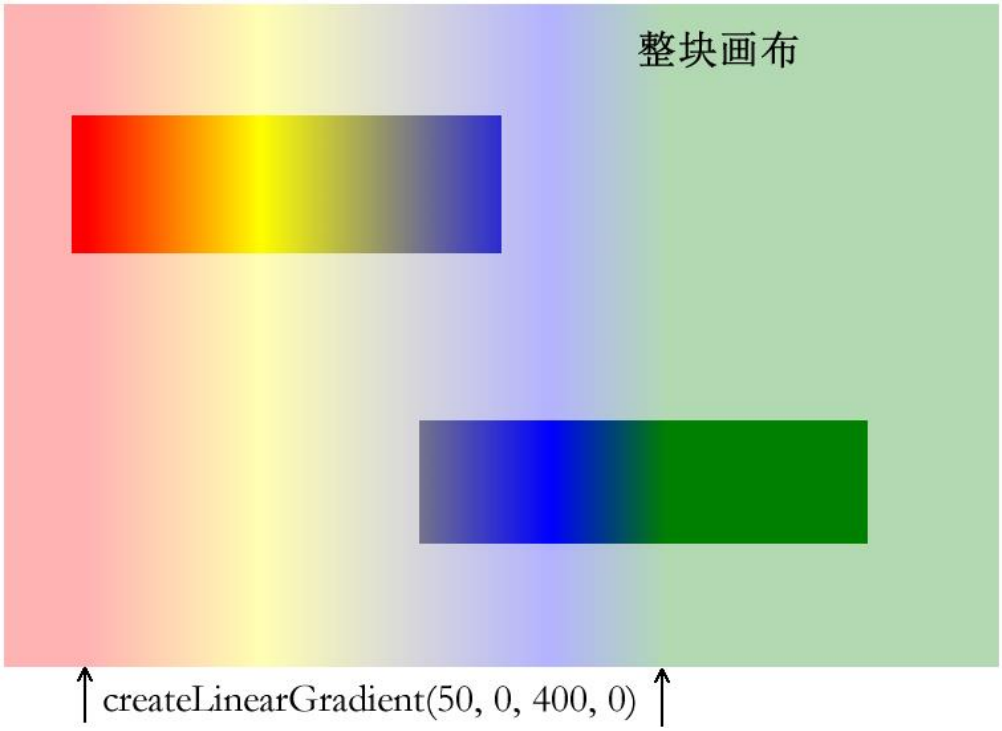


图 4.146 渐变针对整块画布

实操练习：绘制一个徐徐升起的太阳，在上升的过程中，太阳的颜色依次变化：暗红→深红→红色→鲜红→红白，就是模拟现实生活当中早晨到中午的太阳。（提醒：创建一个垂直方向的线型渐变）

径向渐变

使用 createRadialGradient()方法创建径向渐变，此方法的定义原型如下：

```
CanvasGradient ctx.createRadialGradient(x0, y0, r0, x1, y1, r1);
```

其中 *x0,y0,r0* 为渐变起点圆形的圆心坐标和半径，*x1,y1,r1* 为渐变终点圆形的圆心坐标和半径，r0 和 r1 孰大孰小没有严格规定，而且可为 0。当两个圆心重合时就是放射型效果，否则就是光照效果。参看图 4.147 进一步理解这些参数。

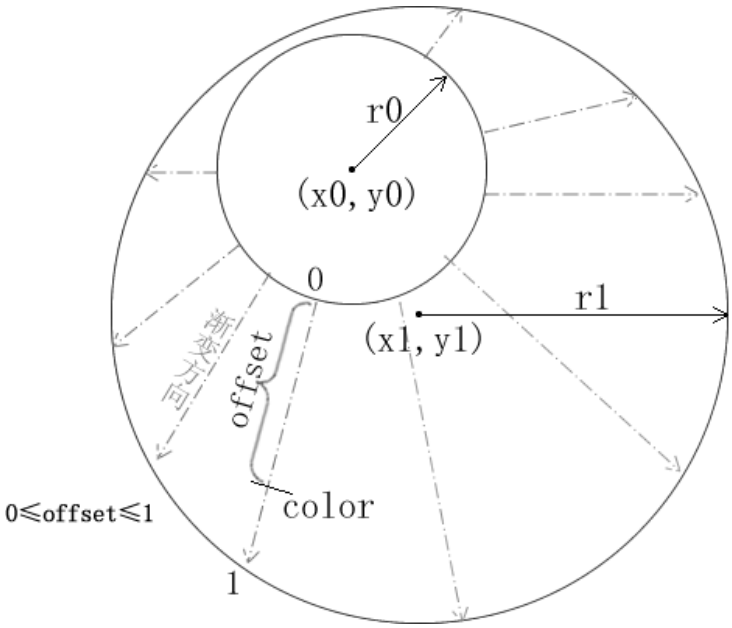


图 4.147 径向渐变方法的各个参数

看例子：

```
var grd = ctx.createRadialGradient(100, 100, 0, 100, 100, 100);
grd.addColorStop(0, "white");
grd.addColorStop(1, "black");
ctx.fillStyle = grd;
ctx.fillRect(0, 0, 200, 200);
```

浏览效果见图 4.148 的最左侧部分。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

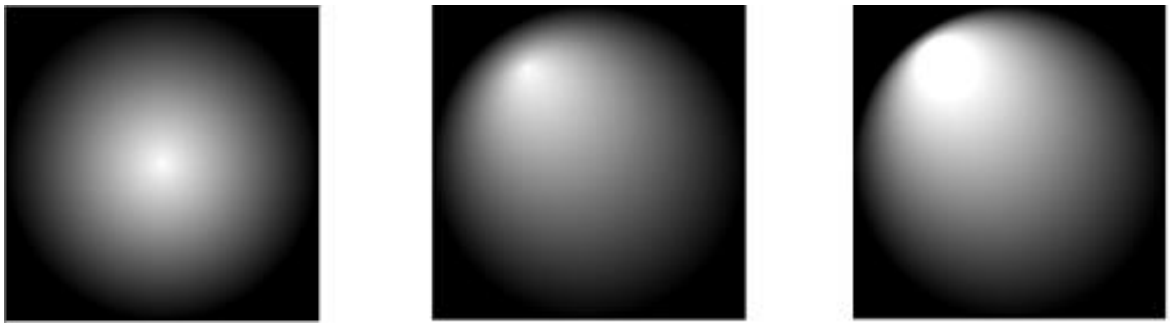


图 4.148 径向渐变举例

如果把上例中的起点圆心坐标改为(60,40)，浏览效果就是图 4.148 中的中间部分，继续修改起点圆的半径为 20，浏览效果就是图 4.148 中的右侧部分。还可以增加更多的过渡色。再来看一个绘制红色气球的例子：

```
var grd = ctx.createRadialGradient(120, 120, 0, 150, 150, 100);
grd.addColorStop(0, "#ddd");
grd.addColorStop(1, "red");
ctx.fillStyle = grd;
ctx.arc(150,150,100,0,2*Math.PI);
ctx.fill();
```

浏览效果如图 4.149。

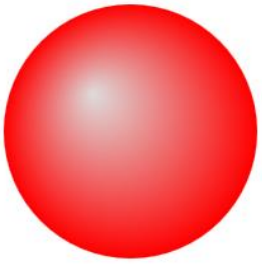


图 4.149 红色气球

对于下面的代码，大家看看浏览的效果是什么？

```
var grd = ctx.createRadialGradient(300, 300, 250, 300, 300, 300);
grd.addColorStop(0, "white");
grd.addColorStop(0.15, "red");
grd.addColorStop(0.30, "green");
grd.addColorStop(0.45, "blue");
grd.addColorStop(0.6, "grey");
grd.addColorStop(0.75, "purple");
grd.addColorStop(0.9, "yellow");
grd.addColorStop(1, "white");
ctx.fillStyle = grd;
ctx.fillRect(0, 0, 600, 300);
```

一个径向渐变对象针对于整个画布，画布上的一个采用此渐变对象的矩形或圆形相当于开了一扇窗户，我们透过这个窗户看到了画布上的径向渐变颜色，移动窗户后看到的渐变颜色是不同的，见图 4.150 所示。

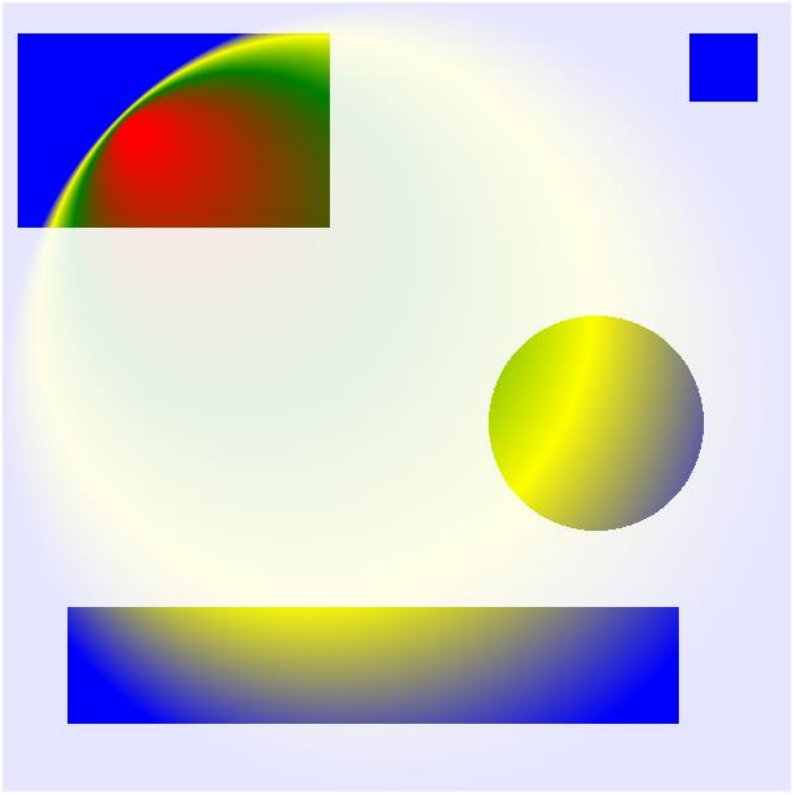


图 4.150 径向渐变针对整块画布

实操练习 1：绘制一个图 4.151 所示的彩虹（提醒：红橙黄绿青蓝紫=red/orange/yellow/green/cyan/blue/purple）。



图 4.151 彩虹

实操练习 2：绘制一个沿着水平方向弹跳的气球（提醒圆心坐标：y=Math.sin(x)）。示意图参见图 4.152。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

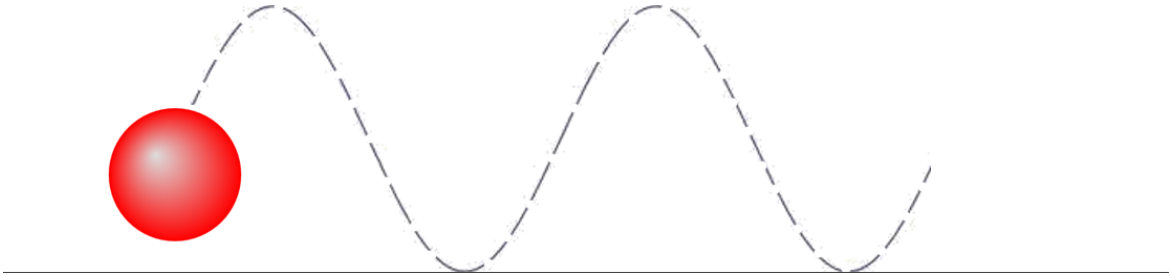


图 4.152 弹跳的气球

模式对象

使用 createPattern()方法创建一个 CanvasPattern 模式对象，此模式对象可以作为填充样式。前面讲到过 fillStyle 和 strokeStyle 这两个绘画上下文属性的值可以是颜色、渐变和 CanvasPattern 模式对象。createPattern()方法的定义原型如下：

```
CanvasPattern ctx.createPattern(image, repetition)
```

其中参数 image 为重复图像源对象，repetition 定义如何重复图像，允许的值有："repeat"垂直和水平方向都重复，"repeat-x"水平方向重复，"repeat-y"垂直方向重复，"no-repeat"都不重复。其实说白了，就是用一个小图片来平铺填充一个区域，就像房间地面贴瓷砖那样。例如：有一个小图片  ，我觉得很好看，现在要拿它来定义填充样式，下面的代码就可以实现我的目标：

```
var img = new Image();
img.src = 'Canvas_createpattern.png';
img.onload = function () {
    var pattern = ctx.createPattern(img, 'repeat');
    ctx.fillStyle = pattern;
    ctx.fillRect(10, 10, 200, 200);
}
```

要把图片文件 Canvas_createpattern.png 放在网页文件所在的那个目录下，然后浏览的效果就是图 4.153 中的右下部分。



图 4.153 用模式对象填充

实操训练：绘制如图 4.154 所示的效果。



图 4.154 地平线上的圆

4.10.4.8 绘制图像

采用 drawImage()方法可以把一张图片导入画布，此方法的定义原型如下：

```
void ctx.drawImage(image[, sx, sy, sWidth, sHeight], dx, dy[, dWidth, dHeight])
```

参数说明：

image 为图像对象，类型可为任何的 canvas 图像源(CanvasImageSource)；sx,sy,sWidth 和 sHeight 参数是可选参数，定义如何对原图剪切，dx,dy,dWidth,dHeight 规定图像如何放置在画布上，其中后两个参数是可选的，请参考图 4.155 进一步理解这些参数。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

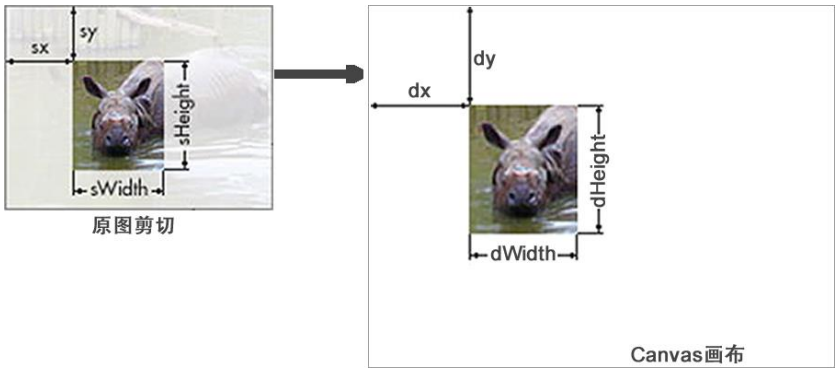


图 4.155 绘制图像方法的参数

drawImage()方法的三种用法：

```
    ctx.drawImage(image, dx, dy);
    ctx.drawImage(image, dx, dy, dWidth, dHeight);
    ctx.drawImage(image, sx, sy, sWidth, sHeight, dx, dy, dWidth, dHeight);
```

请看下面的例子：

```
    var img = new Image();
    img.src = 'eggs.jpg';
    img.onload = function () {
        ctx.drawImage(img,10,10);
        ctx.drawImage(img,320,10,200,200);
        ctx.drawImage(img,160,140,100,100,530,10,100,180);
    }
```

浏览的效果见图 4.156。

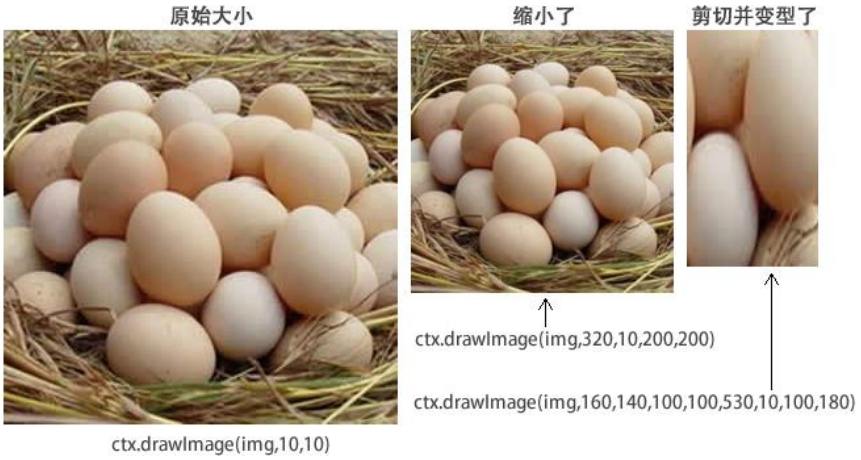


图 4.156 绘制图像

再来看一个例子：

```
<canvas id="myCan" width="800px" height="600px">不支持 canvas</canvas>
<img id="source" src="https://mdn.mozillademos.org/files/5397/rhino.jpg">
<script>
    var canvas = document.getElementById("myCan");
    var ctx = canvas.getContext("2d");
    var image = document.getElementById('source');
    setTimeout("ctx.drawImage(image, 33, 71, 104, 124, 21, 20, 87, 104);",500);
</script>
```

之所以要延时 500 毫秒再调用 drawImage()方法，那是因为要等网页上的图像加载完毕。浏览的效果见图 4.157。



图 4.157 导入页面上的图片

实操练习：网页上有 img 和 canvas 元素，截取 img 元素的图像的右下角部分，然后放大 2 倍显示在 canvas 画布上。

4.10.4.9 阴影

阴影由绘画上下文中的属性 shadowBlur、shadowColor、shadowOffsetX 和 shadowOffsetY 定义，对于这些属性的理解请看图 4.158。

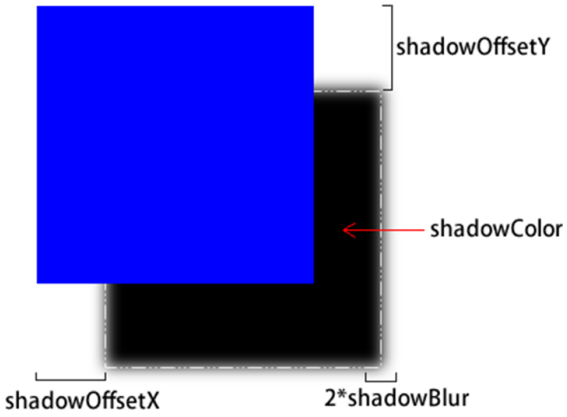


图 4.158 与阴影有关的几个上下文属性

如果模糊为 0，即 shadowBlur=0，那么阴影的大小就是图形的大小，即那个白色虚线框的尺寸等于蓝色矩形的尺寸，否则以虚线框为中心，内外各延申 shadowBlur 定义的尺寸为模糊区域，比如 shadowBlur=10，那么模糊宽度总共 20。

shadowBlur、shadowOffsetX 和 shadowOffsetY 默认都为 0，shadowColor 默认为全透明，也就是没有阴影。shadowOffsetX 和 shadowOffsetY 可为负数，此时阴影在偏左或者偏上。看例子：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
ctx.shadowBlur = 0;
ctx.shadowColor = "black";
ctx.shadowOffsetX = 10;
ctx.shadowOffsetY = 10;
ctx.fillStyle = "blue"
ctx.fillRect(100, 100, 100, 50);
```

浏览的效果如图 4.159。



图 4.159 带阴影的矩形

下面我们再来给前面曾经举的例子（绘制红色气球，见图 4.149）增加阴影，这样看起来立体感更强一些，灰色部分为原来的代码。

```
ctx.shadowBlur = 150;
ctx.shadowColor = "grey";
ctx.shadowOffsetX = 5;
ctx.shadowOffsetY = 300;

var grd = ctx.createRadialGradient(270, 100, 0, 300, 150, 100);
grd.addColorStop(0, "#ddd");
grd.addColorStop(1, "red");
ctx.fillStyle = grd;
ctx.arc(300, 150, 100, 0, 2 * Math.PI);
ctx.fill();
```

浏览效果如图 4.160。

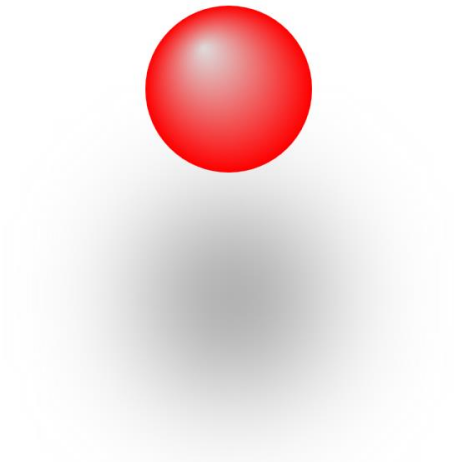


图 4.160 带阴影的气球

再看一个综合的例子，浏览的效果如图 4.161。

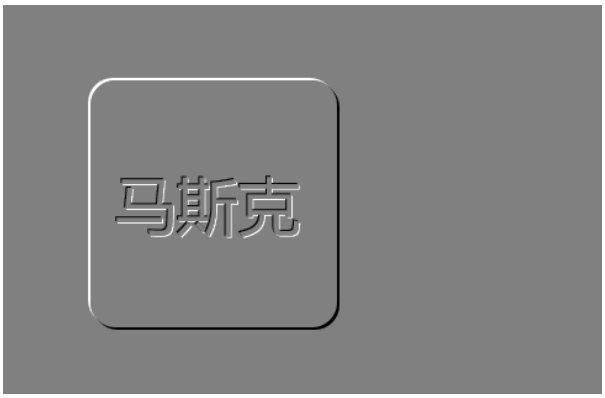


图 4.161 多重阴影效果

实现的代码参见 chapter-4\index-4.html，不过画布元素的背景色也要改为"grey"。

实操练习：绘制如图 4.162 所示的效果。



图 4.162 多重阴影效果实操

4.10.4.10 变换

在 canvas 画布上隐含一个变换矩阵，在作画的第二步（即绘制，第一步是创建）会使用此变换矩阵把图形绘制出来。这个变换矩阵的名字在 firefox 浏览器中为 mozCurrentTransform，在 chrome 浏览器中被隐藏。可以通过 translate()、rotate()、scale()、transform()、setTransform()和 resetTransform()方法来修改矩阵中的数据，下面我们一一介绍。

translate()方法

移动当前坐标系，其定义原型如下：

```
void ctx.translate(x, y)
```

x 水平移动距离，y 垂直移动距离，x、y 可正可负，理解请参见图 4.163。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

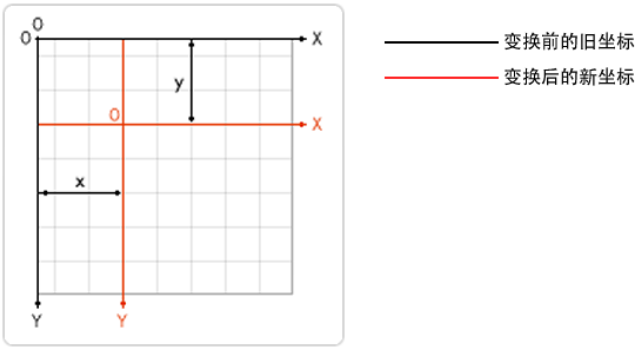


图 4.163 变换坐标系

请看下面的例子：

```
ctx.fillRect(0, 0, 60, 60);
ctx.translate(100, 100);
ctx.fillRect(0, 0, 60, 60);
```

浏览的效果如图 4.164。

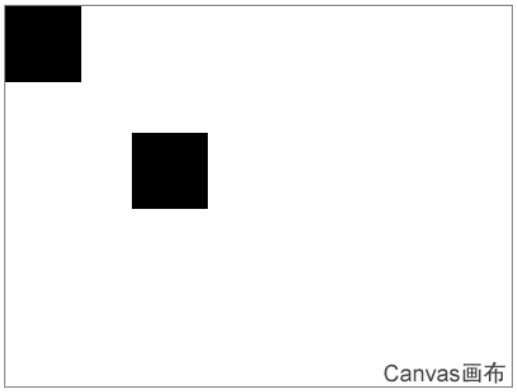


图 4.164 在两个坐标系中绘制同一个矩形

由于坐标系向右下方移动了，所以旧坐标系的点(100,100)成了移动后的新坐标系的原点，第二条绘制矩形的语句与第一条参数完全一样，但是在新坐标系中执行的，所以最终看到的位置不同。

rotate()方法

以坐标原点为转轴旋转当前的坐标系，从而产生新的坐标系，此后作画就在这个新坐标系上。此方法的定义原型如下：

```
void ctx.rotate(angle)
```

参数 angle 为旋转的弧度，正数表示顺时针，用负数表示逆时针，参考图 4.165。

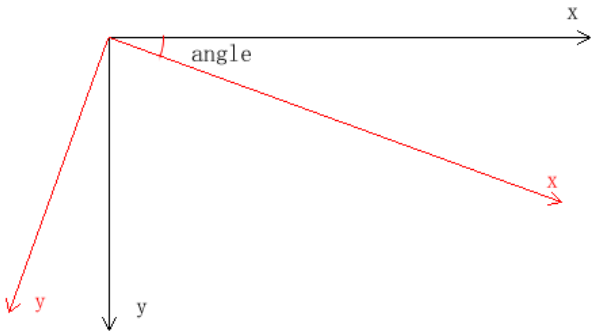


图 4.165 旋转变换

如果你想通过角度值计算，可以使用公式 degree * Math.PI / 180 转换为弧度，比如旋转 45° 角，就是 ctx.rotate(45*Math.PI/180)。请看例子：

```
ctx.rotate(45 * Math.PI / 180);
ctx.fillRect(70, 0, 100, 30);
```

浏览效果如图 4.166 所示，其中红色部分是为了解析，我特意加上去的。

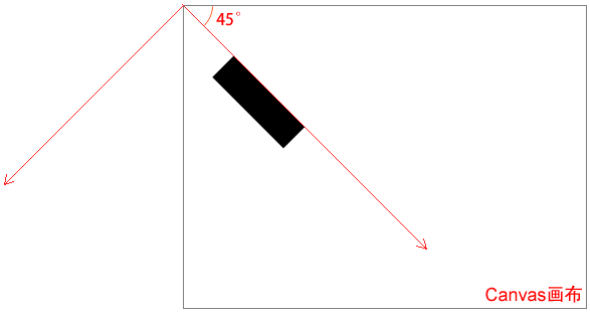


图 4.166 旋转变换后再绘制矩形

scale()方法

缩放当前坐标系的长度计量单位，从而产生一个新的坐标系。默认一个单位等于屏幕上的一个像素，比如在 fillRect(10,20,60,40)中，矩形宽 60 就是 60px。用 scale()方法可以改变一个单位等于几个像素，其定义原型如下：

```
void ctx.scale(x, y)
```

参数 x 定义水平缩放因子，y 定义垂直缩放因子，均为正实数。比如 scale(0.5,2)定义水平方向的一个单位等于 0.5 个像素，垂直方向的一个单位等于 2 个像素，此后绘制的图形的尺寸，在水平方向缩小一半，垂直方向放大两倍。

请看例子：

```
ctx.fillRect(10, 10, 50, 50);
ctx.scale(0.5, 2);
ctx.fillStyle = "red";
ctx.fillRect(10, 10, 50, 50);
```

红色下划线的两条语句一摸一样，但是后一条语句在缩放变换后的新坐标上绘制矩形，x 坐标值和宽度都得乘以 0.5，y 坐标值和高度都必须乘以 2，所以相当于旧坐标系的

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

ctx.fillRect(10*0.5,10*2,50*0.5,50*2)语句，即 ctx.fillRect(5, 20, 25, 100)，因此最终看到的两个矩形完全不同，参考图 4.167。



图 4.167 缩放变换前后绘画对比

允许对坐标系做连续变换，变换的效果是叠加的，因为每一次变换都是操纵当前的坐标系。比如把画布的右下角作为坐标的原点（见图 4.168 所示），这样就要先移动，再旋转。

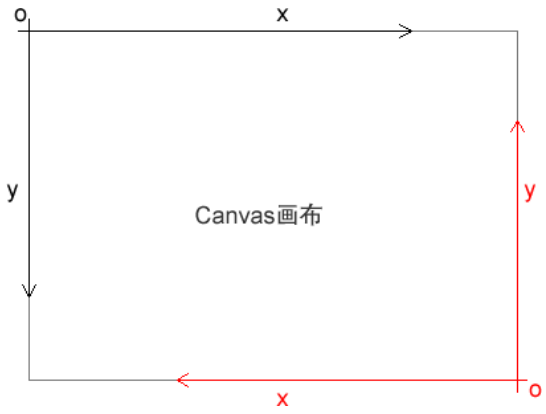


图 4.168 多重变换

代码如下：

```
    var can = document.getElementById("myCan");
    var ctx = can.getContext("2d");
    ctx.translate(can.clientWidth, can.clientHeight);
    ctx.rotate(Math.PI);
    ctx.fillRect(10, 10, 100, 50);
```

浏览效果如图 4.169。

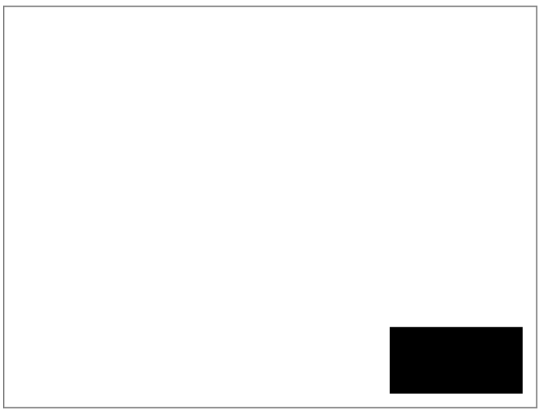


图 4.169 右下角为坐标原点

再来看一个连续变换的例子：

```
ctx.translate(160, 140);
ctx.fillStyle = "rgba(0,140,0,0.5)";
ctx.fillRect(0, 0, 10, 5);

for (i = 0; i < 20; i++) {
    ctx.scale(1.2, 1.2);
    ctx.rotate(20 * Math.PI / 180);
    ctx.fillRect(0, 0, 10, 5);
}
```

浏览的效果如图 4.170。

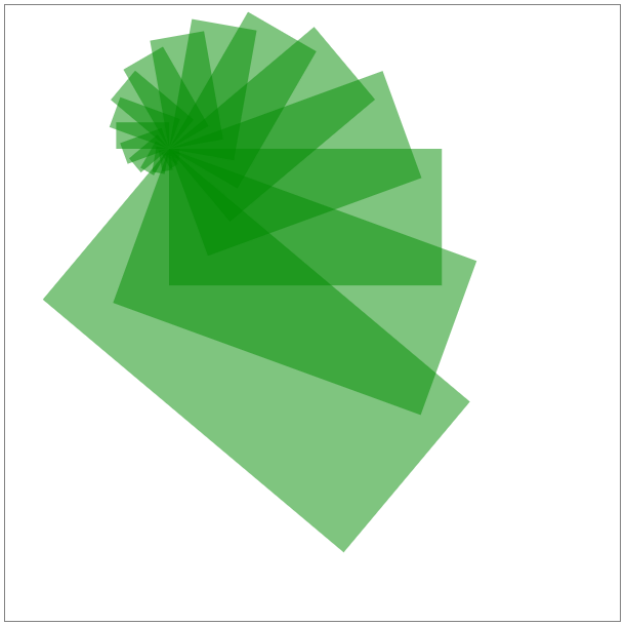


图 4.170 连续变换

transform()方法

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

通用变换方法，可以一次性完成各种变换，比如不但能移动、旋转、缩放，还能扭曲、切变等更多的复杂变换。而上面所介绍的几种方法，每种只能完成一种变换，算是 transform()方法的特例。此方法的定义原型如下：

```
void ctx.transform(a, b, c, d, e, f)
```

参数 a,b,c,d,e,f 在变换矩阵中的位置如下：

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

其中 a,b,c,d,e,f 是变换矩阵中的 6 个有效数字，作为一维数组保存在绘画上下文参数 mozCurrentTransform 中，用下面的代码可以打印这六个数字的默认值：

```
console.log(ctx.mozCurrentTransform);
```

然后在火狐浏览器中的开发者工具的控制台上看到图 4.171 所示的一维数组的值。

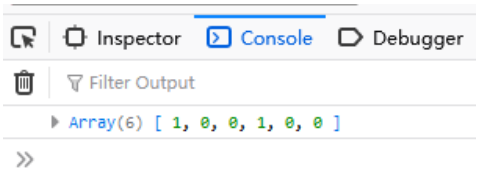


图 4.171 变换矩阵的默认值

因此[a,b,c,d,e,f]=[1,0,0,1,0,0]，对应的默认变换矩阵就是一个单位矩阵：

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

假设(x,y)变换后的坐标为(x1,y1)，那么两个坐标值之间的计算公式如下：

$$\begin{bmatrix} x1 \\ y1 \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \xrightarrow{可得} \begin{cases} x1 = a \times x + c \times y + e \\ y1 = b \times x + d \times y + f \end{cases}$$

把默认的变换矩阵代入上述公式可得 x1=x，y1=y，即画布默认是没有变换的。由上述公式还可知：a 是水平缩放因子，d 为垂直缩放因子，e 为水平移动距离，f 为垂直移动距离，b 为垂直切变因子，c 为水平切变因子。

因此有以下推论：

1、平移变换 trnanslate(x,y)等价于 transform(1,0,0,1,x,y)。

$$\begin{cases} x1 = x + e \\ y1 = y + f \end{cases} \Rightarrow \begin{cases} x1 = 1 \times x + 0 \times y + e \\ y1 = 0 \times x + 1 \times y + f \end{cases} \Rightarrow tranform(1,0,0,1,e,f)$$

2、缩放变化 scale(x,y)等价于 transform(x,0,0,y,0,0)。

$$\begin{cases} x1 = a \times x \\ y1 = d \times y \end{cases} \Rightarrow \begin{cases} x1 = a \times x + 0 \times y + 0 \\ y1 = 0 \times x + b \times y + 0 \end{cases} \Rightarrow tranform(a,0,0,b,0,0)$$

3、旋转变换 rotate(x)等价于transform$(\cos x, \sin x, -\sin x, \cos x, 0,0)$。

推导过程见图 4.172 所示。

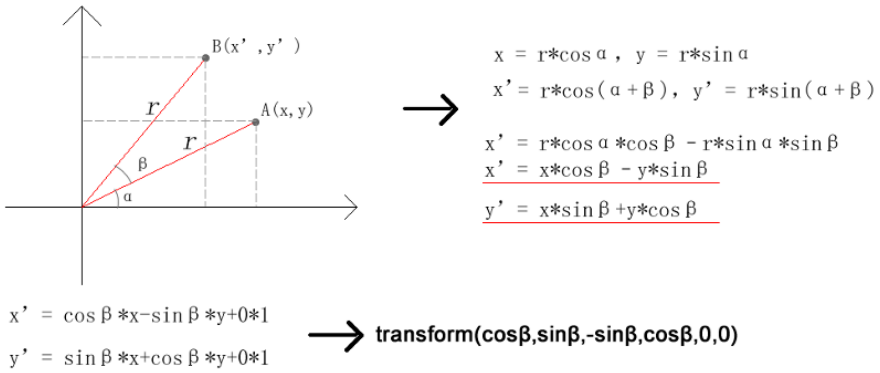


图 4.172 旋转变换的推导过程

由于方法里的参数为弧度，而我们更容易接受角度单位，所以在编程时要转换为弧度，比如 x° 角转化为弧度就是 x*Math.PI/180，此时方法调用就是：

```
transform(Math.cos(x * Math.PI / 180), Math.sin(x * Math.PI / 180), -
Math.sin(x * Math.PI / 180), Math.cos(x * Math.PI / 180), 0, 0)
```

同构变换的条件是 b=-c

使用 transform()可以实现切变，这时要运用 b 和 c 两个参数了，前面已经讲过，b 在垂直方向上切变，c 在水平方向上切变。我们看一个例子：

```
ctx.transform(1, 0.5, 0, 1, 0, 0);
ctx.fillRect(10, 10, 50, 30);
```

浏览效果见图 4.173。



图 4.173 切变变换

方法 setTransform(a, b, c, d, e, f)就是在单位矩阵（默认矩阵）的基础上做变换，而 transform()方法是在当前变换矩阵上继续变换，具有叠加效应。

方法 resetTransform()把变换矩阵重置为默认矩阵（单位矩阵），此方法没有参数。

4.10.4.11 综合例子

例子 1

结合我们已经学过的 canvas 知识，现在做一个综合练习，即实现一个字帖，可以通过移动鼠标写字，参考图 4.174。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

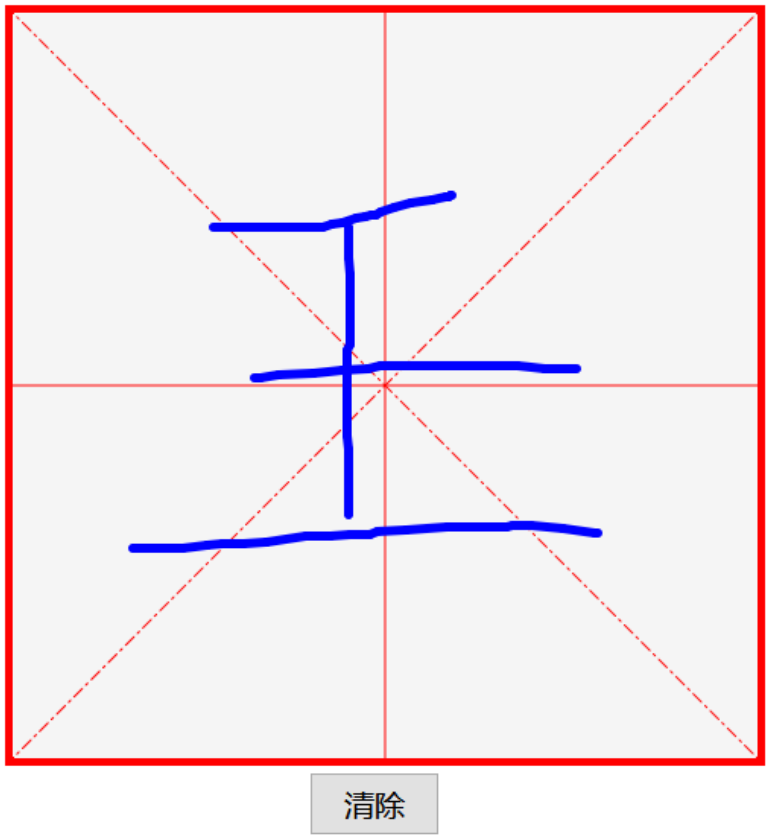


图 4.174 字帖

提示：按下鼠标左键会触发<canvas>元素对象的 mousedown 事件，移动鼠标会触发 mousemove 事件（大概 50 毫秒触发一次），所以在 50 毫秒内可视为鼠标在直线上移动，因此绘制一根直线就可以了，比如我们把鼠标移动的路径上一小部分加以放大，类似图 4.175，宏观上的一条曲线实际上由众多的直线段组成：

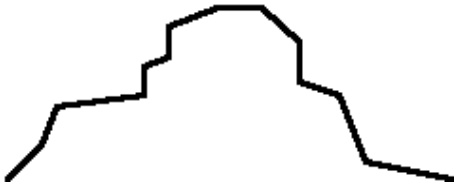


图 4.175 曲线在微观上看作由若干直线段组成

实现代码参考 chapter-4\copybook.html 文件，对于按下或移动鼠标的事件，浏览器会把 MouseEvent 对象实例作为实参调用事件处理函数，而 MouseEvent 对象的属性 offsetX 和 offsetY 保存了当前鼠标在 canvas 画布上的坐标。核心的代码主要是这几条：

```
    //鼠标点下
    canvas.onmousedown = function (e) {
        isDraw = true;
        oldPos = {x:e.offsetX, y:e.offsetY};
    }
    //鼠标移动
    canvas.onmousemove = function (e) {
```

```
        if (isDraw) {
            context.beginPath();
            context.moveTo(oldPos.x, oldPos.y);
            context.lineTo(e.offsetX, e.offsetY);
            context.stroke();
            oldPos = {x:e.offsetX, y:e.offsetY};
        }
    }
```

实操练习：改进上面的例子，增加修改笔迹颜色的功能。

例子 2

实现蚂蚁线——移动的虚线。要求达到的效果如图 4.176 所示，虚线在绕文字移动：



图 4.176 绕行文字的蚂蚁线

实现代码参考 chapter-4\ant.html 文件，我们用到了定时器、绘画上下文属性 lineDashOffset 和虚线。对于虚线的偏移参数 lineDashOffset，请参考图 4.177 好好理解。

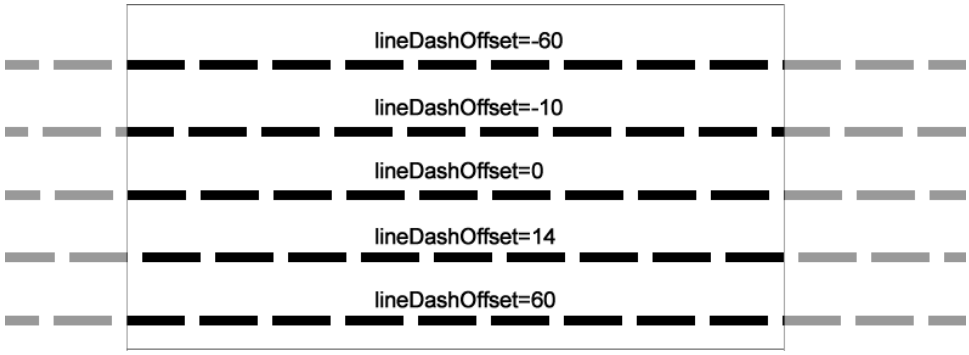


图 4.177 虚线的 lineDashOffset 参数

图 4.177 中，虚线的最小循环体中的线段长度 46，空白长度 14，总长度为 60。lineDashOffset 的值为正时虚线向右移动，比如 lineDashOffset=14，虚线右移 14 个像素。lineDashOffset 的值为负数时，虚线左移，比如 lineDashOffset=-10，则左移了 10 个像素。实际上 lineDashOffset=0、lineDashOffset=-60、lineDashOffset=60 三者的效果一样。进一步推导 lineDashOffset=x+n*60 的效果都一样，其中 n 为 0、1、2、3、……。

因此我们采用定时器不断增大 lineDashOffset 的值，就可以实现移动效果了。摘录的几条核心代码如下：

```
    var offset = 0;
    ctx.textBaseline = "top";
    ctx.setLineDash([46, 14]);
    function draw() {
        offset = (offset + 1) % 60;//虚线最小循环体的总长度为 60
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
        ctx.clearRect(0, 0, canvas.width, canvas.height);
        ctx.lineDashOffset = -offset;
        ctx.strokeRect(10, 10, 396, 56);
        ctx.fillText("看我周围的蚂蚁线", 15, 15);
        requestAnimationFrame(draw);
    }
    requestAnimationFrame(draw);
```

练习

1、修改上例中的蚂蚁行走方向。

2、画一根 30° 斜蚂蚁线（如图 4.178 左侧部分）。再挑战下自己的智商，能否实现折线蚂蚁线呢（见图 4.178 右侧部分）？

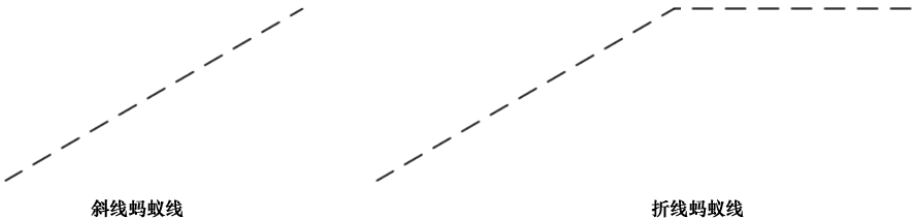


图 4.178 蚂蚁斜线和折线

4.10.6 嵌入页面

采用<iframe>元素在当前页面中嵌入另一个页面，其使用语法如下：

```
<iframe allow="" height="" name="" referrerpolicy="" sandbox="" src="" srcdoc=""
 width=""></iframe>
```

<iframe>元素的属性介绍如下：

allow 属性 - 因涉及安全的考虑，由本属性指定允许使用浏览器的哪些功能，其值格式为："功能名称 值[;功能名称 值;……]"，其中的“值”为如下几种：

*：本页面及嵌套页面都允许使用此功能，而不管是否同源。

'self'：本页面及嵌套页面都允许使用此功能，只要是同源。

'src'：与<iframe>元素 src 属性的值同源的页面允许使用此功能。

'none'：本页面及嵌套页面禁止使用此功能。

<origin(s)>：用空格分开的若干源，此功能只允许这些源使用。

下面是一些常用的浏览器的功能名称：

autoplay：多媒体自动播放功能，battery：电池状态 API 调用，camera：摄像功能，display-capture：捕获屏幕内容，document-domain：设置 document.domain，encrypted-media：加密媒体扩展（EME）

API 调用，fullscreen：进入全屏，geolocation：地理位置接口，gyroscope：通过陀螺仪接口收集有关设备方向的信息，microphone：使用话筒，oversized-images：下载和现实大图片，payment：使用支付请求 API，picture-in-picture：画中画视频播放，sync-xhr：发起同步请求 XHR，usb：使用 WebUSB API，wake-lock：使用唤醒锁 API 以指示设备不要进入电源节省模式，screen-wake-lock：使用屏幕唤醒锁 API 以指示设备不要关机或关屏。看例子：

```
<iframe allow="fullscreen *; microphone 'none'; geolocation 'none'" src="regist
er.html"></iframe>

<iframe allow="usb https://google-developers.appspot.com" src="product.html">
</iframe>
```

例子中的第一条代码表明能进入全屏模式，但是不能使用语音输入，也不定使用地理位置定位功能，第一条代码表明当浏览谷歌的开发网站时允许使用 USB 设备。

sandbox 属性 - 给页面施加额外的限制，其值可为空（施加所有限制）或者以空格分开的若干许可项，可用的许可项有：allow-forms：允许提交表单，allow-modals：允许弹出模态窗口，allow-orientation-lock：锁住横/竖屏切换，allow-popups：弹窗，allow-scripts：执行脚本，等等。

src 属性 - 嵌入页面的 URL，如果值为"about:blank"，则嵌入空白页面。

srcdoc 属性 - 内嵌的 HTML 文本，本属性会覆盖 src 属性，如果浏览器不支持 srcdoc 属性，则继续使用 src 属性。

width、height 属性 - 定义嵌入窗口大小，默认值是宽 300，高 150。单位为 CSS 像素。

referrerpolicy 属性请参考<script>元素的同名属性。

看例子：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>嵌入页面</title>

</head>

<body>
    <p>我是父页面</p>
    <iframe sandbox="allow-popups" src="https://www.baidu.com" srcdoc="<h1> 我 是
子页面</h1>"></iframe><br>
    <iframe allow="fullscreen *" src="https://www.baidu.com" width="700px" heig
ht="300px"></iframe>
</body>

</html>
```

浏览的效果如图 4.179。

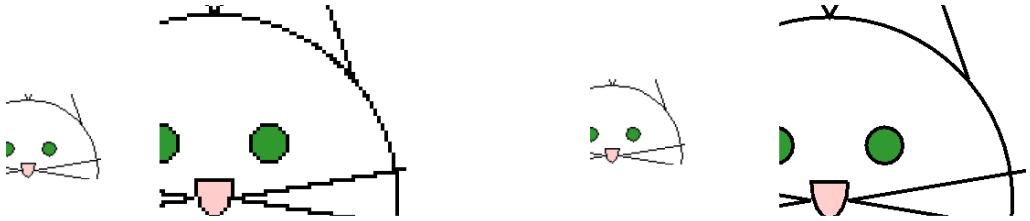
本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 4.179 <iframe>元素用法举例

4.10.7 SVG 矢量图

SVG 为 Scalable Vector Graphics 三个单词的首字母缩写，意为可缩放矢量图，不同于位图文件（png，jpeg 等）那样直接存放每个像素的颜色，矢量图文件中保存的是绘制图形的指令，浏览时必须临时执行这些指令来生成图形，比如工程制图软件 cad 就是处理矢量图。与传统的位图相比，矢量图有一个显著的优点，那就是能无极缩放而不会导致图片失真，参见图 4.180 所示。



位图放大　　矢量图放大

图 4.180 矢量图能无极缩放

SVG 几乎与 CSS 同时诞生，但是由于之前的页面主要以文字为主，所以 CSS 发展远超 SVG，而现在进入图文并茂的页面时代，SVG 发展后劲十足。

SVG 采用 XML 语言编写绘制图形的指令，指令实际上就是各种绘制图形的标签（为了区分，本书统一规定：HTML5 中诸如 的各种标记称为元素，而 SVG 中诸如<circle>的各种标记称为标签），例如下面的指令就是绘制一个半径为 50 个单位的红色圆形，即采用了<circle>标签。

```
<circle cx="60" cy="60" r="50" fill="red" />
```

所有的绘图指令必须包裹在<svg>元素内，即语法如下：

```
<svg[ 属性="" …]>
    一系列的绘图指令
</svg>
```

<svg>元素允许嵌套，它作为 HTML5 的元素，可以直接放在 HTML5 的网页文件中，与普通的元素（如 ）一样使用。绘图指令也可以单独保存在以 svg 为扩展名的文件中（如 myGraphic.svg），然后直接在浏览器中打开这个文件就可以看到图形。但在网站开发领域，更普遍的用法把 svg 图形嵌入到页面中，即通过以下几种方式之一把外部的 svg 文件嵌入到页面。

1、作为图形资源插入，最终被浏览器渲染为位图：

```
<img src="myGraphic.svg" …… />
```

或者

```
div {
    background-image: url("myGraphic.svg");
    background-size: 100% 100%;
}
```

2、作为矢量图嵌入，可无极缩放并可以使用 JS 在两者之间交互：

```
<object data="myGraphic.svg" type="image/svg+xml" ……>
    <mark>你的浏览器不支持 object 元素</mark>
    <img src="myGraphic.png" ……/>
</object>
```

或者

```
<iframe src="myGraphic.svg" …… >浏览器不支持 iframe 元素</iframe>
```

为了发挥 SVG 矢量图的优势，尽量采用第 2 种方法。下面的例子绘制一个黑色圆形：

```
<!DOCTYPE html>
<html>

<head>
    <title>SVG</title>
</head>

<body>
    <svg>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
        <circle cx="100" cy="70" r="70" />
    </svg>
</body>

</html>
```

4.10.7.1 坐标系

由于 SVG 矢量图一般要嵌入到页面中，矢量图本身存在宽/高尺寸（假设为$x_1 \times y_1$），即画布大小，嵌在页面中的<svg>元素也存在宽/高尺寸（假如为$x_2 \times y_2$）——称为 SVG 视口（viewport），而且这两个尺寸一般不相等（$x_1 \neq x_2$、$y_1 \neq y_2$），甚至宽/高比率也不等（$\frac{x_1}{y_1} \neq \frac{x_2}{y_2}$），这时需要完成两个操作：首先按要求缩放 SVG 画布（画布上有矢量图），然后按照某种对齐方式显示在 SVG 视口中，参见图 4.181。



图 4.181 SVG 画布和视口

在实际的项目中，一般由专门的人设计矢量图，矢量图画布通常为正方形，网页由另外的人开发，往往一个矢量图会嵌入到多个页面中，而且不同的页面可能嵌入的区域大小都不同。到底如何缩放又如何对齐，由<svg>元素的一个属性确定。先看<svg>元素的完整使用语法：

```
<svg
    width=""
    height=""
    viewbox=""
    preserveAspectRatio=""
    version="1.1"
    baseProfile=""
    xml:lang=""
    x=""
    y=""
    xmlns="http://www.w3.org/2000/svg">
      一系列的 SVG 标签
</svg>
```

<svg>元素的全部属性都是可选的，即允许省略。下面逐个介绍这些属性。

width、height 属性 - 定义页面上的视口大小，即$x_2 \times y_2$，width=x_2为视口宽度，height=y_2为视口高度，比如 width="80%" height="200px"，也可以通过样式定义。如果没有指定大小也省略了 viewbox 属性，那么默认尺寸就是 300px*150px。如果省略 width、height 属性，但存在 viewbox 属性，那么默认就是 100%。

viewbox 属性 - 定义 SVG 画布大小，取值的格式为四个用逗号或空格分开的数字，即：viewbox="min-x, min-y, width, height"或 viewbox="min-x min-y width height"。其中 min-x 和 min-y

一般都为 0，其具体的含义后面再做详细介绍，width 和 height 为 SVG 画布的宽度和高度，必须大于 0。由于为矢量图，允许无极缩放，所以 width 和 height 不需要单位，否则是非法的，作为绘画作者，你就认为是一个“用户单位”就可以了。画布遵循与<canvas>元素画布一样的坐标系，参见图 4.182 所示。

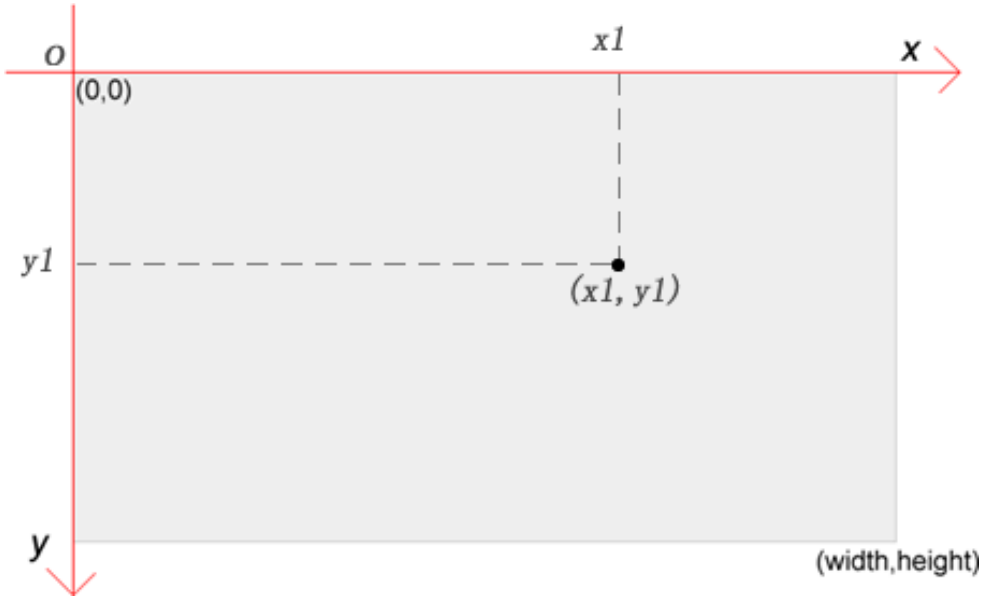


图 4.182 <svg>元素的画布

如果省略 viewbox 属性，画布尺寸就是 SVG 视口尺寸，即<svg>元素的属性 width 和 height 定义的尺寸。例如定义 800px*600px 的画布： viewbox="0, 0, 800, 600" 或者 viewbox="0 0 800 600"。

preserveAspectRatio 属性 - 定义缩放和对齐方式，它取值为"none"或者格式为"水平对齐方式垂直对齐方式 [缩放方式]"。水平对齐方式有三种：xMin 左对齐、xMid 居中对齐、xMax 右对齐，垂直对齐方式也有三种：YMin 上对其、YMid 居中对齐、YMax 下对其，三种水平对齐方式和三种垂直对齐方式组合在一起共有九种对齐方式，加上"none"，一共十种，见表 4.20。

表 4.20 对其方式

对齐方式	**说明**
none	缩放矢量图与页面上的区域一样大，即$x_1 = x_2$、$y_1 = y_2$，这样图形可能会变形
xMinYMin	左上角对齐
xMinYMid	水平方向左对齐，垂直方向居中对齐
xMinYMax	左下角对齐
xMidYMin	水平方居中对齐，垂直方向上对齐

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

xMidYMid	水平和垂直方向都居中对齐
xMidYMax	水平方向居中对齐，垂直方向下对齐
xMaxYMin	右上角对齐
xMaxYMid	水平方向右对齐，垂直方向居中对齐
xMaxYMax	右下角对齐

对于这十种值的理解，请参见图 4.183，黑色虚线框表示页面上拟嵌入 SVG 的区域。

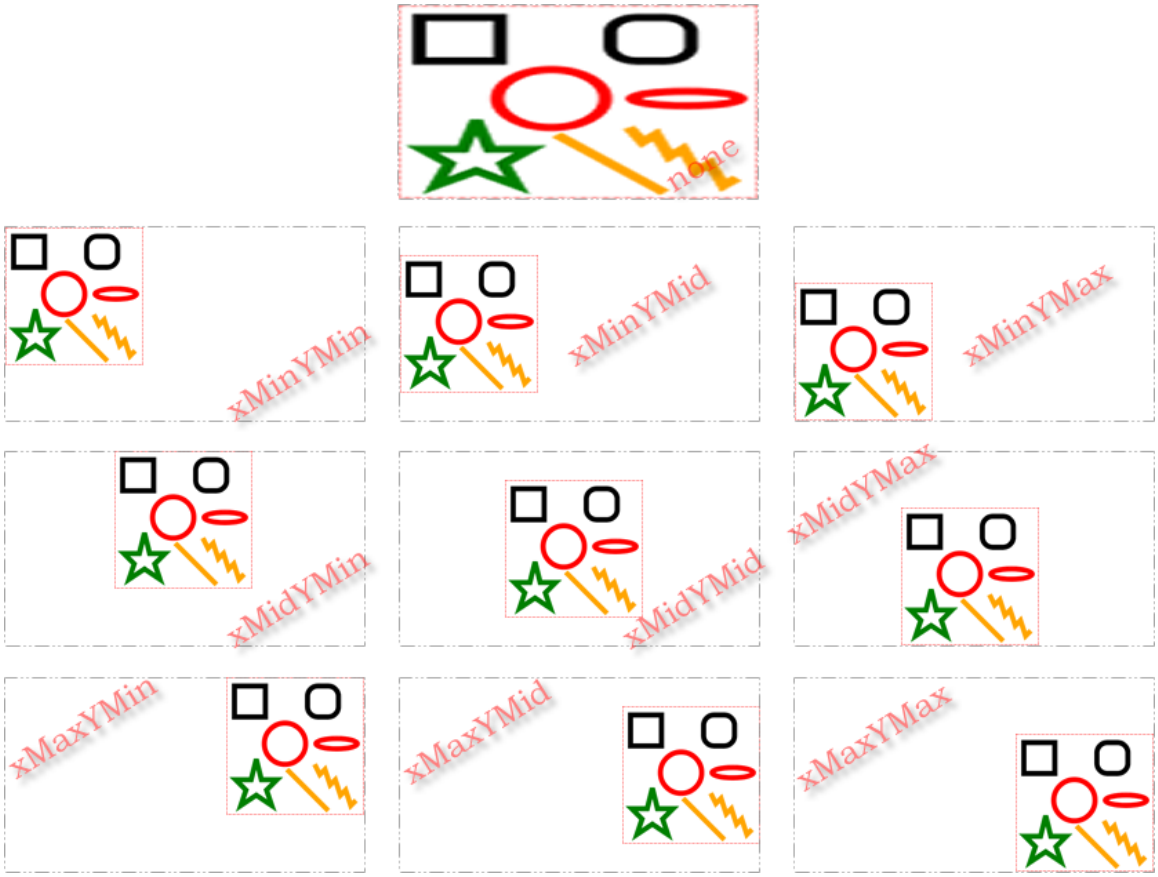


图 4.183 对齐方式

等等，我要像图 4.184 那样摆放，那又怎么搞？

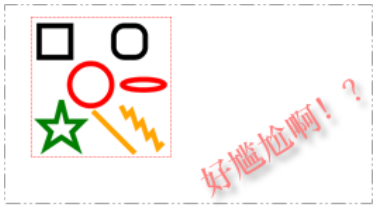


图 4.184 任意位置摆放

这时就要 viewbox 属性的值中的 min-x 和 min-y 出场了，之前都假定它们为 0，如果不为 0 就能满足各种“摆放”要求了。先看一个例子：

```
<svg width="400px" height="180px" viewbox="0 0 160 160" preserveAspectRatio="xM
inYMin meet">
    <circle cx="80" cy="80" r="80" fill="red" />
</svg>
```

显然是左对齐，浏览效果见图 4.185 左侧部分：



图 4.185 调正对其方式

现在修改上面代码中的 viewbox 属性的值为"-20 0 160 160"，浏览效果见图 4.185 中间部分，再改为"20 0 160 160"，效果见图 4.185 右侧部分。viewbox="-20 0 160 160"为什么会产生图 4.185 中间部分的效果呢？解释参考图 4.186。



图 4.186 min-x 和 min-y 的作用

尽管 SVG 画布没变（即图 4.186 中的黑色坐标），继续在这个画布上绘制图形，但是显示在页面区域时，对齐方式的参考边线变了，即上图中的蓝色边框成了对齐参考边线，所以水平方向左对齐的效果就是图 4.186 的中间部分，右对齐的效果就是图 4.186 的右侧部分。因此可以说 min-x 和 min-y 定义了参考边线左上角的坐标。

实操练习

修改 viewbox 属性的值中的第 2 个数字不为 0，看看效果如何（提醒：参考图 4.187）。

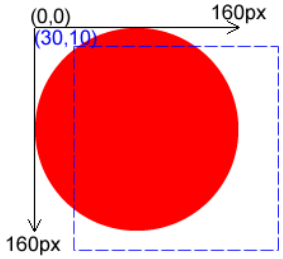


图 4.187 min-x 和 min-y 大于 0

preserveAspectRatio 属性定义的缩放方式有两种：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

"meet" - 同比率缩放 SVG 矢量图，整个图形显示在页面上的区域中且图形保持最大，这时可能要结合对齐方式来摆放图形。

"slice" - 同比率缩放 SVG 矢量图，页面上的区域被图形全部覆盖且图形保持最小，这时可能会截掉部分图形，参见图 4.188 左侧部分，具体会截掉哪部分，要看对齐方式，比如 preserveAspectRatio="xMinYMin slice"就会截掉下侧或右侧部分。

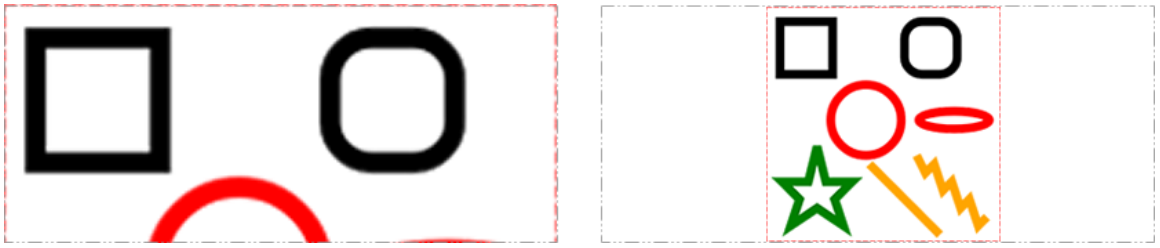


图 4.188 缩放方式

如果省略 preserveAspectRatio 属性，默认值就是"xMidYMid meet"，参见图 4.188 右侧部分。

由此可知，如果 SVG 画布尺寸等于页面上的区域尺寸，那么矢量图大小保持不变，如果 SVG 画布尺寸小于页面区域尺寸，那么矢量图会放大，否则就会缩小。

version 属性 - 定义 SVG 的版本，目前全功能的 SVG 版本是 1.1，虽然 2018 年推出了 2.0 版本，但是目前主流的浏览器支持不是很好，所以建议大家暂时还是用 1.1 版，1.1 版→2.0 版类似于 CSS→CSS3 的转变，变化比较大，可以肯定再过几年，2.0 版会被大量使用。

baseProfile 属性 - 定义渲染图形时需要的 SVG 功能特征，它可取四个值之一：“none”-最小功能特征，“full”-全功能（适用于电脑），“basic”-轻量级（适用于平板），“tiny”-更轻量级（适用于手机）。注意：在 SVG 2.0 版本中，version 和 baseProfile 属性被淘汰。

xml:lang 属性 - 定义语言，可取值"en-US"、"zh-CN"等。

x、y 属性 - 定义本<svg>元素在父<svg>元素中的位置，即本画布左上角在父画布中的坐标。如果没有父<svg>元素，x 和 y 属性无效。

xmlns 属性 - 定义 xml 名字空间的 URIs，目前可固定为"http://www.w3.org/2000/svg"。

看一个例子：

```
    <h3>下面是你购买的 CAD 设计图</h3>
    <p>1、预览效果：</p>
<svg width="500px" height="180px" viewbox="0 0 160 160" version="1.1"
preserveAspectRatio="xMidYMid meet"
xmlns="http://www.w3.org/2000/svg">
        <circle cx="80" cy="80" r="80" fill="red" />
    </svg>
    <p>2、请点击任何一个站点下载：
        <a href="downlaod1.html">下载 1</a>
        <a href="download2.html">下载 2</a>
        <a href="download3.html">下载 3</a>
    </p>
```

浏览下效果如图 4.189。



图 4.189 svg 举例

实操练习

修改本例中的 viewbox 和 preserveAspectRatio 属性的值，看看效果如何变化。

4.10.7.2 SVG 标签

SVG 绘制矢量图就是采用各种 SVG 标签完成的，而且这些标签必须作为<svg>元素的子标签。SVG 标签类似 HTML5 的元素，语法也差不多，类似于：

```
<标签 属性="" 属性="" … >内容</标签>
<标签 属性="" 属性="" … />
```

不过同一个 SVG 标签既可以采用第 1 种语法，也可以采用第 2 种语法。

标签的通用属性有 id（定义唯一标识符）、lang（定义语言）、tabindex（定义焦点顺序号，为非负整数）、class（归属的组名）、style（定义内嵌样式，值格式“"样式名: 值; 样式名: 值; … ; ""”）、<样式属性>（样式名可单独作为标签的属性，如 fill="red"）、<事件属性>（定义事件处理程序，如 onclick="handler()"）。不过不建议采用事件属性，而采用对象的 addEventListener() 方法给标签对象添加事件处理程序。

SVG 标签共有 61 个（淘汰的以及还在实验阶段的除外），具体参见“《HTML5&CSS3 开发大起底》.xlsx ”文件中的“SVG 标签”工作表。对于表中的难点这里再作介绍。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

4.10.7.3 绘制图形

表 4.21 中的标签用来绘制图形。

图 4.21 绘制图形的标签

标签名	简要说明
<circle>	绘制圆
<ellipse>	绘制椭圆
<line>	画线
<polygon>	绘制多边形
<polyline>	绘制折线
<rect>	绘制矩形
<image>	嵌入式外部图形（位图或矢量图）
<path>	通过路径绘制图形，所有的基本图形（如圆、线）都可用<path>绘制
<use>	复制图形

下面我从最简单的画线开始介绍这些标签。

1、 画线-<line>

<line>标签的语法如下：

```
<line x1="" y1="" x2="" y2=""[ pathLength="" 通用属性]/>
```

在(x1,y1)和(x2,y2)两点之间画一根直线，看例子：

```
<svg width="800" height="600" version="1.1" xmlns="http://www.w3.org/2000/svg">
    <line x1=" 10" y1="10" x2="400" y2="200" stroke="black" />
</svg>
```

本例在(10,10)和(400,200)两点之间画了一根黑色线段。另外线宽属性 stroke-width、虚线属性 stroke-dasharray 和透明度属性 stroke-opacity 也是常用的属性，前面已经讲过，这些属性名也是样式名称。因此下面两条语句绘制完全相同的线段：

```
<line x1="30" y1="20" x2="400" y2="20" stroke="red" stroke-width="3" stroke-
opacity="0.4"/>
<line x1="30" y1="20" x2="400" y2="20" style="stroke:red; stroke-
width: 3; stroke-opacity: 0.4;"/>
```

作者可以采用<line>标签的 pathLength 属性指定路径的总长度，假设浏览器计算(x1,y1)和(x2,y2)两点之间的长度为 Z，而作者却把 Z 当着 pathLength，这样作者拟绘制的长度 L（作者看来就是这个长度）的线段，浏览器实际上绘制的长度则是$L \times \frac{Z}{pathLength}$，而且凡是涉及的长度都会作相应调整。在现实生活当中我们经常遇到这种事情：对于一根 1 米长的木棍，在没有尺子测量的情况下，甲认为木棍有 1.2 米长，乙认为有 1.3 米长，可能丙认为不足 1 米，那么我们可以这样推断：以后甲说他手里有一根 1 米的绳子，实际上绳子的长度应该为$1 \times \frac{1}{1.2} = 0.833$米。看例子：

```
<svg viewBox="0 0 1000 600" xmlns="http://www.w3.org/2000/svg">
    <style>
        line {
            fill: none;
            stroke: black;
            stroke-width: 2;
            stroke-dasharray: 20; /*长度为 20 个用户单位*/
        }
    </style>
    <line x1="0" y1="0" x2="200" y2="0" />
    <line x1="0" y1="30" x2="200" y2="30" pathLength="250" />
    <line x1="0" y1="60" x2="200" y2="60" pathLength="150" />
    <line x1="0" y1="90" x2="200" y2="90" pathLength="100" />
    <line x1="0" y1="120" x2="200" y2="120" pathLength="50" />
</svg>
```

样式 stroke-dasharray 定义了线段和空白等长（均是 20 个用户单位）的虚线，然后连续用五个<line>标签绘制 5 条 200 个用户单位长的虚线，但是后面四个<line>标签作者采用 pathLength 属性主观上认为虚线的长度变了。拿最后一条虚线来说，200 的真实长度他认为只有 50，因此他眼中的 1 就代表真实的 4，所以他看到的虚线的线段和空白 20（见“stroke-dasharray: 20;”语句），实际上最终绘制成了 80。参考图 4.190 的左侧，右侧是解析。

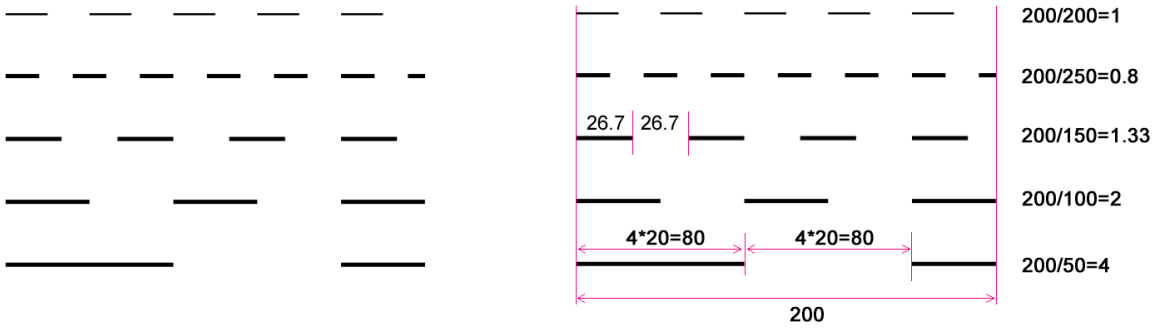


图 4.190 <line>标签 pathLenth 属性的作用

2、 画圆-<circle>

<circle>标签的语法如下：

```
<circle cx="" cy="" r=""[ pathLength="" 通用属性]/>
```

以(cx,cy)为圆心 r 为半径画一个圆，pathLength 属性与<line>标签的同名属性一样。看例子：

```
    <circle cx="100" cy="400" r="50" />
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    <circle cx="210" cy="400" r="50" fill="red"/>
    <circle cx="320" cy="400" r="50" fill="#0F0" fill-opacity=".5"/>
    <circle cx="430" cy="400" r="50" fill="transparent" stroke="#00F" />
```

属性 fill 定义填充色，fill-opacity 定义填充色的透明度。浏览效果如图 4.191。

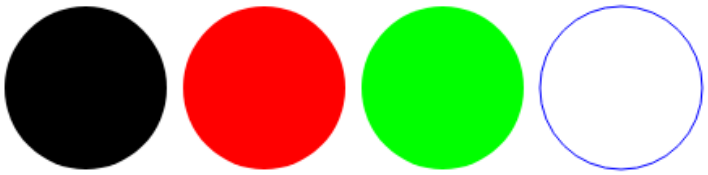


图 4.191 <circle>标签举例

3、 画矩形-<rect>

<rect>标签的语法如下：

```
    <rect x="" y="" width="" height="" rx="" ry=""[ pathLength="" 通用属性]/>
```

各个属性的含义见图 4.192 所示，如果省略 x、y、rx 和 ry，默认都为 0。rx 为水平方向的圆角半径，如果省略，就等于 ry（如果存在），ry 为垂直方向的圆角半径，如果省略，就等于 rx（如果存在）。

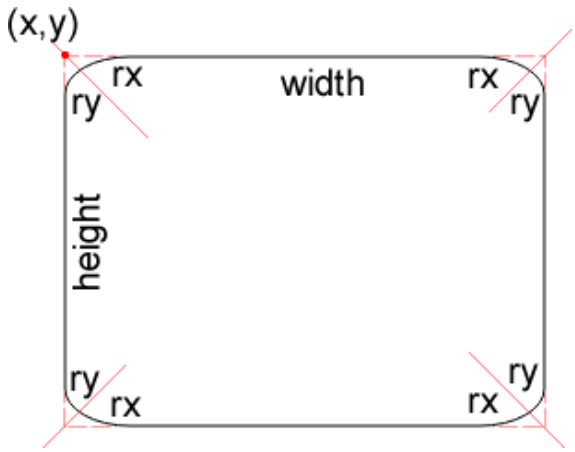


图 4.192 <rect>标签的属性说明

看例子：

```
  <rect width="100" height="100" />
  <rect x="110" y="110" width="100" height="100" rx="15" />
  <rect x="220" y="220" width="100" height="100" rx="15" ry="10" fill="red" />
  <rect x="330" y="330" width="100" height="100" fill="none" stroke="green" />
```

4、 画椭圆-<ellipse>

<ellipse>标签的语法如下：

```
<ellipse cx="" cy="" rx="" ry=""[ pathLength="" 通用属性]/>
```

各个属性的含义见图 4.193 所示，如果省略 cx、cy，默认都为 0。rx 为水平方向的椭圆半径，如果省略，就等于 ry（如果存在），ry 为垂直方向的椭圆半径，如果省略，就等于 rx（如果存在）。

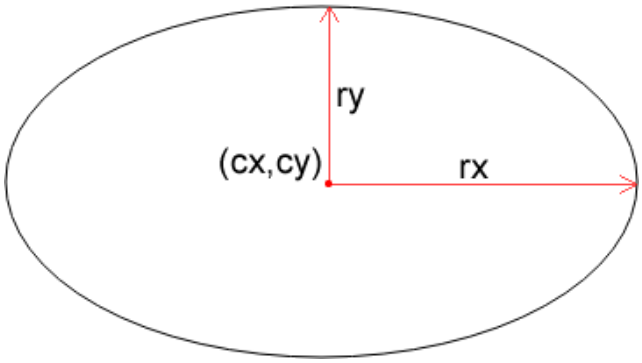


图 4.193 <ellipse>标签的属性说明

看例子：

```
<ellipse cx="220" cy="200" rx="180" ry="100" />
```

5、 画多边形-<polygon>

<polygon>标签的语法如下：

```
<polygon points="" [ pathLength="" 通用属性]/>
```

points 属性定义多边形顶点的坐标，坐标之间用空格分开，每个坐标的格式为 x,y，看例子：

```
<polygon points="0,100 50,25 50,75 100,0" fill="red" />
<polygon points="200,0 150,100 280,100" fill="none" stroke="blue" />
<polygon points="100,10 40,180 190,60 10,60 160,180" style="fill:lime;stroke:pu
rple;stroke-width:5;fill-rule:evenodd;" />
```

浏览效果如图 4.194 所示（为了便于理解，附加了每点的坐标）：

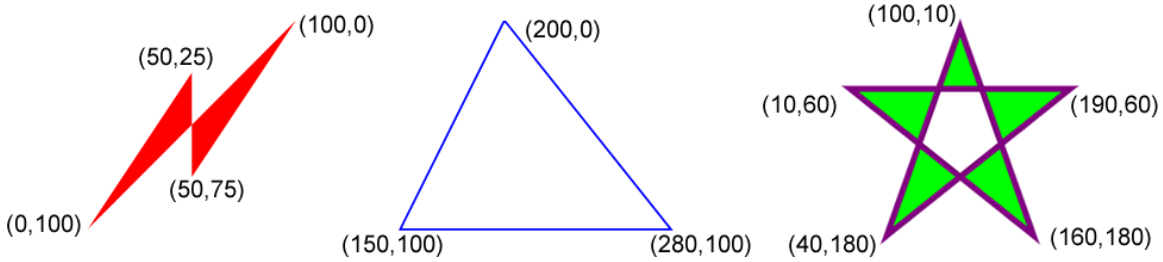


图 4.194 <polygon>标签举例

6、 画折线-<polyline>

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

<polyline>标签的语法如下：

```
<polyline points="" [pathLength="" 通用属性]/>
```

points 属性定义转折点的坐标，坐标之间用空格分开，每个坐标的格式为 x,y。

7、 嵌入外部图形-<image>

<image>标签可以引入外部的位图图形（不支持 gif）或者矢量图形，它的语法如下：

```
<image x="" y="" width="" height="" href="" preserveAspectRatio="" [ 通用属性]/>
```

(x,y)、width 和 height 属性定义了画布上一个矩形区域的左上角的坐标、宽度和高度，引入的图形只能显示在此区域之内，href 为外部图形的 URL，preserveAspectRatio 的含义与<svg>标签的同名属性相同，它的默认值为"xMidYMid meet"，即同比率缩放并居中对齐。x 和 y 属性允许省略，默认为 0，而 width 和 height 不能省略。参见图 4.195 所示。

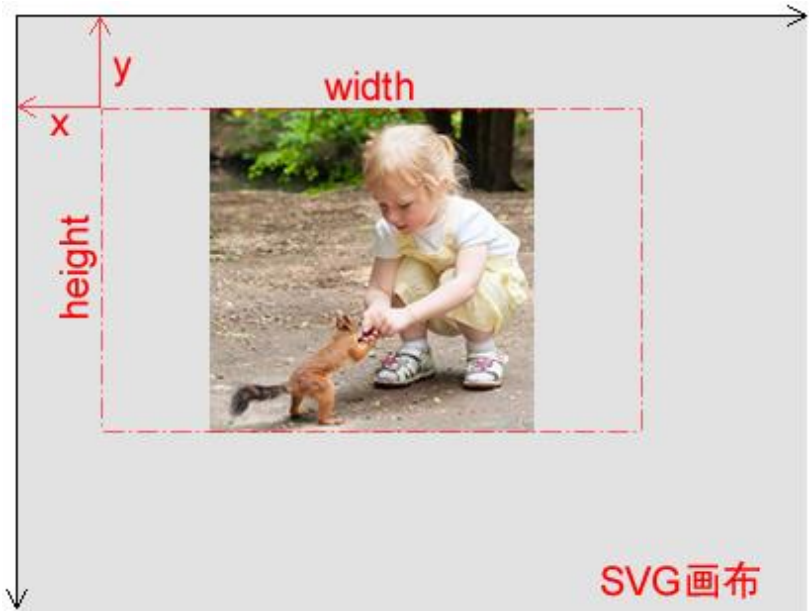


图 4.195 <image>标签的属性说明

看例子：

```
<image x="50" y="50" width="200" height="120" href="brother.jpg" />
```

8、 绘制路径-<path>

<path>是通用的绘图标签，除了直线、矩形、圆形、多边形等外，还可以绘制任何图形，它的语法如下：

```
<path d="" [pathLength=" 通用属性]/>
```

d 属性定义路径，它的值是以空格分开的一系列路径命令，每条路径命令的格式为“命令 参数”，其中“命令”用一个字母表示（区分大小写），参数是一系列的数字，如“M 10,20”。命令介绍具体请参见“《HTML5&CSS3 开发大起底》.xlsx ”的“SVG 之 path”工作表。

定位命令 M 或 m，相当于画家在画布上落笔的那个点，此后他可以连续绘制图形直到拿起笔来。语法“M (x, y)+”中的“+”表示坐存在一个或多个坐标，展开这个语法就是 M x_1,y_1 x_2,y_2 x_3,y_3 …，即先定位到(x_1,y_1)坐标，然后画线到(x_2,y_2)，再画线到(x_3,y_3)，以此类推，最后那个坐标又成为当前坐标。下面的例子先定位到坐标(10,10)，然后连续画了几根直线，最后坐标(200,80)成为当前坐标：

```
<path d="M 10,10 100,10 100,80 200,80" fill="none" stroke="black" />
```

浏览效果见图 4.196 左侧部分。

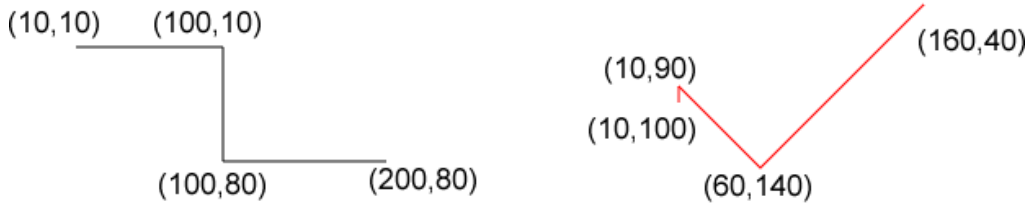


图 4.196 <path>标签用法举例

m 命令是增量定位，假设当前坐标为(x_0, y_0)，那么 m dx,dy 最终定位到坐标(x_1+dx, y_1+dy)

画直线的命令有 L、l、H、h、V、v 共六个，L 命令的语法为“L (x, y)+”，展开来就是 L x_1,y_1 x_2,y_2 x_3,y_3 …，即从当前坐标画一根线到(x_1,y_1)，再画线到(x_2,y_2)，再画到(x_3,y_3)，以此类推，最后那个坐标又成为当前坐标。看例子：

```
<path d="M 10,100 L 10,90 60,140 160,40" fill="none" stroke="red" />
```

浏览效果见图 4.196 右侧部分。H 命令是画水平线，例如 H 100 就是在(x_0, y_0)和(100, y_0)两点之间画一根线，最后(100, y_0)成为了当前坐标。

C、c、S、s 四个命令用于绘制三次贝塞尔曲线，一根三次贝塞尔曲线需要四个坐标：起点坐标、终点坐标、控制起点坐标和控制终点坐标，控制点位置不同会导致曲线形状不同，见图 4.197 左侧部分所示。所以“C (x_1,y_1, x_2,y_2, x,y)+”表示当前坐标(x_0, y_0)就是起点坐标、(x, y)为终点坐标、(x_1,y_1)为控制起点坐标，(x_2,y_2)控制终点坐标。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

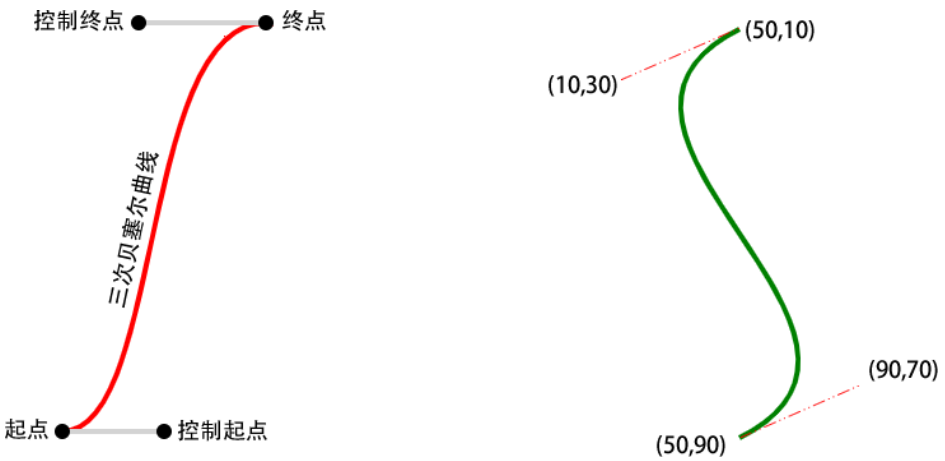


图 4.197 三次贝塞尔曲线

下面的例子，浏览效果参见图 4.197 右侧部分：

```
<path fill="none" stroke="green" d="M 50,90 C 90,70 10,30 50,10" />
```

命令 A 和 a 用来绘制椭圆弧，语法“A (rx ry angle large-arc-flag sweep-flag x,y)+”，各个参数的解释参见图 4.198，其中(x_0, y_0)表示当前坐标。参数 angle 表示弧线以椭圆形中心为转轴旋转的角度，可以这样理解：先绘出弧线，然后再旋转。最后(x,y)成为当前坐标。

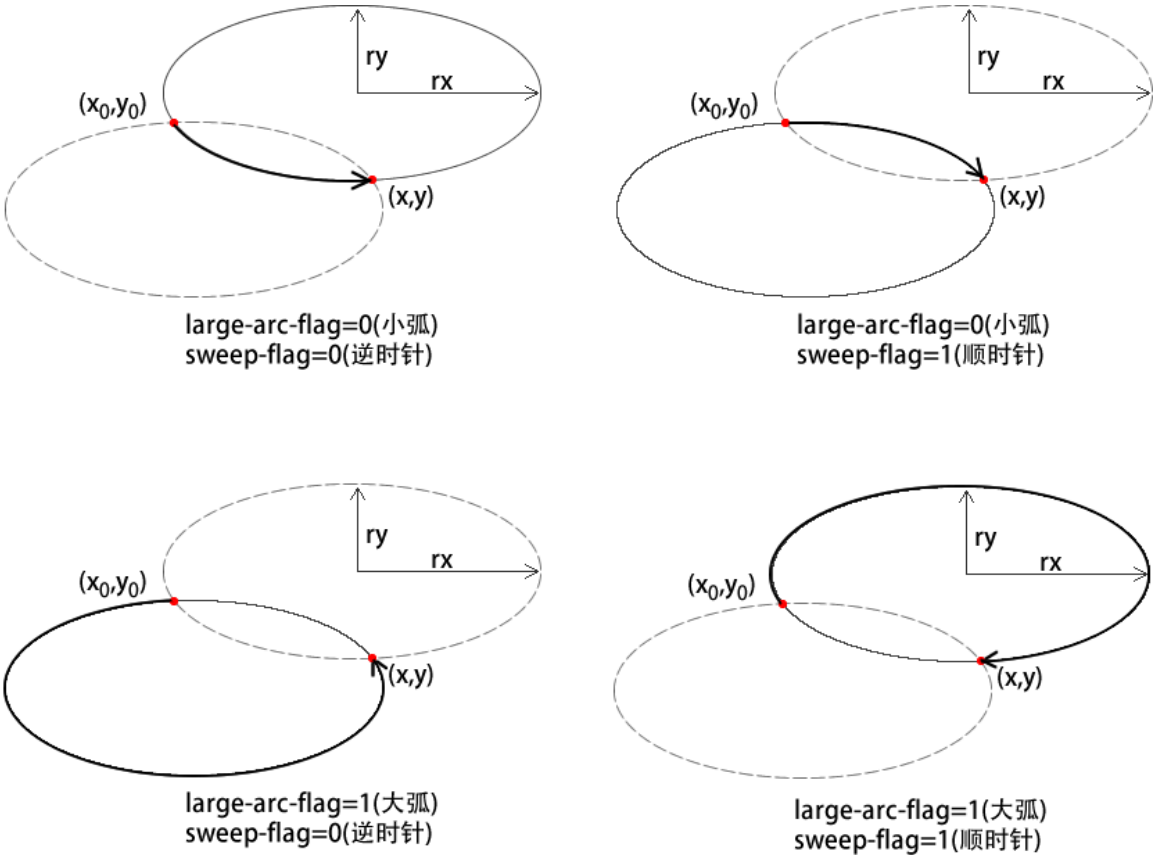


图 4.198 绘制椭圆弧命令的参数说明

看例子：

```
<path fill="none" stroke-width="2" stroke="red" d="M 120,82 A 130 60 0 1 1
 260,120" />
```

浏览的效果见图 4.199 左侧部分。

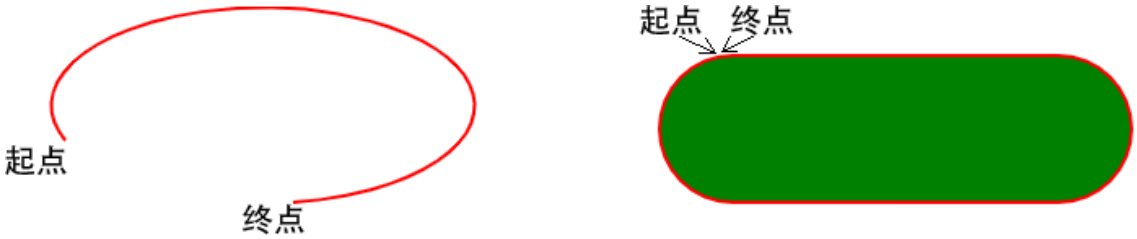


图 4.199 绘制椭圆弧举例

命令 Z 和 z 一样，从路径的终点画一根直线到起点，从而封闭路径。

再看一个综合例子，浏览的效果见图 4.199 右侧部分。

```
   <path fill="green" stroke-width="2" stroke="red"
        d="M 100,10
           A 45 45 0 1 0 100,100
           H 300
           A 45 45 0 1 0 300,10
           Z"
   />
```

9、 复制图形-<use>

<use>复制同一文档内的<svg>元素或者<svg>内的一个标签或片段到<use>标签所在的位置，一个文档内允许存在多个<svg>元素，复制允许跨<svg>元素。它的语法如下：

```
<use href="" x="" y="" width="" height="" [ 通用属性]/>
```

href 指明待复制的标签或 svg 片段，x、y 属性定义<use>标签左上角的坐标，被复制对象内的坐标以 x 和 y 为坐标原点，width 和 height 定义宽度和高度，除非被复制的对象支持 viewbox 属性（如<svg>或<symbol>标签），否则 width 和 height 无效。当被复制对象支持 viewbox 和 preserveAspectRatio 属性时（如果省略也是存在默认值），那么<use>标签的 width 和 height 属性就定义了一个区域，如果尺寸不同于被复制对象 viewbox 定义的尺寸，就再根据 preserveAspectRatio 属性做缩放和对齐。另外<use>标签上的通用属性不能覆盖被复制对象上的同名通用属性。看例子：

```
    <circle id="myCircle" cx="50" cy="50" r="40" stroke="blue" />
    <use href="#myCircle" x="100" y="50" fill="blue" stroke="red" />
```

注意：<use>标签上的 stroke="red"无效，因为被复制对象上的同名属性优先。浏览效果如图 4.200。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

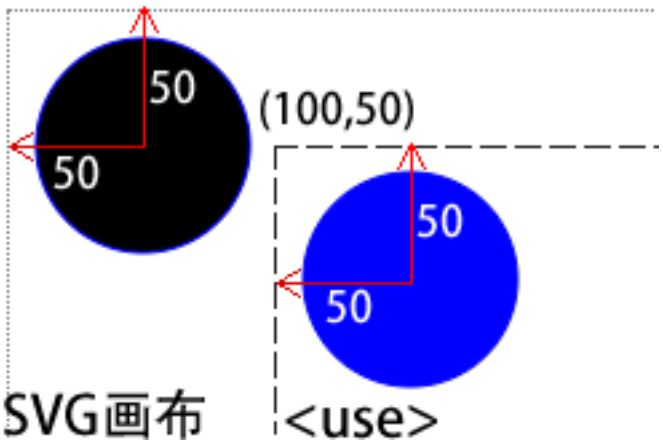


图 4.200 <use>标签用法举例

4.10.7.4 绘制文本

在<svg>标签，只有被<text>标签包裹的文本才会被显示出来，其他地方的文本都不会显示。表 4.22 列出了三个绘制文本的标签。

图 4.22 绘制文本的标签

标签名	简要说明
<text>	绘制文本
<textPath>	绘制路径文本。必须作为<text>的子标签
<tspan>	作为<text>的子标签，包含的文本可单独附加样式

1、 绘制文本-<text>

<text>标签的语法如下：

```
<text x="" y="" dx="" dy="" rotate="" lengthAdjust="" textLength="" [ 通用属性]>
文本内容</text>
```

(x,y)为文本基线起点的坐标（文本基线的理解请参考“§4.5.4.3 绘画上下文”小节的上下文参数 textBaseline），默认值都为 0；dx 和 dy 为相对于(x,y)的水平和垂直方向的偏移量；rotate 属性旋转每一个字形（以每个字形基线上的起点为转轴），单位为度，顺时针为正角度，逆时针为负角度；textLength 属性规定文本显示出来的宽度，单位为像素；当本文实际宽度与 textLength 不相等时，由 lengthAdjust 属性控制字符缩放，可取的值有"spacing"(默认值)和"spacingAndGlyphs"，"spacing"不缩放字符而缩放字符间的空间，这样有可能会导致字符部分重叠，而"spacingAndGlyphs"缩放字符。看例子：

```
<text x="10" y="20">先要学会做好一个人！right?</text>
<text x="10" y="20" rotate="45">先要学会做好一个人！right?</text>
```

```
<text x="10" y="60" dx="20" dy="5" textLength="130">先要学会做好一个人！right?</text>
<text x="10" y="80" dx="20" textLength="130" lengthAdjust="spacingAndGlyphs">先要学会做好一个人！right?</text>
<text x="10" y="100" textLength="300">先要学会做好一个人！right?</text>
<text x="10" y="120" textLength="300" lengthAdjust="spacingAndGlyphs">先要学会做好一个人！right?</text>
```

浏览的效果如图 4.201。

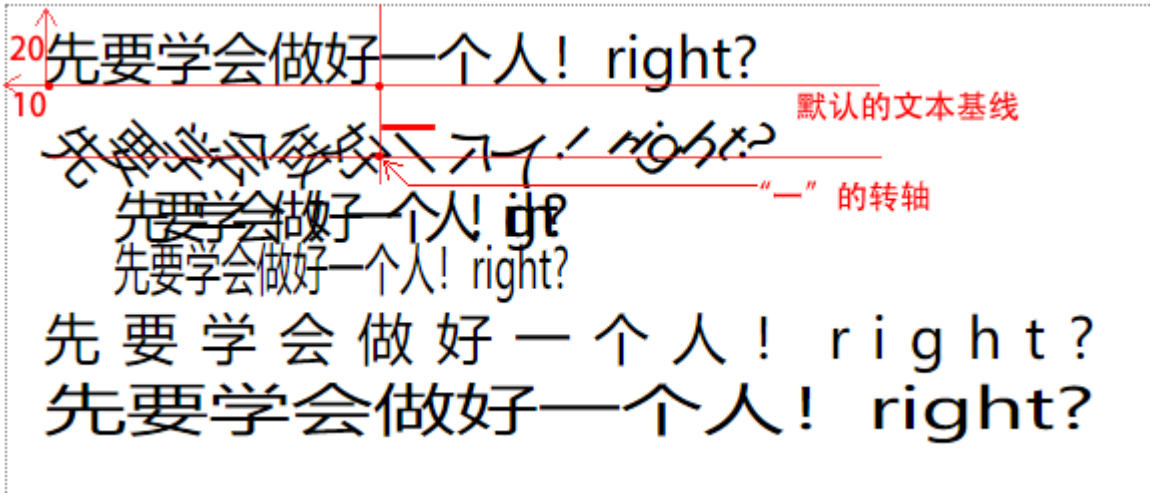


图 4.201 <text>标签用法举例

<text>标签可通过样式 dominant-baseline 修改文本基线的位置，具体参考 SVG 的样式。

2、 路径文本-<textPath>

<textPath>标签实现文本沿着路径显示，它必须要被<text>标签包裹，其语法如下：

```
<textPath href="" lengthAdjust="" method="" path="" side="" spacing=""
 startOffset="" textLength="" [ 通用属性]>文本内容</textPath>
```

href 引用路径对象或基本图形对象，path 定义一条路径，path 属性会覆盖 href 属性，method 属性目前还在实验阶段，大部分浏览器不支持，它用来定义字形沿着路径渲染的方法，可取值"align"（默认值）或"stretch"，前者只对字形做旋转和移动变换，后者还会做拉伸、切变，从而会导致字形变形；side 属性定义文本显示在路径上的哪一侧，取值"left"（左侧）或"right"（右侧），面朝路径方向来区分左右侧，不过只在 firefox 浏览上测试有效（相当于调转了路径方向后的左侧）；spacing 属性定义如何确定沿路径呈现的字形之间的间隔，可取值"auto"（默认值）或"exact"；startOffset 属性定义文本起点偏移路径头部多远，可以用百分数或长度单位；lengthAdjust 和 textLength 属性与<text>标签的同名属性含义相同。看例子：

```
<path id="cirl" fill="none" stroke="red" d="M100,200 A 95 95 0 1 1 280,200" />
<path id="cirl2" fill="none" stroke="green" stroke-width="3"
          d="M350,150 A 60 60 0 1 0 470,150 A 60 60 0 1 1 590,150" />

<text font-family="Brush Script MT" style="font-size: 40pt;">
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    <textPath href="#cirl" method="align" side="left" spacing="exact">Do you lo
ve English?
    </textPath>
</text>
<text style="font-size:30px;stroke:blue; fill: blue" >
<textPath href="#cirl2" textLength="44%" startOffset="20"> 你 蠢 我 哪 敢 聪 明 ！
</textPath>
</text>
```

浏览的效果如图 4.202。

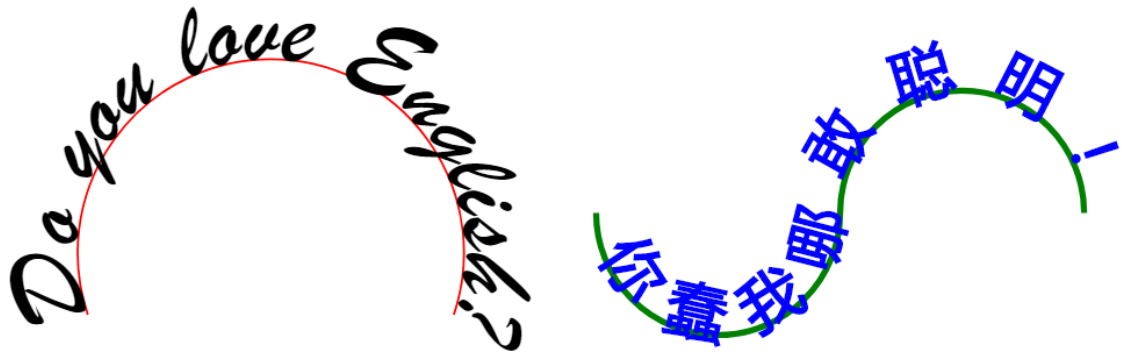


图 4.202 <textPath>标签用法举例

3、 绘制子文本-<tspan>

用<tspan>标签包裹<text>标签文本中的一段子文本，以便运用不同的样式，<tspan>允许嵌套。它的语法如下：

```
<tspan x="" y="" dx="" dy="" rotate="" lengthAdjust="" textLength="" [ 通用属性]>
文本内容</tspan>
```

属性与<text>标签完全一样。看例子：

```
<svg width="800" height="600" version="1.1" xmlns="http://www.w3.org/2000/svg">
     <style>
         text  { font: italic 20px serif; }
         tspan { font: bold 30px sans-serif; fill: red; }
     </style>

     <text x="10" y="30">
          You are <tspan>not</tspan> a banana!
     </text>
</svg>
```

4、 提示文本-<title>

<title>标签给标签兑现提供提示信息，当鼠标停靠其上时显示出来。它的语法如下：

```
                <title href="" href="" [ 通用属性]>文本内容</title>
```

(x,y)为文本基线起点的坐标(文本基线的理解请参考<canvas>画布元素的上下文参数 textBaseline)

4.10.7.5 容器标签

容器标签用来包裹若干标签对象，以便作为“组”共同施行某种操作。表 4.23 列出了这些标签。

表 4.23 容器标签

标签名	简要说明
<a>	定义超链接
<defs>	包裹若干图形对象，这些对象不会直接渲染，此后引用它时才会显示（如采用<use>标签引用某个对象）
<g>	组合若干图形标签，组合在一起的标签作为整体同时做转换
<marker>	定义用于在给定的<path>、<line>、<polyline>或<polygon>标签上绘制箭头或多点标记的图形
<mask>	蒙版
<pattern>	创建允许平铺的图形对象
<switch>	按顺序判断子标签的 requiredFeatures、requiredExtensions 和 systemLanguage 属性值，然后渲染第一个判断结果为真的那个标签
<symbol>	定义图形模板对象，以后采用<use>标签对之实例化。与<defs>不同，<defs>包裹的每一个对象可以单独引用

1、 超链接-<a>

<a>标签类似于 HTML5 的<a>元素，与某个资源（比如页面、文件、邮件地址或者文档内的锚等）建立超链接，由于它是容器标签，允许包裹任何标签对象。它的语法如下：

```
<a href="" hreflang="" target="" type="" [ 通用属性]>内容</a>
```

href 属性定义资源的 URL；hreflang 定义资源的语言，可取值"en-US"、"zh-CN"等；target 属性定义打开资源的方式，可取值"_self"（默认值）、"_parent"、"_top"、"_blank"和<XML-Name>，除了<XML-Name>外，其他值都与 HTML5 的<a>元素相同，<XML-Name>指定一个浏览上下文

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

的名称，如果此名已经存在，就重新使用它并替换现有内容，如果不存在，则创建它（与'_blank'相同，但现在有了一个名称）。type 属性与 HTML5 的<a>元素的同名属性含义一样。看例子：

```
<a href="goto.svg" target="_black">
      <circle cx="50" cy="50" r="47"/>
      <text x="17" y="120" fill="blue">详细页面</text>
</a>
```

2、 声明图形对象-<defs>

<defs>标签声明若干图形对象，供以后做实例化渲染显示，在 SVG 中，采用<use>标签实例化，尽量不要在声明时附加样式，而在实例化时添加样式。有点类似 JS 中的函数定义，一次定义可多次调用。它的语法如下：

```
<defs[ 通用属性]>内容</defs>
```

<defs>没有自己的专用属性，看例子：

```
<defs>
     <circle id="wheel" cx="130" cy="69" r="40" />
     <path id="road" d="M 0,100 h 100 A 44 44 0 0 0 160,100 h 100"/>
</defs>
<use x="100" y="100" href="#wheel" fill="red" />
<use x="100" y="100" href="#road" fill="none" stroke="black" stroke-width="6" />
```

浏览的效果如图 4.203。



图 4.203 <def>标签用法举例

3、 组合图形对象-<g>

<g>标签用于组合 SVG 标签，从而使得这些标签作为整体一起施行变换或被<use>引用。<g>标签类似于 HTML5 的 元素，施加在<g>上的样式会对所有子标签产生影响。它的语法如下：

```
<g[ 通用属性]>内容</g>
```

<g>没有自己的专用属性，看例子：

```
<g id="sun" fill="red" stroke="blue" stroke-width="6">
     <circle id="wheel" stroke="none" cx="105" cy="60" r="40" />
     <path id="road" d="M 0,100 h 80 M 130,100 h 80 M 55,114 h 100"/>
</g>
<use x="300" y="0" href="#sun" />
```

浏览的效果如图 4.204，其中右侧是采用<use>标签复制<g>的。



图 4.204 <g>标签用法举例

4、 标记图形-<marker>

<marker>定义一个图形，用来给<path>、<line>、<polyline>和<polygon>标签的顶点添加箭头或连接点，这些标签只要添加 marker-start、marker-mid 和 marker-end 属性即可。<marker>的语法如下：

```
<marker markerHeight="" markerUnits="" markerWidth="" orient="" preservAspectRatio="" refX="" refY="" viewBox="" [ 通用属性]>内容</marker>
```

在 viewBox 定义的坐标系中绘制标记图形，然后根据 preservAspectRatio 定义的对齐和缩放方式在 markerWidth╳markerHeight 矩形内渲染，最后根据(refX,refY)坐标附着在路径上，请参考示意图 4.205 进一步理解这些属性。

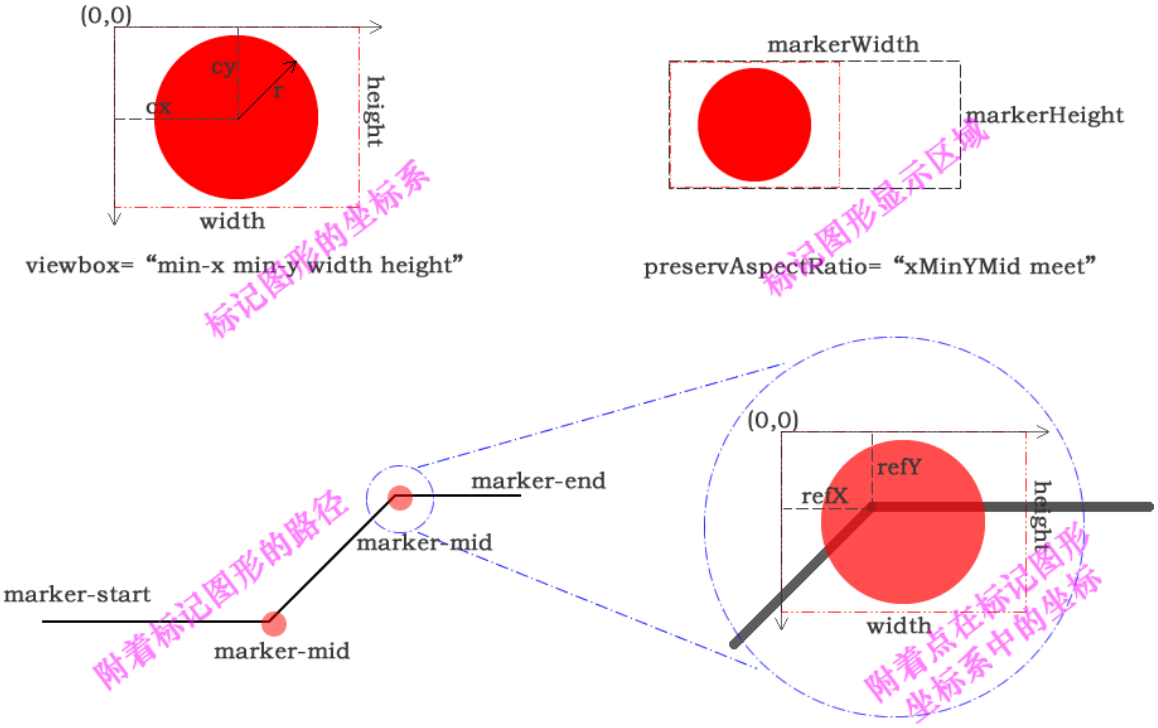


图 4.205 <marker>标签各属性说明

markerUnits 属性定义 markerWidth 和 markerHeight 属性的长度单位，可取值"userSpaceOnUse"或"strokeWidth"，前者使得标记大小固定不变，而后者会根据路径线宽（stroke-width 样式定义）同比率缩放，比如线宽为 3，那么标记图形扩大三倍，参考示意图 4.206。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

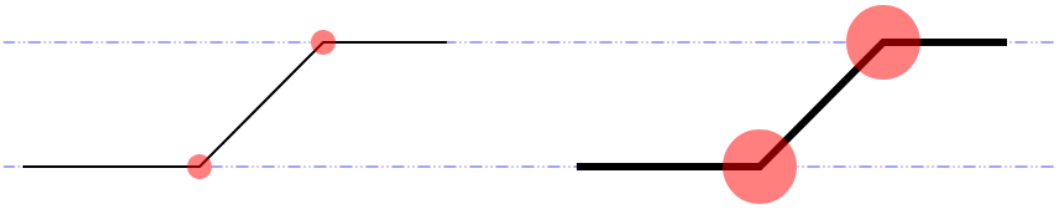


图 4.206 <marker>标签的 markerUnits 属性

当标记图形附着在路径上时，由 orient 属性定义如何旋转标记图形。它可取下面的几个值之一：

"auto" - 标记图形坐标系中的 x 轴的方向与路径图形上附着点处的路径方向一致，路径图形中间折线处的附着点的路径方向就是“反射面”（如图 4.207 左侧和中间部分的蓝色箭头）。

"auto-start-reverse " - 标记图形附着在路径头部时，标记图形翻转 180°，其他与"auto"相同。

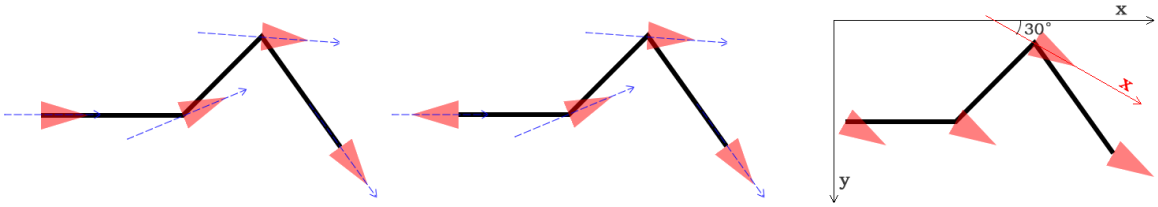


图 4.207 <marker>标签的 orient 属性

"<angle>" – 标记图形坐标系中的 x 轴与路径图形所在的坐标系中的 x 轴之间的夹角，以度为单位，正角度表示顺时针旋转标记图形，否则逆时针旋转。如 orient="30"，等价于"30deg"，参见图 4.207 右侧部分，红色 x 表示标记图形坐标系中的 x 轴，黑色 x 表示路径图形坐标系中的 x 轴，两根轴之间的夹角就是 30 度。

看例子：

```
<defs>
     <!-- 定义箭头 -->
     <marker id="arrow" viewBox="0 0 20 20" refX="10" refY="10" markerWidth="12
" markerHeight="12"  orient="auto-start-reverse">
          <path d="M 0 0 L 20 10 L 0 20 z" />
     </marker>

     <!-- 定义连接点 -->
     <marker id="dot" viewBox="0 0 20 20" refX="10" refY="10" markerWidth="10"
markerHeight="10">
           <circle cx="10" cy="10" r="10" fill="red" />
     </marker>
</defs>

<!-- 绘制坐标 -->
<polyline points="20,20 20,180 300,180" fill="none" stroke="black" marker-
start="url(#arrow)"  marker-end="url(#arrow)" />

<!-- 绘制路径 -->
<polyline points="30,160 58,100 86,120 114,60 142,80 170,30" fill="none" stroke
="#333" marker-start="url(#dot)"  marker-mid="url(#dot)" marker-
end="url(#dot)" />
```

浏览效果如图 4.208。

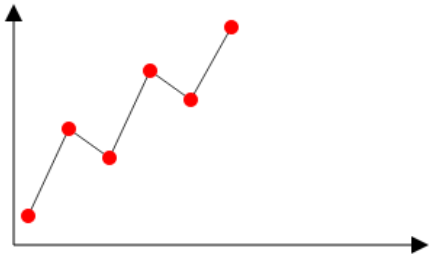


图 4.208 <marker>标签综合举例

5、 蒙版-<mask>

用过图像处理软件的人都知道，蒙版附着在一个图层上，蒙版中的白色部分盖住的图层完全可见，而黑色部分遮盖的图层或超出蒙版尺寸之外的图层则完全消失（全透明），灰色部分盖住的图层是半透明的，参见图 4.209 所示，图层 1 是绿色背景，图层 2 是一幅照片，然后给图层 2 加上左侧部分的蒙版，这样黑色的“心”遮盖的图层 2 部分完全变成全透明，因为形成一个洞，直接看到了背景。

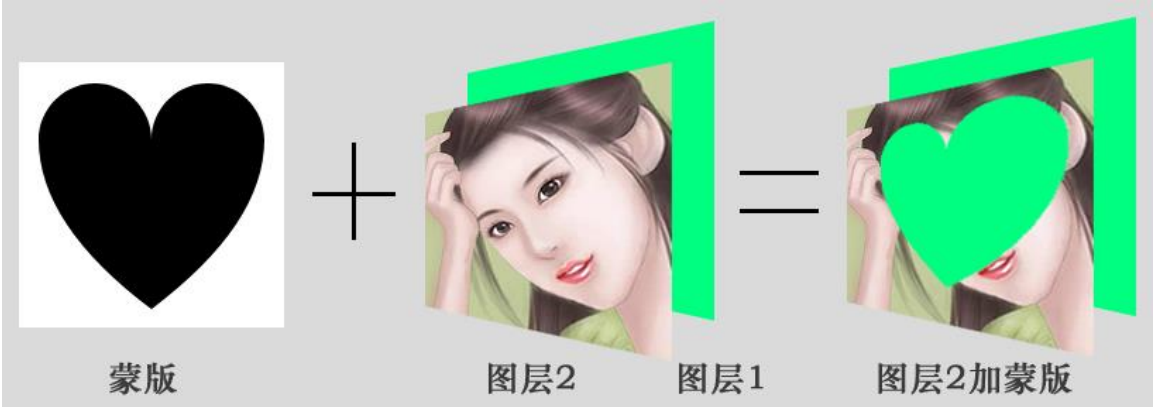


图 4.209 蒙版

在 SVG 中，首先采用<mask>标签定义蒙版，然后把蒙版附着在其他图形上。<mask>标签的语法如下：

```
<mask height="" maskContentUnits="" maskUnits="" x="" y="" width=""[ 通用属性]>内容</mask>
```

width、height 属性分别定义蒙版的宽度和高度，默认值都是 120%，(x,y)为蒙版左上角的坐标。默认值为(-10%,-10%)，maskUnits 属性定义 x、y、width 和 height 属性属于哪个坐标系，可以取值：

"userSpaceOnUse" - 当前坐标系，也就是创建蒙版时的坐标系，这时<mask>标签的 x、y、width 和 height 属性与其他普通图形标签（如<rect>）的同名属性一样，都在同一个坐标系中。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

"objectBoundingBox"（默认值） - 要应用蒙版的图形对象的边界框的比率或百分比，可以将边界框视为与将<mask>的内容绑定到 viewbox="0 0 1 1"，因此百分数与等值小数效果一样，如 x="20%"等价于 x="0.2"，height= "60%"表示蒙版的高度为图形对象高度的 60%，但是 y="20"就远远超出了 viewbox="0 0 1 1"。要牢记：实际上是在 viewbox="0 0 1 1"内绘制，然后渲染到应用蒙版的图形对象的边界框内。

图形对象的边界框就是包裹图形对象及其后代的最小矩形，它的左上角坐标为(minx,miny)，右下角坐标为(maxx,maxy)。计算边界框时不包括裁剪、遮罩、滤镜效果、不透明度和笔划宽度。对于弯曲的形状，边界框会包围形状的所有部分，而不仅仅是端点。对于文本，每个字形均被视为单独的图形标签，计算假定所有字形都占据整个字形单元，例如，对于水平文本，计算时假定每个字形都垂直延伸到字体的完整上升和下降值，请看示意图 4.210。

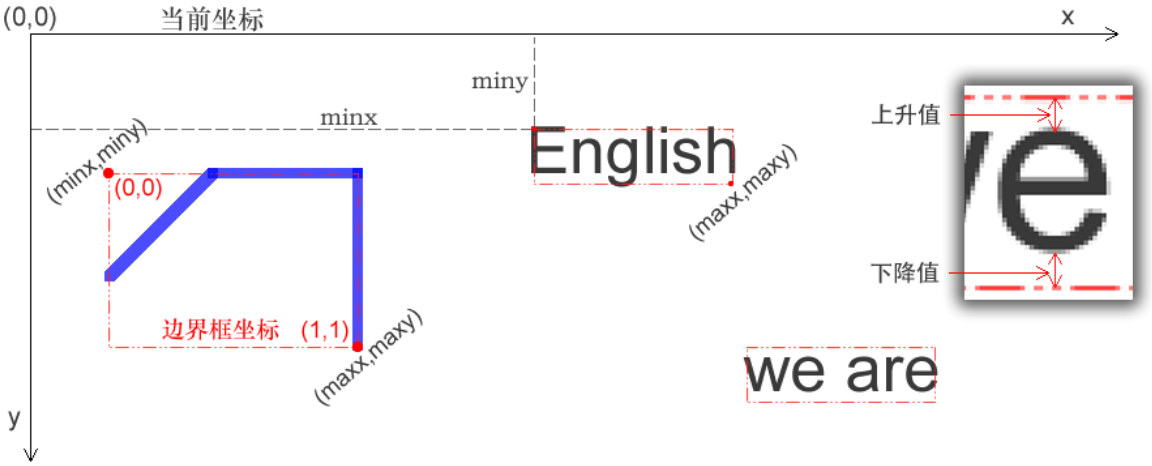


图 4.210 图形对象边界框

<mask>标签的 maskContentUnits 属性定义蒙版内容所使用的坐标系，它的取值与 maskUnits 属性相同。但是当取值"objectBoundingBox"时，蒙版内容中的百分数还是与取值"userSpaceOnUse"时的效果一样，所以尽量采用小数，比如 x="0.5"表示水平方向的中间位置，y="1"落在下边界，而 y="2"已经超出底部了。新手很容易犯一个错误：假如要应用蒙版的图形对象的边界框高度为 100px，这时他给蒙版内的一个圆心纵坐标设为 cy="50"，以为刚好在图形对象的中间，这是错误的，中间应该是 cy="0.5"。另外 maskContentUnits 的默认值为"userSpaceOnUse"。

看例子：

```
<mask id="Mask1" maskUnits="userSpaceOnUse" maskContentUnits="userSpaceOnUse">
<path d="M30,105 A60,60,0,0,1,150,105 A60,60,0,0,1,270,105 Q270,195,150,285 Q30
,195,30,105 Z" fill="white" />
</mask>
<image mask="url(#Mask1)" href="spot.jpg" y="30" width="450" />
```

本例利用图 4.211 中的左侧部分，加上蒙版产生右侧的效果。

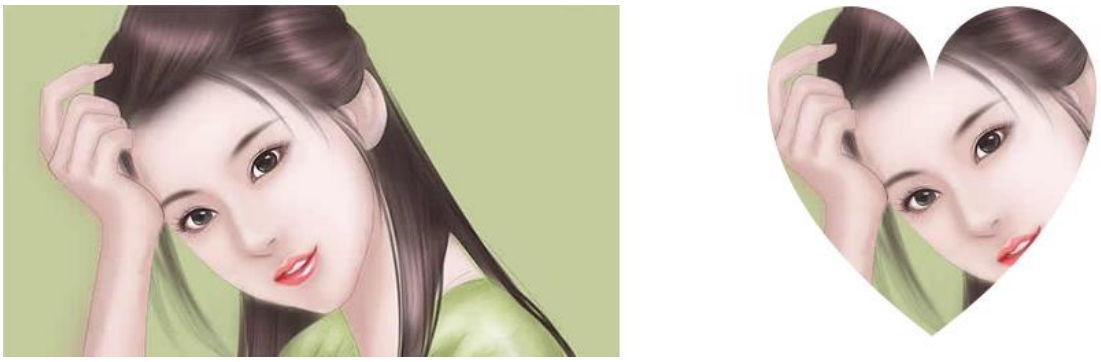


图 4.211 <mask>标签用法举例

再看例子：

```
<mask id="myMask" x="0" y="0" width="1" height="1" maskContentUnits="objectBoun
dingBox">
       <circle cx="0.5" cy="0.5" r="0.4" fill="white" />
</mask>
<image mask="url(#myMask)" href="mask.jpg" />
```

由于 maskUnits 的默认值就是"objectBoundingBox"，所以代码中省略了。浏览的效果如图 4.212，即把左侧的图形应用一个圆形蒙版，从而产生右侧的效果。

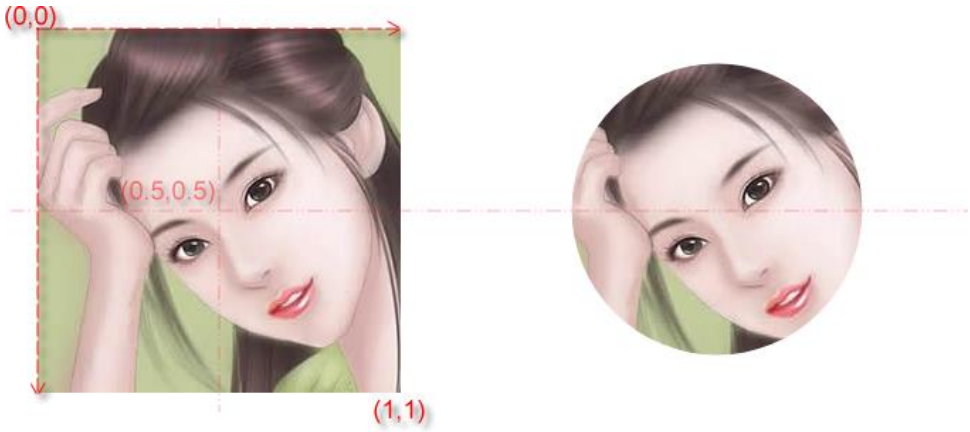


图 4.212 圆形蒙版

6、 图案-<pattern>

<pattern>标签用于产生一个图案对象，以后可以用来平铺一个区域，其他标签通过 stroke 或 fill 属性引用此图案，其使用语法如下：

```
<pattern height=""    href=""    patternContentUnits=""    patternTransform=""
patternUnits="" preserveAspectRatio="" viewBox="" width="" x="" y="" [ 通用属性]>
内容</pattern>
```

width、height 属性定义图案的宽和高，x、y 定义左上角的坐标；href 属性引用同一文档中的另一个<pattern>标签图案，本<pattern>标签不存在的属性和对应的值将从被引用的图案标签继承过来，另外，如果本<pattern>没有子标签，那么被引用的<pattern>标签中的子标签也会继承过

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

来（如果它有的话），而且这种继承是可传递的，即 A 继承 B，B 又继承 C，则 A 间接继承 C 的属性和子标签；patternContentUnits 属性定义<pattern>标签内容的坐标系，默认值为"userSpaceOnUse"，如果同时存在 viewBox 属性，则本属性失效，patternContentUnits 的含义与<mask>标签的 maskContentUnits 属性相同；patternUnits 属性定义<pattern>标签的 x、y、width 和 heigth 的坐标系，默认值为"objectBoundingBox"，本属性的含义与<mask>标签的 maskUnits 属性相同；patternTransform 属性包含额外的变换函数，这些函数在映射到目标坐标系时执行；preserveAspectRatio 和 viewbox 属性的含义与<svg>标签的同名属性相同。看例子：

```
<defs>
     <pattern id="star" viewBox="0,0,10,10" width="0.1" height="0.1">
       <polygon points="0,0 2,5 0,10 5,8 10,10 8,5 10,0 5,2"/>
     </pattern>
</defs>

<circle cx="120"  cy="120" r="100" fill="url(#star)" />
<circle cx="340" cy="120" r="90" fill="none" stroke-
width="20" stroke="url(#star)"/>
<rect x="480" y="20" width="430" height="200" fill="url(#star)" stroke="grey" />
```

注意<pattern>标签一些属性的默认值：x="0"、y="0"、patternUnits="objectBoundingBox"、patternContentUnits="userSpaceOnUse"、preserveAspectRatio="xMidYMid meet"、patternTransform="none"。图案宽高比为 1:1，而例子中的矩形宽高比为 2.15:1，所以每个小矩形（见图 4.213 中的红色框）的宽高比也是 2.15:1，因此图案同比率缩放并居中对齐。浏览的效果如图 4.213。

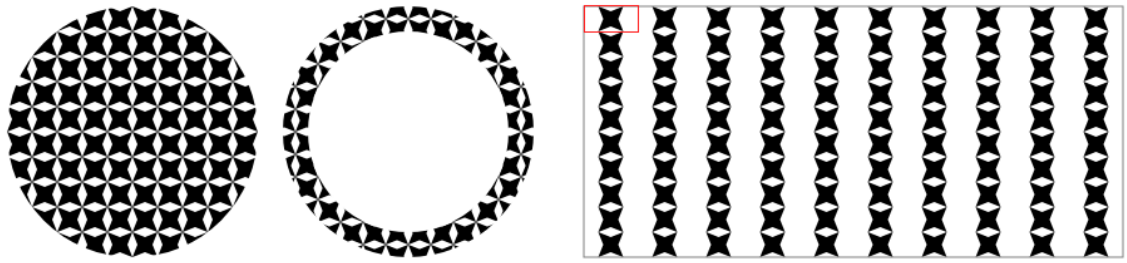


图 4.213 <pattern>标签用法举例

我们在上面的例子中增加变换，去掉两个圆形：

```
<defs>
     <pattern id="star" viewBox="0,0,10,10" width="10%" height="10%" patternTra
nsform="rotate(20) skewX(30)">
         <polygon points="0,0 2,5 0,10 5,8 10,10 8,5 10,0 5,2" />
     </pattern>
</defs>
<rect x="20" y="20" width="430" height="200" fill="url(#star)" stroke="black" />
```

变换函数有移动 translate()、旋转 rotate()、缩放 scale()、扭曲 skew()等，具体参考后续章节。浏览的效果如图 4.214。

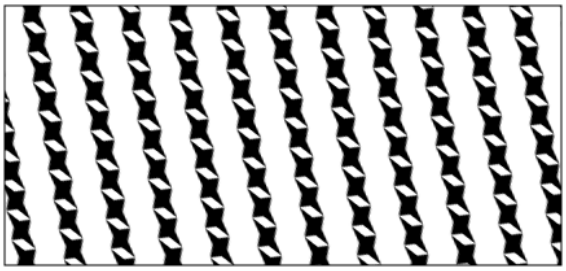


图 4.214 给<pattern>增加扭曲样式

7、 多选一-<switch>

<switch>标签在多个子标签中选择一个渲染，选择的过程是这样：从第一个子标签开始，依次判断各个子标签的 requiredExtensions、systemLanguage 属性，如果为真就渲染这个子标签连同它的后代标签（如果存在后代标签的话），剩下的子标签就全部忽略，如果某个子标签没有 requiredExtensions 和 systemLanguage 属性，那么就恒真，这样的恒真标签一般放在最后。其使用语法如下：

```
<switch [ 通用属性]>内容</switch>
```

看例子：

```
<switch>
     <text systemLanguage="ar" y="20">مرحبا</text>
     <text systemLanguage="zh-cn" y="20">又双叒叕</text>
     <text systemLanguage="en-us" y="20">Howdy!</text>
     <text systemLanguage="en" y="20">Hello!</text>
     <text systemLanguage="es" y="20">Hola!</text>
     <text systemLanguage="fr" y="20">Bonjour!</text>
     <text y="20">😆😉😆😉😆😉</text>
</switch>
```

如果 systemLanguage 属性定义的语言等于浏览器设置的语言，就为“真”，而我的 chrome 浏览器的语言设置为“简体中文”，所以本例子最终显示“又双叒叕”。requiredExtensions 属性定义一系列的浏览器支持的功能，通过这个属性我们可以根据浏览器的功能特征来渲染相应的图形对象。

8、 图形模板对象-<symbol>

<symbol>定义图形模板对象，以后可采用<use>标签对之实例化，与<defs>不同，<defs>包裹的每一个对象可以单独引用。对于文档中经常用到的结构和语义适合打包成图形模板对象，在用到的地方实例化即可，类似于 JS 语言的函数声明和调用。其使用语法如下：

```
<symbol height="" preservAspectRatio="" refX="" refY="" viewBox="" width="" x="" y="" [ 通用属性]>内容</symbol>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

width、height 定义宽和高，(x,y)为左上角的坐标，其余四个属性与<marker>标签的同名属性含义相同。看例子：

```
<symbol id="myDot" width="60" height="60" viewBox="0 0 2 2">
     <circle cx="1" cy="1" r="1" fill="red" />
</symbol>

<path d="M 30,200 A 200 170 0 0 1 430,200" fill="none" stroke="blue" stroke-
width="5" />
<!-- 多次实例化 -->
<use href="#myDot" x="0" y="170" style="opacity:0.1" />
<use href="#myDot" x="13" y="112" style="opacity:0.2" />
<use href="#myDot" x="48" y="64" style="opacity:0.3" />
<use href="#myDot" x="94" y="26" style="opacity:0.4" />
<use href="#myDot" x="149" y="4" style="opacity:0.5" />
<use href="#myDot" x="208" y="2" style="opacity:0.6" />
<use href="#myDot" x="267" y="10" style="opacity:0.7" />
<use href="#myDot" x="322" y="34" style="opacity:0.8" />
<use href="#myDot" x="366" y="76" style="opacity:0.9" />
<use href="#myDot" x="392" y="130" style="opacity:0.95" />
```

浏览的效果如图 4.215。

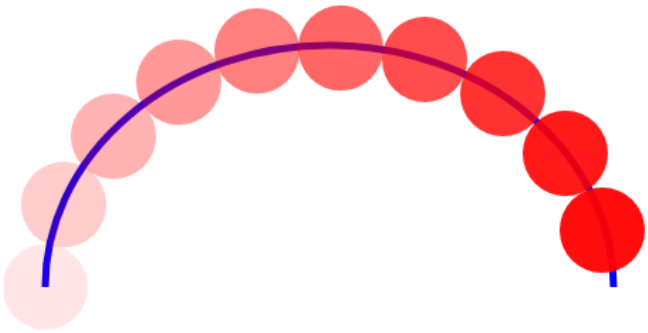


图 4.215 <symbol>标签用法举例

4.10.7.6 渐变

渐变就是若干颜色沿着直线或者放射线发生过渡。表 4.24 列出了与渐变油管的标签。

表 4.24 渐变标签

标签名	简要说明
<linearGradient>	线性渐变
<radialGradient>	径向渐变
<stop>	渐变方向上的颜色点，作为<linearGradient>、<radialGradient>的子标签

1、 线性渐变-<linearGradient>

<linearGradient>标签结合<stop>标签创建线性渐变对象，它可用于填充或描边，其使用语法如下：

```
<linearGradient gradientUnits=""     gradientTransform=""     href=""     id=""
spreadMethod="" x1="" x2="" y1="" y2="" [ 通用属性]>
    <stop offset="" stop-color="" stop-opacity="" />
    <stop offset="" stop-color="" stop-opacity="" />
    ……
</linearGradient>
```

(x1,y1)和(x2,y2)为渐变线段的端点坐标；gradientUnits 定义端点坐标所在的坐标系，取值"objectBoundingBox"（默认值）时，端点坐标在应用渐变对象的那个标签的边界框坐标系中，如果坐标值落在[0,1]范围之外，就超出了目标标签的边界框，百分数等价于等值小数，如 x2="60%"等价于 x2="0.2"，见图 4.216 左侧部分。

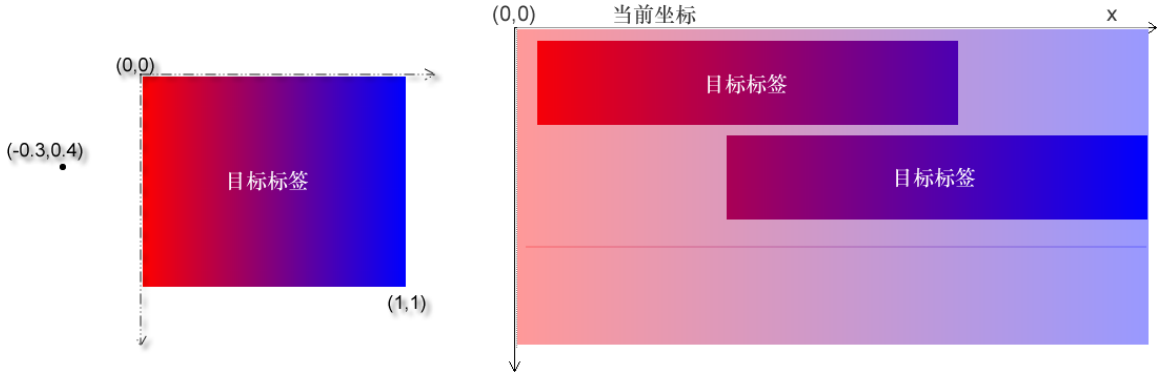


图 4.216 <linearGradient>标签用法说明

当 gradientUnits 取值"userSpaceOnUse"时，端点坐标就在当前坐标系中，但这时要特别注意，应用渐变对象的标签的位置不同，效果也不同，实际上渐变对象是作用于整个画布的，应用渐变的标签等于开了一扇窗户，透过窗户就看到了画布上的渐变，当然窗户开的位置不同，通过窗户看到画布上的小块渐变，其效果也不同，示意图参考上图右侧部分。

另外如果边线颜色要应用渐变，这时 gradientUnits 属性只能等于"userSpaceOnUse"，因为值"objectBoundingBox"要求标签的边界框具有非 0 的宽度和高度，而线是没有高度的。

应用渐变的标签超出渐变端点的区域如何处理，由<linearGradient>标签的 spreadMethod 属性规定，它可取值"pad"（默认值）、"reflect"或"repeat"，分别表示纯色、镜像和重复，具体参考示意图 4.217。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

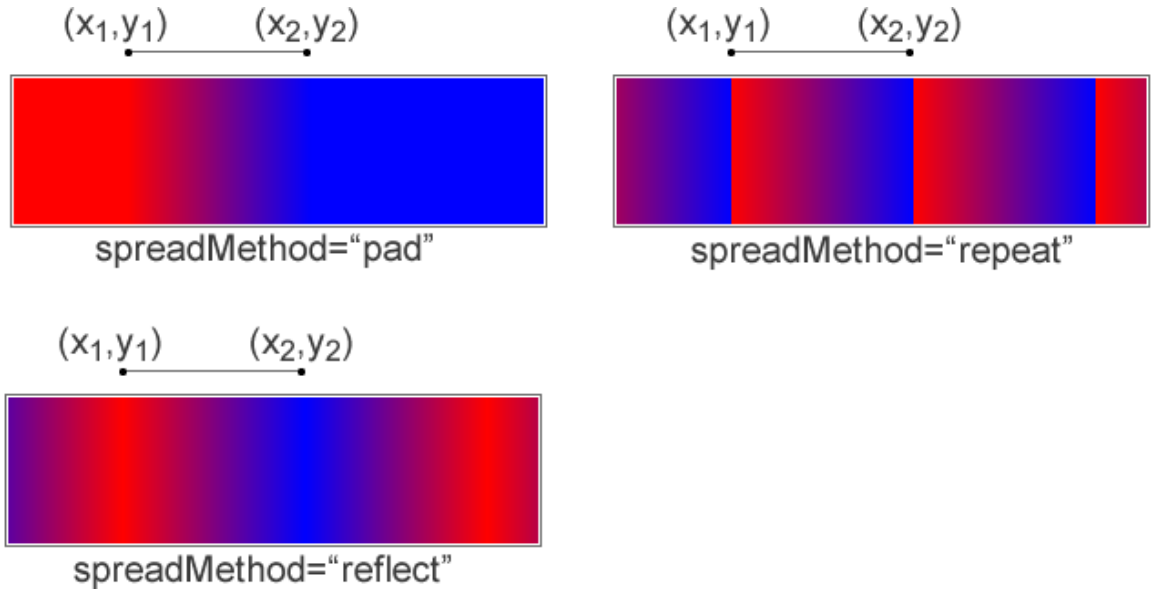


图 4.217 <linearGradient>标签的 spreadMethod 属性

href 属性继承另一个已经定义好的渐变对象（属性包括颜色点），然后重设 x1、x2、y1、y2 属性，以达到复用的目的，例如：

```
<linearGradient id="me" x1="10" y1="10" x2="100" href="#hisId" />
```

<stop>标签用来定义渐变线段上的颜色点，offset 属性取值范围是[0,1]，代表离渐变起点的距离占比，因此 0 就落在起点，0.5 落在起/止点的中间，1 落在终点，参见示意图 4.218；stop-color 属性定义颜色，stop-opacity 属性定义透明度，取值范围为[0,1]，默认为 1，不透明。注意 stop-color 和 stop-opacity 也可以作为样式使用。

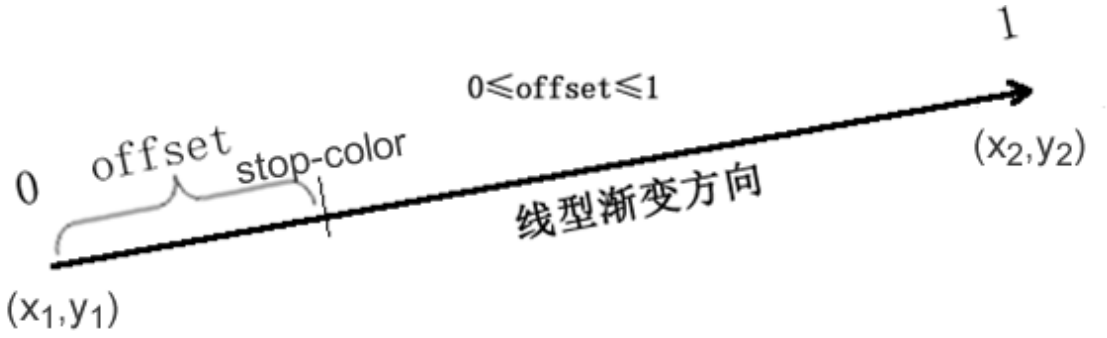


图 4.218 定义渐变的颜色点

看例子：

```
<linearGradient id="lin1" gradientUnits="objectBoundingBox">
      <stop offset="0" stop-color="red" />
      <stop offset="0.5" stop-color="green" />
      <stop offset="1" stop-color="blue" />
</linearGradient>
<linearGradient id="lin2" gradientUnits="userSpaceOnUse" href="#lin1" />

<rect x="100" y="10" width="500" height="80" fill="url(#lin1)" />
<line x1="50" y1="95" x2="650" y2="95" stroke="url(#lin1)" />
<line x1="50" y1="110" x2="650" y2="110" stroke="url(#lin2)" />
<rect x="100" y="120" width="500" height="80" fill="url(#lin2)" />
```

本例定义了两个渐变对象#lin1 和#lin2，而#lin2 继承#lin1，然后覆盖 gradientUnits 属性，最后分别应用在两个相同大小的矩形标签上，浏览效果如图 4.219。

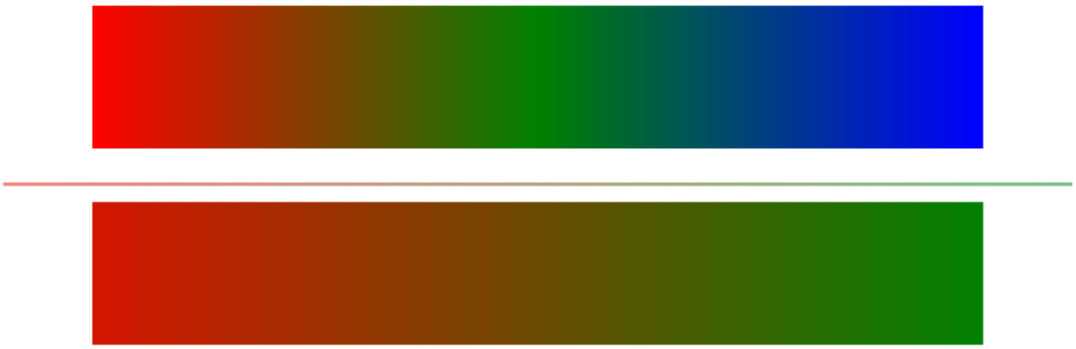


图 4.219 <linearGradient>标签用法举例

为什么只有一条线？再看一个例子：

```
<linearGradient id="Grad" x2="10%" spreadMethod="repeat">
     <stop offset="0" stop-color="black" />
     <stop offset="0.5" stop-color="black" />
     <stop offset="0.5" stop-color="white" />
     <stop offset="1" stop-color="white" />
</linearGradient>

<rect x="20" y="20" width="400" height="100" fill="url('#Grad')" />
```

浏览的效果如图 4.220。



图 4.220 <linearGradient>标签举例——斑马线

2、 径向渐变-<radialGradient>

<radialGradient>标签结合<stop>标签创建径向渐变对象，它可用于填充或描边，其使用语法如下：

```
<radialGradient cx=""     cy=""     fr=""     fx=""     fy=""     gradientUnits=""
gradientTransform="" href="" id="" r="" spreadMethod="" [ 通用属性]>
    <stop offset="" stop-color="" stop-opacity="" />
<stop offset="" stop-color="" stop-opacity="" />
……
</radialGradient>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

径向渐变的起点为一个点或者圆，终线是一个圆，大部分参数参见图 4.221 所示，其他属性与线性渐变的同名属性含义相同。默认值：cx="50%"、cy="50%"、r="50%"、fx="50%"、fy="50%"、fr="0"、gradientUnits="objectBoundingBox"、spreadMethod="pad"、href="none"

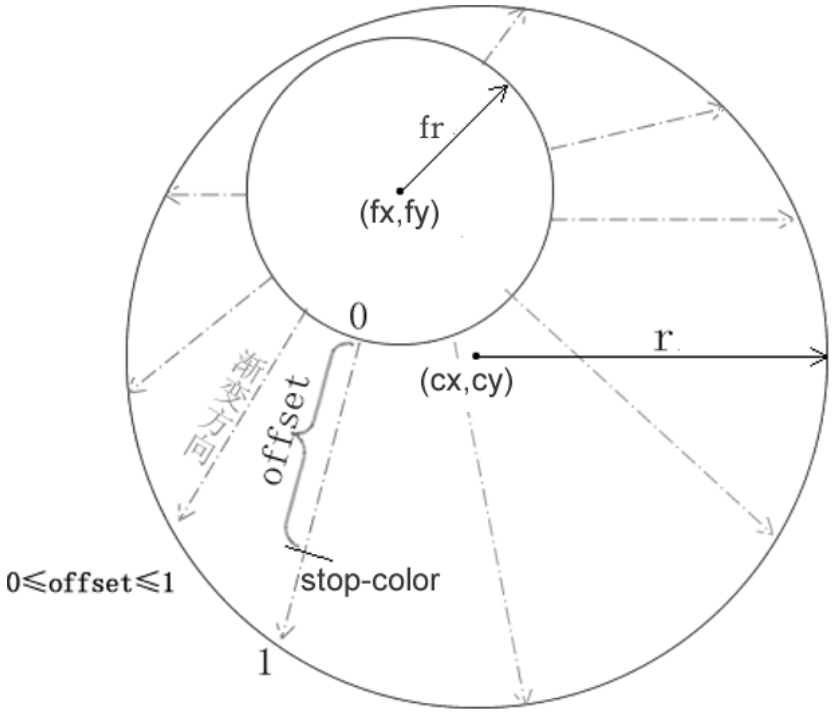


图 4.221 <radialGradient>标签用法说明

看例子：

```
<defs>
      <radialGradient id="Grad1" r="100%">
          <stop offset="0" stop-color="white" />
          <stop offset="90%" stop-color="black" />
      </radialGradient>
      <radialGradient id="Grad2" href="#Grad1" fx="0.3" fy="0.3" />
</defs>

<circle cx="100" cy="100" r="80" fill="url('#Grad1')" />
<circle cx="300" cy="100" r="80" fill="url('#Grad2')" />
```

浏览效果如图 4.222 左、中部分。

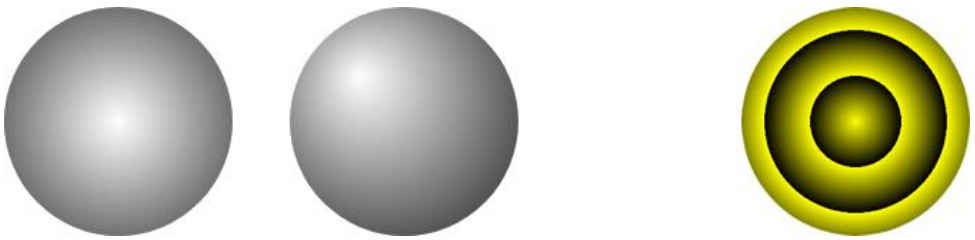


图 4.222 <radialGradient>标签用法举例

再看例子，浏览效果见图 4.222 右侧部分。

```
<radialGradient id="Grad" r="20%" spreadMethod="repeat">
     <stop offset="0" stop-color="yellow" />
```

```
      <stop offset="1" stop-color="black" />
  </radialGradient>

  <circle cx="200" cy="200" r="150" fill="url('#Grad')" />
```

4.10.7.7 动画

把标签的属性 x 的值由 y 改为 z，并把 y 和 z 当作关键帧，在这两个关键帧之间插入一系列的过渡帧 k_i，这样就产生了动画效果，即 y→k_1→k_2→……→z，示意图参考图 4.223。

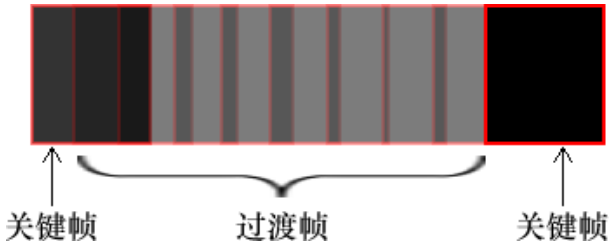


图 4.223 动物示意图

表 4.25 列举了与动画油管的 SVG 标签。

表 4.25 SVG 动画标签

标签名	简要说明
<animate>	对一个父标签的某个属性做动画改变
<animateMotion>	定义一个标签沿着路径运动
<animateTransform>	给标签设置变换属性，从而允许动画控制平移，缩放，旋转和/或切变
<discard>	定义销毁某个标签的时间点
<mpath>	作为<animateMotion>的子标签，用于引用一条外部路径
<set>	在特定的事件段内修改父标签的某个属性

1、 属性动画-<animate>

<animate>标签对父标签或 href 引用的标签的某个属性做动画改变，一个标签内可以有多个<animate>子标签，这样可以并行动画改变一个或者多个属性的值。其使用语法如下：

```
<xyz … >
    <animate attributeName="" 其他专用属性 [ 通用属性]/>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    <animate attributeName="" 其他专用属性 [ 通用属性]/>
    …
</xyz >
```

或者

```
<xyz id="xxx" … />
<animate href="#xxx" attributeName="" 其他专用属性 [ 通用属性]/>
<animate href="#xxx" attributeName="" 其他专用属性 [ 通用属性]/>
…
```

即<animate>标签采用它的 attributeName 属性指定父标签或 href 引用的标签的某个属性（下文简称为动画属性），并对此属性做动画改变。先看一个例子，让我们对 SVG 的动画有一个感性认识：

```
  <rect width="60" height="60" x="50" y="50">
       <animate dur="4s" attributeName="x" repeatCount="3" to="400" />
  </rect>
```

你会看到一个黑色正方形从左向右移动 3 次，每次持续 4 秒钟，正方形的水平坐标 x 在 50~400 之间变化。

<animate>标签的专用属性比较多，见下表 4.26。

表 4.26 <animate>标签的专用属性

分类	属性名	说明
时间类	begin	什么时候开始动画，值为分号分开的列表，列表钟的每个值指明到了这个时点就重新开始动画，默认"0s"
	end	什么时候结束动画，值的格式与 begin 相同。
	dur	循环一次动画持续时间，或取值"indefinite"，时间单位有 s(秒)、min(分钟)、h(小时)，还可取[hh:]mm:ss[.iii]
	min	动画持续的最小时间
	max	动画持续的最大时间
	restart	能否重启动画，取值"always"(默认值)-总能重启，"whenNotActive" - 动画只有停止后才能重启，"never" - 除了首次启动外再也不能重启。
	repeatCount	定义动画循环次数
	repeatDur	定义总的持续时间，或取值"indefinite"，时间单位有 s(秒)、min(分钟)、h(小时)，还可取[hh:]mm:ss[.iii]
	fill	动画最后的状态，"freeze"保留最后帧的状态，"remove"保留第一个帧的状态

取值类	calcMode	定义在两个相邻关键帧之间的插值方法，默认是线性插值法"linear"。如果属性不支持指定的插值法，则 calcMode 属性无效。另外的取值还有离散插值法"discrete"（瞬间完成）、匀速变化插值法"paced"、三次贝塞尔插值法"spline"。
	values	列出动画过程中的关键帧（用分号隔开的属性的值），这时 from、to、by 属性失效
	keyTimes	列出各关键帧的时间比值，取值范围为[0,1]，0 表示一轮动画的开始，1 表示一轮动画的结尾，0.5 表示一轮动画持续时间的一半。如果省略 keyTimes 属性，那么关键帧平均分配在一轮的时间轴上。
	keySplines	以相邻两个关键帧为锚点，定义该时间段内的三次贝塞尔曲线的两个控制点
	from	动画开始时属性的值（对应第一个关键帧）。如果存在 values 属性，from 失效
	to	动画结束时属性的值（对应最后一个关键帧）
	by	相对于属性初始值的改变量，初始值要么由 from 指定，要么由动画属性定义。by 的值可正可负
事件类	onbegin	动画开始时触发
	onend	动画结束时触发
	onrepeat	进入下一轮循环时触发
其他类	attributeName	指明参与动画的属性名称或者样式名称，如 attributeName="width"，即动画改变宽度。
	additive	定义关键帧的属性值是绝对的还是增量，取值"sum"-增量，"replace"（默认值）-绝对值
	accumulate	定义属性的值在每轮循环后是否累加，取值"sum"-累加，"none"（默认值）-不累加。如果只有 to 属性，本属性失效。

下面进一步介绍一些复杂的属性。

（1）、begin 属性的值的格式为"时点 1[;时点 2;时点 3;…]"，即到了“时点 n”就重新开始动画，如果此时动画正在一轮循环当中，那么就立即结束本轮循环，重新开始新一轮循环。如果相邻时点之间的时间跨度大于一轮循环的时间长度（由 dur 定义），那么该循环就不受影响，一切照旧，一直到了下一个“时点”时才会重新开始新一轮循环。“时点 n”可为如下格式之一：

①文档加载后过多少时间开始动画，不能为负数。格式可为："n"（等价于"ns"，即多少秒，如"4s"、"3"）、"nmin"（多少分钟，如"1.5min"）、"nms"（多少毫秒，如"500ms"）、"nh"（多少小时，如"2h"）、"[hh:]mm:ss[.iii]"（如 1 分 30 秒就是"01:30"、"01:30.050"表示 1 分 30 秒 50 毫秒、"02:01:30"表示 2 小时 1 分 30 秒、"02:01:30.200"）。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

②另一个动画开始或结束后过多少时间开始动画。格式为 id.{begin|end}[±offset]，其中 id 为另一个<animate>标签的 id 属性的值，begin 和 end 只能选择一个，前者表示另一个动画开始，后者表示另一个动画结束，±offset 是可选的，表示延后（正数）或者提前（负数）多少时间，时间的格式同①。如"third.begin+5"表示 id 为"third"的<animate>动画开始后再过 5 秒开始本动画，"third.end-01.500"表示 id 为"third"的<animate>动画结束前 1 秒 500 毫秒开始本动画。

③发生某个事件后过多少时间开始动画。格式为[id.]event[+offset]，即 id 指到的标签发生 event 时再过 offset 开始动画，如果省略 id 就是本动画所在的标签，event 常见的有 click（点击鼠标）、dblclick（双击）、mouseover（鼠标停靠）等。如 begin="click"表示点击鼠标开始动画、begin="go.click+2"表示点击 id 值为"go"的那个标签后再过 2 秒开始动画。

④另一个动画结束第几轮循环前/后多少时间开始动画。格式为 id.repeat(n)[±offset]，即 id 指到的那个<animate>动画结束第 n 轮循环时延后 offset（正数）或者提前 offset（负数）开始动画。如"rise.repeat(3)"表示 id 为"rise"的动画进入第 4 轮之初开始动画，"rise.repeat(3)+2"表示 id 为"rise"的动画结束第 3 轮后再过 2 秒开始动画。

⑤"indefinite"。当<animate>的属性 begin="indefinite"时，只有两种方法才能启动动画：一是采用 js 调用动画标签对象的 beginElement()方法，二是点击 svg 的<a>标签（此时 href="#<animate-id>"）。

（2）、end 属性的值的格式与 begin 属性相同，只不过是到了“时点 n”就结束动画，如果动画结束后再也没有重启，那么“时点 n”之后的所有时点无效，因为结束已经结束的动画没有实际意义。图 4.224 进一步解释了 begin 和 end 属性的关系。

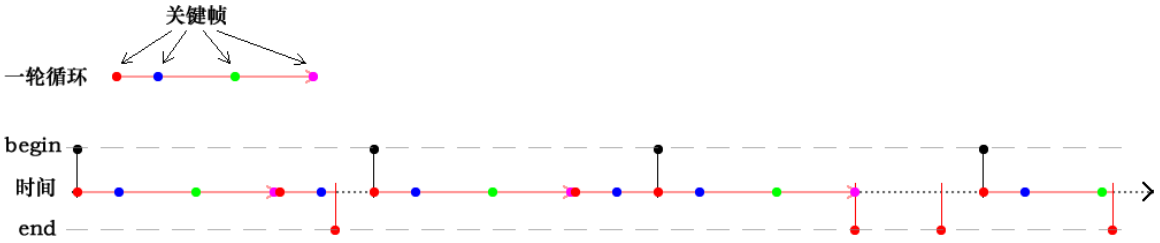


图 4.224 <animate>标签的 begin 和 end 属性的关系

（3）、属性 values 或者 from\to\by 定义关键帧，关键帧由动画属性的值确定，比如对圆的半径 r 做动画改变，那么“from="20" to="100"”定义了两个关键帧：第一个关键帧为半径等于 20 的圆，最后一个关键帧为半径等于 100 的圆。values 属性的值是以分号隔开的数列，数列中的每个“数”定义一个关键帧，只要存在 values 属性，from/to/by 属性失效，比如 values="10;30;70;100"定义了四个关键帧。

（4）、keyTimes 属性定义每个关键帧的时间占比，即在动画的一轮循环中，从开始到关键帧所在时间点位的时间长度除以一轮动画的时间总长度（由 dur 属性定义），所以取值范围为[0,1]，即第 1 个帧为 0，最后一个帧为 1，其他的帧都为小数，满足 $0 \leqslant t_1 \leqslant t_2 \leqslant t_3 \leqslant \cdots \leqslant 1$。如果省略 keyTimes 属性，那么关键帧平均分配在时间轴上，即相邻关键帧之间的时间长度都一样。参见图 4.225。

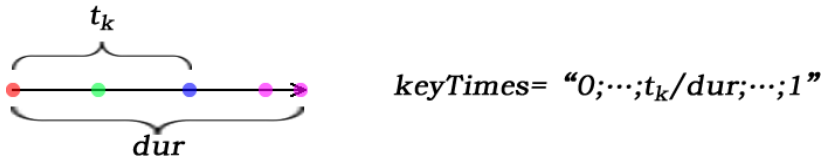


图 4.225 <animate>标签的 keyTimes 属性说明

属性 begin、end、values、keyTimes 的关系可用示意图 4.226 表示。

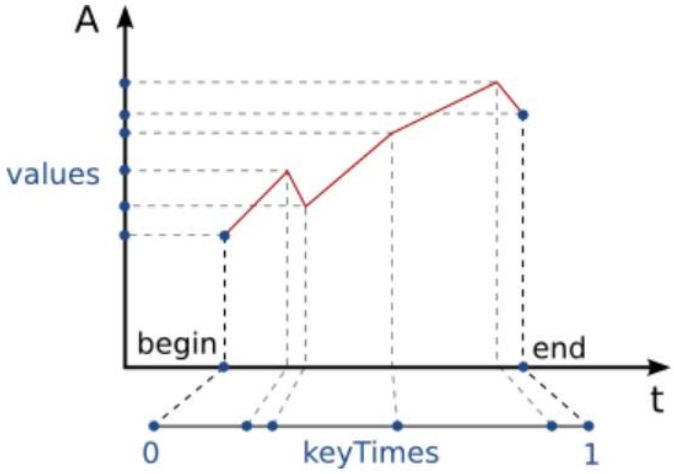
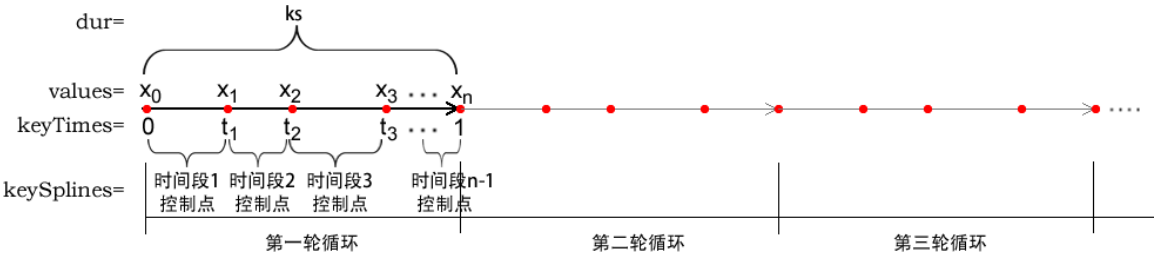


图 4.226 属性 begin、end、values、keyTimes 的关系

（5）、calcMode 属性定义在两个相邻关键帧之间的插值方法，即控制动画速度。它的取值可为"discrete"、"linear"（默认值）、"paced"或"spline"，其中 discrete 表示无插值，即从一个关键帧瞬间到下一个帧；linear 为线性插值，即从一个关键帧匀速到下一个关键帧，但有可能各个时间段内（关键帧之间）的速度不同；paced 表示从第 1 个关键帧匀速到最后一个关键帧，即整个动画过程中的速度保持不变，此时 keyTimes 和 keySplines 属性无效；spline 表示三次贝塞尔插值，即动画的速度的变化符合三次贝塞尔函数——以两个相邻关键帧的时间点(由 keyTimes 属性定义)为锚点，由 keySplines 属性定义两个控制点。keySplines 属性的值为用分号分隔的数列，但“数”的个数比 values 属性少一个，其格式为"时间段 1 的控制点;时间段 2 的控制点;时间段 3 的控制点;…;时间段 n-1 的控制点"，其中各个控制点的格式为 $x_1\ y_1\ x_2\ y_2$，满足 $0 \leqslant x_1,y_1,x_2,y_2 \leqslant 1$，默认为 0 0 1 1，即匀速变化，可参见示意图 4.227 进一步理解。



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

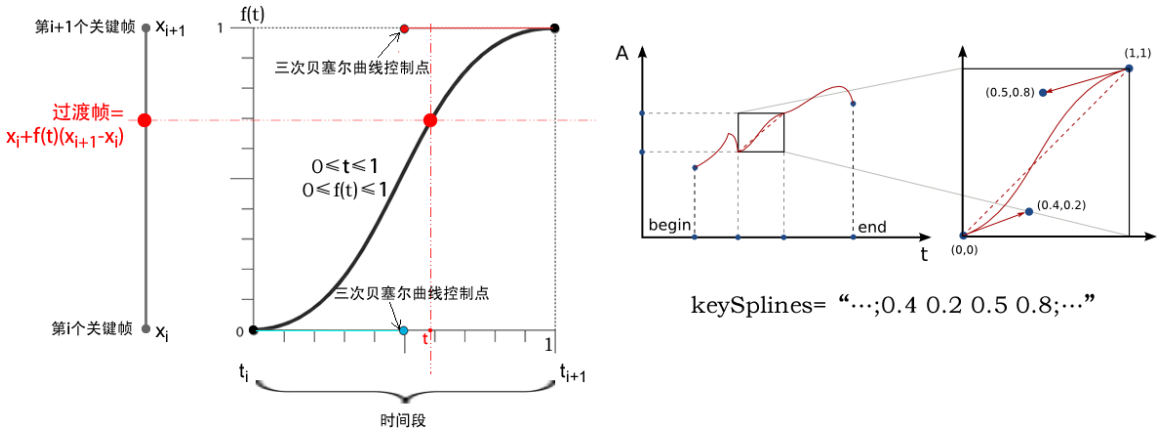


图 4.227 <animate>标签之三次贝塞尔动画

看下面的赛马例子：

```
<g font-size="60">
    <line x1="420" y1="0" x2="420" y2="260" stroke="red" />
    <line x1="0" y1="0" x2="0" y2="260" stroke="red" />
    <text y="60" x="360">马
        <animate attributeName="x" dur="5s" values="0; 135; 360" calcMode="disc
rete" />
    </text>
    <text y="120" x="360" fill="blue">马
        <animate attributeName="x" dur="5s" values="0; 135; 360" keyTimes="0; 0
.8; 1" calcMode="linear" />
    </text>
    <text y="180" x="360" fill="red">马
        <animate attributeName="x" dur="5s" values="0; 135; 360" calcMode="pace
d" />
    </text>
    <text y="240" x="360" fill="green">马
        <animate attributeName="x" dur="5s" values="0; 135; 360" keyTimes="0; 0
.8; 1" calcMode="spline" keySplines="0.5 0 0.5 1; 0 0 1 1" />
    </text>
</g>
```

浏览效果图 4.228，注意观察它们的速度。

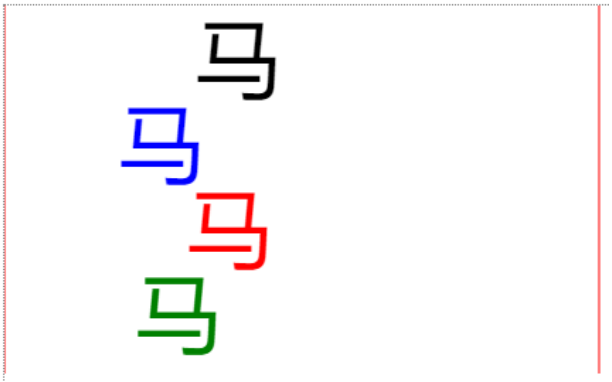


图 4.228 动画——赛马

最后那匹绿色马的动画采用三次贝塞尔插值法，只不过在两个时间段内的控制点不一样，马跑的速度看起来有点怪异。首先由<animate>标签的 values 属性可知存在三个关键帧，对应的水平坐标 x 的值分别为 0、135 和 360，再由 keyTimes 属性可知三个关键帧的时间点分别为 0 秒、4 秒和 5 秒，4 秒就是 0.8╳5 秒得来的。分成两个时间段，即第一时间段为 0s~4s 共 4 秒，第二时间段为 4s~5s 共 1 秒，在第一时间段的 4 秒钟内，绿色“马”的横坐标 x 由 0 过渡到 135，跨度为 135，而第二时间段的 1 秒钟内，x 由 135 过渡到 360，跨度达到 225。显然在最后 1 秒钟内，动画速度要比第一时间段快得多，而且是匀速的，因为控制点为 0 0 1 1，对应的三次贝塞尔曲线为一直线。而在第一时间段内，控制点为 0.5 0 0.5 1，这是条曲线，反应在马的速度上，就是先慢后快再慢。请参考示意图 4.229。

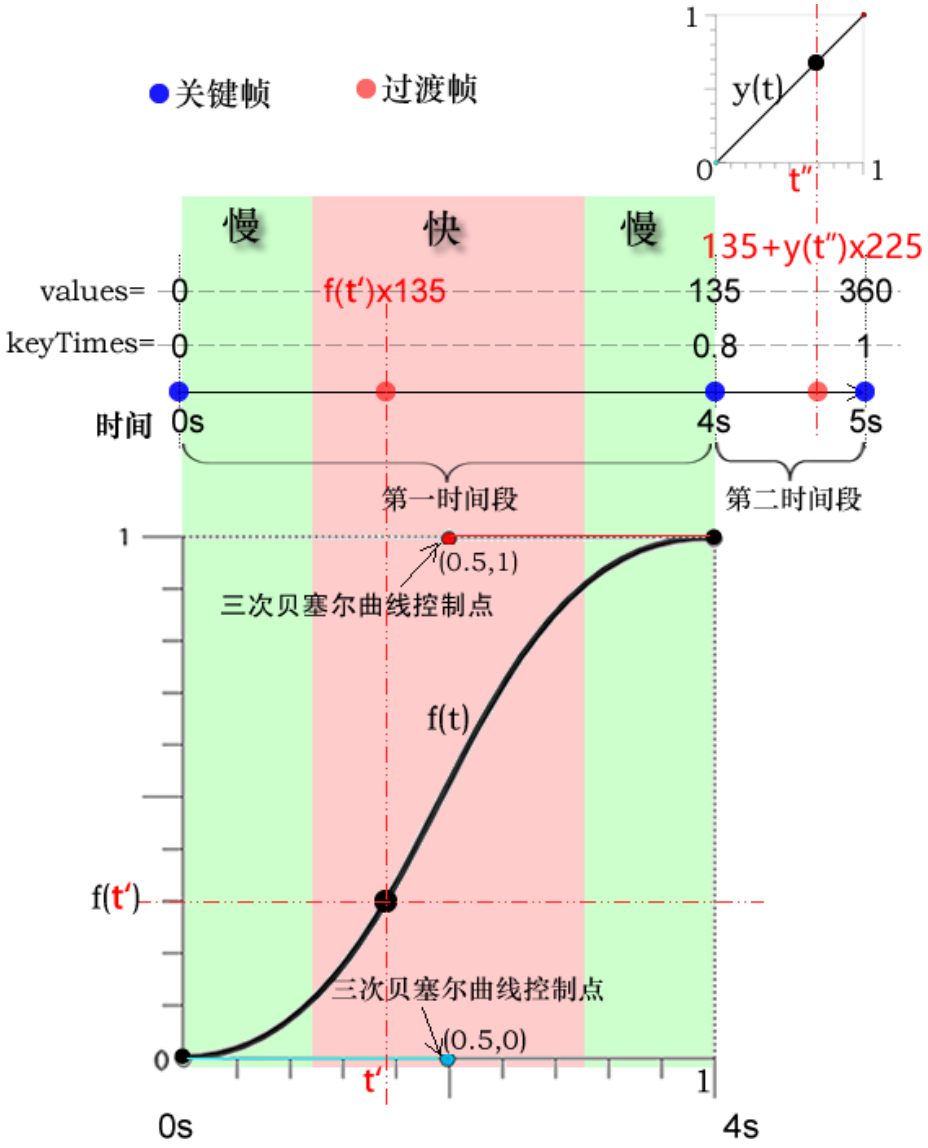


图 4.229 按三次贝塞尔曲线运动的马

再看一个模拟心跳的例子：

```
  <circle cx="175" cy="75" r="20" fill="red">
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
      <animate attributeName="r" values="20;50;20" keyTimes="0;.85;1" calcMode
="spline"  keySplines=".5 0 .5 1;.5 0 .5 1" dur="1" repeatCount="indefinite" />
  </circle>
```

1 秒种之内完成扩张和收缩动作，而且每个动作的速度都遵循慢-快-慢的规则。

（6）、additive 属性定义参与动画的属性的值（由 from、to、by 和 values 定义）是绝对值还是相对值，可取值"sum"和"replace"（默认值），前者表示相对值，后者表示绝对值。看例子。

```
  <circle cx="100" cy="100" r="30" fill="red">
      <animate attributeName="cx" dur="5s" begin="1" from="0" to="300" additiv
e="replace" fill="freeze" />
  </circle>
  <circle cx="100" cy="200" r="30" fill="blue">
      <animate attributeName="cx" dur="5s" begin="1" from="0" to="300" additiv
e="sum" fill="freeze" />
  </circle>
```

红色球中的 from 和 to 属性是绝对值，也就是说 cx 在 0~300 之间变化，蓝色球中的 from 和 to 是相对值，即 cx 在(200+0)~(200+300)之间变化，其中基础值 200 由<circle>标签的 cx 定义。

（7）、accumulate 属性定义动画属性的值在每轮循环后是否累加，取值"sum" - 累加，"none"（默认值）- 不累加。如果只有 to 属性或者动画属性不支持加法运算，则本属性失效。看下面的例子。

```
<path d="M 20,0 20,140 M 120,0 120,140 M 220,0 220,140" stroke="grey" />
<rect x="20" y="10" width="80" height="50" fill="red">
     <animate attributeName="x" id="s1" from="20" to="100" dur="3s" accumulate=
"sum" repeatCount="3" begin="click" fill="freeze">
     </animate>
</rect>
<rect x="20" y="70" width="80" height="50" fill="blue">
     <animate attributeName="x" begin="s1.begin" from="20" to="100" dur="3s" re
peatCount="3" begin="click"  fill="freeze">
     </animate>
</rect>
```

点击红色矩形开始动画。对于红色矩形，由于采用了属性 accumulate="sum"，所以矩形的 x 的值在每轮循环后不断累加，即第 1 轮循环后 x="100"，第 2 轮循环 x 在 120~200 之间变化（即 100+from~100+to），第 3 轮循环 x 在 220~300 之间变化。浏览的效果见图 4.230 左侧部分。

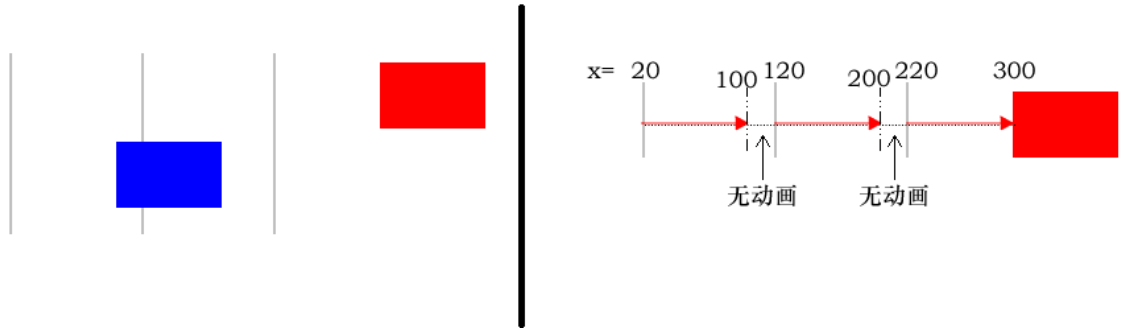


图 4.230 <animate>标签的 accumulate 属性

（8）、onend 属性为事件属性，表示动画结束时触发。在上述例子中的<animate>标签里增加 onend="console.log('动画结束了！')"，这样当动画结束时就在浏览器的控制台上打印“动画结束了！”。

同一个标签的同一个属性允许被多个<animate>标签使用，从而形成串行混合动画效果，类似下面的例子：

```
  <path d="M 300,0 300,200 M400,0 400,200" stroke="red" />
  <circle cx="100" cy="100" r="30" >
        <animate attributeName="cx" to="400" dur="7" fill="freeze" />
        <animate attributeName="cx" begin="2" to="300" dur="1" />
  </circle>
```

同一个标签的同一个属性是不可能产生并行的动画效果的，比如一个球不可能既快速水平运动又慢速水平运动，只可能产生串行的动画效果。那么在同一个时间点，同时有多个活动的动画存在，这时优先级最高的动画起作用。在启动动画时确定优先级，越早启动的动画，其优先级越低，如果同时启动，此时存在两种情况：①存在 begin="animate-id.begin"属性的动画优先级高于 id="animate-id"属性所在的动画；②在文档中排在前面的动画优先级较低。但要注意的是进入动画下一轮循环不叫启动动画，只有开始新的动画过程才是启动动画，比如由 begin 属性定义的时间点就是启动动画的时间点。当最高优先级的动画结束时，剩下的动画中的最高优先级的动画起作用，依次类推，不过存在 additive="sum"属性的动画会受到低优先级动画的影响。

以动画形式改变标签的属性的值，其实并没有修改 DOM 中的标签对象的同名属性值，前者修改的是展示值，而 DOM 中的那个值我们称之为原始值，原始值一直不变，这点可以通过 JS 编程加以验证。

最后看一个综合的例子：弹跳的红色气球。

```
<line x1="0" y1="342" x2="1040" y2="342" stroke="grey" stroke-width="5" />
<radialGradient id="Grad" r="100%" fx="0.3" fy="0.3">
     <stop offset="0" stop-color="red" />
     <stop offset="90%" stop-color="black" />
</radialGradient>

<circle id="heart" cx="10" cy="300" r="40" fill="url('#Grad')">
     <animate attributeName="cx" from="10" to="1000" dur="10" repeatCount="inde
finite" />
     <animate attributeName="cy" values="300;100;300" keyTimes="0;.5;1" calcMod
e="spline"  keySplines=".5 0 .5 1;.5 0 .5 1" dur="2" repeatCount="indefinite" />
</circle>
```

下面的例子是改变形状：

```
<path fill="#c44">
     <animate attributeType="XML" attributeName="d" dur="6s" repeatCount="indef
inite"  values="M 0,0 L 40,0 80,0 40,80;
                M 0,80 L 40,0 80,80 40,80;
                M 0,0 L 40,0 80,0 40,80" />
</path>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

2、 路径动画-<animateMotion>

<animateMotion>标签定义沿着路径运动的动画，其使用语法如下：

```
<xyz … >
    <animateMotion 其他专用属性 [ 通用属性]/>
    <animateMotion 其他专用属性 [ 通用属性]/>
    …
</xyz >
```

或者

```
<xyz id="xxx" … />
<animateMotion href="#xxx" 其他专用属性 [ 通用属性]/>
<animateMotion href="#xxx" 其他专用属性 [ 通用属性]/>
…
```

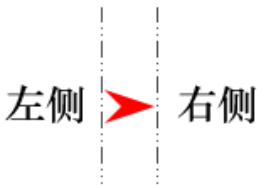
与<animate>标签相比，少了 attributeName 属性，多了 keyPoints、path 和 rotate 三个属性，同名属性与<animate>标签一样，只不过 calcMode 默认值为 paced，这里不再赘述，下面具体讲讲这新增的三个属性。

keyPoints 属性 - 定义 keyTimes 中的每个时间点对应的运动图形离路径起点的距离与路径总长度的比例，格式为用分号隔开的数列，因此取值范围为[0,1]，0 表示起点，1 表示终点，0.5 到达路径中点。如 keyPoints="0;0.5;1" keyTimes="0;0.15;1"，即前半程只用了总时长的 15%，后半程用了 85%。

path 属性 - 定义动画路径，格式与<path>标签的 d 属性一样，但是路径本身不会显示，如果要显示路径，建议采用<path>标签，它的 d 属性的值与 path 属性的值相同即可。也可以采用<mpath>子标签引用外部的一条路径，如：

```
  <path id="theMotionPath" …… />
  <animateMotion dur="6s" repeatCount="indefinite" rotate="auto">
       <mpath href="#theMotionPath" />
  </animateMotion>
```

rotate 属性 - 定义运动图形沿着路径运动时的方向，可取值"auto"、"auto-reverse"或<number>，默认是 0，即不做任何旋转改变。auto 指示图形动态旋转以使其右侧始终对着前进方向，auto-reverse 指示图形动态旋转以使其左侧始终对着前进方向，<number>为以数字，单位是度。参考示意图 4.231。



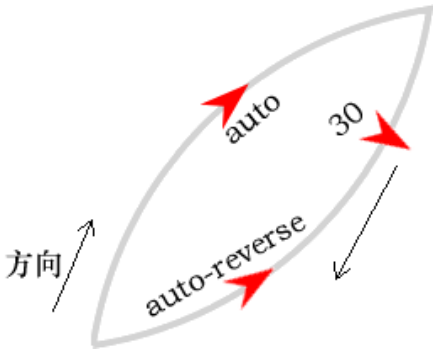


图 4.231 <animateMotion>标签的 rotate 属性

看一个例子：

```
<path fill="none" stroke="lightgrey" d="M20,50 C20,-50 180,150 180,50 C180-
50 20,150 20,50 z" />

<circle r="5" fill="red">
     <animateMotion dur="10s" repeatCount="indefinite" keyTimes="0;0.2;1" keyPo
ints="0;0.5;1" calcMode="linear"       path="M20,50 C20,-50 180,150 180,50 C180-
50 20,150 20,50 z" />
</circle>
```

浏览的效果如图 4.232。



图 4.232 <animateMotion>标签用法举例

至于 from/to/by/values 属性，在<animateMotion>标签中用于定义运动路径的顶点，因此只能是折线路径。from/to/by 的值的格式为"x,y"(顶点坐标)，而 values 的值的格式为"$x_1,y_1;x_2,y_2;x_3,y_3;\ldots$"，定义了以$(x_i,y_i)$为顶点的折线。如果 path 属性存在，这些属性无效。看下面的射击例子：

```
   <circle r="5" fill="red">
        <animateMotion dur="5" values="10,400;500,10;500,620" repeatCount="inde
finite" keyTimes="0;0.1;1" keyPoints="0;0.5;1" calcMode="linear" />
   </circle>
```

3、 变换动画-<animateTransform>

<animateTransform>标签以动画形式对目标对象做变换，如缩放、旋转、移动和扭曲等。本标签的语法与<animate>相同，属性也相同，但增加了定义变换类型的 type 属性，它可取值"translate"（默认）、"scale"、"rotate"、"skewX"或"skewY"。type 取值不同，from/by/to/values 属性的值的格式也不同，当①type="translate"或者省略 type 时表示移位变换，from/by/to="x,[y]"；②type="scale"时表示缩放变换，from/by/to="s_x,[s_y]"，s_x 为 x 轴方向的缩放比率，同理 s_y 为 y 轴方向的缩放比率；③type="rotate"时表示旋转变换，from/by/to=" <rotate-angle> [c_x c_y]"，其中(c_x,c_y)为转轴；④type=" skewX"或"skewY"时表示扭曲变换，from/by/to="<skew-angle>"。values 属性的值为用分号分隔的数列，里面每一个“数”的格式与 from 属性相同。

下面的例子是高度和宽度不断改变的椭圆：

```
<svg width="1200" height="600" version="1.1" viewbox="-100 -100 400 400">
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    <circle cx="0" cy="0" r="25" fill="#c44">
        <animateTransform attributeName="transform" type="scale" dur="6" repe
atCount="indefinite" values="1 2; 2 1; 1 2" />
    </circle>
</svg>
```

下面的例子为一个正方形在移动的同时旋转：

```
<rect x="5" y="5" width="20" height="20" rx="4" fill="#c44">
     <animateTransform attributeName="transform" attributeType="XML" type="tra
nslate" values="0,0; 275,0; 0,0"  dur="6s" repeatCount="indefinite" additive="s
um" />
    <animateTransform attributeName="transform" attributeType="XML" type="rota
te" from="0 15,15" to="360 15,15"  dur="4s" repeatCount="indefinite" additive="
sum" />
</rect>
```

4、 变换动画-<discard>

定义销毁某个标签对象的时间点，标签一旦被销毁，它占用的资源（如内存）就被释放出来，<discard>自身也被同时销毁。<discard>的语法如下：

```
<xyz … >
    …
    <discard begin="" />
</xyz >
```

或者

```
<xyz id="xxx" … />
<discard href="#xxx" begin="" />
```

begin 属性定义销毁标签对象的时间，格式同<animate>标签的同名属性。不过<discard>标签还没有激活前，允许被其他<discard>销毁。看例子：

```
<ellipse id="ec" cx="98.5" cy="17.5" rx="20.5" ry="17.5" fill="blue" stroke="bl
ack" transform="translate(9 252) translate(3 -296)">
    <animateTransform attributeName="transform" begin="0s" dur="2s" fill="remo
ve" calcMode="linear"  type="translate" additive="sum" from="0 0" to="-18 305" />
    <discard begin="2s" />
</ellipse>
```

不过很多浏览器还不支持<discard>标签。

5、 临时改值-<set>

<set>标签提供了一种简单的方法，允许在指定的持续时间内设置属性的值，没有动画效果。它支持所有的属性，而有的属性是无法通过插值形成动画效果的，比如值为布尔类型和字符串类型的属性就没法做动画。对于能通过合理插值的属性，如果要实现动画效果，请优先使用<animate>标签。<set>标签的使用语法与<animate>标签相同，但只有这些属性：attributeName、

begin、dur、end、min、max、restart、repeatCount、repeatDur、fill、onbegin、onend、onrepeat 和 to。看例子：

```
<rect id="me" width="100" height="100">
     <set attributeName="x" to="300" begin="me.click" dur="2s" />
</rect>
```

浏览的效果是：点击黑色正方形后 x="300"，2 秒后恢复 x="0"。

4.10.7.8 滤镜

SVG 的滤镜是非常强大的功，也是比较难掌握的，因为涉及很多光学和图像处理方面的知识。表 4.26 列出了全部与滤镜油管的标签。

表 4.26 滤镜标签

标签名	简要说明
<filter>	通过分组原子滤镜来定义滤镜效果
<feBlend>	按照某种组合模式混合两个图形对象
<feColorMatrix>	基于 5x5 的变换矩阵来改变象数的颜色
<feComponentTransfer>	每个像素执行颜色分量数据的重新映射，以便进行诸如亮度调节，对比度调节，色彩平衡或阈值设置之类的操作
<feComposite>	组合两个图形，完成诸如 over、in、atop、out、xor 和 lighter 操作
<feConvolveMatrix>	应用矩阵卷积过滤器效果。通过卷积可以实现各种各样的成像操作，包括模糊、边缘检测、锐化、浮雕和倒角等
<feDiffuseLighting>	使用 alpha 通道作为凹凸贴图来照亮图像。 生成的图像是 RGBA 不透明图像，其效果取决于输入的凹凸贴图的灯光颜色、灯光位置和表面几何形状
<feDisplacementMap>	使用 in2 属性指定的图像中的像素值对 in 属性定义的图像进行空间移位
<feFlood>	使用由 Flood-color 和 Flood-opacity 属性定义的颜色和不透明度填充滤镜子区域
<feFuncA>	为父标签<feComponentTransfer>的输入图形的 Alpha 分量定义传递函数
<feFuncB>	为父标签<feComponentTransfer>的输入图形的 Blue 分量定义传递函数
<feFuncG>	为父标签<feComponentTransfer>的输入图形的 Green 分量定义传递函数

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

<feFuncR>	为父标签<feComponentTransfer>的输入图形的 Red 分量定义传递函数
<feGaussianBlur>	模糊图形，模糊距离由 stdDeviation 属性定义
<feImage>	从外部源获取图像数据并输出像素数据。如果外部源是 SVG 矢量图，则将其位图化处理
<feMerge>	并行应用滤镜效果，而不是串行应用滤镜效果
<feMergeNode>	获取另一个过滤器的结果，该过滤器将由其父<feMerge>标签处理
<feMorphology>	用于腐蚀或膨胀图像
<feOffset>	偏移图像操作，由 dx 和 dy 属性指定偏移量。
<fePointLight>	定义一个光源，该光源可以创建点光源效果。 <feDiffuseLighting> 或 <feSpecularLighting>标签会用到点光源
<feSpecularLighting>	使用 alpha 通道作为凹凸贴图来照亮源图形
<feSpotLight>	定义一个光源，该光源可以创建聚光灯效果。 <feDiffuseLighting> 或 <feSpecularLighting>标签会用到聚光灯
<feTile>	通过重复平铺源图案来填充一个矩形，效果类似于<pattern>
<feTurbulence>	使用 Perlin 噪声函数创建图像。 它可以合成人造纹理，如云层或大理石。 生成的图像将填充整个滤镜基元子区域

1、 滤镜-<filter>

<filter>标签组合若干原子滤镜来自定义滤镜效果，类似<fe*>的标签就是原子滤镜。它本身不会被渲染，但必须由 SVG 标签的 filter 属性或同名样式来引用。<filter>标签的语法如下：

```
<filter x="" y="" width="" height="" filterUnits="" primitiveUnits="" href="">
      描述性标签
      原子滤镜标签
      <animate>或<set>标签
</filter>
```

其中的专用属性介绍如下：

x、y 属性 – 滤镜左上角的坐标，默认值(-10%,-10%)。

width、height 属性 – 滤镜的宽度和高度，默认值(120%,120%)。

filterUnits 属性 – 定义 x、y、width 和 height 属性的坐标系统，可取值"userSpaceOnUse"或"objectBoundingBox"（默认值），前者为当前坐标系，后者为滤镜所应用的图形的边界框坐标系（边界框的左上角为坐标原点），具体介绍参见<mask>标签的 maskUnits 属性。

primitiveUnits 属性 - 定义<filter>标签内容所使用的坐标系，它的取值与 filterUnits 属性相同，默认取值"userSpaceOnUse"，详细解释请参考<mask>标签的 maskContentUnits 属性（在“§4.5.5.4 容器标签”小节）。

href 属性 – 引用本<svg>元素内的另一个<filter>标签，被引用的<filter>标签有而本<filter>标签没有的属性都会继承过来，另外如果本<filter>标签内没有子标签，那么也继承被引用<filter>标签内的全部子标签。允许多层继承，如 A 引用 B，B 引用 C，那么 A 可能继承了 C。

看一个例子：

```
<filter id="blurMe">
    <desc>
       好一个例子
    </desc>
    <feGaussianBlur stdDeviation="5" />
</filter>

<circle cx="60" cy="60" r="50" fill="green" />
<circle cx="170" cy="60" r="50" fill="green" filter="url(#blurMe)" />
```

其中<filter>标签内的<feGaussianBlu>子标签是一个原子滤镜标签，后面会介绍，第二个圆通过 filter 属性引用<filter>滤镜，浏览的效果如图 4.233。



图 4.233 滤镜<filter>用法举例

2、 油漆桶-<feFlood>

<feFlood>原子滤镜使用由 Flood-color 和 Flood-opacity 定义的颜色和透明度填充滤镜子区域，滤镜子区域是由属性 x、y、width 和 height 属性定义的矩形范围（限定在父标签<filter>的规定的范围之内）。其试用语法为：

```
    <feFlood x="" y="" width="" height="" flood-color="" flood-opacity=""/>
```

属性 x、y、width 和 height 参考<filter>标签的同名属性，不过它们的意义还与父<filter>标签的属性 primitiveUnits 有关。flood-color 属性定义颜色（也可以采用同名样式），默认值为 black，flood-opacity 属性定义透明度（也可以采用同名样式），取值范围为[0,1]，默认值为 1。看例子：

```
  <filter id="flood">
       <feFlood flood-color="skyblue" x="0" y="0" width="100" height="200" />
  </filter>
  <rect x="0" y="0" width="200" height="200" style="filter: url(#flood);"/>
```

3、 填充-<feImage>

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

<feImage>原子滤镜从外部源获取图像数据并输出像素数据(这意味着如果外部源是 SVG 图像，则将其位图化）。其试用语法为：

```
<feImage x="" y="" width="" height="" href ="" preserveAspectRatio=""/>
```

属性 x、y、width 和 height 参考<filter>标签的同名属性，href 属性定义外部图形，可为 SVG 矢量图或者位图，preserveAspectRatio 属性定义对齐方式，当图片大小或长宽比率与本标签中的 width、height 不一致时，需要指定具体的对齐方式。看一个例子：

```
<defs>
     <filter id="f" primitiveUnits="objectBoundingBox">
          <feImage href="demo.jpg" x="0" y="0" width="1" height="1" />
     </filter>
</defs>
<rect x="30" y="30" width="150" height="150" filter="url(#f)" />
<rect x="30" y="30" width="150" height="150" stroke="#00f" fill="none" />
```

4、 混合-<feBlend>>

<feBlend>原子滤镜按照某种组合模式混合两个图形对象，效果类似图形处理软件中的混合图层操作。<feBlend>标签作为<filter>标签的子标签，它的使用语法如下：

```
<feBlend in="" in2="" mode="" />
```

其中属性 in 定义输入源，可取值：

"SourceGraphic" - 本<filter>标签内的原始输入的图形标签（如<feImage>）作为本滤镜的输入源。

"SourceAlpha" - 与"SourceGraphic"一样，不过只使用 alpha 通道。

"BackgroundImage" - 应用<filter>标签时，滤镜区域下的图像快照作为输入源。滤镜区域由<filter>标签的 x、y、width、height 和 filterUnits 属性定义。

"BackgroundAlpha" - 与"BackgroundImage"一样，不过只使用 alpha 通道。

"FillPaint" - 应用滤镜效果的那个目标元素上的 fill 属性的值作为输入源。

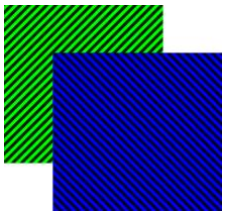
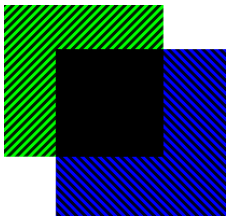
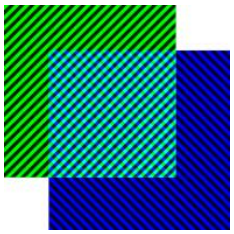
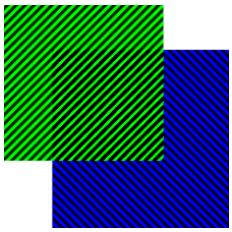
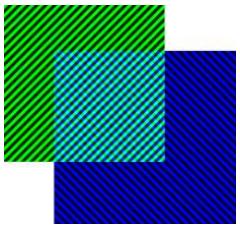
"FStrokePaint" - 应用滤镜效果的那个目标元素上的 stroke 属性的值作为输入源。

"<filter-primitive-reference>" - 同一个<filter>标签中的另一个原子滤镜的输出作为输入源，“另一个原子滤镜”带有属性 result="<filter-primitive-reference>"。

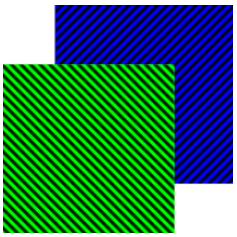
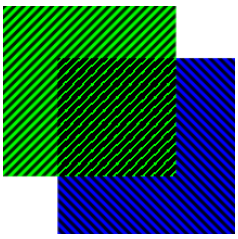
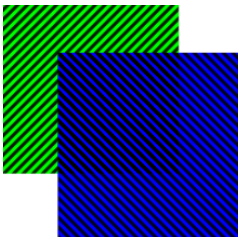
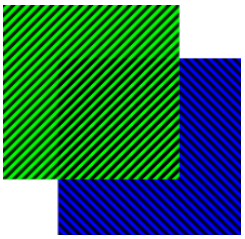
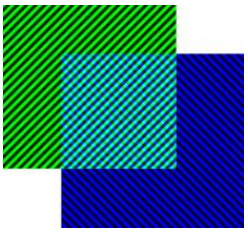
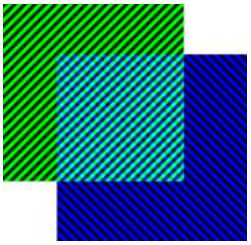
<feBlend>标签的 in2 属性定义第二个输入源，取值与 in 属性相同。属性 mode 定义混合模式，影响重叠部分图形的色彩效果，其取值见表 4.26。

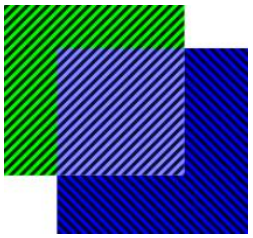
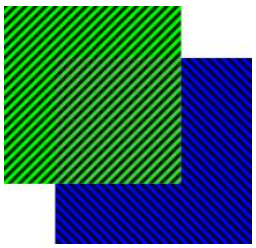
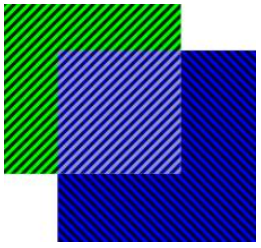
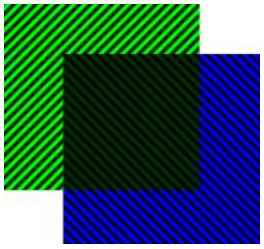
表 4.26 <feBlend>标签的 mode 属性的取值

取值	说明	图例

normal （默认值）	顶层完全覆盖下层，像纸叠加在一起	
multiply	颜色相乘（实际上要再除以 255，以保证颜色最终落在[0,255]）。如 rgb(120,200,220)╳rgb(51,150,30)=rgb(24,118,26)	
screen	颜色相乘后反转	
overlay	如果底色较深，则与取值"multiply"效果一样；如果底色较浅，则与取值"screen"效果一样。	
darken	最终颜色由每个颜色通道的最暗值组成。rgb(120,200,220)与 rgb(151,150,30)叠加的结果 rgb(151,200,220)	
lighten	最终颜色由每个颜色通道的最亮值组成。rgb(120,200,220)与 rgb(151,150,30)叠加的结果 rgb(120,150,30)	

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

color-dodge	底色除以顶色的反转色	
color-burn	反转底色后除以顶色得 X，最后再反转 X	
hard-light	如果顶色较深，则与取值"multiply"效果一样；如果顶色较浅，则与取值"screen"效果一样。	
soft-light	最终颜色类似于"hard-light"，但更柔和。效果类似于用散射的聚光灯照射背景图。	
difference	浅色减去深色后的结果。黑色层没有效果，而白色层会反转另一层的颜色。	
exclusion	最终颜色类似于"difference"，但对比度较低。	

hue	最终颜色具有顶部颜色的色相，同时使用底部颜色的饱和度和光度。	
saturation	最终颜色具有顶部颜色的饱和度，同时使用底部颜色的色相和亮度。纯灰色背景（不饱和）将不起作用。	
color	最终颜色具有顶部颜色的色相和饱和度，同时使用底部颜色的光度。该效果保留了灰度等级，可用于为前景着色。	
luminosity	最终颜色具有顶部颜色的亮度，同时使用底部颜色的色相和饱和度。此混合模式等效于颜色，但是交换了图层。	

看例子

```
<filter id="blend1" x="0" y="0" width="100%" height="100%">
      <feFlood result="floodFill" x="0" y="0" width="100%" height="100%" flood-
color="seagreen" flood-opacity="1" />
      <feBlend in="SourceGraphic" in2="floodFill" mode="multiply" />
</filter>
<filter id="blend2" x="0" y="0" width="100%" height="100%">
      <feFlood result="floodFill" x="0" y="0" width="100%" height="100%" flood-
color="seagreen" flood-opacity="1" />
      <feBlend in="SourceGraphic" in2="floodFill" mode="color-dodge" />
</filter>

<image href="mask.jpg" width="256" height="256" />
<image href="mask.jpg" width="256" height="256" style="filter:url(#blend1); tra
nsform:translateX(300px);" />
<image href="mask.jpg" width="256" height="256" style="filter:url(#blend2); tra
nsform:translateX(600px);" />
```

浏览的效果见图 4.234。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 4.234 <feBlend>标签的 mode 属性用法举例

5、 色彩矩阵-<feColorMatrix>

<feColorMatrix>原子滤镜通过色彩矩阵来修改颜色，类似 photoshop 中的颜色通道编辑功能。众所周知，每个像素的颜色是一个四元组[R, G, B, A]，分别表示红色、绿色、蓝色和透明度，此四元组加上分量 1 作为一个 1×5 矩阵，再乘以一个 5×5 色彩矩阵得到一个新的 1×5 矩阵[R' G' B' A' 1]，最终得到元素的新颜色四元组[R', G', B', A']。矩阵相乘的公式如下：

$$\begin{bmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{bmatrix} = \begin{bmatrix} r_1 & r_2 & r_3 & r_4 & r_5 \\ g_1 & g_2 & g_3 & g_4 & g_5 \\ b_1 & b_2 & b_3 & b_4 & b_5 \\ a_1 & a_2 & a_3 & a_4 & a_5 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix}$$

由此得到：

$R' = r_1*R + r_2*G + r_3*B + r_4*A + r_5$

$G' = g_1*R + g_2*G + g_3*B + g_4*A + g_5$

$B' = b_1*R + b_2*G + b_3*B + b_4*A + b_5$

$A' = a_1*R + a_2*G + a_3*B + a_4*A + a_5$

<feColorMatrix>标签作为<filter>标签的子标签，它的使用语法如下：

```
<feColorMatrix in="" type="" values="" />
```

其中属性 in 的含义与<feBlend>的同名属性一样，type 属性定义色彩矩阵类型，可取值：

"matrix" – 色彩矩阵。values 属性定义了一个 4×5 的色彩矩阵，按先行后列次序保存了 20 个数字（浏览器会加上一行"0 0 0 0 1"，成为一个 5×5 矩阵再参与运算），用空格或逗号隔开，即"r_1 r_2 r_3 r_4 r_5 g_1 g_2 g_3 g_4 g_5 b_1 b_2 b_3 b_4 b_5 a_1 a_2 a_3 a_4 a_5"，默认值为单位矩阵，即"1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0"。

"saturate" – 饱和度。values 属性定义饱和度，值为一实数，取值范围为[0,1]，默认值为 1。

"hueRotate" – 色相旋转度。由 values 属性定义具体的度数，值为实数，默认值为 0。参考图 4.235。

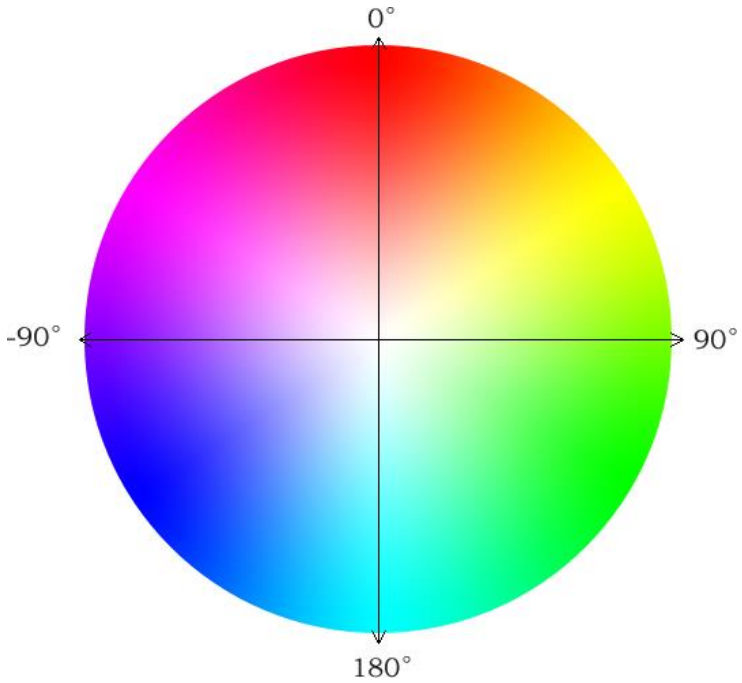


图 4.235 色相图

"luminanceToAlpha" – 透明度。这时 values 属性无效，色变矩阵固定为"0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0.2126 0.7152 0.0722 0 0"。

我们来看一个例子：

```
<filter id="Saturate">
     <feColorMatrix type="saturate" values="0.5"/>
</filter>
<filter id="HueRotate">
     <feColorMatrix type="hueRotate" values="90"/>
</filter>
<filter id="colorMatrix">
     <feColorMatrix in="SourceGraphic" type="matrix" values=
       "0 0 0 0 0
        0 1 0 0 0
        0 0 1 0 0
        0 0 0 1 0" />
</filter>
<circle cx="60" cy="60" r="50" fill="red"/>
<circle cx="180" cy="60" r="50" fill="red" filter="url(#Saturate)"/>
<circle cx="300" cy="60" r="50" fill="red" filter="url(#HueRotate)"/>
<circle cx="420" cy="60" r="50" fill="red" filter="url(#colorMatrix)"/>
```

浏览的效果如图 4.236。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

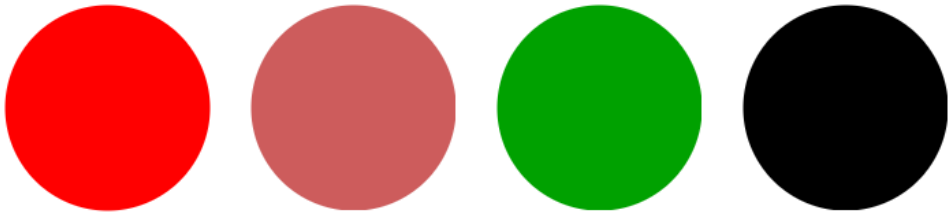


图 4.236 <feColorMatrix>标签用法举例

对于第二、三个圆的颜色很容易理解，为什么第四个圆是黑色？我们需要通过色彩矩阵的计算过程来理解。

$$\begin{bmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 255 \\ 0 \\ 0 \\ 0.5 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0.5 \\ 1 \end{bmatrix}$$

因此$[R', G', B', A']$= [0, 0, 0,0.5]，即 rgba(0,0,0,0.5)，这就是黑色半透明颜色。

6、 映射-<feComponentTransfer>/<feFuncR>/<feRuncG>/<feFuncB>/<feFuncA>

<feComponentTransfer>原子滤镜为每个像素执行颜色分量的重映射，允许进行诸如亮度调节、比度调节、色彩平衡或阈值设置之类的操作。通过将每个颜色通道（R、G、B 和 A）更改为子标签<feFuncR>、<feFuncB>、<feFuncG>和<feFuncA>返回的结果，来修改颜色。它们的使用语法如下：

```
<feComponentTransfer[ in=""]>
    [<feFuncR   type=""[   tableValues=""   slope=""   intercept=""   amplitude=""
exponent="" offset="" ]/>]
    [<feFuncG   type=""[   tableValues=""   slope=""   intercept=""   amplitude=""
exponent="" offset="" ]/>]
    [<feFuncB   type=""[   tableValues=""   slope=""   intercept=""   amplitude=""
exponent="" offset="" ]/>]
    [<feFuncA   type=""[   tableValues=""   slope=""   intercept=""   amplitude=""
exponent="" offset="" ]/>]
</feComponentTransfer>
```

属性 in 的含义与<feBlend>标签的同名属性一样。颜色分量转移函数标签<feFunc?>实际上就是一个函数：$C' = \mathrm{f(C)}$，即把初始颜色 C 转换为C'，C 和C'的取值范围都是[0,1]。type 属性定了函数 f，函数不同，能搭配的其他属性也不同，具体参见图 4.237。

type取值与其他属性搭配表

其他属性 / 类型type	tableValues	slope	intercept	amplitude	exponent	offset	解释
identity							等值映射
table	√						分段线性插值
discrete	√						离散数据映射
linear		√	√				线性函数
gamma				√	√	√	指数函数
	一系列数字	函数的斜率	函数的常量	函数的幅度	函数的指数	偏移量	

图 4.237 <feFunc?>标签的属性 type

"identity"定义的函数就是$C' = C$;

"table"定义分段线性插值函数，它能近似模拟任何曲线插值函数，相当于在 C 轴上的等分采样点。段点由 tablevalues 属性定义，它的值为 n+1 个用空格或逗号分开的数字组成，即"$v_0\ v_1\ v_2\ \cdots\ v_n$"，n+1 个段点把 C 的[0,1]区间平均分成 n 段，每一段的长度均为$1/n$，见图 4.238 左侧部分，n 越大，近似都越高。现假设初始颜色为 c，则c'满足：

$$c' = v_k + \frac{\left(c - \frac{k}{n}\right)}{\frac{1}{n}} \times (v_{k+1} - v_k) = v_k + \left(c - \frac{k}{n}\right) \times n \times (v_{k+1} - v_k)$$

其中k满足：

$$\frac{k}{n} \le c < \frac{k+1}{n}$$

即 c 落在 C 坐标上的$[\frac{k}{n}, \frac{k+1}{n})$区段内，但当 c=1 时，$c' = v_n$。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

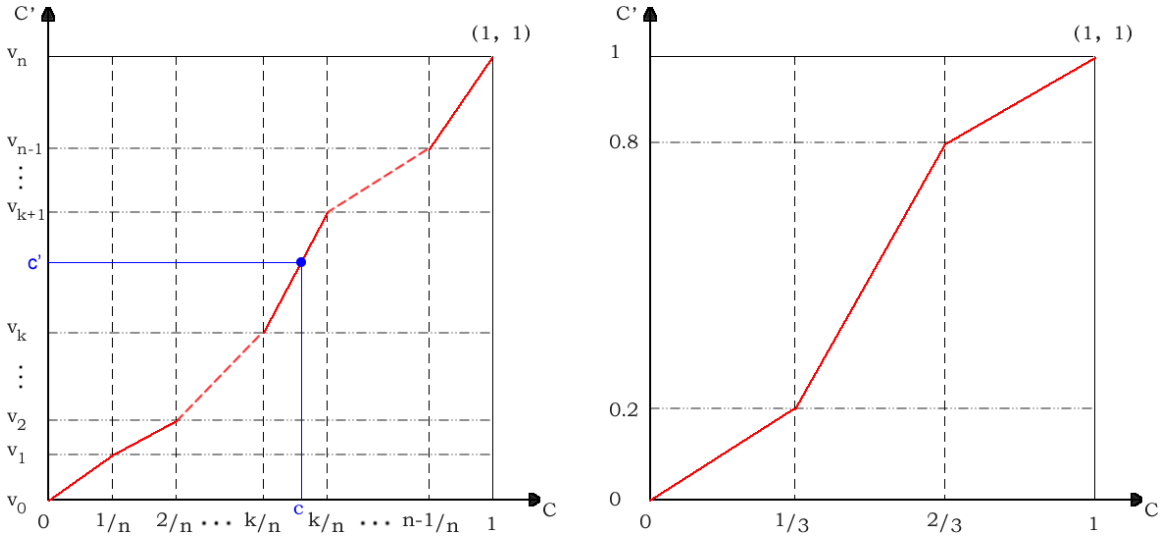


图 4.238 <feFunc?>标签的属性 type 取值"table"

看一个例子：

```
<feFuncR type="table" tableValues="0 0.2 0.8 1"></feFuncR>
```

例子对应的函数曲线参见上图右侧部分。

"discrete"定义离散函数，离散值保存在 tablevalues 属性中，离散值格式为 n 个用空格或逗号分开的数字，即"v_0 v_1 v_2 … v_{n-1}"。现假设初始颜色为 c，则c'满足：

当 $\frac{k}{n} \le c < \frac{k+1}{n}$ 时，$c' = v_k$，当 c=1 时$c' = v_{n-1}$

看例子：

```
<feFuncG type="discrete" tableValues="1 1 0 0"></feFuncG>
```

只要输入的绿色通道颜色小于 0.5，输出颜色就是 1，只要输入的绿色通道颜色大于或等于 0.5，输出颜色就是 0。

"linear"定义线性转换函数：$c' = slope \times c + intercept$，slope 为直线的斜率，由 slope 属性定义，默认之为 1，intercept 为偏移量，由 intercept 属性定义，默认之为 0。如果胜率 slope 和 intercept 属性，"linear"就等价于"identity"。

"gamma"定义指数转换函数：$c' = amplitude \times c^{exponent} + offset$，其中 amplitude、exponent 和 offset 为三个属性，默认值分别为 1、1 和 0，如果全部省略这三个属性，指数函数就退化为$c' = c$了（即"identity"）。看例子：

```
<feFuncB type="gamma" amplitude="4" exponent="2" offset="0"></feFuncB>
```

对于蓝色通道，本例的转换函数为 $c' = 4 \times c^2$。

下面举一个综合的例子，代码参见随书配套资源 chapter-4\svg.html 文件，浏览此文件看到的效果如图 4.239。

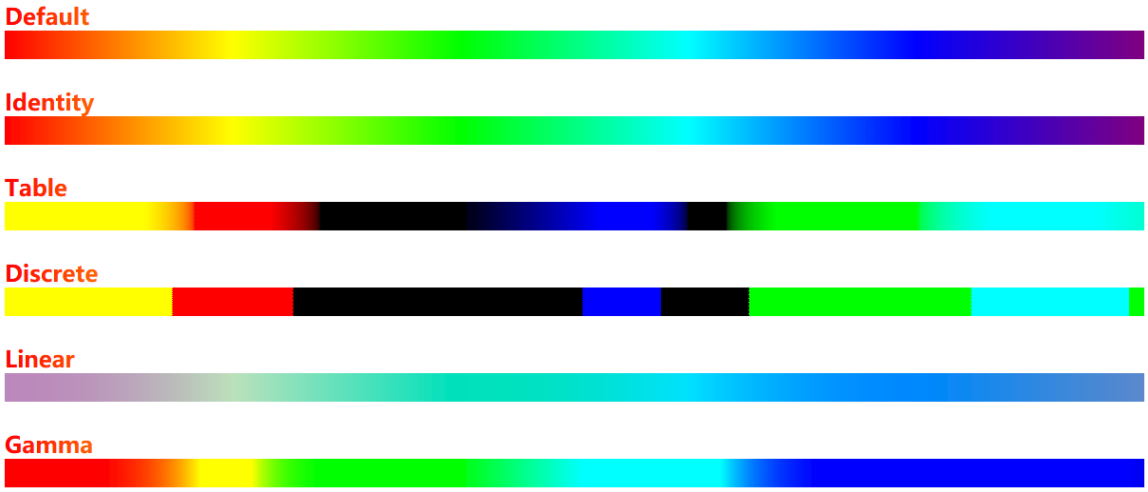


图 4.239 滤镜——映射综合举例

7、 合成-<feComposite>

<feComposite>原子滤镜使用 Porter-Duff 合成操作（over、in、atop、out 和 xor）之一在图像空间中按像素执行两个输入图像的合成。另外，也可以对各颜色分量做算术运算（结果限制在[0..1]之间）。算术运算对于将<feDiffuseLighting>和<feSpecularLighting>滤镜的输出与纹理数据结合在一起很有用。 <feComposite>标签的语法如下：

```
<feComposite in="" in2="" operator=""[ k1="" k2="" k3="" k4="" ]/>
```

属性 in 和 in2 参考<feBlend>标签的同名属性，属性 operator 定义合成操作，有以下几种操作方法：

"over" – 源图（由 in 属性指定）置于目标图（由 in2 属性指定）之上。这是默认值。

"in" –源图中与目标图不重叠的部分消失。

"out" – 源图中与目标图重叠的部分消失，效果类似于目标图在源图之上。

"atop" – 源图中与目标图不重叠的部分消失。

"xor" – 重叠部分做异或运算，效果类似"out"。

"arithmetic" – 源图和目标图按照下面的公式组合：

result = k1*i1*i2 + k2*i1 + k3*i2 + k4

其中 i1 和 i2 表示输入图像的相应像素通道值，输入图像分别由 in 和 in2 属性定义。k1、k2、k3 和 k4 为四个同名属性的值，这些属性的默认值为 0。

看一个综合例子：

```
<filter id="Over" x="0%" y="0%" width="100%" height="100%">
     <feImage href="dst-blue.png" result="dst" />
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
     <feComposite in="SourceGraphic" in2="dst" operator="over" />
</filter>
<image href="src-red.png" filter="url(#Over)" />

<filter id="In" x="0%" y="0%" width="100%" height="100%">
     <feImage href="dst-blue.png" result="dst" />
     <feComposite in="SourceGraphic" in2="dst" operator="in" />
</filter>
<image href="src-red.png" filter="url(#In)" x="120" />

<filter id="Out" x="0%" y="0%" width="100%" height="100%">
     <feImage href="dst-blue.png" result="dst" />
     <feComposite in="SourceGraphic" in2="dst" operator="out" />
</filter>
<image href="src-red.png" filter="url(#Out)" x="240" />

<filter id="Atop" x="0%" y="0%" width="100%" height="100%">
     <feImage href="dst-blue.png" result="dst" />
     <feComposite in="SourceGraphic" in2="dst" operator="atop" />
</filter>
<image href="src-red.png" filter="url(#Atop)" x="360" />

<filter id="Xor" x="0%" y="0%" width="100%" height="100%">
     <feImage href="dst-blue.png" result="dst" />
     <feComposite in="SourceGraphic" in2="dst" operator="xor" />
</filter>
<image href="src-red.png" filter="url(#Xor)" x="480" />

<filter id="Arithmetic" x="0%" y="0%" width="100%" height="100%">
     <feImage href="dst-blue.png" result="dst" />
     <feComposite in="SourceGraphic" in2="dst" operator="arithmetic"
k2="3" k3="90" />
</filter>
<image href="src-red.png" filter="url(#Arithmetic)" x="600" />
```

浏览效果见图 4.240 上半部分，源图和目标图都是半透明，背景全透明。再修改源图和背景图为不透明，然后刷新页面，就会看到图 4.240 的下半部分。不透明红色的像素颜色值为 RGBA=[255,0,0,1]，蓝色为 RGBA=[0,0,255,1]，半透明红色为 RGBA=[255,0,0,0.5]，蓝色为 RGBA=[0,0,255,0.5]，大家根据这些数字去好好理解本例子。

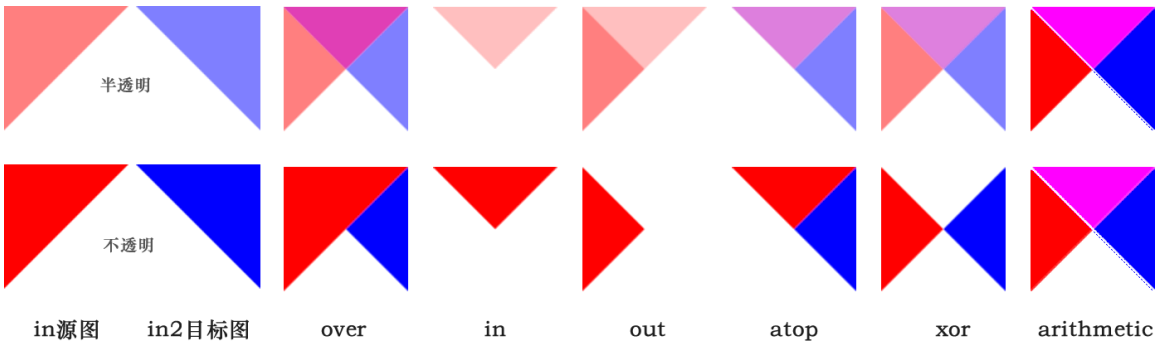


图 4.240 <feComposite>标签用法举例

8、 矩阵卷积-<feConvolveMatrix>

<feConvolveMatrix>原子滤镜产生矩阵卷积滤镜效果，卷积将输入图像中的每个像素与它相邻像素合并计算以产生结果图像。 通过卷积可以实现各种各样的成像操作，包括模糊、边缘检测、锐化、浮雕和倒角。

矩阵卷积使用一个n × m矩阵（简称核矩阵），该矩阵描述输入图像中给定像素值如何与其相邻像素值组合以产生结果像素值。适用于给定像素（假设在图片中的坐标为(X,Y)）的每个颜色通道的值的基本卷积公式为：

$$COLOR_{X,Y} = \left(\sum_{i=0}^{orderY-1} \sum_{j=0}^{orderX-1} SOURCE_{X-targetX+j,Y-targetY+i} \times kernelMatrix_{orderX-j-1,orderY-i-1} \right) \div divisor + bias \times ALPHA_{X,Y}$$

其中 orderX、orderY、targetX、targetY、kernelMatrix、divisor 和 bias 均为同名标签属性的值，$SOURCE_{x,y}$ 表示源图中坐标为(x,y)的像素的颜色值，$ALPHA_{x,y}i$ 表示相应像素的 alpha 通道的值。

<feConvolveMatrix>标签的使用语法如下：

```
<feConvolveMatrix kernelMatrix=""[ in="" order="" divisor="" bias="" targetX="" targetY="" edgeMode="" preserveAlpha="" ]/>
```

对于这些属性的理解参考下图 4.241。图片由像素组成，每个像素的颜色采用 RGBA 表示，矩阵卷积计算就是对于图片的每一个像素的每一个颜色通道采用上面的公式计算结果颜色通道值，这时周围的像素也参与计算。“周围”的像素具体是哪些呢？由该矩阵确定。矩阵卷积计算时，先旋转核矩阵 180 度，然后将核矩阵中坐标为(targetX,targetY)的点盖在图片中的目标像素上（不妨假设目标像素的坐标为(X,Y)，即下图中的红色小方框内的像素），这样核矩阵盖住的图片像素就是要参与计算的周围像素了——把重叠在一起的点相乘后相加。

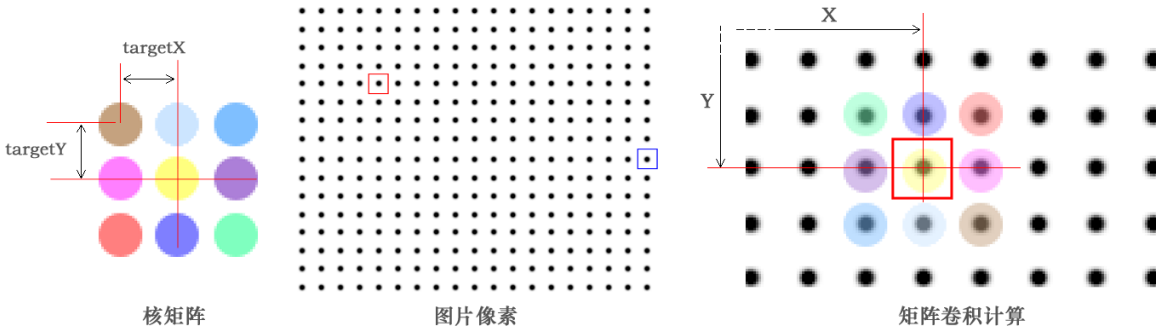


图 4.241 <feConvolveMatrix>标签的用法说明

order 属性定义核矩阵的行数和列数，值的格式为"m[n]"，如果只有一个数字，表示行/列数相同，如果是两个数字，前面的数字表示列数，后面的表示行数，比如 order= "4 3"表示 3 行 4 列。如果省略 order 属性，核矩阵默认为 3 行 3 列。核矩阵不应该太大，否则非常消耗 CPU 资源，而且太大效果不一定会城北增加。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

kernelMatrix 属性定义核矩阵中每个元素的值，用空格或逗号分开，元素的个数要等于 order 属性定义的行/列数的乘积，比如 order="3" kernelMatrix="1 1 0 0 0 0 0 0 -1"。

属性 targetX 和 targetY 定义核矩阵中要盖住图片目标像素的那个元素的下标，下标从 0 开始，参见上图所示。默认值为$\left\lfloor\frac{orderX}{2}\right\rfloor$、$\left\lfloor\frac{orderY}{2}\right\rfloor$。

当目标像素靠近图片边沿时（比如图 4.241 中蓝色小方框内的像素），计算时核矩阵的部分元素可能超出了图片的范围，这时由 edgeMode 属性确定超出部分的元素与哪些像素相乘，此属性的取值有："duplicate"（默认值）- 直接与临近边界上的像素相乘，"wrap" – 与对面边界上的像素相乘，"none" – 乘以 0。参考图 4.242。

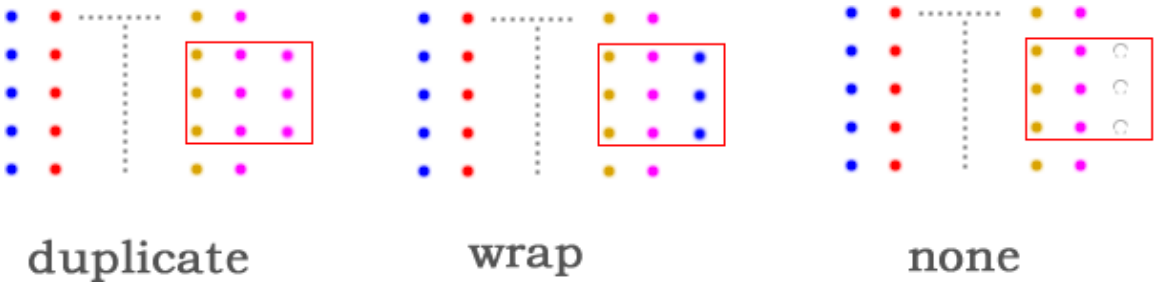


图 4.242 <feConvolveMatrix>标签的 edgeMode 属性

divisor 属性定义上面卷积公式中的除数，默认值为核矩阵中全部元素的累加和，如果累加和为 0，那么默认值就是 1。

preserveAlpha 属性定义要不要对 alpha 通道做同样的卷积计算，取值"true"时不改变 alpha 通道，"false"对 alpha 通道作同样的卷积计算。

bias 定义偏移量，属性默认为 0。

示意图 4.243 展示了卷积计算过程（假设其他属性都采用默认值）。

```
 1 2 3
 4 5 6
 7 8 9
```

核矩阵

```
0    20  40 235 235
100 120 140 235 235
200 220 240 235 235
225 225 255 255 255
225 225 255 255 255
```

图片像素的R通道

```
(9*  0 + 8* 20 + 7* 40 +
 6*100 + 5*120 + 4*140 +
 3*200 + 2*220 + 1*240) / (9+8+7+6+5+4+3+2+1)
```

矩阵卷积计算

图 4.243 卷积计算过程

下面看一个综合例子：

```
<defs>
     <filter id="blurry">
          <!--模糊-->
          <feConvolveMatrix kernelMatrix="1 1 1 1 1 1 1 1 1" edgeMode="none" />
     </filter>
     <filter id="sharp">
          <!--尖锐-->
          <feConvolveMatrix kernelMatrix="0 -2 0 -2 15 -2 0 -
2 0" edgeMode="none" />
     </filter>
     <filter id="Marginalization">
```

```
          <!--边沿化-->
          <feConvolveMatrix kernelMatrix="1 1 1 1 -6 1 1 1 1" edgeMode="none" />
     </filter>
     <filter id="Relief">
          <!--浮雕-->
          <feConvolveMatrix kernelMatrix="-2 -1 0 -
1 1 1 0 1 2" edgeMode="none" />
     </filter>
</defs>

<image href="demo.png" x="304" y="210" />
<image href="demo.png" filter="url(#blurry)" x="304" />
<image href="demo.png" filter="url(#sharp)" y="210" />
<image href="demo.png" filter="url(#Marginalization)" x="608" y="210" />
<image href="demo.png" filter="url(#Relief)" x="304" y="420" />
```

浏览的效果见图 4.244。

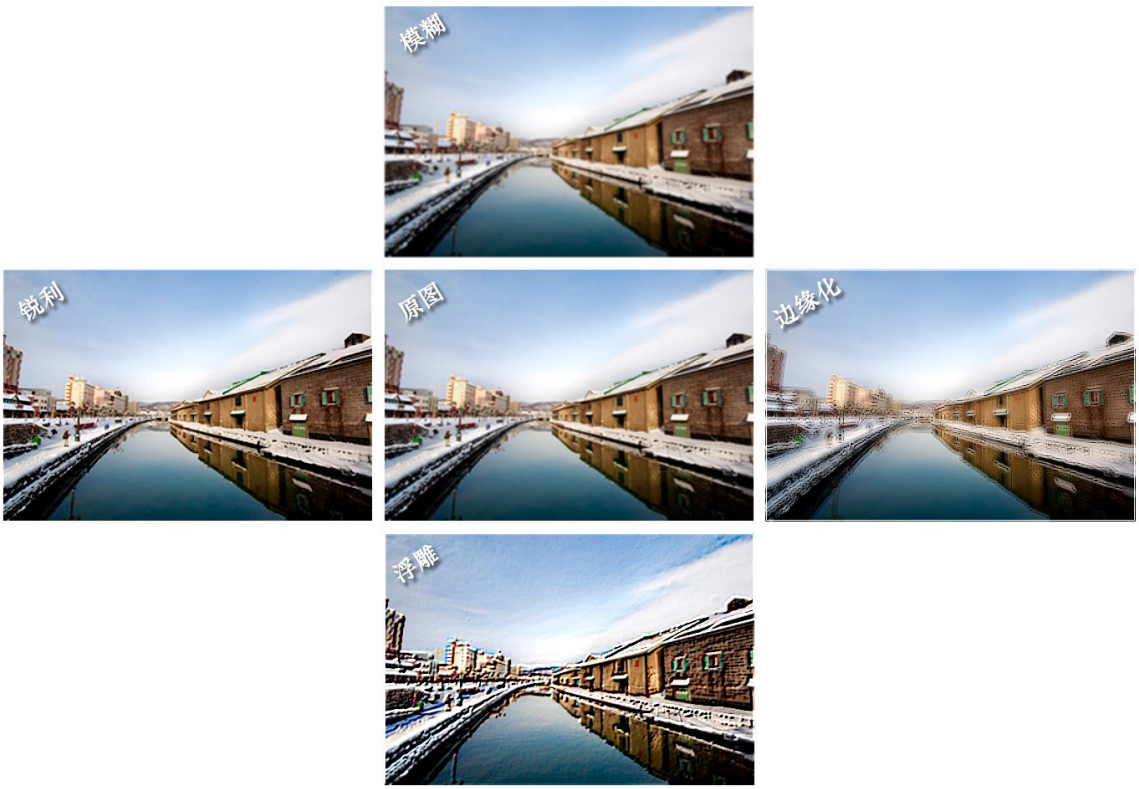


图 4.244 <feConvolveMatrix>标签综合举例

9、 聚光灯-<feSpotLight>

<feSpotLight>原子滤镜定义一个光源，从而产生聚光灯效果，一般作为原子滤镜<feDiffuseLighting>或<feSpecularLighting>的子标签。它的使用语法如下：

```
<feConvolveMatrix x="" y="" z="" pointsAtX="" pointsAtY="" pointsAtZ=""
specularExponent="" limitingConeAngle="" />
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

(x,y,z)为光源在三维空间中的坐标，z 轴垂直屏幕并指向人的眼睛，(pointsAtX,pointsAtY,pointsAtZ)为照射点的坐标，注意：坐标属性的值的含义还与祖先标签<filter>的 primitiveUnits 属性有关。specularExponent 属性控制光源的焦点，值越大，光线越亮。limitingConeAngle 属性定义聚光灯轴（即光源与其指向的点之间的轴）与聚光灯锥之间的角度，以度为单位，因此，它定义了一个限制锥，没有光投射到圆锥体外部。对这些参数的进一步理解请参见下图 4.245。

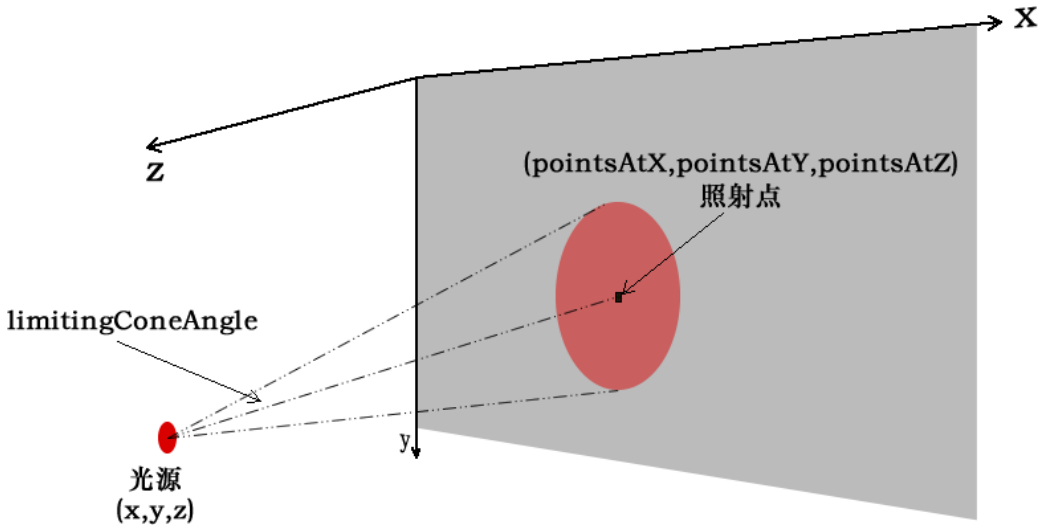


图 4.245 <feSpotLight>标签用法说明

看例子：

```
<filter id="spotLight1" x="0" y="0" width="100%" height="100%">
     <feDiffuseLighting diffuseConstant="2">
          <feSpotLight x="10" y="10" z="50" pointsAtX="100" pointsAtY="100" lim
itingConeAngle="10" />
     </feDiffuseLighting>
</filter>
<filter id="spotLight2" x="0" y="0" width="100%" height="100%">
     <feDiffuseLighting diffuseConstant="2">
          <feSpotLight x="10" y="10" z="50" pointsAtX="100" pointsAtY="100" lim
itingConeAngle="30" />
     </feDiffuseLighting>
</filter>

<rect x="0" y="0" width="200" height="200" style="filter: url(#spotLight1);" />
<rect x="0" y="0" width="200" height="200" style="filter: url(#spotLight2); tra
nsform: translateX(220px);" />
```

浏览的效果如图 4.246。

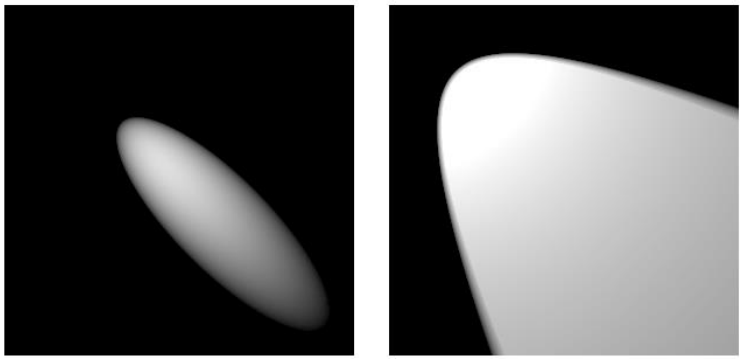


图 4.246 <feSpotLight>标签用法举例

10、 点光源-<fePointLight>

<fePointLight>原子滤镜定义一个光源，该光源可以创建点光源效果，一般作为原子滤镜<feD<feDiffuseLighting>或<feSpecularLighting>的子标签。它的使用语法如下：

```
<feConvolveMatrix x="" y="" z="" />
```

(x,y,z)为光源在三维空间中的坐标，z 轴垂直屏幕并指向人的眼睛。坐标属性的值的含义还与祖先标签<filter>的 primitiveUnits 属性有关。看例子：

```
<filter id="Pointlight">
    <feSpecularLighting result="pointlight" specularConstant="1.5" specularExpo
nent="80" lighting-color="#FFF">
        <fePointLight x="50" y="50" z="150" />
    </feSpecularLighting>
    <feComposite in="SourceGraphic" in2="pointlight" operator="arithmetic" k1="
0" k2="1" k3="1" k4="0" />
</filter>
<rect width="200" height="200" filter="url(#Pointlight)" />
```

浏览效果如图 4.247。

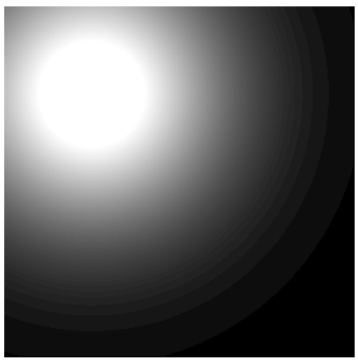


图 4.247 <fePointLight>标签用法举例

11、 散射光照-<feDiffuseLighting>

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

<feDiffuseLighting>原子滤镜使用 alpha 通道作为凹凸贴图来照亮图像。生成的图像是 RGBA 不透明图像，图形效果取决于输入的凹凸贴图的灯光颜色、灯光位置和表面几何形状。可使用<feComposite>原子滤镜的乘法运算（operator="arithmetic"）将<feDiffuseLighting>生成的光照贴图与纹理图像组合起来。 在将多个光照贴图应用于纹理图像之前，可以将这些光照贴图相加，从而模拟多个光源。它的使用语法如下：

```
<feDiffuseLighting in="" surfaceScale="" diffuseConstant="" />
```

surfaceScale 属性表示原子滤镜的表面高度，默认值为 1。diffuseConstant 属性表示 Phong 照明模型中的 k_d 值（k_d 用于确定给定像素的最终 RGB 值），可取任何非负数，照明颜色越亮，该数字应该越小。关于 Phong 照明模型的介绍请参考这个网站 http://en.wikipedia.org/wiki/Phong_reflection_model。看例子：

```
<filter id="diffuseLighting1" x="0" y="0" width="100%" height="100%">
    <feDiffuseLighting in="SourceGraphic" diffuseConstant="1">
        <fePointLight x="60" y="60" z="20" />
    </feDiffuseLighting>
</filter>
<filter id="diffuseLighting2" x="0" y="0" width="100%" height="100%">
    <feDiffuseLighting in="SourceGraphic" diffuseConstant="1.5">
        <fePointLight x="60" y="60" z="20" />
    </feDiffuseLighting>
</filter>

<rect x="0" y="0" width="200" height="200" style="filter: url(#diffuseLighting1
);" />
<rect x="0" y="0" width="200" height="200" style="filter: url(#diffuseLighting2
); transform: translateX(220px);" />
```

浏览的效果如图 4.248。

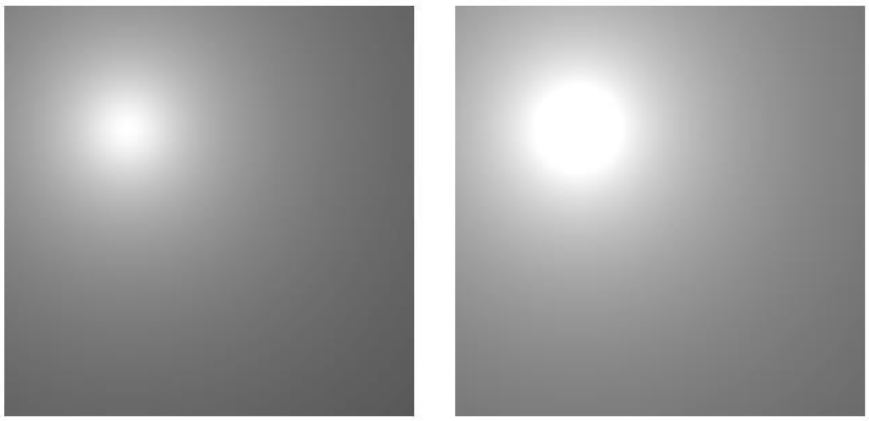


图 4.248 <feDiffuseLighting>标签用法举例

12、 Perlin 湍流-<feTurbulence>

<feTurbulence>原子滤镜使用 Perlin 湍流功能创建图像，它可以合成人造纹理，如云层或大理石，生成的图像将填充整个原子滤镜子区域。其使用语法如下：

```
<feTurbulence baseFrequency="" numOctaves="" seed="" stitchTiles="" type=""/>
```

baseFrequency 属性定义基础频率，默认为 0。numOctaves 属性定义八度数，默认为 1。八度是一种由频率和幅度定义的噪声函数，通过积累几个八度音来建立湍流，而这些八度音的频率会不断增加而幅度不断减小。八度的数量越多，噪声看起来越自然，但是计算机资源的消耗也越大。seed 属性定义伪随机数生成器的起始数字，默认为 0。spellTiles 属性定义 Perlin 噪波图块在边框处的行为，可取值："noStitch"（默认值）- 边框处的图块不作任何处理，这样会产生清晰的边界线，"Stitch"– 处理边界土块，使边界间形成过度。type 属性指示本原子滤镜执行噪声函数还是湍流函数，分别对应取值"fractalNoise"和"turbulence"，默认是"turbulence"。

看例子：

```
<filter id="Noise1" x="0" y="0" width="100%" height="100%">
    <feTurbulence baseFrequency="0.025" type="fractalNoise" />
</filter>
<filter id="Noise2" x="0" y="0" width="100%" height="100%">
    <feTurbulence baseFrequency="0.05" type="fractalNoise" />
</filter>
<filter id="Turb1" x="0" y="0" width="100%" height="100%">
    <feTurbulence baseFrequency="0.025" />
</filter>
<filter id="Turb2" x="0" y="0" width="100%" height="100%">
    <feTurbulence baseFrequency="0.05" />
</filter>

<rect x="0" y="0" width="200" height="200" style="filter: url(#Noise1);" />
<rect x="0" y="0" width="200" height="200" style="filter: url(#Noise2); transfo
rm: translateX(220px);" />
<rect x="0" y="0" width="200" height="200" style="filter: url(#Turb1); transfor
m: translateX(440px);" />
<rect x="0" y="0" width="200" height="200" style="filter: url(#Turb2); transfor
m: translateX(660px);" />
```

浏览的效果见图 4.249。

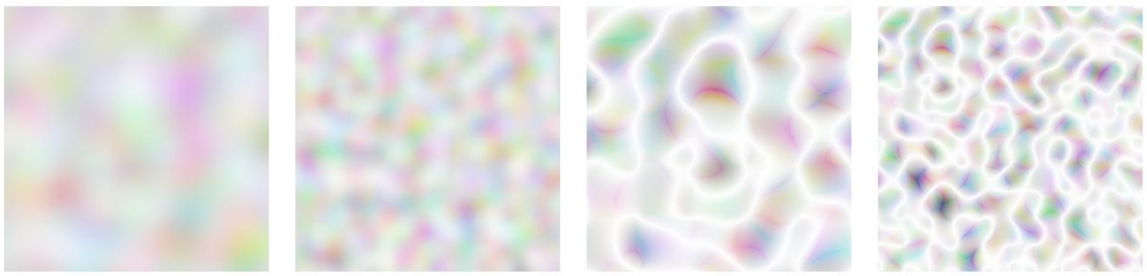


图 4.249 <feTurbulence>标签用法举例

13、 移位-<feDisplacementMap>

<feDisplacementMap>原子滤镜使用 in2 属性指定的图像的颜色通道对 in 属性指定的源图像素进行移位。对于源图中的的P(x,y)像素，移位后变为：

$$P'\big(x+scale\times(XC(x,y)-0.5),y+scale\times(YC(x,y)-0.5)\big)$$

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

其中 XC（x，y）和 YC（x，y）是 xChannelSelector 和 yChannelSelector 属性指定的颜色通道的值（来自 in2 属性定义的图形）。其使用语法如下：

```
<    feDisplacementMap    in=""    in2=""    scale=""    xChannelSelector=""
yChannelSelector=""/>
```

xChannelSelector 和 yChannelSelector 属性指定 in2 中图形的颜色通道，可取值"R"、"G"、"B" 和"A"，默认值为"A"。

看例子：

```
<filter id="Dis1" x="-20%" y="-20%" width="140%" height="140%">
    <feTurbulence baseFrequency="0.05" numOctaves="2" result="turbulence" />
    <feDisplacementMap in2="turbulence" in="SourceGraphic" scale="10" />
</filter>
<filter id="Dis2" x="-20%" y="-20%" width="140%" height="140%">
    <feTurbulence baseFrequency="0.05" numOctaves="2" result="turbulence" />
    <feDisplacementMap in2="turbulence" in="SourceGraphic" scale="30" />
</filter>

<image id="mv" href="meinv.png" />
<use href="#mv" style="filter: url(#Dis1); transform: translateX(200px);" />
<use href="#mv" style="filter: url(#Dis2); transform: translateX(400px);"/>
```

浏览的效果如图 4.250。



图 4.250 <feDisplacementMap>标签用法举例

14、 高斯模糊-<feGaussianBlur>

<feGaussianBlur>原子滤镜对图形做模糊处理，其使用语法如下：

```
<feGaussianBlur in="" stdDeviation="" edgeMode=""/>
```

stdDeviation 属性定义模糊偏移量，默认值为 0，值的格式为"offsetX[offsetY]"，offsetX 为 x 轴方向的偏差，offsetY 为 y 轴的偏差，如果只有一个数字，那么 x、y 轴的偏差相同。edgeMode 属性与<feConvolveMatrix>标签的同名属性含义相同。看例子：

```
<filter id="Gs1">
    <feGaussianBlur stdDeviation="1" />
</filter>
<filter id="Gs2">
    <feGaussianBlur stdDeviation="5" />
```

```
</filter>

<image href="meinv.png" style="filter: url(#Gs1);transform: translateY(20px);"
/>
<image href="meinv.png" style="filter: url(#Gs2);transform: translate(200px,20p
x);" />
```

浏览的效果如图 4.251。



图 4.251 <feGaussianBlur>标签用法举例

15、 偏移-<feOffset>

<feOffset>原子滤镜对图形作偏移处理。其使用语法如下：

```
<feOffset in="" dx="" dy=""/>
```

属性 dx、dy 定义 x 和 y 轴上的偏移量。看例子：

```
<defs>
    <filter id="Offset" width="180" height="180">
        <feOffset in="SourceGraphic" dx="60" dy="60" />
    </filter>
</defs>

<rect x="0" y="0" width="100" height="100" stroke="black" fill="green" />
<rect x="0" y="0" width="100" height="100" stroke="black" fill="green" filter="
url(#Offset)" />
```

16、 并发-<feMerge>/<feMergeNode>

<feMerge>标签允许同时应用滤镜效果，而不是顺序应用滤镜效果，这是因为其他过滤器通过 result 属性存储它的输出，而在<feMergeNode>子标签中对其进行访问来实现的。<feMerge>结合若干<feMergeNode>相当于建立了多个显示管道。它们的使用语法如下：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    <feMerge>
        <feMergeNode in=""/>
        ……
        <feMergeNode in=""/>
    </feMerge>
```

我们来看一个例子：

```
<filter id="Test" filterUnits="objectBoundingBox" x="0" y="0" width="1.5" heigh
t="4">
    <feOffset result="Off1" dx="15" dy="20" />
    <feFlood style="flood-color:#ff0000;flood-opacity:0.8" />
    <feComposite in2="Off1" operator="in" result="C1" />
    <feOffset in="SourceGraphic" result="Off2" dx="30" dy="40" />
    <feFlood style="flood-color:#00ff00;flood-opacity:0.4" />
    <feComposite in2="Off2" operator="in" result="C2" />
    <feMerge>
        <feMergeNode in="SourceGraphic" />
        <feMergeNode in="C1" />
        <feMergeNode in="C2" />
</feMerge>
</filter>
</defs>
<text x="30" y="100" style="font:36px verdana bold;fill:blue;filter:url(#Test)">
    您好，SVG ！
</text>
```

浏览的效果如图 4.252。

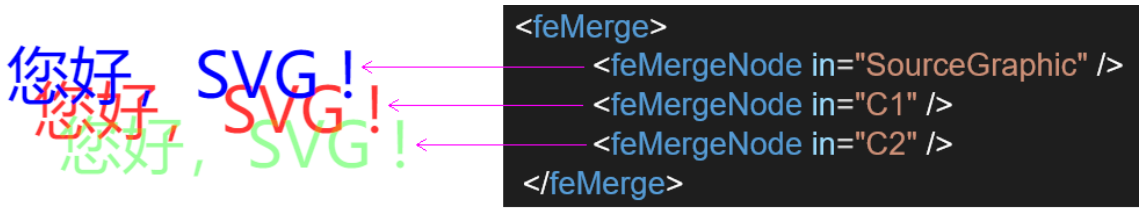


图 4.252 <feMerge>标签用法举例

17、 腐蚀-<feMorphology>

<feMorphology>原子滤镜用于腐蚀或扩大输入图像，尤其在于增脂或稀释方面非常有用。其使用语法如下：

```
<feMorphology in="" operator="" radius=""/>
```

operator 定义行为，即"erode"（腐蚀）还是"dilate"（扩张/淡化），默认是"erode"。radius 属性定义半径，值的格式为"n[m]"，n 为 x 轴方向的半径，m 为 y 轴的半径，如果只有一个数字，那么 x、y 轴的半径相同，radius 含义还与祖先标签<filter>的 primitiveUnits 属性有关。

看例子：

```
<filter id="Erode">
   <feMorphology operator="erode" radius=".45" />
</filter>
<filter id="Dilate">
```

```
   <feMorphology operator="dilate" radius="0.9" />
</filter>
<text y="1em">Hello 世界! </text>
<text y="2em" filter="url(#Erode)">Hello 世界! </text>
<text y="3em" filter="url(#Dilate)">Hello 世界! </text>
```

浏览的效果如图 4.253。



图 4.253 <feMorphology>标签用法举例

18、 光照-<feSpecularLighting>

<feSpecularLighting>原子滤镜使用 alpha 通道作为凹凸贴图来照亮源图形，照明计算遵循 Phong 照明模型的标准镜面反射分量。生成的图像是基于浅色的 RGBA 图像，其中包含照明计算的镜面反射部分，实际效果与输入凹凸贴图的灯光颜色、灯光位置和表面几何形状有关。本原子滤镜假定查看者在 z 方向上处于无穷远的位置。

可使用<feComposite>原子滤镜的加法运算（operator="arithmetic"）将此类贴图与纹理组合。 在将多个光照贴图应用于纹理图像之前，可以将这些光照贴图相加，从而模拟多个光源。

其使用语法如下：

```
<feMorphology in="" surfaceScale="" specularConstant="" specularExponent="" />
```

surfaceScale 属性表示原子滤镜的表面高度，默认值为 1。specularConstant 属性控制镜面照明的反射率，它表示 Phong 照明模型中的 k_s 值，值越大，反射越强。specularExponent 属性控制光源的焦点，值越大，光线越亮。例子参见<fePointLight>标签。

19、 平铺-<feTile>

<feTile>原子滤镜通过平铺图像来填充目标矩形，目标矩形由 x、y、width 和 height 属性定义，效果类似于<pattern>。其语法如下：

```
<feTile in="" x="" y="" width="" height=""/>
```

属性都可以省略，这时就去默认值。看例子：

```
<defs>
    <filter id="Tile" primitiveUnits="objectBoundingBox" x="0" y="0" width="1"
height="1">
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
        <feImage href="hel.png" width=".2" height=".4" />
        <feTile />
    </filter>
</defs>
<rect x="30" y="30" width="400" height="200" filter="url(#Tile)" />
<rect x="30" y="30" width="400" height="200" stroke="#00f" fill="none" />
```

<feImage>原子滤镜的输出作为<feTile>原子滤镜的输入,把100*100的图片像贴瓷砖一样平铺,浏览的效果如图 4.254。



图 4.254 <feTile>标签用法举例

4.10.7.9 其他标签

<svg>元素的其他标签见表 4.27。

表 4.27 <svg>元素的其他标签

标签名	简要说明
<script>	包裹 JS 代码
<style>	包裹样式表
<view>	定义浏览图形的方法，比如大图、缩略图等
<clipPath>	定义剪切路径，其他标签通过 clip-path 属性引用。剪切路径限制了绘画的区域，区域之外的图形看不到
<foreignObject>	引用另一个 XML 名字空间中的标签

1、 嵌入脚本-<script>

<script>标签在 SVG 文档中嵌入脚本代码，在同一个文档中的任何<script>标签内的函数和变量都是全局的。其使用语法如下：

```
<script crossorigin="" href="" type=""/>
```

或

```
<script crossorigin="" type="">脚本代码</script>
```

属性 crossorigin、href、type 分别与 HTML5 的元素<script>的属性 crossorigin、src 和 type 含义相同。看例子：

```
<svg width="800" height="600" version="1.1" xmlns="http://www.w3.org/2000/svg">
<script>
   window.addEventListener('DOMContentLoaded', () => {
      function getColor() {
         const R = Math.round(Math.random() * 255).toString(16).padStart(2, '0')
         const G = Math.round(Math.random() * 255).toString(16).padStart(2, '0')
         const B = Math.round(Math.random() * 255).toString(16).padStart(2, '0')
         return `#${R}${G}${B}`
      }

      document.querySelector('circle').addEventListener('click', (e) => {
         e.target.style.fill = getColor()
      })
   })
</script>

<circle cx="50" cy="50" r="40" />
</svg>
```

当初始的 HTML 或 svg 文档被完全加载和解析完成之后，DOMContentLoaded 事件被触发，而无需等待样式表、图像和子框架完成加载，所以说发生在 loaded 事件之前。例子中的“()=>{ … }”为匿名函数“function(){ … }”的另一种写法。在浏览器中打开此例子，会显示一个圆，当鼠标点击圆时改变颜色。

2、 嵌入样式-<style>

<style>标签在 SVG 文档中嵌入样式代码，其使用语法如下：

```
<style type="" media="" title="">样式代码</style>
```

三个属性请参考 HTML5 的元素<style>的同名属性。看例子：

```
<svg width="800" height="600" version="1.1" xmlns="http://www.w3.org/2000/svg">
    <style>
        path {
            fill: gold;
            stroke: maroon;
            stroke-width: 10px;
        }
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    </style>
    <path d="M10,30 A20,20,0,0,1,50,30 A20,20,0,0,1,90,30 Q90,60,50,90 Q10,60,1
0,30 Z" />
</svg>
```

3、 浏览图形-<view>

<view>标签定义一种查看图像的方式，例如缩放，它实际上就是修改父标签<svg>的 vieBox 属性的值。其使用语法如下：

```
<view viewBox="" preserveAspectRatio=""/>
```

两个属性请参考<svg>的同名属性。把下面代码单独保存在 mySvg.xml 文件中，然后在浏览器中打开它，点击“放大 2 倍”看看效果。

```
<svg width="600" height="200" viewBox="0 0 600 200" xmlns="http://www.w3.org/20
00/svg" xmlns:xlink="http://www.w3.org/1999/xlink">
    <defs>
        <radialGradient id="gradient">
            <stop offset="0%" stop-color="#8cffa0" />
            <stop offset="100%" stop-color="#8ca0ff" />
        </radialGradient>
    </defs>
    <circle r="50" cx="180" cy="50" style="fill:url(#gradient)" />
    <view id="halfSizeView" viewBox="0 0 1200 400" />
    <view id="normalSizeView" viewBox="0 0 600 200" />
    <view id="doubleSizeView" viewBox="0 0 300 100" />
    <a href="#halfSizeView">
        <text x="5" y="20" font-size="20">缩小一半</text>
    </a>
    <a href="#normalSizeView">
        <text x="5" y="40" font-size="20">正常大小</text>
    </a>
    <a href="#doubleSizeView">
        <text x="5" y="60" font-size="20">放大 2 倍</text>
    </a>
</svg>
```

4、 剪贴作画-<clipPath>

<clipPath>标签定义一条剪切路径，剪切路径限制了可以施加涂料的区域，因此剪切路径所限定的区域之外的图形不会显示。其他标签由 clip-path 属性引用剪贴路径。其使用语法如下：

```
<clipPath clipPathUnits="" id=""/>
```

属性 clipPathUnits 定义内部子标签的坐标系，可取值"userSpaceOnUse"、"objectBoundingBox"，默认值为"objectBoundingBox"，各个值的含义参见<mask>标签。看例子：

```
<defs>
    <clipPath id="myClip">
        <path d="M10,30 A20,20,0,0,1,50,30 A20,20,0,0,1,90,30 Q90,60,50,90 Q10,
60,10,30 Z" />
    </clipPath>
```

```
    <linearGradient id="Grad" y2="1.5%" x2="0" spreadMethod="repeat">
        <stop offset="0" stop-color="black" />
        <stop offset="0.5" stop-color="black" />
        <stop offset="0.5" stop-color="white" />
        <stop offset="1" stop-color="white" />
    </linearGradient>
</defs>

<rect width="400" height="400" fill="url(#Grad)" clip-path="url(#myClip)" />
```

浏览的效果见图 4.255。



图 4.255 <clipPath>标签用法举例

5、 嵌入外部内容-<foreignObject>

<foreignObject> 标签包裹来自不同 XML 名称空间的标签（由 xmlns 属性指定），在浏览器的上下文中，也可能是（X）HTML。目前主要的用途包括：嵌入多行文字、页面截图。其使用语法如下：

```
    <foreignObject x="" y="" width="" height="">被嵌入的内容</foreignObject>
```

下面的例子嵌入多行文字：

```
<style>
   div {
      color: white;
      font: 18px serif;
      height: 100%;
      overflow: auto;
   }
</style>

<polygon points="5,5 195,10 185,185 10,195" />
<foreignObject x="20" y="20" width="160" height="160">
   <div xmlns="http://www.w3.org/1999/xhtml">
      许多人认为，要赢得他人的忠诚，最好的办法是给其恩惠。其实，这是对人性的误解，在现实中真正对你忠诚的，都是曾经给过你恩惠的人。因为乐善好施的人心地不坏。
   </div>
</foreignObject>
```

svg 的名字空间 xmlns="http://www.w3.org/2000/svg，而例子中被嵌入的 的名字空间为 xmlns="http://www.w3.org/1999/xhtml"。本例浏览的结果如图 4.256。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

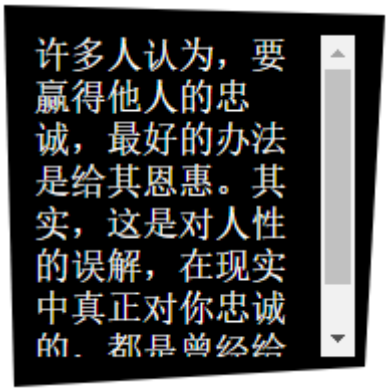


图 4.256 <foreignObject>标签用法举例

4.10.8 嵌入外部资源

使用<object>元素（有时结合<param>元素）嵌入外部资源，如图片、pdf 文件、Flash、Java applets、ActiveX 和多媒体等，建议不要使用另一个非标准化的类似元素<embed>。<object>元素的使用语法如下：

```
<object data="" form="" height="" width="" name="" type="" typemustmatch usemap
="">
    [<param name="" value="">
      ……
     <param name="" value="">
     浏览器不支持]
</object>
```

其中<param>元素给处理外部资源的插件提供参数，name 属性的值为参数名称，value 属性的值为参数的值。如果给播放音频的插件提供参数 loop=true：

```
<param name="loop" value="true">
```

<object>元素的 data 属性定义外部资源的有效 URL，data 和 type 不能全部省略，属性 typemustmatch 表明 type 属性定义的类型必须与 data 属性提供资源一致。form 属性参考<input>元素的同名属性，type 属性参考<link>的同名属性，usemap 属性参考“§4.5.2 创建图片热点”小节中的<img>元素。看例子：

```
<object data="beijing.mp3">
     <param name="controls" value="true">
     <param name="loop" value="true">
     浏览器不支持
</object>
<object data="机动车商业保险示范条款.pdf" width="600px" height="300px"></object>
```

本例子嵌入了音频和 PDF 两个外部资源，浏览效果如图 4.258。



图 4.258 <object>元素用法举例

4.11 表格元素

“HTML5 元素速查表 • 表格”分类元素可以产生二维表格以满足我们对于表格的任何需求。

本节介绍这些元素：<table>、<tbody>、<caption>、<tr>、<td>、<th>、<thead>、<tfoot>、<colgroup>、<col>。

这些元素的相互关系见图 4.259 和图 4.260 所示。



图 4.259 表元素之间的关系

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

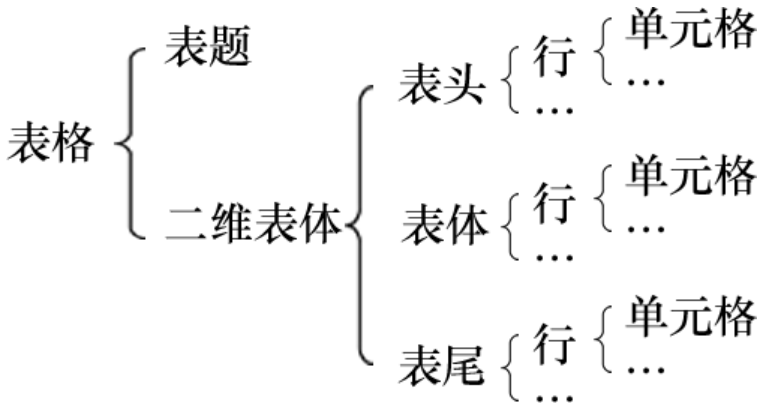


图 4.260 二位表结构

这些元素的联合用法如下：

```
<table>
    <caption>表题</caption>
    <colgroup[ span=""]>
         <col[ span=""]>
         <col[ span=""]>
         ……
    </colgroup>
     <thead>
          <tr>
              <th[ abbr="" colspan="" headers="" rowspan="" scope=""]>列名 1</th>
              <th[ abbr="" colspan="" headers="" rowspan="" scope=""]>列名 2</th>
              ……
          </tr>
          ……
     </thead>
     <tbody>
          <tr>
              <td[ colspan="" headers="" rowspan=""]>数据 1</td>
              <td[ colspan="" headers="" rowspan=""]>数据 2</td>
              ……
          </tr>
          ……
     </tbody>
     <tfoot>
          <tr>
              <td[ colspan="" headers="" rowspan=""]>数据 1</td>
              <td[ colspan="" headers="" rowspan=""]>数据 2</td>
              ……
          </tr>
          ……
     </tfoot>
</table>
```

元素<table>、<caption>、<thead>、<tbody>、<tfoot>、<tr>除了通用属性外没有专用属性，<thead>包裹表头，<tbody>包裹表体，<tfoot>包裹表尾，表头、表体和表尾统称为表段，这三个元素只能是<table>的子元素，且允许出现多个。另外<table>、<tr>和<td>三个元素不能省略外，其他元素允许省略，它们的作用更多侧重于语义方面。<tr>元素包裹表中的一行，<th>元素包裹表头中的一个列名（头部单元格），它的可选属性有：

abbr 属性 – 对本单元格内容的简要说明，便于读屏器、爬虫这种非可视化的 WEB 客户端识别。

colspan 属性 – 本单元横跨多少列（相当于合并单元格），值为正整数，默认为 1。

headers 属性 – 对于多行的表头，从第二行开始的单元格，用本属性指定隶属的头部单元格，值为用空格分开的 id。例如对于上图，“平时分”单元格隶属于“成绩”头部单元格，假如标注“成绩”单元格的<th>元素的 id 属性的值为"scores"，那么标注“平时分”的<th>元素的 headers 的值就是"scores"，参考图 4.261 所示。

助学金名单公示

姓名	学号	班级	成绩		
			平时分	期末分	综合分
王靖雯	2020090101	19应用1班	86	90	88.4
李茂升	2020090110	19网络2班	90	96	93.6
张百万	2020090111	19软件1班	92	85	87.8
一共3人					

<th id=”scores” …>…
<th id=”work” headers=”scores” …>…
<td headers=”scores work” …>…

图 4.261 表格元素的 headers 属性

rowspan 属性 – 本单元横跨多少行（相当于合并单元格），值为非负整数，默认为 1，如果设为 0，表示跨越整个表段（即<thead>、<tbody>或<tfoot>）。例如对于图 4.261 中的“姓名”头部单元格，可以设置为 rowspan=2 或者 rowspan=0。

scope 属性 – 定义本头部单元格“管辖”哪些数据单元格，取值可为"row"-管辖所在的行、"col"-管辖所在的列、"rowgroup"-管辖所在行组中的所有数据单元格、"colgroup"-管辖所在列组中的所有数据单元格和"auto"，默认是"auto"。请参看示意图 4.262。

scope=”col”

标题单元格	标题单元格
数据单元格	数据单元格
数据单元格	数据单元格
数据单元格	数据单元格

scope=”colgroup”

标题单元格		标题单元格	
标题单元格	标题单元格	标题单元格	标题单元格
数据单元格	数据单元格	数据单元格	数据单元格
数据单元格	数据单元格	数据单元格	数据单元格

scope=”rowgroup”

标题单元格		
数据单元格	数据单元格	数据单元格
数据单元格	数据单元格	数据单元格
标题单元格		
数据单元格	数据单元格	数据单元格
数据单元格	数据单元格	数据单元格

scope=”row”

标题单元格	数据单元格	数据单元格
标题单元格	数据单元格	数据单元格

图 4.262 表格元素的 scope 属性

<td>元素包裹行中的一个数据单元格，它的三个可选专用属性 colspan、headers 和 rowspan 的作用与<th>元素的同名属性一样。例如对于图 xyz-2 中的包裹“92”的<td>元素，其 headers 属性的值为"scores work"（假设包裹“成绩”和“平时分”的<th>的 id 的值分别为"scores"、"work"）。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

<colgroup>和<col>元素对列进行分组，然后施加不同的样式（一般是背景色），以便于我们从一个大表中定位目标单元格。这两个元素的 span 属性指定包含的列数，省略 span 属性，默认包含的列数就是 1。如果<colgroup>元素内包含了<col>子元素，那么<colgroup>的 span 属性无效。在一个<table>元素内允许出现多个<colgroup>子元素。

下面我们一步一步来生成图 4.259 所示的表格。

第 1 步：产生只包含三行数据的最简单表格

只用<table>、<tr>和<td>三个元素，代码如下：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>表格</title>

</head>

<body>
    <table>
        <tr>
            <!--第 1 行-->
            <td>王靖雯</td>
            <td>2020090101</td>
            <td>19 应用 1 班</td>
            <td>86</td>
            <td>90</td>
            <td>88.4</td>
        </tr>
        <tr>
            <!--第 2 行-->
            <td>李茂升</td>
            <td>2020090110</td>
            <td>19 网络 2 班</td>
            <td>90</td>
            <td>96</td>
            <td>93.6</td>
        </tr>
        <tr>
            <!--第 3 行-->
            <td>张百万</td>
            <td>2020090111</td>
            <td>19 软件 1 班</td>
            <td>92</td>
            <td>85</td>
            <td>87.8</td>
        </tr>
    </table>
</body>

</html>
```

浏览的效果见图 4.263 左侧部分，连表格线都没有。

王靖雯 2020090101 19应用1班 86 90 88.4
李茂升 2020090110 19网络2班 90 96 93.6
张百万 2020090111 19软件1班 92 85 87.8

王靖雯	2020090101	19应用1班	86	90	88.4
李茂升	2020090110	19网络2班	90	96	93.6
张百万	2020090111	19软件1班	92	85	87.8

图 4.263 先生成表体部分

在<head>内增加如下的样式表，从而添加表格线，然后浏览效果参见图 4.263 右侧部分。

```
    <style>
        table {
            border-collapse: collapse;
        }
        td, th {
            border: 1px solid grey;
            padding: 4px;
            text-align: center;
        }
        #grey1 {
            background-color: #EEE;
        }
        #col2 {
            background-color: #FFE599;
        }
        tfoot>tr>td {
            background-color: white;
        }
        caption {
            font-size: 26px;
            font-weight: 700;
            padding-bottom: 20px;
        }
    </style>
```

关于 CSS3 样式，请参考第 5 章。

第 2 步：增加表题

针对在第 1 步产生的代码，再<table>元素内增加子元素<caption>，代码如下：

```
<caption>助学金名单公示</caption>
```

第 3 步：增加一行表头

针对之前的代码，增加下面灰色除外的代码：

```
    <table>
        <caption>助学金名单公示</caption>
        <thead>
            <tr>
                <th>姓名</th>
                <th>学号</th>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
                <th>班级</th>
                <th>平时分</th>
                <th>期末分</th>
                <th>综合分</th>
            </tr>
        </thead>
        <tbody>
            <tr>
                <!--第 1 行-->
                <td>王靖雯</td>
                <td>2020090101</td>
                <td>19 应用 1 班</td>
                <td>86</td>
                <td>90</td>
                <td>88.4</td>
            </tr>
            ……
        </tbody>
    </table>
```

浏览效果见图 4.264 左侧部分。

助学金名单公示

姓名	学号	班级	平时分	期末分	综合分
王靖雯	2020090101	19应用1班	86	90	88.4
李茂升	2020090110	19网络2班	90	96	93.6
张百万	2020090111	19软件1班	92	85	87.8

助学金名单公示

姓名	学号	班级	成绩		
			平时分	期末分	综合分
王靖雯	2020090101	19应用1班	86	90	88.4
李茂升	2020090110	19网络2班	90	96	93.6
张百万	2020090111	19软件1班	92	85	87.8

图 4.264 增加表头

第 4 步：产生完整的表头

在<thead>内添加下面的除灰色之外的代码，浏览效果见图 4.264 右侧部分。

```
        <thead>
            <tr>
                <th rowspan="0">姓名</th>
                <th rowspan="2">学号</th>
                <th rowspan="2">班级</th>
                <th colspan="3">成绩</th>
            </tr>
            <tr>
                <th>平时分</th>
                <th>期末分</th>
                <th>综合分</th>
            </tr>
        </thead>
```

第 5 步：增加表尾

继续在<table>元素内增加子元素<tfoot>，代码如下：

```
        <tfoot>
            <tr>
                <td colspan="6">一共 3 人</td>
            </tr>
        </tfoot>
```

浏览效果见图 4.265 左侧部分。

助学金名单公示

姓名	学号	班级	成绩		
			平时分	期末分	综合分
王靖雯	2020090101	19应用1班	86	90	88.4
李茂升	2020090110	19网络2班	90	96	93.6
张百万	2020090111	19软件1班	92	85	87.8
一共3人					

助学金名单公示

姓名	学号	班级	成绩		
			平时分	期末分	综合分
王靖雯	2020090101	19应用1班	86	90	88.4
李茂升	2020090110	19网络2班	90	96	93.6
张百万	2020090111	19软件1班	92	85	87.8
一共3人					

图 4.265 增加表尾并分组着色

第 6 步：给某些列增加不同的背景色

在<table>元素内增加<colgroup>元素，代码如下，浏览的效果见图 4.265 右侧部分。

```
        <colgroup span="3" id="grey1"></colgroup>
        <colgroup>
            <col span="2">
            <col id="col2">
        </colgroup>
```

着色通过样式完成，参见第 1 步中的样式代码。

4.12 互动元素

<dialoag>、<details>和<summary>元素实现与人简单交互。

<details>、<summary>元素

使用<details>和<summary>元素可以轻松设计出如图 4.266 的页面，点击那个红色的小三角形展开详细内容，再点击一次隐藏详细内容。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 4.266 <details>元素效果展示

<details>和<summary>元素的一般用法是：

```
<dedails[ open]>
     <summary>永远显示的内容</summary>
     其他内容（这些内容会隐藏或显示）
</details>
```

<details>元素内容默认是关闭的，如果加上 open 属性，就是打开的。<summary>元素允许出现在<details>内任何位置。看例子：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>details</title>
    <style>
        div {
            width: 100px;
            height: 100px;
            background-color: red;
            display: inline-block;
        }
    </style>
</head>

<body>
    <details>
        <summary>销售前 5 名</summary>
        <div>1</div>
        <div>2</div>
        <div>3</div>
        <div>4</div>
        <div>5</div>
    </details>

</body>

</html>
```

标题和标题前的标记都可以通过给 summary 元素添加样式来改变，适用的样式属性有 list-style-type、list-style-position 和 list-style-image，比如下面的样式修改关闭时的标记和标题颜色：

```
summary {
   color: white;
   list-style-image: url("/images/close1.png");
```

```
}
```

下面的样式修改打开时的标记：

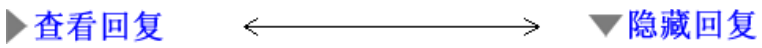
```
details[open]>summary{
    list-style-image: url("/images/open1.png");
}
```

浏览的效果图：



对于列表样式，请参考“§5.8 列表”。

<details>元素对象支持 toggle 事件，当发生状态改变时（关闭或打开）触发，利用此事件可以改变标题，见下图：



实现代码参考：

```
let detailsObj = document.getElementById("allComments");
detailsObj.addEventListener("toggle", () => {
    detailsObj.querySelector("summary").innerText = detailsObj.open ? "隐藏回复" : "查看回复";
});
```

<dialog>元素

专门用于设计对话框。我们知道采用 JS 的 alert()函数、confirm()函数、prompt()函数可以弹出一个简单的对话框，比如语句 alert("你好啊!")，浏览的效果见图 4.267。



图 4.267 alert()交互方法

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

那么采用<dialog>元素可以设计个性化的对话框，其语法是：

```
<dialog [open]>
    其他 html5 元素和内容
</dialog>
```

如果省略 open 属性，默认对话框是隐藏的。采用 JS 编程，可通过元素对象的 returnValue 属性来传递信息。<dialog>对象的方法有以下几个（不过新版 firefox 不支持）：

show()：以非模态方式显示对话框。

showModal()：以模态方式显示对话框。

close(返回值)：关闭对话框，并把"返回值"赋给 returnValue 属性。

<dialog>对象有一个 onclose 事件，有一个::backdrop 伪元素，用于设置模态遮罩层的样式。

看例子：

```
<!DOCTYPE html>
<html>

<head>
<meta charset="utf-8">
    <title>Dialog 对话框</title>
</head>

<body>
    <button type="button" id="show">删除</button>
    <dialog id="mydialog">
        <h2>我的对话框</h2>
        <p>你好，这是我设计的对话框，好看吗？ </p>
        <button type="button" id="clos1" style="float:right;">很难看</button>
        <button type="button" id="clos2" style="float:right;">好看</button>
    </dialog>
    <script defer>
        var x = document.getElementById("mydialog");
        x.returnValue = 100;
        x.addEventListener("close",function(){alert(x.returnValue);})

        document.getElementById("show").addEventListener("click",show1);
        function show1() {
            x.show();
            //x.showModal();
        }
        document.getElementById("clos1").addEventListener("click",()=>{diaclose
(1)});
        document.getElementById("clos2").addEventListener("click",()=>{diaclose
(2)});
        function diaclose(value) {
            console.log(x.returnValue);
            x.close(value);
        }
    </script>
</body>
```

```
</html>
```

在 chrome 浏览器中的效果如图 4.268 所示。



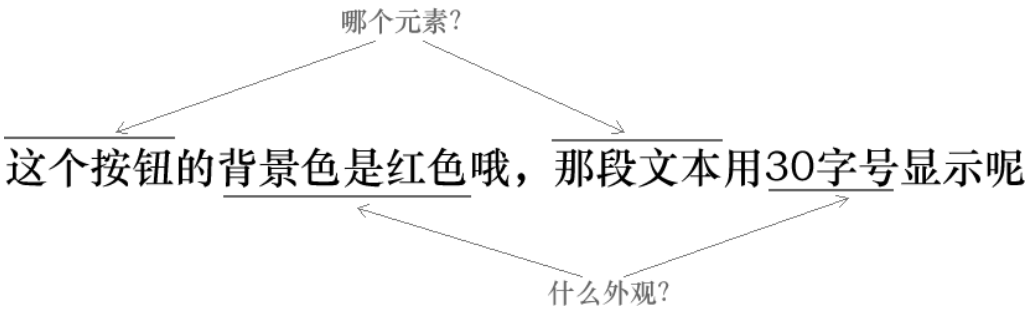
图 4.268 <dialog>元素用法举例

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

第 5 章

网页样式：CSS3

CSS3 样式就是通过 HTML5 元素来定义相应内容的外观，比如颜色、位置、背景、边框、阴影、动画、2d/3d 变换等等，本质上就是你告诉浏览器：“这个按钮的背景色是红色哦，那段文本用 30 字号显示呢，对了，还有这个图形要带阴影啦，……。”因此，你需要跟浏览器说清楚两个问题：（1）页面当中的哪个元素？（2）对此元素设置什么外观？



好了，稍作歇息之后，我们再次上路吧。小伙伴们，我将带领大家去绚丽的 CSS3 世界里走一走、看一看，那里能满足我们对美的追求！

5.1 css3 简介

一个元素的样式来源渠道有五个，它们的优先级为：嵌入到“元素开始”中的样式（如图 2.18 所示）→嵌入到 HTML 文件中的样式（如图 2.21 所示）→样式文件中的样式（如图 2.20 所示）→继承父元素的样式→浏览器的默认样式。当某个样式属性存在于多个来源渠道时，优先级高的起作用，比如颜色样式“color”既在嵌入到元素中的样式中定义为红色，又在样式文件中设为蓝色，那么最终浏览器以红色显示。如果要强制低优先级的样式起作用，就必须额外附加“!important”，比如刚才的那个例子，要强制以蓝色显示，就修改样式文件中的样式表如下：

```
p {
    color: blue !important;
}
```

除了嵌入到元素中的样式外，其他四个样式来源渠道都面临第（1）个问题，对于第（1）个问题的解决办法就是采用样式选择器，具体介绍参考下一小节。至于如何定义外观的问题，CSS3 提供了各种各样的样式属性，如 color、border、left、font-size、width 等，每个样式属性都有规定的取值范围，比如 width 属性表示元素的宽度，它的值只能是长度单位或者百分数，而不能是字符串。因此学习 CSS3 就是学习样式选择器和各种样式属性的用法，一个样式表由样式选择器和若干样式属性组成，其语法如图 5.1 所示。

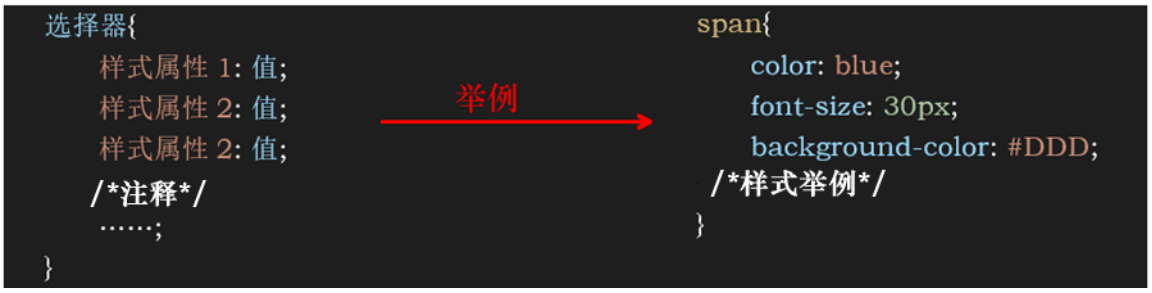


图 5.1 CSS3 样式语法

最后一个分号允许省略。如果样式表嵌入在 html 网页文件中，则必须作为<style>元素的内容，而<style>元素最好放在<head>元素中，参见“图 2.19 嵌入在网页文件中的样式表”。样式表允许写在一行内，比如上面的例子也可以写成：

```
span{ color: blue; font-size: 30px; background-color: #DDD;}
```

在定义样式的代码中，采用/*说明内容*/作注释，注释内容可以一行也可以多行。一个元素渲染后，首先它与周边元素之间存在间距（即外边距），由样式属性 margin(对应分属性 margin-top、margin-right、margin-bootom 和 margin-left)定义；其次存在边框，由样式属性 border（对应分属性 border-top、border-right、border-bottom 和 border-left）定义；再次元素的边框与元素内容之间存在内边距，由 padding（对应分属性 padding-top、padding-right、padding-bottom 和 padding-left）定义，最后元素内容区域大小默认由元素的 width 和 height 定义，还有对齐方式、颜色、背景色等等都有相应的样式属性，具体请参阅“《HTML5&CSS3 开发大起底》.xlsx”文件中的“CSS3 样式速查表”。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

5.2 选择器

CSS3 可以通过各种方式来定位元素，这种定位元素的方式称为样式选择器，共有 5 大类 76 种，具体请参见“《HTML5&CSS3 开发大起底》.xlsx”文件中的“样式选择器”工作表。

我们先用下面的例子热热身。

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>样式选择器</title>
</head>

<body>
    <header draggable="true">
        <img id="logo" src="logo.png" />
        <nav>
            <ul style="display: inline">
                <a href="home">
                    <li class="active">首页</li>
                </a>
                <a href="product">
                    <li>产品</li>
                </a>
                <a href="contact">
                    <li>联系</li>
                </a>
                <a href="login">
                    <li>登陆</li>
                </a>
            </ul>
        </nav>
    </header>
    <section>
        <article style="display: inline; position: relative;">
            <h2>关于微算</h2>
            <section title="abcd">
                <h3>创立</h3>
                <p>成立于 2001 年，由……</p>
                <p>注册资本一亿，……</p>
            </section>
            <section>
                <h3>使命</h3>
                <p>良心产品，造福国民。</p>
            </section>
        </article>
        <aside data-name="zansan">
            <ol>
                <a href="a1.html">
                    <li>aaaa</li>
```

```
                </a>
                <a href="a2.html">
                    <li>bbbb</li>
                </a>
                <a href="a3.html">
                    <li>cccc</li>
                </a>
            </ol>
        </aside>
    </section>
    <footer>关于我们       版权所有 2019</footer>
</body>

</html>
```

针对上面的代码，下面是一些选择器的例子：

1、选择全部的 p 元素：p { … }

2、选择属于类 active 的元素：.active { … }

3、选择 header 和 footer 元素：header, footer { … }

4、选择 article 元素下的 section 子元素：article > section { … }

5、选择 nav 下的 a 后代元素：nav a { … }

6、选择 li 兄弟中的最后一个 li 元素：li:last-of-type { … }

7、选择索引号为偶数的 a 元素：a:nth-of-type(even) { … }

8、选择带有属性 data-name 且该属性的值为 zansan 的元素： [data-name=”zhangsan”] { … }

5.3 样式属性

CSS3 支持的全部样式属性请参阅“《HTML5&CSS3 开发大起底》.xlsx”文件中的“CSS3 样式速查表”，下面各章节按照表中的一级分类逐一介绍。对了，你还得去再买一箱方便面，继续苦修，到了现在，放弃就可惜了！

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

5.4 文字内容

文字内容的样式涉及字体、字号、颜色、字型、加重、对齐以及阴影、装饰等，具体请参阅“CSS3 样式速查表”中的“文字内容”一级分类——为了简洁，下文统一采用点分法，即“CSS3 样式速查表・一级分类・二级分类”。

5.4.1 字体

“CSS3 样式速查表・文字内容・字体”分类列举了全部的定义字体的样式。

font 属性定义元素字体的全部特性，包括字符集、字号、字型、变体、行高等。其使用语法如下：

```
font:                                                                    必须                                必须
  ❶[ <'font-style'> || <'font-variant'> || <'font-weight'> || <'font-stretch'> ]? <'font-size'> [ / <'line-height'> ]? <'font-family'>
  ❷caption | icon | menu | message-box | small-caption | status-bar
```

只能选择①或②。语法②直接选择操作系统字体，此时可能的值有 caption（系统控件的字体，如按钮）、icon（用于标记图标的系统字体）、menu（系统菜单的字体）、message-box（系统对话框的字体）、small-caption（用于标记小型控件的系统字体）、status-bar（系统状态栏的字体）。此时诸如 font-style 等其他属性不再其作用。参考图 5.2。



图 5.2 系统字体说明

语法①中，字号<font-size>和字符集<font-family>不能省略，其他的都是可选的，顺序不能乱。语法①表明 font 又是 font-style、font-variant、font-weight、font-stretch、font-size、line-height 和 font-family 样式的复合属性。下面具体介绍这些属性的用法。

font-style 属性定义字型，其语法如下：

```
font-style: normal | italic | oblique <angle>?
```

normal 为正体，italic 为斜体，oblique 为倾斜体，而且后面还可以附加倾斜的角度。斜体是字符集（由 font-family 属性指定）本身蕴含的特征，比如宋体就包含了 normal 和 italic 字型点阵，有的字符集根本没有斜体。而倾斜体是由应用软件（如浏览器、word 等）直接倾斜 normal 字体得到的，所以我们可以告诉浏览器倾斜多大的角度，角度范围为(-90°,90°)。例如 font-style: oblique 和 font-style: italic 效果差不多，但是 font-style: oblique -20deg 表示向左倾斜 20 度。

font-variant 属性设置字体的所有变种。在日常项目中很少用到此属性，拟详细了解的读者可参考 https://drafts.csswg.org/css-fonts-3/#propdef-font-variant。

font-weight 属性定义字体的粗细程度，取的值与字符集相关（由 font-family 属性定义）。其使用语法如下：

```
font-weight: normal | bold | lighter | bolder | <number>
```

normal 正常粗度，相当于 400，bold 加粗，相当于 700，lighter 比父元素的字体细，bolder 比父元素要粗，<number>指定具体的粗度，取值范围为[1,1000]。

font-stretch 属性从字符集中选择普通、压缩或扩展的外观，如果字符集不支持相应的外观，font-stretch 属性无效。它的是使用语法如下：

```
font-stretch: normal | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded | <percentage>
```

normal 属于正常外观（100%），semi-condensed、condensed、extra-condensed 和 ultra-condensed 为压缩外观，压缩率分别为 87.5%、75%、62.5%和 50%，而 semi-expanded、expanded、extra-expanded 和 ultra-expanded 表示扩展外观，放大率分别为 112.5%、125%、150%和 200%。<percentage>表示百分数，取值范围在[50%,200%]，比如 font-stretch: 90%。

font-size 属性设置字号，会影响与字号关联的长度单位，如 em、ex 和 rem。其使用语法如下：

```
font-size: xx-small | x-small | small | medium | large | x-large | xx-large | xxx-large | larger | smaller | <length> | <percentage>
```

xx-small、x-small、small、medium、large、x-large、xx-large 和 xxx-large 与用户的默认字号相比的尺寸，类似于衣服的尺寸表示。larger 比父元素的字号大，smaller 比父元素的字号小。<length>为正数，常用的单位有 px、em、ex、rem 等。比如 font-size: x-larger、font-size: 1.5rem。

line-height 属性定义行高，行内文本垂直方向居中对齐，其他语法如下：

```
line-height: normal | <number> | <length> | <percentage>
```

<number>为无量纲的数字，表示当前字号的倍数。

font-family 属性为元素指定若干个字体，中间用逗号分隔，如果字体名字包含空格，则必须用引号括住，越左的优先级越高。浏览器会使用优先级尽量高且已安装或通过@font-face 规则能下载的字体——即从左至右搜索用户需要的字型（font-style 定义）、变体（font-variant 定义）和尺寸（font-size 定义）的字体。如果没找到，浏览器会使用默认的一种字体。所以最好的做法是，在最右侧指定一种通用字体，通用字体不能用引号括住。用法如下：

```
font-family: 字体 1, 字体 2, "字 体 3", …, 通用字体
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

目前常用的通用字体有 serif、sans-serif、monospace、cursive、fantasy、math 和 fangsong 等。看例子：

```
<p style="font-family: 华文中宋,华文行楷,'Bradley Hand ITC',serif">
    作为一个人的存在，你首先要了解自己，社会一切问题的根源最终都可追溯到人的问题。人也是组成家庭、组织、国家、乃至全人类社会的基本单位。所以，认识你自己是认识其他对象的开端。</p>
<p style="font-family: 华文行楷,华文中宋,monospace">
    那就阅读英国作家大卫·休谟的《人性论》吧，此书有点深奥，谁让你是一个人呢？人的深奥那才是深不见底。另外向你推荐美国雅克·蒂洛、基思·克拉斯曼合著的《伦理学与生活（第 11 版）》</p>
```

华文中宋和华文行楷字体都已经安装了（在 Windows 的 C:\Windows\Fonts 目录中），第 1 段采用“华文中宋”字体，但第 2 段“华文行楷”排在前面，所以第 2 段采用“华文行楷”字体。浏览的效果如图 5.3。

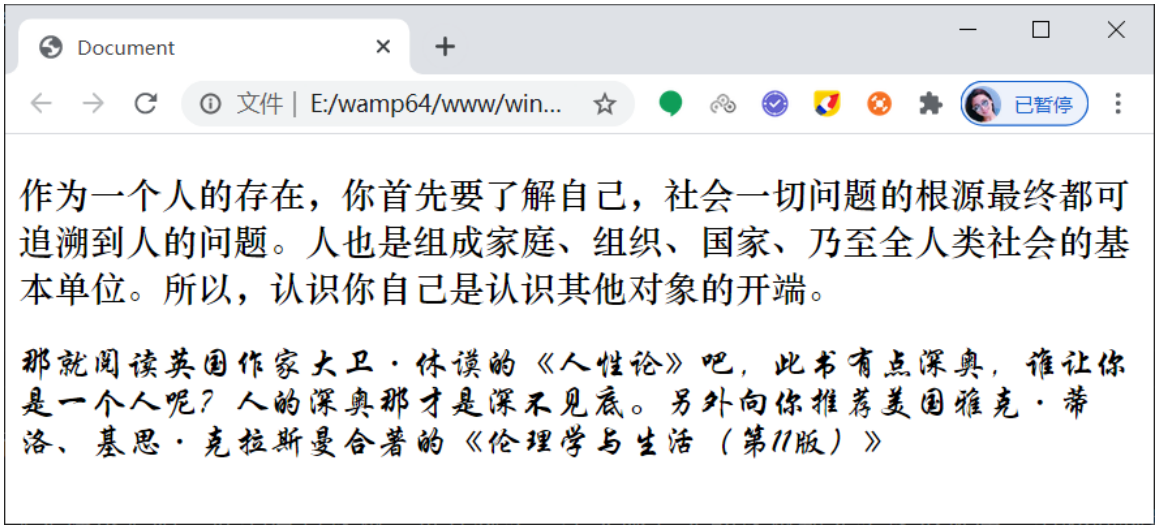


图 5.3 字体属性用法举例

针对 font 复合属性，下面举一些例子（建议使用 firefox 浏览器做实验）：

```
font: 24px 华文中宋,serif;/*字号 24 个像素*/
font: 24px/1.5 "Couruer New",serif; /*1.5 倍行高*/
font: italic 24px/34px 宋体,"Couruer New",sans-serif;/*行高 30 像素*/
font: oblique -20deg 700 condensed 20px/2 华文中宋,AmstelvarAlpha,sans-serif;/*向左倾斜 20 度*/
```

font-synthesis 属性告诉浏览器当一种字体的变种（比如中文、日文等没有斜体或加粗）缺失而用户又要使用时，如何处理。其使用语法如下：

```
font-synthesis: none | [ weight || style ]
```

none 不合成粗体和斜体，weight 如果需要就合成粗体，style 如果需要就合成斜体。比如 font-synthesis: weight、font-synthesis: weight style。

line-height-step 属性定义行高因子，行距必须是行高因子的整数倍。其语法如下：

```
line-height-step: <length>
```

例如：

```
<p style= "line-height-step:18px; font-size:20px;">实际行高为 36px</p>
```

5.4.2 文本

“CSS3 样式速查表 • 文字内容 • 文本”分类列举了全部的定义文本的样式。

1、writing-mode/direction/unicode-bidi/text-orientation — 定义排版和书写方向

文本书写涉及四个要素，即排版方向、多行的行进方向、行内文字的书写方向和单个字符的朝向，参考示意图 5.4。

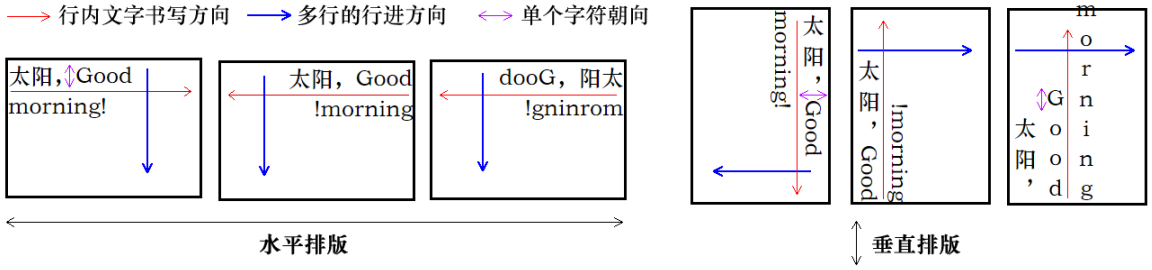


图 5.4 文本书写的四个要素

由 writing-mode 属性定义排版方向和多行的行进方向，排版方向只有水平排版和垂直排版两种，行进方向对于水平排版来说只有从上往下（目前没有从下往上行进的人类语言），对于垂直排版存在从左往右或从右往左两种行进方向。writing-mode 属性的语法如下：

```
writing-mode: horizontal-tb | vertical-rl | vertical-lr
```

值 horizontal-tb 表示水平排版从上往下行进，vertical-rl 表示垂直排版从右往左行进，vertical-lr 表示垂直排版从左往右行进。另外两个值 sideways-rl 和 sideways-lr 目前处于实验阶段。

由 direction 和 unicode-bidi 两个属性定义一行内的文字的书写方向。它们的使用语法如下：

```
direction: ltr | rtl
unicode-bidi: normal | embed | isolate | bidi-override | isolate-override |
plaintext
```

值 ltr 表示在水平排版中从左往右、垂直排版中从上往下；值 rtl 表示从右往左（水平排版）或从下往上（垂直排版）。但是如果不配合 unicode-bidi: bidi-override，那么 direction: rtl 对于“天生”从左往右书写的语言（如英语、中文等）来说，只是右对齐或者下对齐而已，参考图中“水平排版”的中间部分和“垂直排版”的右侧部分。对于“天生”书写方向从右往左的语言（如阿拉伯语、波斯语等）来说才会有效果。下面的例子说明阿拉伯语“天生”的书写方式：

```
<p style="direction: rtl;">أن يعني امرأة تكون أن كأ تبدو أن يجب امرأة لتكوني
تكون مثل المرأة ، فهي تصر على لعب كرة القدم ورفع الأثقال والملاكمة والمشاركة في
الأنشطة المشينة للرجال. تفاحته ولحيته وعضلات بطنه وعجولته مثل الرجال تمامًا. </p>
```

浏览的效果如图 5.5 左侧部分。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

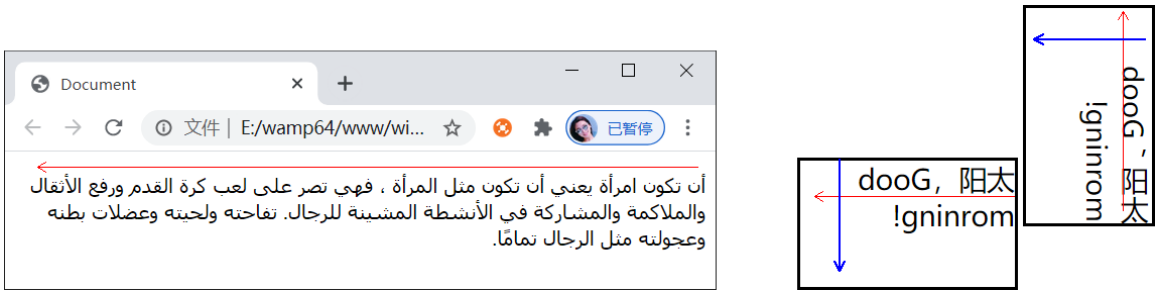


图 5.5 文本书写规则举例

当 direction: rtl 结合 unicode-bidi: bidi-override 时，就不管是什么“天生”方向的语言，都遵循从右往左（水平排版）或者从下往上（垂直排版）的规则，下面的例子浏览效果参见图右侧部分：

```
span {
    display: inline-block;
    inline-size: 170px;
    block-size: 100px;
    border: solid;
    direction: rtl;
    unicode-bidi: bidi-override;
    font-size: 22px;
}
……
<span style="writing-mode: horizontal-tb;">太阳，Good morning!</span>
<span style="writing-mode: vertical-rl;">太阳，Good morning!</span>
```

text-orientation 属性定义单个字符的朝向，只对垂直排版有效果，它的语法如下：

```
text-orientation: mixed | upright | sideways
```

mixed 表示顺时针旋转 90 度，这是默认值，upright 保持字符的字型方向不变（垂直），参见图中“垂直排版”最右侧的方框。sideways 相当于把水平文本整体顺时针旋转 90 度，会影响中文、英文等全部语言。

```
span {
    display: inline-block;
    inline-size: 170px;
    border: solid;
    block-size: 100px;
    font-size: 22px;
    writing-mode: vertical-rl;
}
……
<span style="text-orientation: mixed;">太阳，Good morning!</span>
<span style="text-orientation: upright;">太阳，Good morning!</span>
<span style="text-orientation: sideways;">太阳，Good morning!</span>
```

浏览效果见图 5.6。

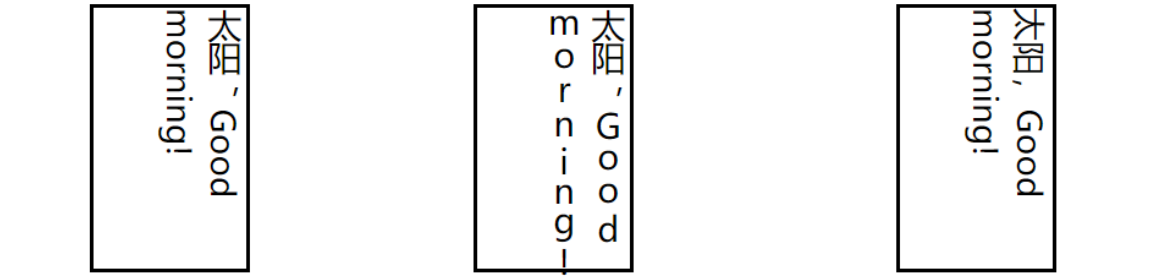


图 5.6 文本方向距离

2、overflow-wrap/word-break/hyphens/text-overflow/line-break — 如何折行

overflow-wrap 属性指示浏览器：当一行放不下一个单词时如何处理。它的语法如下：

```
overflow-wrap: normal | break-word | anywhere
```

normal 不在单词内折行，anywhere 表示如果行中没有其他可接受的断点，那就可在单词内部任意位置折行，break-word 与 anywhere 差不多，但是在计算 min-content 时忽略用户插入的软折行符，关于 min-content 请参考“§5.10.3.1 应用在容器上的样式”。

word-break 属性定义单词内的折行模式，其语法如下：

```
word-break: normal | break-all | keep-all
```

normal 采用浏览器默认的折行规则，break-all 可在单词内部任意位置折行以免本行溢出，keep-all 对于 CJK（Chinese/Japanese/Korean）文本不折行，对于非 CJK 文本，处理方式与 normal 一样。

那么 overflow-wrap: anywhere 和 word-break: break-all 有什么区别呢？用一个例子说明：

```
p {
    inline-size: 220px;
    border: 1px solid black;
    display: inline-block;
}
……
<p style="overflow-wrap: break-word;">
The longest word pneumonoultramicroscopicsilicovolcanoconiosis means 矽肺又称硅肺，是尘肺中最为常见的一种类型，是由于长期吸入大量游离二氧化硅粉尘所引起。</p>
<p style="word-break: break-all;">
The longest word pneumonoultramicroscopicsilicovolcanoconiosis means 矽肺又称硅肺，是尘肺中最为常见的一种类型，是由于长期吸入大量游离二氧化硅粉尘所引起。</p>
```

主要差别在处理长单词 pneumonoultramicroscopicsilicovolcanoconiosis，见图 5.7。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

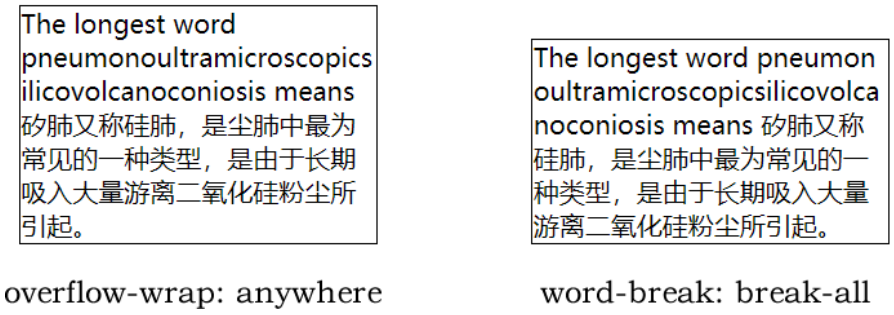


图 5.7 overflow-wrap 和 word-break 属性的区别

当一个单词跨行时，由 hyphens 属性定义连字方法，其语法如下：

```
hyphens: none | manual | auto;
```

none 不在单词中间折行，即使插入了建议折行符也不行。manual 可能的话在用户建议的地方折行，比如用户在长单词之中插入一个“软”连字符，告诉浏览器如果要折行，就这里折吧，如果用户没有插入建议折行符，那么此时 manual 等价于 none。auto 就是由浏览器自行决定折行的位置，但如果用户插入了建议折行符，那么这时 auto 等价于 manual。注意<wbr>元素优先级最高。看例子：

```
dd {
    inline-size: 65px;
    border: 1px solid black;
    display: inline-block;
}
……
<dd style="hyphens: none;">An extreme&shy;ly long English word</dd>
<dd style="hyphens: manual">An extreme&shy;ly long English word</dd>
<dd style="hyphens: auto">An extremely long English word</dd>
```

浏览的效果如图 5.8。

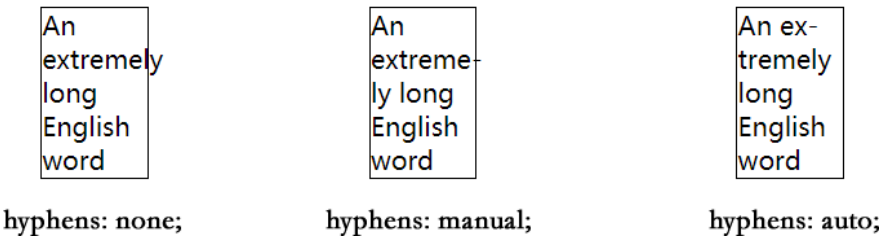


图 5.8 hyphens 属性用法举例

text-overflow 属性指示浏览器如何处理溢出部分的文本内容，它的语法如下：

```
text-overflow: clip | ellipsis
```

clip 剪掉溢出的内容，ellipsis 在行尾显示…表示存在被剪掉的内容。注意：只有 overflow 属性设置隐藏时本属性才有效果，overflow 属性参考“§5.5 元素定位”小节。

针对带有标点符号的中文、日文或韩文文本，由 line-break 属性定义如何换行。其使用语法如下：

```
line-break: auto | loose | normal | strict | anywhere
```

3、text-align/text-align-last/text-justify/vertical-align — 对齐方式

text-align 属性设置块元素或表格单元内的文本的水平对齐方式，其语法如下：

```
text-align: start | end | left | right | center | justify | match-parent
```

值 left、right、center 依次表示左对齐、右对齐和居中对齐，justify 通过增加单词间的空格实现两端对齐，start 和 end 是逻辑样式，表示书写方向上的起点对齐和终点对齐，当从左往右书写时 start 就是左对齐，end 就是右对齐，如果从上往下书写，那么 start 就是上对齐了。match-parent 继承父元素的对齐方式。

text-align-last 属性定义文本块的最后一行或强行换行符之前的那行的对齐方式，这两类文本行往往留下大量空白空间。此属性的语法如下：

```
text-align-last: auto | start | end | left | right | center | justify
```

auto 表示参照 text-align 属性，但如果 text-align 属性的值为 justify，那么此时 auto 相当于 start。text-align-last 其他值与 text-align 属性的同名值一样。

看例子：

```
p {
    inline-size: 220px;
    border: 1px solid black;
    display: inline-block;
    padding-block-end: 30px;
}
<p style="text-align: justify; text-align-last: right">大清灭亡后军阀割据及后续的民国时期，中<br>国迎来了第三次文化、思想爆发的黄金时代，大师名人如雷贯耳，人文社科成果斐然。</p>
```

浏览的效果如图 5.9。

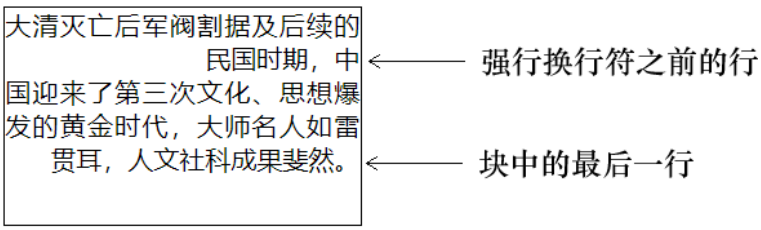


图 5.9 text-align 和 text-aligin-last 属性用法举例

当 text-align: justify 时（即两端对齐），由 text-justify 属性定义在何处增加空格，比如在单词之间还是字母之间。它的语法如下：

```
text-justify: auto | inter-character | inter-word | none
```

auto 由浏览器自行决定，inter-character 增加字符之间的间距，inter-word 增大单词之间的间距，none 使 text-aligin: justify 作废（不起作用）。例如把上面例子中的<p>元素更换为：

```
<p style="text-align-last: justify; text-justify: inter-character;">
This is my favorite music!</p>
```

读者尝试把代码中的 inter-character 更换为其他值，注意观察浏览效果由什么变化。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

vertical-align 属性定义内联元素（含内联块元素）或者表格单元的垂直对齐方式，注意本属性作用于元素本身，是与父元素比较垂直方向上的位置，而前面介绍的三种对齐属性作用于元素里面的内容，是与元素的边界比较水平方向上的位置。它的使用语法如下：

```
vertical-align: baseline | sub | super | text-top | text-bottom | middle | top |
bottom | <percentage> | <length>
```

针对内联元素和表格单元，这些值有不同的涵义。

(1)、针对内联元素

baseline 表示元素的基线对齐父元素的基线，如果元素（如<img>）没有基线，那么元素的底部对齐父元素的基线。元素的基线可以这样确定：在元素里放置字母 x，那么 x 的底部就是元素的基线。参考示意图 5.10。



图 5.10 vertical-align 属性说明

看例子：

```
div{
    border-style: solid none;
    inline-size: 340px;
}
<div>Oxford English<span style="vertical-align: baseline;color: red;font-
size:40px;">xofspan</span></div>
```

浏览的效果如图 5.11。



图 5.11 父/子元素基线对齐

sub 表示元素的基线对齐父元素的下标文本的基线，**super** 表示元素基线对齐父元素上标文本的基线。用元素<sub>包裹下标文本，用元素<sup>包裹上标文本。参见图 5.12。

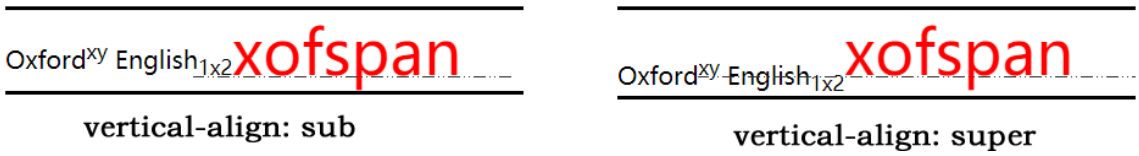


图 5.12 sub/super 对齐举例

text-top 表示元素的顶部与父元素中的文本顶部对齐，类似 text-bottom 表示元素底部与父元素中的文本的底部对齐，middle 表示元素的中线与父元素的基线之上 x 高度的一半处对齐，大部分情况下就是父元素的中线，top 表示元素的顶部与元素所在的行的顶部对齐，类似地 bottom 表示底部对齐。<length>表示元素基线与父元素的基线+length 处对齐，例如 vertical-align: 4px 表示元素与父元素基线之上 4 个像素处对齐，<length>可去负数。<percentage>表示元素的基线对

齐父元素的基线+percentage*属性 line-height 的值，例如 line-height:30px; vertical-align:30%表示基线之上 9 个像素处，<percentage>可取负数。

(2)、针对表格单元

baseline 表示本单元格的基线与本行中所有符合条件的单元格的基线对齐，此条件就是应用了样式 vertical-align: baseline 的单元格。sub、super、text-top、text-bottom、<length>和<percentage>作同样的理解。

top 表示本单元格的顶部 padding 边线对齐本行的顶部，bottom 表示本单元格的底部 padding 边线对齐本行的底部，而 middle 表示本单元格的 padding 框中线对齐本行的中线。

4、word-spacing/letter-spacing/text-indent/white-space/tab-size — 定义间隔

word-spacing 属性定义单词之间的间距，其语法如下：

```
word-spacing: normal | <length> | <percentage>
```

<length>表示在 normal 的基础上额外增加的长度，<percentage>表示额外增加默认间距的百分数，例如 word-spacing: 200%，默认间距假设为 10px，那么最终间距就是 30px。<length>和<percentage>可取负数。

letter-spacing 属性定义字符之间的间距，使用语法如下(<length>理解参考 word-spacing 属性)：

```
letter-spacing: normal | <length>
```

text-indent 属性定义段落缩进，其语法如下：

```
text-indent: <length> | <percentage>
```

<length>缩进长度，<percentage>缩进元素宽度的百分数，都可取负数。例如 text-indent: 2em 就是缩进两个字符。

white-space 属性指示浏览器如何处理空白字符，它的语法如下：

```
white-space: normal | nowrap | pre | pre-wrap | pre-line | break-spaces
```

normal：连续多个空白字符只保留一个(合并空格)，换行符视同空白字符，自动换行；nowrap：合并空格，不自动换行；pre：保留空格不自动换行；pre-wrap：保留空格，自动换行；pre-line：合并空格并自动换行，换行符起作用，这点与 normal 不同；break-spaces：与 pre-wrap 基本相同，但区别是 break-spaces 会在连续的空白字符中间换行，所以会影响 min-content 和 max-content。

为了更好地理解这几个值的区别，我们举一个例子，用不同的值渲染如图 5.13 所示的这段文字。

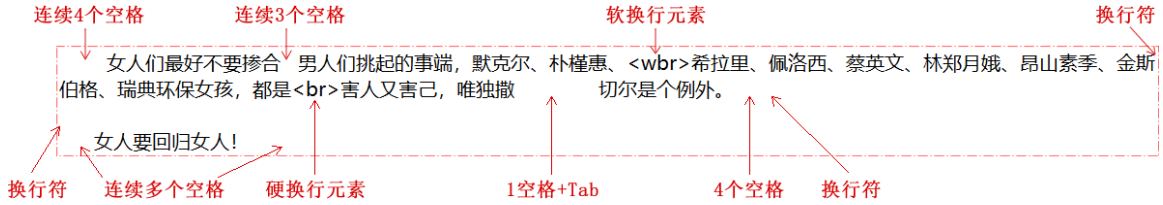


图 5.13 white-space 属性用法举例 1

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

代码参见 chapter-5\white-space.html 文件，在浏览器中打开它，然后直接在开发者工具中修改 white-space 属性的值，你会看到如图 5.14 所示的效果，选中这些文字是为了方便看到空白字符。

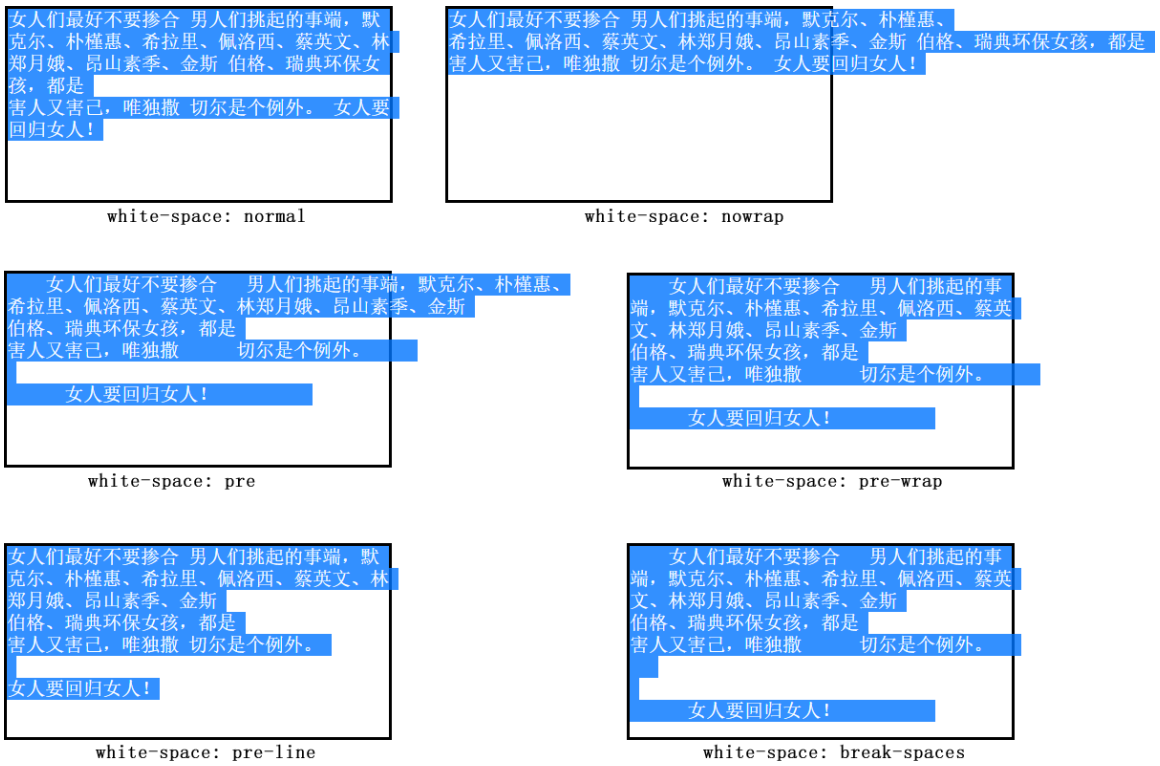


图 5.14 white-space 属性用法举例 2

最后用表 5.4 总结一下。

表 5.4 white-space 属性取值总结

white-space 属性的值	连续空白字符	换行符	 /<wbr>	容器边界处	行尾空格
normal	合并	忽略	起作用	自动换行	忽略
nowrap	合并	忽略	起作用	不换行	忽略
pre	保留	保留	起作用	不换行	保留
pre-wrap	保留	保留	起作用	自动换行	悬挂
pre-line	合并	保留	起作用	自动换行	忽略
break-spaces	保留	保留	起作用	自动换行	换行

tab-size 属性定义制表符的长度，其语法如下：

```
tab-size: <integer> | <length>
```

<integer>直接指名多少个空格，比如 tab-size: 6 表示按一下 tab 键右移六个空格，<length>直接指名具体长度，不能为负数，如 tab-size: 2em。

5、text-transform — 定义文本大小写转换规则

text-transform 属性使用语法如下：

```
text-transform: none | capitalize | uppercase | lowercase | full-width
```

none 不做变换，这时默认值；capitalize 每个单词首字母转为大写；uppsercase 全部字母转为大写；lowercase 全部转变为小写字母；full-width 字符占据相同的宽度。

看例子：

```
<p>iiiiiiiii oxford English 男人们成了娘炮</p>
<p>wwwwwwwww oxford English 男人们成了娘炮</p>
<p style="text-transform: full-width;">iiiiiiiii oxford English 男人们成了娘炮</p>
<p style="text-transform: full-width;">wwwwwwwww oxford English 男人们成了娘炮</p>
```

浏览的效果如图 5.15。

iiiiiiiii oxford English 男人们成了娘炮
wwwwwwwww oxford English 男人们成了娘炮
ｉｉｉｉｉｉｉｉｉ ｏｘｆｏｒｄ Ｅｎｇｌｉｓｈ 男人们成了娘炮
ｗｗｗｗｗｗｗｗｗ ｏｘｆｏｒｄ Ｅｎｇｌｉｓｈ 男人们成了娘炮

图 5.15 text-transform 属性用法举例

6、color/opacity — 设置颜色

color 属性定义文本的颜色，同时会改变 currentcolor 的值，其他使用颜色的样式可以直接使用 currentcolor，从而间接引用 color 属性，比如 border-corlor: currentcolor。color 属性的语法如下：

```
color: <rgb()> | <rgba()> | <hsl()> | <hsla()> | <hex-color> | <named-color> |
currentcolor | <deprecated-system-color>
```

用来表示颜色的方法非常多，最常见的有：

<hex-color>：用#引导的 3 位或 4 位或 6 位或 8 位十六进制数表示的 RGB[A]配色，见图 5.16，其中透明度 FF 表示不透明，00 全透明，80 半透明，反正越小越透明。如果省略透明度，默认是不透明。例如#DDD、#F0FA、#FF43D4、#FF0000A0。

#XYZ[A] ⟶ #XXYYZZ[AA]
红色分量 绿色分量 蓝色分量 透明度

图 5.16 十六进制颜色表示法

<named-color>：颜色名字，如 red（红色）、green（绿色）、blue（蓝色）、white（白色）、black（黑色）、gray（灰色）、purple（紫色）等等，CSS3 增加了很多名字，具体参考 https://developer.mozilla.org/en-US/docs/Web/CSS/color_value。

<rgb()>：RGB 颜色模型的样式函数 rgb(r, g, b)，三个参数分别表示红色分量、绿色分量和蓝色分，可以采用纯数字或百分数，取值范围[0,255]或者[0%,100%]，例如 color: rgb(255,100,10)、rgb(80%, 50%,40%)、red=#F00=#FF0000=rgb(255,0,0)。

<rgba()>：样式函数 rgba(r, g, b, a)在 rgb()的基础上增加了透明度参数 a，a 的取值范围是[0,1]或者[0%,100%]，1 和 100%表示不透明，0 和 0%表示全透明，0.5 或.5 或 50%表示半透明，值越大越不透明。例如 color: rgba(150,150,100,.6)。

<hsl()>：HSL 颜色模型的样式函数 hsl(h, s, l)根据色相、饱和度和亮度成分定义给定的颜色，参数 h 代表色相，单位为度，比如 0deg 和 360deg 表示红色，120deg 代表绿色，单位 deg 可以省

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

略，参数 s 代表饱和度，取值范围是[0%,100%]，100%完全饱和，0%就是灰色了。参数 l 代表亮度，取值也是 0%~100%，100%最亮（即白色）、0%最暗（黑色），参考图 5.17。例如 color: hsl(120deg, 100%, 50%)。

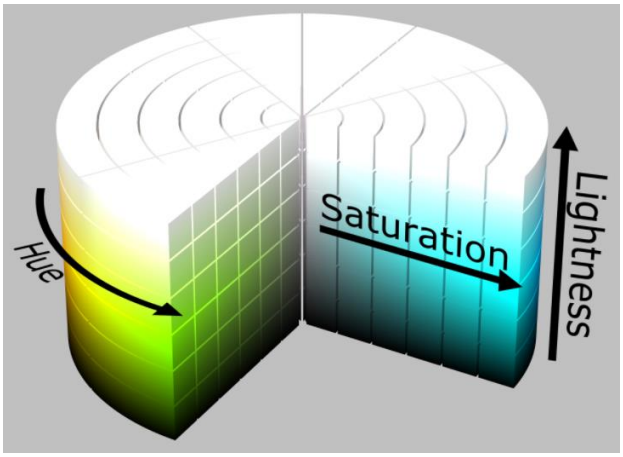


图 5.17 HSL 颜色模型

<hsla()>：在 hsl()的基础上增加了透明度。透明度的介绍参见 rgba()。

可选的 alpha 组件代表颜色的透明度。

currentcolor：引用父元素的 color 属性的值。

<deprecated-system-color>：在强制颜色模式下，大多数颜色都限制在浏览器定义的调色板中，可通过下面的关键字直接选取：ActiveText（活动链接文本的颜色）、ButtonFace（按钮的颜色）、ButtonText（按钮文本的颜色）、Field（输入框背景色）、FieldText（输入框文本的颜色）、GrayText（被禁用文本的颜色）、Highlight（被选中文本的背景色）、HighlightText（被选中文本的前景色），其他的还有 Canvas、CanvasText、LinkText、Mark、MarkText、VisitedText。例如 color: Hightlight。

7、user-select — 是否允许用户选中页面文本

此属性的语法如下：

```
user-select: auto | text | none | contain | all
```

none 表示元素及其子元素的内容不能被选择，text 允许选中文本，all 全部被自动选中，比如点击鼠标，元素的所有内容就被选中，contain 选择的起点落在元素内才行，比如用户在元素外开始按下鼠标，然后移动鼠标进入元素，此时不能选中元素内部的内容，目前只有 Edge 浏览器支持 contain 值。

5.4.3 装饰

CSS3 样式速查表·文字内容·装饰”分类列举了全部的定义文本的样式。

1、text-shadow — 定义文本阴影

文本的阴影包括普通的阴影和倒影两种，其中普通的阴影效果采用样式属性 text-shadow 定义，它的使用语法是：

```
text-shadow: none|offsetX offsetY [blur] color [, x-offset y-offset [blur] color, ……]
```

默认取值 none，表示没有阴影。同一文本允许定义多个阴影，中间用逗号分隔。各个参数的理解参见图 5.18。

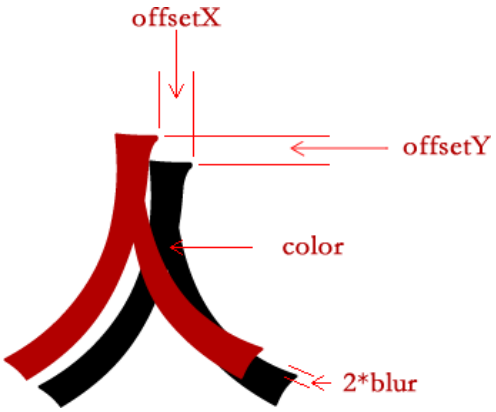


图 5.18 阴影说明

offsetX 和 offsetY 是水平和垂直方向的偏移，可为负数；blur 是模糊半径，就是以阴影的边沿为中心向内向外各模糊 blur，总模糊距离就是 2*blur；color 就是阴影的颜色，默认为黑色。看例子：

```
text-shadow: 3px 3px black;
```

浏览效果如图 5.19。

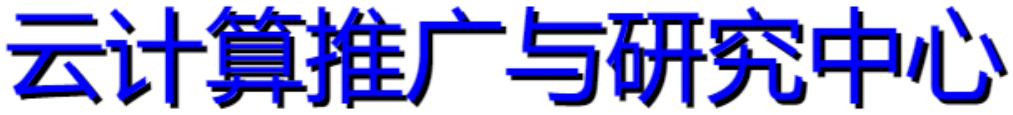


图 5.20 文本阴影-基本

在上面的例子中添加模糊：

```
text-shadow: 3px 3px 4px black;
```

效果如图 5.21。



图 5.21 文本阴影-模糊

下面的例子添加两种阴影：

```
text-shadow: -2px -2px #333, 2px 2px #DDD;
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

背景色和文本颜色都采用灰色，浏览效果如图 5.22。



图 5.22 给文本附加两种阴影

实操练习

完成如图 5.23 所示的文本效果。



图 5.23 阴影实验

2、-web-box-reflect — 设置文本倒影

使用-webkit-box-reflect 设置文本倒影，不过目前只有 chome 浏览器支持，它的使用语法如下：

```
-webkit-box-reflect: below|above|left|right[ length][ image]
```

其中“below | above | left | right”指明倒影出现在文本的“下方|上方|左边|右边”，length 为文本与倒影之间的距离，image 为遮盖图形，一般用渐变对象。看例子：

```
p{
    -webkit-box-reflect: below;
    color: black;
    font-size: 100px;
}
……
<p>看到倒影了</p>
```

采用 chrome 浏览，效果如图 5.24。

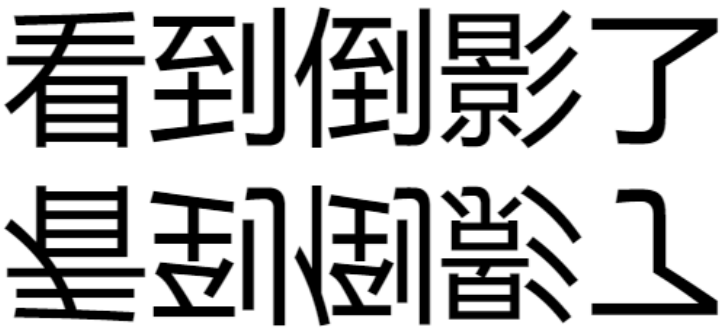


图 5.24 基本倒影

继续修改距离：

```
-webkit-box-reflect: below -10px;
```

浏览的效果如图 5.25。



图 5.25 调整文本与倒影之间的距离

再继续添加遮盖：

```
-webkit-box-reflect: below -10px linear-gradient(transparent,   transparent 40%,
rgba(255, 255, 255, .5));
```

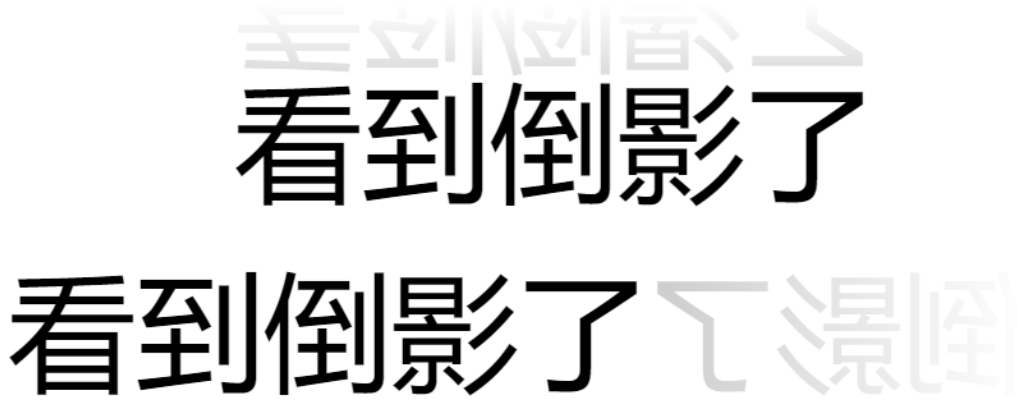
最后看到的效果如图 5.26。



图 5.26 增加遮盖

实操练习

完成如图 5.27 所示的效果（提醒：右侧倒影，给<p>元素定义宽度）。



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 5.28 阴影/倒影实验

3、text-decoration/ text-decoration-*/ text-underline-* — 设置文本划线

text-decoration 样式用于给文本添加划线，它是一个复合属性，对应的分属性为 text-decoration-line、text-decoration-color 和 text-decoration-style，其中 text-decoration-line 定义文本划线位置，比如上划线、下划线什么的，text-decoration-style 定义划线类型，诸如实线、波浪线等，text-decoration-color 定义划线的颜色。text-decoration-line 属性的语法如下：

```
text-decoration-line: none | [ underline || overline || line-through || blink ]
```

none 没有划线、underline 下划线、overline 上划线、line-through 中划线、blink 闪烁。允许同时定义多根画线，如 text-decoration-line: underline overline blink。

text-decoration-style 属性的语法如下：

```
text-decoration-style: solid | double | dotted | dashed | wavy
```

solid 实线、double 双实线、dotted 点画线、dashed 虚线、wavy 波浪线，只能选一种。

看例子：

```
    text-decoration-line: underline blink;
    text-decoration-style: wavy;
    text-decoration-color: red;
```

浏览的效果如图 5.29。

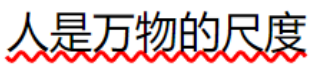


图 5.29 文本装饰用法举例

实操练习

完成如图 5.30 所示的文本装饰。

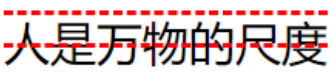


图 5.30 文本装饰实验

复合属性 text-decoration 的语法如下：

```
<'text-decoration-line'> || <'text-decoration-style'> || <'text-decoration-color'>
```

例如：

```
text-decoration: double blink solid blue
```

text-underline-position 属性定义划线的位置，比如文字竖排时，在左边还是右边。它的使用语法如下：

```
text-underline-position: auto | from-font | [ under || [ left | right ] ]
```

auto 由浏览器自行决定，如$C_8H_{10}N_4O_2$。from-font 如果字体文件中存在划线位置的信息，就使用它，否则由浏览器自行决定。under 在下标以下（如果存在下标的话），如$C_8H_{10}N_4O_2$。left 竖排文本的左侧，right 竖排文本的右侧。

text-underline-offset 属性定义划线位置偏移量，负值表示更靠近文本。其使用语法如下：

```
text-underline-offset: auto | <length> | <percentage>
```

<length>通常采用 em 单位，这样随字号大小自动调整偏移量，如 text-underline-offset: 0.3em。<percentage>指名 1em 长度的百分数，比如 50%实际上就是 0.5em。

4、text-emphasis* — 设置着重强调记号

text-emphasis-style 属性定义着重强调记号的形状，它的使用语法如下：

```
text-emphasis-style: none | [ [ filled | open ] || [ dot | circle | double-circle | triangle | sesame ] ] | <string>
```

none 无记号，filled 实心记号，open 空心记号，dot 小圆圈记号，如'•'、'◦'，circle 大圆圈记号，如'●'、'○'，double-circle 双圆圈记号，如'◉'、'◎'，triangle 小三角记号，如'▲'、'△'，sesame 芝麻记号，如'﹅'、'﹆'，<string>自定义的字符串作记号，不过最好不要超过一个字符或一个汉字，如'*'。

text-emphasis-color 属性设置记号的颜色，颜色值请参考 color 属性。

text-emphasis 是上面两个属性的复合属性，它的语法如下：

```
text-emphasis: <'text-emphasis-style'> || <'text-emphasis-color'>
```

对于 chrome 浏览器，这些属性都要加上前缀-webkit-。例如：

```
span {
    text-emphasis: filled double-circle red;
    -webkit-text-emphasis: filled double-circle red;
}
<p>小时候恨<span>蒋介石</span>是因为不明白事理！</p>
```

浏览的效果参见图 5.31 的左侧。

text-emphasis-position 属性定义强调记号相对文本的位置，如上面、下面等，它的语法如下：

```
text-emphasis-position: [ over | under ] && [ left | right ]
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

over 横排的上方，under 横排的下方，left 竖排的左侧，right 竖排的右侧。在上面的例子中增加样式-webkit-text-emphasis-position: under;，这时浏览的效果见图 5.31 的右侧。

小时候恨蒋介石是因为不明白事理！　　小时候恨蒋介石是因为不明白事理！

图 5.31 text-emphasis*属性用法举例

5.5 元素定位

“CSS3 样式速查表 • 元素定位”分类列举的样式用于确定元素再文档流中的位置。

1、display — 元素类型及其子元素排列

我们都知道一个元素里面可能还包含更多的子元素，那么这个元素及其子元素如何排列显示（比如排成一行、排成一列、排成网格、浮动等）是由 CSS3 的 display 属性定义的。参考示意图 5.32。

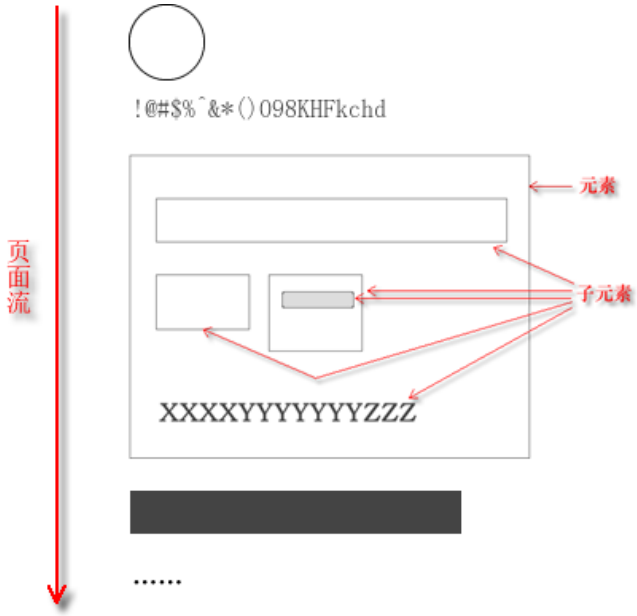


图 5.32 渲染后的文档流

不但涉及元素本身在页面流中的排列，还涉及元素里的子元素如何排列的问题，所以 display 样式可取的值特别多，参考图 5.33。

1.元素外在表现	2.影响子元素排列	3.列表	4.元素内部结构	5.是否渲染	6.CSS2遗留
display: [<display-outside> || <display-inside>] | <display-listitem> | <display-internal> | <display-box> | <display-legacy> ;					
block inline	flow-root flex grid table	list-item	table-row-group table-header-group table-footer-group table-row table-cell table-column-group table-column table-caption	none	inline-block (inline flow-root) inline-list-item (inline list-item) inline-table (inline table) inline-flex (inline flex) inline-grid (inline grid)

图 5.33 display 样式用法

可取的值分为 6 类，其中 1、2 类允许合并使用，成为双值，如 display: inline flow-root，其他类是互斥的，比如只能 display: none。第 6 类表示 css2 遗留用法，小括号内的双值是等价的 CSS3 用法，建议大家不要采用 CSS2 的用法，尽管浏览器依然支持。

目前比较常用的值有 1 类（block/inline）、2 类（flex/grid）、5 类和 1/2 类组合的双值（inline flow-root/inline flex/inline grid），下面分别介绍这些值，对于一些即将淘汰或者很少使用的值这里不做介绍。

影响元素本身在页面流中的排列的值有四个（简称为 display 的外部值）：block、inline、inline flow-root、none，其中 none 导致元素不被渲染，即不占用页面流空间，如果某个元素要频繁第显示和隐藏，不建议采用 display: none|block 样式，而推荐采用 content-visibility: hidden|visible 样式，因为 display:none 会彻底破坏渲染效果，下次显示时又得再次渲染，而后者不会。下面详细介绍其他三个值。

display: block

使元素成为块元素——元素在文档流中要占据一行，尽管元素本身宽度可能很小，这时左右留下空白。容器类型的元素基本上默认都是这个值，比如除了常见的 之外，还有<h1>到<h6>、<p>、<form>、<header>、<footer>、<section>和<article>等等。一个块元素可以通过添加 display:inline 样式强行转化为内联元素。块元素可以设置 width、height 值，尽管 width 可能设置得非常小，但还是占据一整行，如下图 5.34 所示。

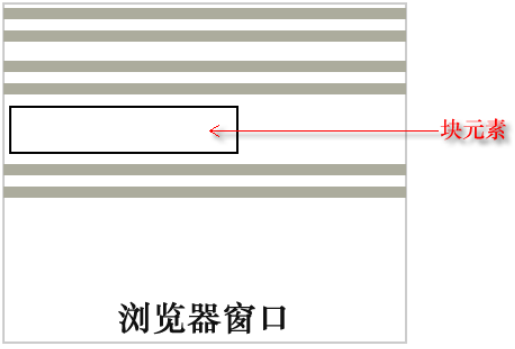


图 5.34 块元素

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

例如下面的样式强行把元素转化为块元素，所以浏览时那个按钮没与在同一行：

```
<span style="display: block">本来我是内联元素的，被你转为块元素了，杯具！</span>
<button>我不能和你同行了</button>
```

display: inline

称为内联元素——元素按照内容实际宽度占用页面空间，所以在页面的同一行内可以排列多个内联元素。最典型的内联元素有、<a>、<img>、<button>等。对内联元素给 width 和 height 属性赋值将无效。一个内联元素可以通过添加 display:block 样式强行转化为块元素。参考示意图 5.35。

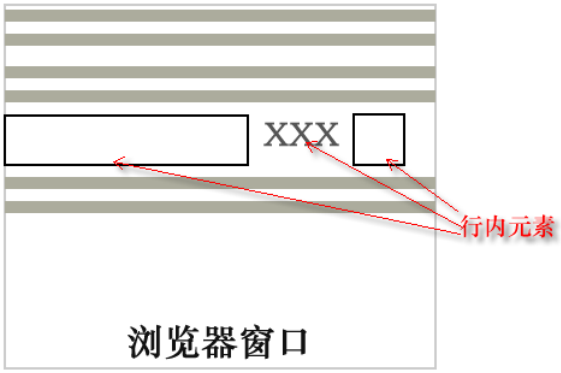


图 5.35 行内元素

display: inline flow-root

成为内联块元素——按照元素内容的实际宽度或者 width 样式属性的值占据一行内的部分空间，同一行内只要有足够多的空间，则允许排列更多的非块元素。给内联块元素的 width 和 height 属性赋值将有效。参考示意图 5.36。

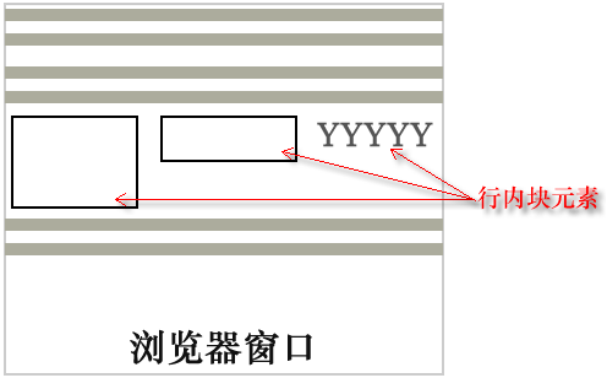


图 5.36 内联块元素

与内联元素最主要的区别是内联元素不能设置宽度和高度（设置了也无效）。

影响元素里的子元素的排列的值有很多（简称为 display 的内部值），但是目前最流行的有两个，即 flex 和 grid，这两个值同时隐含着元素是块元素，如果要转变为内联元素的话，可以采用双

值：inline flex 和 inline grid。其他的内部值逐渐被淘汰，所以这里不再讲解 了。flex 的讲解请参考“§5.10.2 弹性布局”章节，grid 的讲解请参考“§5.10.3 网格布局”章节。

2、float — 浮动方式

如果一个元素的 float 样式的值不为 none，那么这样的元素称为浮动元素，浮动元素脱离了正常的文档流，向左或向右移动，直到碰到父元素的左/右边线或者另一个浮动元素为止。尽管浮动元素脱离了文档流，但是在页面渲染后，浮动元素要继续占据其最终停靠时所在的内联空间，这样其他的文本或内联元素内容会环绕在周围，参考图 5.37 所示。

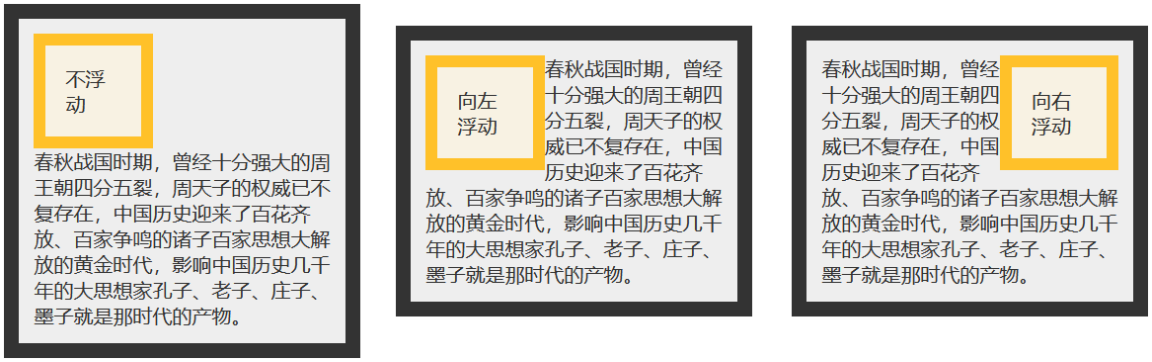


图 5.37 浮动元素示意图

但是后续的块元素会占据浮动元素覆盖的空间。

float 样式用法：

```
float: left | right | none | inline-start | inline-end
```

这五个值表示依次左浮动、右浮动、不浮动、向父元素的起始位置浮动（如父元素是 ltr 语系，如英语，起始位置就是左侧）、向父元素的终止位置浮动。看例子：

```
<div style="width:200px; border: 1px solid black;">
    <div style="float: left; width: 60px;height: 60px; border-
radius: 50%;background-color: red;"></div>
    <div style="float: left; width: 60px;height: 60px; background-
color: blue;"></div>
    生为女人就该像个女人样，不好好刺绣，硬要去踢足球搞举重打拳击参合男人不光彩的勾当，喉结胡子腹肌腿肚子与男人一样，这下子女权运动成功了！
</div>
```

浏览的效果见图 5.38



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

图 5.38 float 属性用法举例

3、clear — 清除侧边元素

并非要删除左/右元素，清除左侧元素就是另起一行显示本元素，类似清除右侧元素，就是本元素所在的行右边不允许放置其他元素。

clear 属性的语法如下：

```
clear: none | left | right | both | inline-start | inline-end
```

分别表示不清除、清除左侧、清除右侧、清除两侧、清除内联轴起始方向、清除内联轴终止方向的元素。在 ltr 语系（如英语、中文等）中，内联轴起始方向就是左侧。看例子：

```
<div style="width: 200px; border: 1px solid;">
    <div style="float: left; width: 60px;height:60px;border: 1px solid;"> 左浮
</div>
    <div style="float: right;width: 50px;height: 120px;border: 1px solid;"> 右浮
</div>
    <div style="clear: left;">
        同性恋、男女同厕、吸大麻、娈童、女权运动、变性从来都是富贵病，一个只有汗流浃背才能得温饱的人是根本没有时间和精力去关注胯下之物的，更何来厌弃它更换它之理?
    </div>
</div>
```

大家直接在浏览器的开发者工具中把 left 修改为 right、both、none，看看效果如何。

4、visibility — 是否显示元素

隐藏或显示元素（包括表的行、列），默认是显示。隐藏的元素继续占据页面空间，这点与 display:none 不同，应用了 display:none 样式的元素根本不渲染，相当于存在了。

visibility 样式的用法:

```
visibility: visible | hidden | collapse
```

前两个值分别表示显示、隐藏，而 collapse 类似 hidden，但是对于<table>元素和弹性容器的子元素来说，不但隐藏而且不占据页面空间，但在计算表中的行高和列宽时，视其继续存在，例如表的总高度 100px，共 5 行，1 行被 collapse 了，最终渲染了 4 行，而且每行的高度仍然是 20px。看例子：

```
td{
  border: 1px solid gray;
}
……
<table style="border: 1px solid red;">
  <tr><td>1.1</td><td style="visibility: collapse;">1.2</td><td>1.3</td></tr>
  <tr style="visibility: collapse;"><td>2.1</td><td>2.2</td><td>2.3</td></tr>
  <tr><td>3.1</td><td>3.2</td><td>3.3</td></tr>
</table>
```

浏览效果见图 5.39

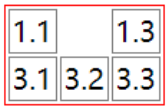


图 5.39 隐藏空的单元格

5、overflow/overflow-x/overflow-y/overflow-block/overflow-inline — 如何处理溢出内容

当元素内容过多从而超出元素空间时，通过这三个属性告诉浏览器如何处理：隐藏、剪除或增加滚动条等。其中 overflow 是 overflow-x 和 overflow-y 的复合属性，overflow-x 只针对内容超过元素左/右边界，overflow-y 只针对超出上/下边界，而 overflow 针对元素的四个边界。他们的使用语法如下：

```
overflow-x: visible | hidden | clip | scroll | auto
overflow-y: visible | hidden | clip | scroll | auto
overflow: [ visible | hidden | clip | scroll | auto ]{1,2}
```

针对溢出的内容，这个五个值分别表示显示、隐藏、剪除、增加滚动条、由浏览器自行处理。对于 auto 值，目前大部分浏览器是这样处理：当内容没溢出时不出现滚动条且不给滚动条预留页面空间，一旦内容溢出就出现滚动条，这会使页面发生“跳变”。但如果本属性取值 scroll，那么效果与 auto 差不多，只是当内容没溢出时会给滚动条预留空间。

clip（剪除）与 hidden（隐藏）类似，但是 clip 还会彻底禁止滚动条（编程增加关东条也不行）。

如果 overflow 属性为单值，那么水平和垂直两个方向的溢出内容做同样处理，如果为双值，那么前一个值处理水平方向，后一个值针对垂直方向。参考示意图 5.40。

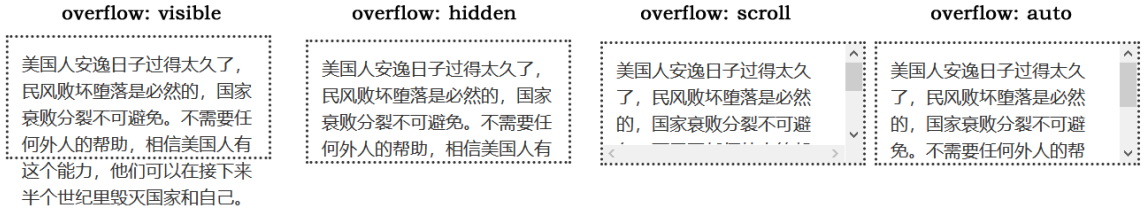


图 5.40 overflow 属性用法举例

overflow-block 属性指示浏览器当行进方向上溢出时如何处理，取值与 overflow-x 属性一样，overflow-inline 属性指示浏览器当书写方向上溢出时如何处理，取值与 overflow-x 属性一样。关于行进方向和书写方向，请参考“§5.4.2 文本”小节。

6、overflow-anchor — 定义滚动锚定

经常上网的人应该都有这个体会：浏览一个内容丰富的网页，由于网速原因，首先出现文本，然后不断加载图片等其他内容，你会看到页面在不断变化。你急不可耐地滚动到文字部分，想在浏览器还在加载页面的同时先浏览文本信息，可是页面在不断调整，因此你正在阅读的文本的位置也在不断变动，你得不断滚动定位，所以有点烦。不过你的烦恼已不在了，因为 CSS3 引入了 overflow-anchor 属性——告诉浏览器是否滚动锚定。默认值 auto 表明开启滚动锚定，此时你滚动定位到的内容固定在屏幕上不动。取值 none 则关闭滚动锚定。scrollbar-anchor 属性一般设置在<body>或包含很多内容的元素上。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

7、position — 元素定位模式

一个元素出现在网页上的位置是由样式属性 position 并结合 left、right、bottom 和 top 样式确定的，该属性有 5 种可能的值：

```
position: static | relative | absolute | sticky | fixed
```

1) static

static 是 position 属性的默认值，即如果省略 position 属性，浏览器就认为该元素是 static 定位。这时，浏览器会按照网页结构源码的顺序，决定每个元素的位置，这称为"正常的页面流"（normal flow）——每个块级元素占据自己的区块（block），元素与元素之间不产生重叠，这个位置就是元素的默认位置。参考图 5.41。



图 5.41 元素 static 定位

注意，static 定位所导致的元素位置，是浏览器自主决定的，所以这时给元素的 top、bottom、left、right 这四个样式属性赋值不起作用。

下面的代码：

```
<button type="button" onclick="ask()" style="position: static;">确认</button>
```

与：

```
<button type="button" onclick="ask()">确认</button>
```

是等价的。

2) relative

relative、absolute、fixed 这三个值有一个共同点，都是相对于某个基点进行定位，不同之处仅仅在于基点不同。所以，只要理解了它们的基点是什么，就很容易掌握这三个属性值。这三种定位都不会对其他元素的位置产生影响，因此元素之间可能会产生重叠。

先来看看 relative。

定位基点就是元素的默认位置（static 定位时的位置），再结合 top、bottom、left、right 四个属性的值使得元素相对于基点发生偏移，显然如果不设置这个四个属性，relative 定位与 static 定位对本元素来说效果是一样的（但对子元素的定位影响会不同），参见图 5.42。

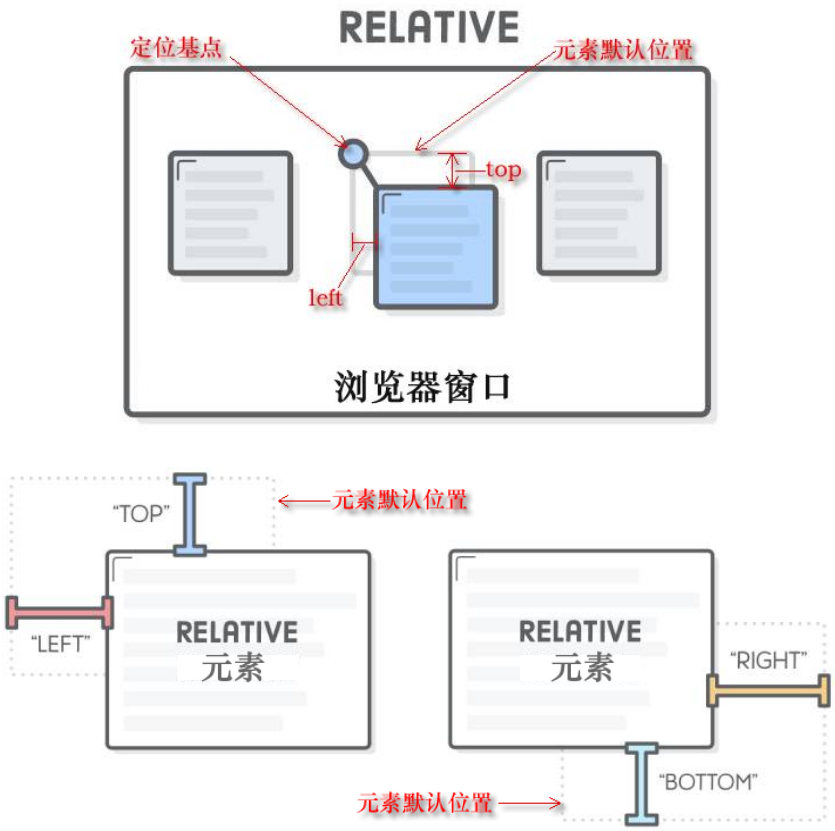


图 5.42 relative 定位说明

例子：

```
<head>
    <title>relative 定位</title>
    <style>
        div {
            position: relative;
            top: 10px;
            left: 20px;
            border: 2px solid red;
            width: 100px;
            height: 100px;
        }
    </style>
</head>

<body>
    <div></div>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
</body>
```

上面的代码中， 元素从默认位置向下偏移 10 个像素向右偏移 20 个像素。可以尝试下，去掉 top 和 left 所在的两行代码，看看浏览的效果会如何变化。

3) absolute

定位基点是辈分最近的非 static 定位的祖宗元素，如果所有的祖宗元素都是 static 定位，那么网页的根元素<html>就被选定为定位基点。在确定基点后可以结合 top、left、right 和 bottom 属性产生偏移，参考图 5.43。

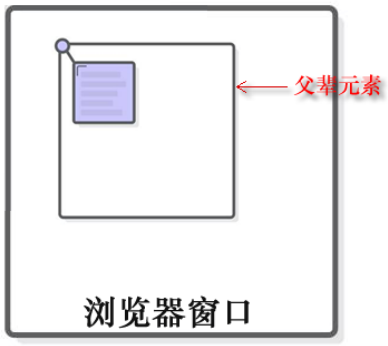


图 5.43 absolute 定位

如何确定辈分最近的非 static 定位的祖宗元素呢？就是先看父元素是不是非 static 定位，如果不是则继续找爷元素是不是非 static 定位，依次类推，直到找到一个非 static 定位的祖宗元素为止。另外要注意的是，absolute 定位的元素会被"正常页面流"忽略，即在"正常页面流"中，该元素所占空间为零，你可以想象此元素好像在另外一个图层上。

4) fixed

相对于视窗（viewport，即浏览器窗口）进行偏移（通过设置 top、left、right、bottom），即定位基点是浏览器窗口，这会导致元素的位置不随页面滚动而变化，好像固定在屏幕上一样。参考图 5.44。

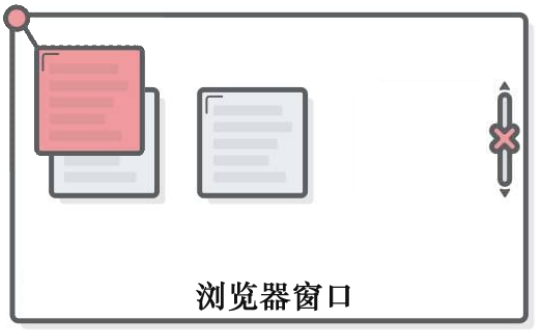


图 5.44 fixed 定位

另外，fixed 定位的元素也脱离了"正常页面流"。下面的例子把 元素固定再浏览器窗口的顶部。

```
div {
     position: fixed;
     top: 0;
}
```

5) sticky

sticky 跟前面四个属性值都不一样，它会产生动态效果，很像 relative 和 fixed 的结合：一些时候是 relative 定位（定位基点是自身默认位置），另一些时候自动变成 fixed 定位（定位基点是浏览器窗口）。因此，它能够形成"动态固定"的效果。比如，网页的搜索工具栏，刚开始在自己的默认位置（relative 定位），参考图 5.45。

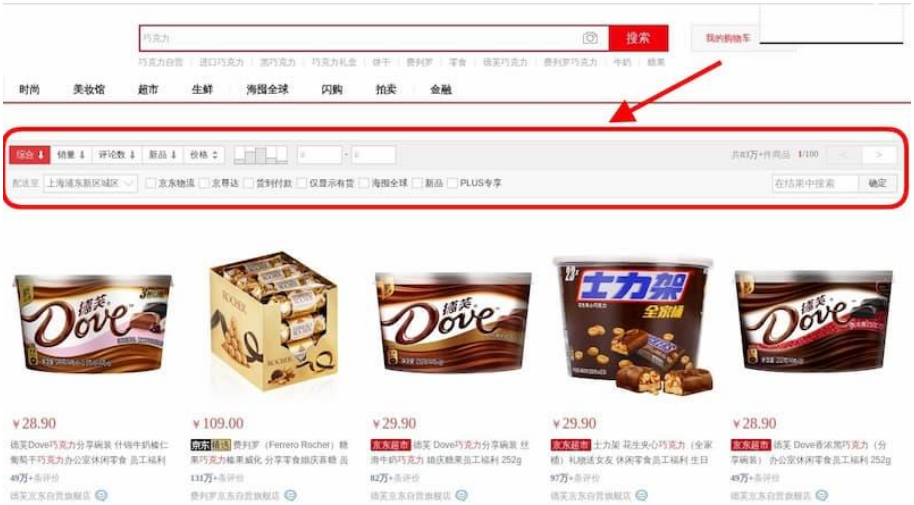


图 5.45 sticky 定位举例之一

当向上滚动页面时，工具栏变成固定位置，始终停留在页面头部（fixed 定位），参考图 5.46。

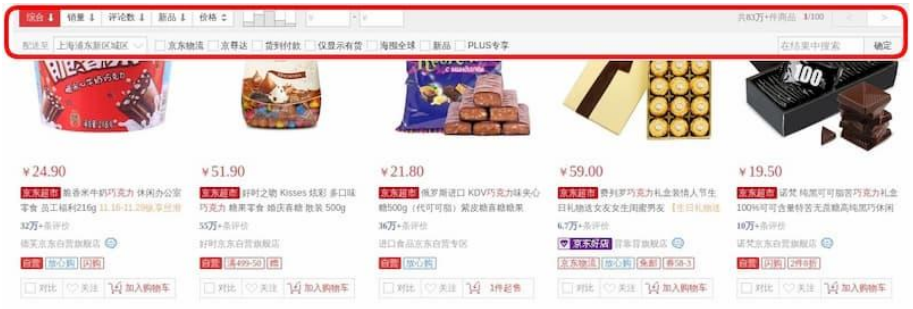


图 5.46 sticky 定位举例之二

等到页面重新向下滚动回到原位时，工具栏也会回到刚开始的位置。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

sticky 生效的前提是，必须搭配 top、bottom、left、right 这四个属性中的一个或多个一起使用，不能省略，否则等同于 relative 定位，不产生"动态固定"的效果。原因是这四个属性用来定义与浏览器窗口边沿的"偏移距离"，比如设置了 top:10px，如果页面向上滚动，当元素离浏览器窗口上边沿等于 10px 时，就触发 fixed 定位，元素就固定不动了。如果存在定位的父元素，则当元素的下边沿与父元素的下边沿重合时，与父元素一起向上滚动，直至离开屏幕。

例子：固定表头，表内容上下滚动。

```
  th {
      position: sticky;
     background-color: black;
     color: white;
     top: 0;
  }
……
  <div style="height:100px"></div>
  <table border="1">
     <thead><tr><th>姓名</th><th>喜欢的颜色</th></tr>
     </thead>
       <tbody>
           <tr><td>Bob</td><td>Yellow</td></tr>
           <tr><td>Michelle</td> <td>Purple</td></tr>
           <tr><td>Bob</td><td>Yellow</td></tr>
           <tr><td>Michelle</td><td>Purple</td></tr>
           <tr><td>Bob</td><td>Yellow</td></tr>
           <tr><td>Michelle</td><td>Purple</td></tr>
           <tr><td>Bob</td><td>Yellow</td></tr>
           <tr><td>Michelle</td><td>Purple</td></tr>
           <tr><td>Bob</td><td>Yellow</td></tr>
           <tr><td>Bob</td><td>Yellow</td></tr>
           <tr><td>Michelle</td><td>Purple</td></tr>
      </tbody>
  </table>
  <div style="height:300px"></div>
```

为了测试效果，尽量缩小浏览器窗口尺寸，参见图 5.47，刚开始整个表一起向上移动，当表头一道顶部时固定不动，而表体继续向上移动。



图 5.47 sticky 定位固定表头

当整张表向上移动离开浏览器窗口时，表头也一起离开，因为<table>是<th>的父元素。再来看一个例子：图片堆叠。

```
<!DOCTYPE html>
<html>
```

```
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>图片堆叠</title>
    <style>
        div {
            position: sticky;
            top: 0;
        }
    </style>
</head>

<body>
    <div><img src="https://picsum.photos/id/10/480/360"></div>
    <div><img src="https://picsum.photos/id/11/480/360"></div>
    <div><img src="https://picsum.photos/id/12/480/360"></div>
    <div><img src="https://picsum.photos/id/13/480/360"></div>
    <div><img src="https://picsum.photos/id/14/480/360"></div>
</body>

</html>
```

浏览效果见图 5.48。

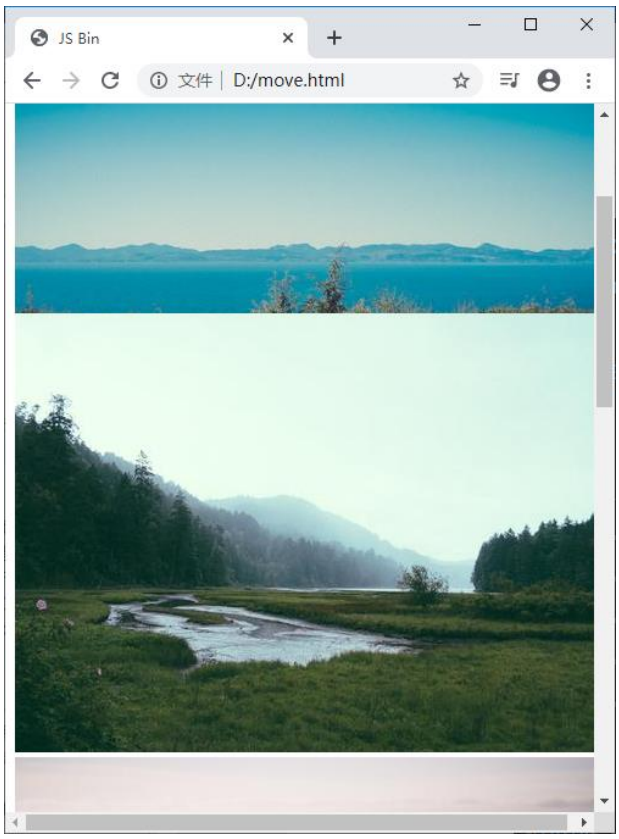


图 5.48 sticky 定位实现堆叠相册

8、top/right/bottom/left —— 元素位置坐标

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

除了 position: static 定位模式外，其他模式都可以通过 top、right、bottom 和 left 样式确定具体的坐标，示意图参考 5.13。top 表示本元素上边界离定位基准的上边界的距离，right 表示本元素右边界离定位基准的右边界的距离，bottom 和 left 的含义类似。这四个属性的取值一样，left 的使用语法如下：

```
left: auto | <length> | <percentage>
```

auto：元素水平位置由 right 样式决定，如果 right 也是 auto，那么元素位置由文档流确定，这时相当于 static 定位模式了。

<length>：某个单位的长度，可正可负，正表示向右偏移，负则向左偏移，如 10px、2rem、1vw\-5ch 等，参见图 5.49。

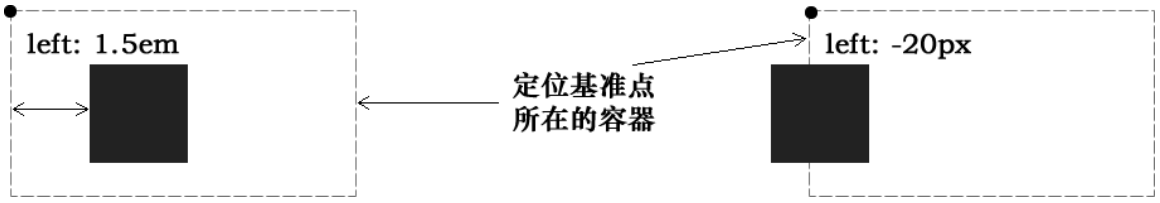


图 5.49 定位基点说明

<percentage>：父元素宽度的百分比，比如 10%表示右移父元素宽度的百分之十，-2%表示左移父元素宽度的百分之二。

其他三个样式属性可与 left 作同样地理解。如果同时定义了 left 和 right 属性，那么 left 起作用，比如 left:-10px、right:20px，则元素左移 10 个像素。如果同时定义了 top 和 bottom 属性，那么 top 起作用。

9、z-index — 元素堆叠序号

z-index 属性定义已定位元素（position 样式的值不是 static）在 Z 轴上（垂直于屏幕并指向人的眼睛）的堆叠序号，序号大的兄弟元素会盖住序号小的兄弟元素。其取值如下：

```
z-index: auto | <integer>
```

auto 由浏览器决定，即在网页文档中，出现在后面的元素会盖住出现在前面的元素。<integer>为无量纲的整数，可正可负也可为 0。

不同辈分的元素的堆叠序号互不干扰。比如 A、B 为兄弟元素，A 的 z-index 为 10，B 的 z-index 为 20，这时 B 会覆盖全部或部分的 B。现在给 A 添加子元素 C，并给 C 的 z-index 设为 50，此时 C 依然在 B 的下面。非兄弟元素之间的 z-index 没有可比性。

5.6 元素尺寸

“CSS3 样式速查表 • 元素尺寸”一级分类中的样式属性用于定义元素尺寸的样式属性，包括内/外边距、边线和元素的尺寸。默认的元素尺寸指内边距所包裹的区域的大小，但是可以通过 box-sizing 属性改变默认计算方法。示意图参考 5.50。

类似于*block*和*inline*的属性名是 CSS3 新加的逻辑属性，*block*是垂直于语言书写方向上的尺寸，而*inline*是语言书写方向上的尺寸，建议多采用这些逻辑样式属性，以便网页适用更多的语言。

这些属性的关系参考图 5.50。

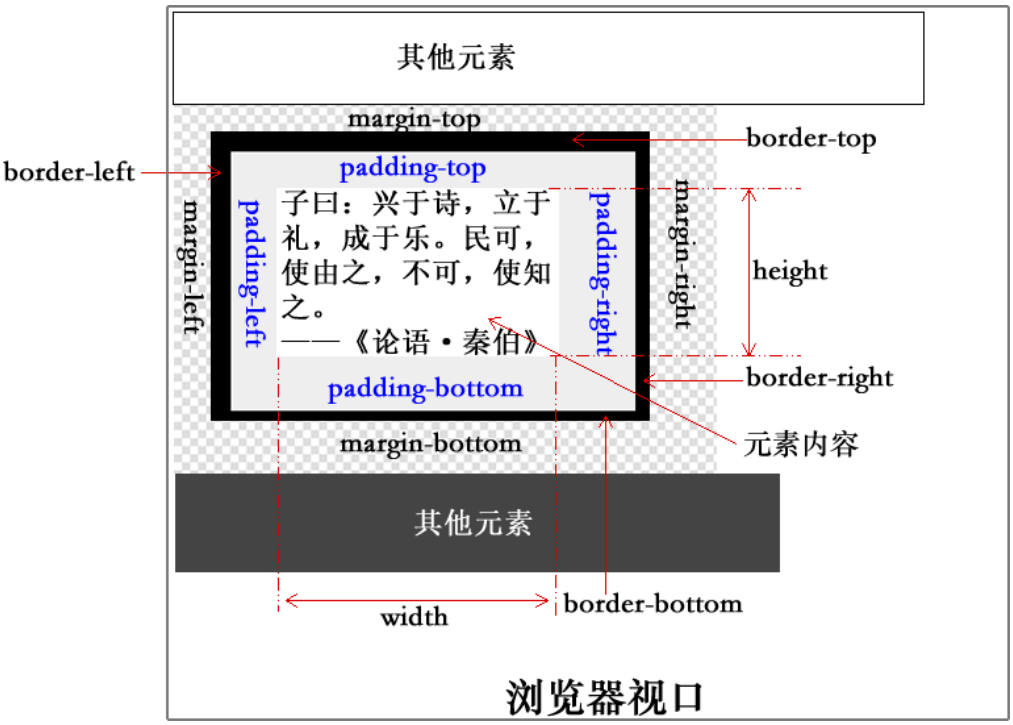


图 5.50 元素尺寸属性的作用

5.6.1 元素的宽和高

根据图 5.50 可知，默认情况下，元素的宽和高是指内边距包围的区域大小，在这个区域中呈现元素内容，由“CSS3 样式速查表 • 元素尺寸 • 尺寸”分类中的样式定义元素的宽和高。

1、width/height/bock-size/inline-size — 定义宽和高

width 和 height 属性的取值定义相同，都是：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
auto | <length> | <percentage> | min-content | max-content | fit-content(<length-percentage>)
```

对于 auto、<length>和<percentage>的介绍请参考“§5.5 元素定位”小节中介绍的 left 属性，对于其余的三个值的理解请参考“§5.10.3.1 应用在容器上的样式”。

对于水平书写的语言，宽就是元素水平方向的尺寸，这很好，但是当页面被翻译成垂直书写的语言（如传统蒙古语）时，这时的“宽”应该是元素的高才对。因此更准确的定义是：元素的宽就是书写方向上的尺寸，高就是行进方向的尺寸，所以为了使页面更好适应国际化的要求，CSS3 推出了一组逻辑样式，其中 inline-size 和 block-size 就是用来定义元素的逻辑宽和高，后面要介绍的 writing-mode 属性定义了书写方向。因此对于绝大多数水平书写的语言（比如英语和中文），inline-size 和 width 样式等价，对于小部分垂直书写的语言，inline-size 和 height 是等价的。看例子：

```
  p {
      border: 1px solid gray;
      display: inline-block;
  }
  .old {
      width: 150px; height: 30px;
  }
  .new {
      inline-size: 150px; block-size: 30px;
  }
……
  <p class="old" style="writing-mode: horizontal-tb;">中国，hello world!</p>
  <p class="old" style="writing-mode: vertical-rl;">中国，hello world!</p>
  <p class="new" style="writing-mode: horizontal-tb;">中国，hello world!</p>
  <p class="new" style="writing-mode: vertical-rl;">中国，hello world!</p>
```

浏览的效果如图 5.51，可见采用 inline-size 技能适应横排也能使用竖写。

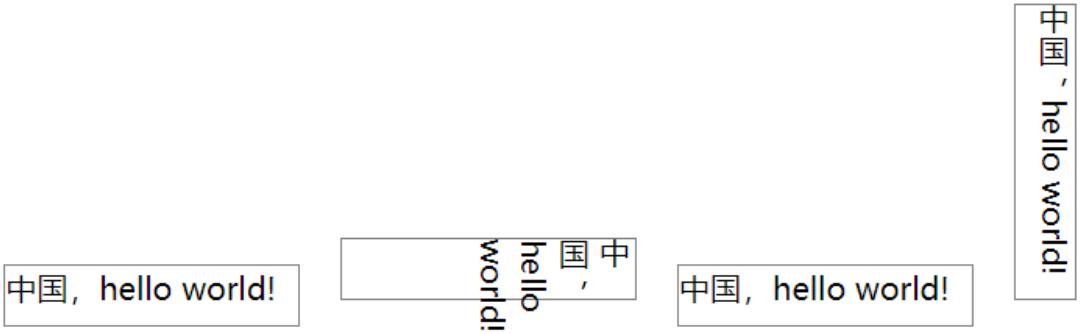


图 5.51 inline-size 属性说明

2、min-width/min-height/max-width/max-height — 定义宽/高的范围

元素的实际尺寸由 width 和 height 决定，但是要满足条件 min-width≤width≤max-width、min-height≤height≤max-height。假如 width 属性的值小于 min-width 属性的值，那么元素的宽度就是 min-width。这四个属性的取值参考上面的 width 属性。

min-block-size、min-inline-size、max-block-size 和 max-inline-size 分别定义了元素的逻辑最小/最大尺寸，必须满足条件 min-block-size≤block-size≤max-block-size、min-inline-size≤inline-size≤max-inline-size。

3、box-sizing/resize/aspect-ratio — 调整尺寸

在默认情况下，元素的 width 和 heigth 属性定义呈现元素内容的区域大小，不包括外边距、边框和内边距，这不利于排版布局页面中的元素。box-sizing 属性可以改变元素尺寸的默认计算方法，它的使用语法如下：

```
box-sizing: content-box | border-box
```

content-box 是默认值，即元素的 width 和 height 只定义元素内容区域大小。border-box 表明元素的 width 和 height 定义的区域包括边框、内边距和内容区，也就是边框所包裹的区域（含边框本身）大小，但不包括外边距。

resize 属性设置元素尺寸能否改变以及朝哪个方向改变，它的语法如下：

```
resize: none | both | horizontal | vertical | block | inline
```

none 不能改变，both 允许横/纵向改变，horizontal 只能横向改变，vertical 只能纵向改变，block 只能朝行进方向改变，inline 只能朝书写方向改变。由 writing-mode 属性定义行进方向，direction 属性定义书写方向。看例子：

```
p {
     overflow: auto;
     border: 1px solid black;
     resize: both;
     inline-size: 322px;
     block-size: 168px;
     text-indent: 2em;
     font: 24px/1.5 隶书, serf;
}
……
<p>代际间的观念传承就是这样从我被、烧到我烧他人，父母教育好小孩是一生中最重要最伟大的使命。</p>
```

浏览的效果如图 5.52。

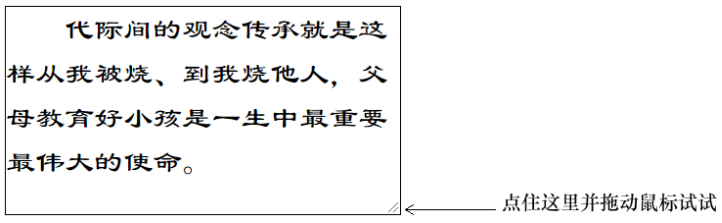


图 5.52 resize 属性举例

对于元素的尺寸，渲染过程中有时需要计算，比如我们定义 width 或 height 属性的值为 auto 时，最终的长度是需要计算的。aspect-ratio 属性设置宽高比，它的使用语法如下：

```
aspect-ratio: auto | <ratio>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

默认值 auto 表示替换元素（如<img>）使用自身固有的宽高比，非替换元素就不存在预设的宽高比，这时要根据元素内容来计算。<ratio>值的格式为宽/高的比值，比如 1/1、4/3 等，此后就采用此比率计算宽或高。看例子：

```
<div style="width:180px;">
    <div style="aspect-ratio: 1/1; border: solid;">
      同卵双胞胎基因几乎一样，性别肯定一样，因而生物层面几乎无差别，可以肯定形成的后天性差别主要在观念上。
    </div>
</div>
```

父元素 只规定了宽度 180px，height 的值是 auto，由于它是非替换元素，所以没有预设的宽高比，因此高度由子元素的高度确定。代码中的子元素 宽和高都是 auto，因此宽等于父元素的宽（因为 是块元素），即 180px，高度等于 180px*1/1=180px。浏览效果见图 5.53 左二。

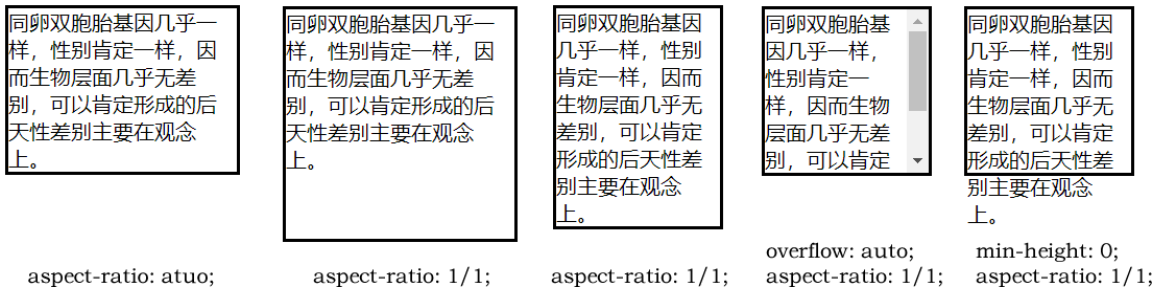


图 5.53 aspect-ratio 属性用法举例

把 aspect-ratio 的值改为 auto，效果见图左一。调整父元素宽度为 130px，宽高比继续为 1/1，这时为了能“装”下内容，高度已经超过 130px 了，见图左三。如果子元素 允许出现滚动条，因此优先满足 1:1 的宽高比的情况下，附加滚动条，见图右二。如果不允许滚动并且重置 min-height 属性（赋值 0），这时满足 1:1 的宽高比，内容超出元素范围，见图右一，为什么呢？

5.6.2 元素的外边距

可通过定义元素的外边距来调正元素之间的距离，示意图请参见图 5.50。“CSS3 样式速查表 •元素尺寸 • 外边距”列举了与外边距有有关的样式。

margin 是其他四个属性的复合属性，用它同时定义四个外边距——元素边框周围的空白区域，用于与其他元素分隔开来。其语法如下：

```
margin: [ <length> | <percentage> | auto ]{1,4}
```

margin 是多值属性，允许出现 1 至 4 个值，如果只有一个值，那么四个边距都相等，具体参见图 5.54

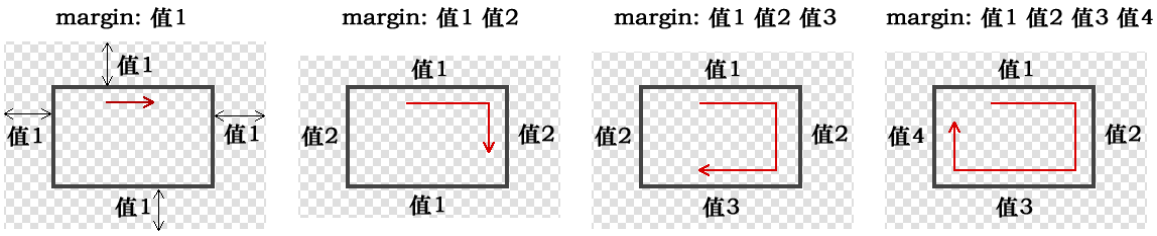


图 5.54 margin 属性用法说明

为便于记忆，我用红色箭头标注：即从上边距开始顺时针分配各个值，没有分配的外边距就等于对端边距。

margin-top 为单值属性，定义上外边距，其值的格式为<length> | <percentage> | auto。其他三个属性作类似理解。

逻辑外边距采用样式 margin-block、margin-block-start、margin-block-end、margin-inline、margin-inline-start 和 margin-inline-end 定义。

5.6.3 元素的内边距

“CSS3 样式速查表 • 元素尺寸 • 内边距” 列出了设置内边距的样式。内边距用于元素边框与内容分隔开来，示意图参见图 5.50。padding 是复合属性，可一次性定义四个内边距。这几个属性的理解与外边距属性一样，这里不再赘述。

逻辑内边距采用样式 padding-block、padding-block-start、padding-block-end、padding-inline、padding-inline-start 和 padding-inline-end 定义。

5.7 元素外观

元素外观涉及边框、背景和元素轮廓三部分。

5.7.1 边框

由于 CSS3 增设了 border-block*、border-inline*逻辑边框属性，所以定义边框的样式特别多，建议大家尽量采用逻辑边框样式，这样更有利于网站的国际化。“CSS3 样式速查表 •元素外观 •边框”列举了用于定义元素边框的样式属性。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

5.7.1.1 边框线

定义元素四条边线的粗度、线型和颜色，其中 border、border-width、border-style 和 border-color 一次性定义四条边线，border-top*定义上边线，border-right*定义右边线，border-bottom*定义下边线，而 border-left*定义左边线。border-top 是 border-top-*的复合属性，border-right 是 border-right-*的复合属性，依此类推，border、border-bottom、border-left 作同样理解。因此这些复合属性使用方法是：

```
border: <border-width> <border-style> <border-color>
border-top: <border-top-width> <border-top-style> <border-top-color>
border-right: <border-right-width> <border-right-style> <border-right-color>
border-bottom: <border-bottom-width> <border-bottom-style> <border-bottom-color>
border-left: <border-left-width> <border-left-style> <border-left-color>
```

允许出现 1 至 3 个值，且顺序没有关系，省略的值就采用默认值，但是对于 border-*-style，浏览器默认的值为 none，即没有边框线。例如下面的例子定义四条边线为红色实线：

```
border: red solid;   /*或者 solid red 或者 1px solid #F00*/
```

下面的例子设置下边线为灰色粗线：

```
border-bottom: 3px solid #666666
```

border-width、border-style 和 border-color 是多值属性，允许对四条边线设置不同的值，使用语法如下：

```
border-width: <line-width>{1,4}
border-style: <line-style>{1,4}
border-color: <line-color>{1,4}
```

比如 border-width 属性的 1 至 4 个值对应关系参见图 5.55，border-style 和 border-color 属性作同样理解。

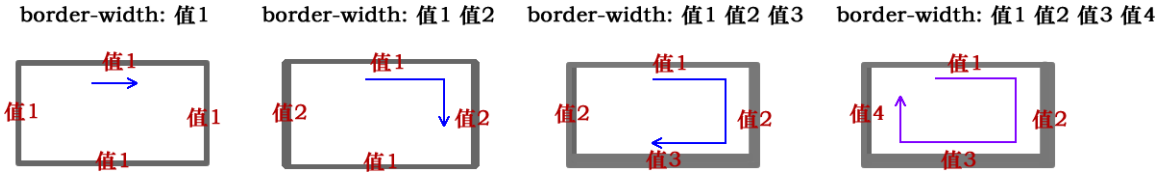


图 5.55 border-width 属性说明

其他各个 border-*-wdth、border-*-style 和 border-*-color 属性的使用语法是（*代表 top、right、bottom 或 left）：

```
border-*-width: <length> | thin | medium | thick
border-*-style: none | hidden | dotted | dashed | solid | double | groove | ridge
| inset | outset
border-*-color: <rgb()> | <rgba()> | <hsl()> | <hsla()> | <hex-color> | <named-
color> | currentcolor | <deprecated-system-color>
```

可用 thin（细线）、medium（中等）、thick（粗线）或具体的数字定义边线的宽度，如 5px。

对于线型，请参考图 5.56，其中 groove、ridge、inset 和 outset，线较粗才效果明显。

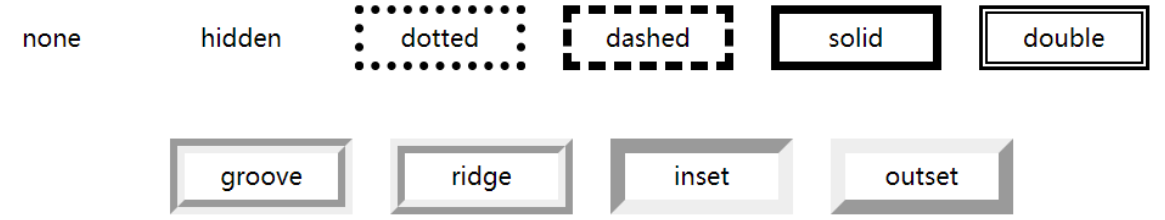


图 5.56 边线类型

线框颜色可用 rgb()函数（如 rgb(200,100,50)）、rgba()函数（如 rgba(0,244,0,0.6)，0.6 表示透明度）、十六进制颜色（如#F0AA33）、颜色名字（如 red）等表示，更详细的介绍请参考 https://developer.mozilla.org/en-US/docs/Web/CSS/color_value。

看一个例子：

```
div {
    border-top: thick dotted red;
    border-right-width: medium;
    border-right-style: solid;
    border-right-color: blue;
    border-bottom: 15px groove;
    border-left-width: 10px;
    border-left-style: ridge;
    border-left-color: green;
    width: 200px;
    height: 100px;
}
……
<div></div>
```

浏览的效果如图 5.57。



图 5.57 边框举例

5.7.1.2 圆角

采用 css3 样式属性 border-radius 定义圆角，每个角的水平半径和垂直半径都可以单独定义。border-radius 是 border-top-left-radius、border-top-right-radius、border-bottom-right-radius 和 border-bottom-left-radius 的复合属性。即使没有边框，border-radius 也会影响元素的背景。见图 5.58。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 5.58 圆角

border-radius 语法如下：

```
border-radius: x[/y] [x[/y] x[/y] x[/y]]
```

x 为水平半径，y 为垂直半径，x、y 可用百分数，可存在 1~4 个值，如果是：

1 个值：四个角都一样；

2 个值：第一个值设左上角和右下角，第二个值设右上角和左下角；

3 个值：第一个值设左上角，第二个值设置右上角和左下角，第三个值设置右下角。

4 个值：分别设置四个角，从左上角开始，顺时针方向，即左上角、右上角、右下角、左下角。

用图 5.59 便于记忆：

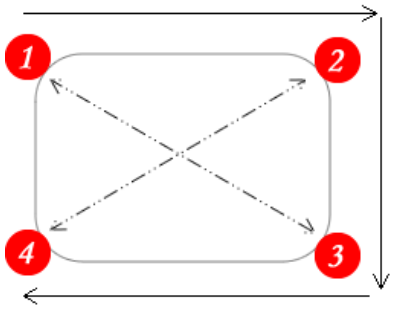


图 5.59 圆角属性定义的顺序

看例子：

```
div{
 …
      line-height: 130px;
      text-align: center;
      border-radius: 30px;
}
…
<div>四个角都一样</div>
```

四个角具有相同的圆角，浏览效果如图 5.60。



图 5.60 四个角圆角一样

下面的例子和相应的效果图 5.61、图 5.62、图 5.63 和图 5.64。

```
    border-radius: 25% 10%;
```



图 5.61 圆角举例之一

```
    border-radius: 10% 20% 30% 40%;
```



图 5.62 圆角举例之二

```
    border-radius: 20% / 50%;
```



图 5.63 圆角举例之三

```
    border-radius: 10px 100px / 120px;
```



图 5.64 圆角举例之四

实操练习

画一个类似图 5.65 所示的橄榄球（提醒：需要旋转 45°）

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 5.65 橄榄球

5.7.1.3 图片边框

切割一幅图片，并用切割出来的各个部分填充元素的边框，从而使得边框呈现色彩斑斓的样子，不过需要增加 border-style 属性才行。请参考图 5.66 进一步理解。

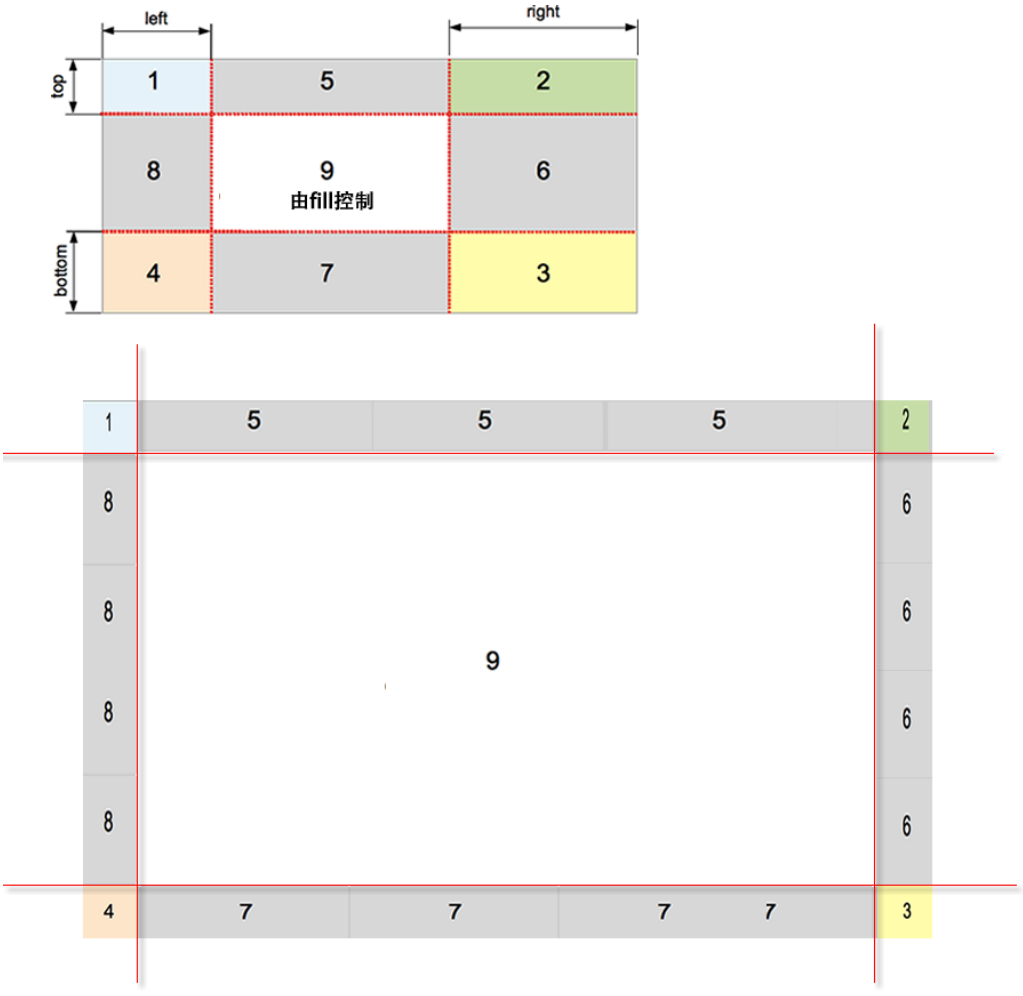


图 5.66 边框图片说明

可以采用属性 border-image 给边框添加图片，此属性是 border-image-source、border-image-slice、border-image-width、border-image-outset 和 border-image-repeat 的复合属性，其语法是：

```
border-image: <'border-image-source'> || <'border-image-slice'> [ / <'border-image-width'> | / <'border-image-width'>? / <'border-image-outset'> ]? || <'border-image-repeat'>
```

根据此语法，可推导出下面这几种用法：

```
border-image: <'border-image-source'> <'border-image-slice'>;

border-image: <'border-image-source'> <'border-image-slice'> <'border-image-repeat'>;

border-image: <'border-image-source'> <'border-image-slice'> / <'border-image-width'>;

border-image: <'border-image-source'> <'border-image-slice'> / <'border-image-width'> / <'border-image-outset'>;

border-image: <'border-image-source'> <'border-image-slice'> / <'border-image-width'> / <'border-image-outset'> <'border-image-repeat'>;
```

其中：

border-image-source 属性定义图片来源，其语法是：

```
border-image-source: none | <image>
```

默认值为 none，<image>表示图片，它的详细用法参考“§5.7.2 背景”中的 background-image 样式。如：

```
border-image-source: url('diamonds.png');
```

或

```
border-image-source:repeating-linear-gradient(45deg,transparent,#4d9f0c 20px);
```

border-image-slice 属性定义分割图片的方法，即分割线与边界的距离，默认值为 100%。其语法是：

```
border-image-slice: [<number>|<percentage>]{1,4}[ fill]
```

其中<number>为无量纲的数字，隐含单位为像素（位图）或坐标（矢量图），fill 指定图片的中间部分作为目标元素的背景图。例如：30（分割线与图形边界的距离都是 30）、10% fill（分割性与图形边界的距离都是 10%，且中间部分作为目标元素的背景图）、20 15%（上下距离 20，左右 15%）、5 10 15（上 5 下 15 左右 10）、7 12 4 15（上 7 右 12 下 4 左 15）。看例子：

```
border-image-slice: 10 5% fill;
```

border-image-width 属性定义元素的图像边框宽度。语法为：

```
border-image-width: [<length>|<percentage>|<number> | auto ]{1,4}
```

其中 length 和 percentage 表示元素边框宽度(如 20px 或 10%)，number 就是目标元素属性 border-width 的倍数，默认值为 1。如果 border-image-width 大于已经定义的 border-width，那么它将向元素内部扩展（会占用 padding/content）。注意：border-image-width 和 border-width 至少有一个存在，否则不会显示边框。参见图 5.67。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 5.67 border-image-width 属性说明

border-image-outset 属性定义边框图像可超出元素边框盒的大小，默认值为 0。语法为：

```
border-image-outset: [<length>|<percentage>|<number> | auto ]{1,4}
```

各个值的意思与 border-image-width 的值相同。在页面中本属性的值不会导致周围其他元素移动。

border-image-repeat 属性定义图片如何填充边框（四个角部除外），语法为

```
border-image-repeat: [ stretch | repeat | round | space ]{1,2};
```

默认是 stretch。其中 round 平铺图像，当不能整数次平铺时，这时剩余空间如果大于二分之一图块，就再增加一个图块，同时所有的图块都相应缩小，当剩余空间小于二分之一图块时，就放大已经铺下的全部图块以便占满剩余空间，参见图 5.68。

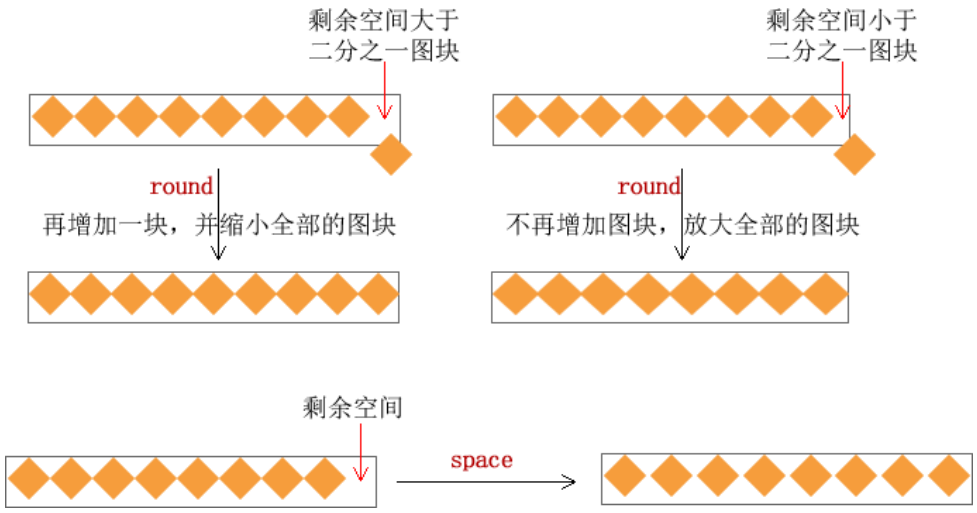


图 5.68 border-image-repeat 属性用法说明

space 类似于 round，但是不改变图片大小，剩余的空间平均分配到各个图块之间。默认值为 stretch。如 border-image-repeat: stretch round，上/下边框中的图块拉伸左/右边平铺。

看例子：

我们切割这个图片  ，形成如图 5.69 所示的边框。



图 5.69 图形边框举例

```
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>元素图形边框</title>
    <style>
        div {
            width: 300px;
            height: 200px;
            background-color: #5b6dcd;
            border-style: solid;
            margin: 50px;
            border-image-source: url('border-diamonds.png');
            border-image-slice: 30 fill;
            border-image-width: 30px;
            border-image-outset: 30px;
            border-image-repeat: space;
        }
    </style>
</head>

<body>
    <div></div>
</body>
```

实操练习

分割下面的图片（参见图 5.70），作为相片的边框（提醒：要增加样式属性 border-style 才会显示边框上的图片），效果参见图 5.71。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

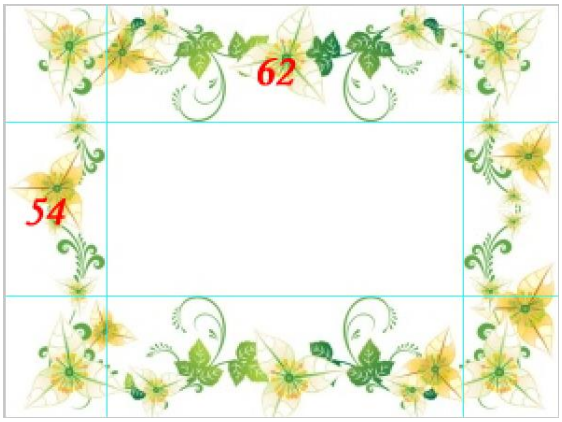


图 5.70 边框图片实验



图 5.71 实验效果图

5.7.1.4 元素阴影

使用 box-shadow 属性定义元素的阴影，其语法如下：

```
box-shadow: offsetX offsetY [blur spread color inset] [,offsetX offsetY [blur spread color inset],…];
```

允许给同一个元素添加多个类型的阴影，各个参数的作用请参考图 5.72。

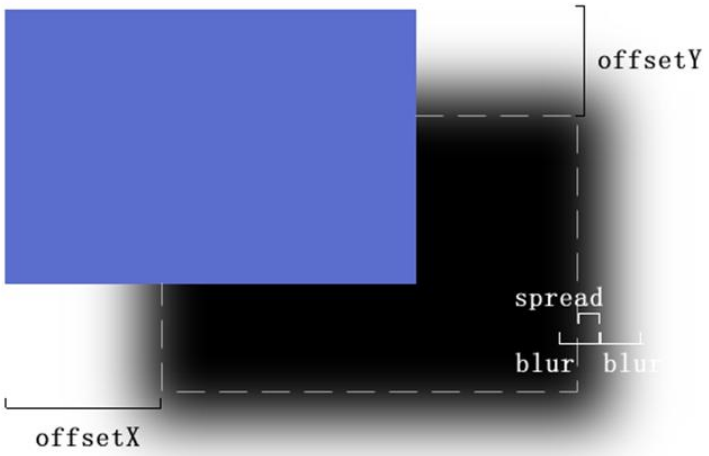


图 5.72 阴影 box-shadow 属性说明

各个参数解释:

offsetX：必需，设置元素阴影水平偏移量，可以为负值，单位是像素。

offsetY：必需，设置元素阴影垂直偏移量，可以为负值，单位是像素。

blur：可选，阴影模糊尺寸，实际的模糊距离是 2*blur，blur 只能为正值，如果为 0px，表示不具有模糊效果，单位像素。

spread：可选，阴影的扩展尺寸，可以为负值（阴影边沿向内收缩），单位是像素。spread 参数直接扩展阴影的边缘，然后 blur 再以这个新边缘为标准进行模糊处理。

color：可选，设置阴影的颜色。如果省略此参数，那么浏览器选取默认颜色，各个浏览器的默认值可能不同，有的为透明，有的为黑色。

inset：可选，可以将外部阴影改为内部阴影。

参见图 5.73 所示的各个例子。

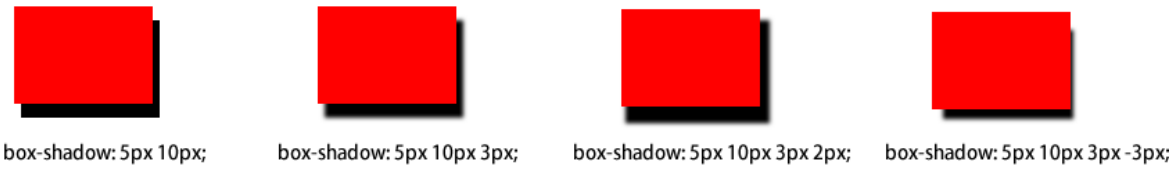


图 5.73 阴影用法举例

再看例子：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    border-radius: 20px;
    box-shadow: -2px -2px 0px white, 2px 2px 0px black;
```

浏览的效果见图 5.74。

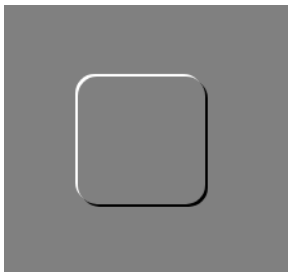


图 5.74 多个阴影

```
    :root {
         background-color: rgb(255, 239, 211);
    }
    div {
         width: 230px;
         height: 300px;
         margin: 20px;
         background-color: white;
         box-shadow: 0px 3px 5px 0px rgba(0, 0, 0, 0.2);
    }
```

浏览效果见图 5.75。

图 5.75 用阴影产生白纸效果

下面的例子产生内部阴影：

```
    box-shadow: 0 0 10px 1px inset black;
```

浏览效果如下：



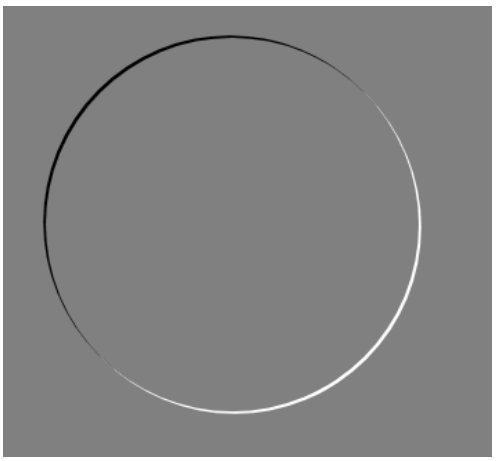
实操练习

1、完成如图 5.76 所示的效果。



图 5.76 漂浮的红气球

2、完成如图 5.77 所示的效果。



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

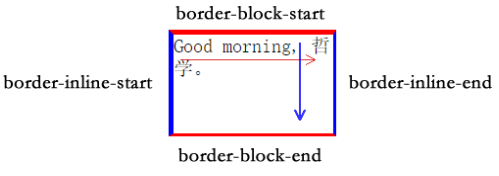
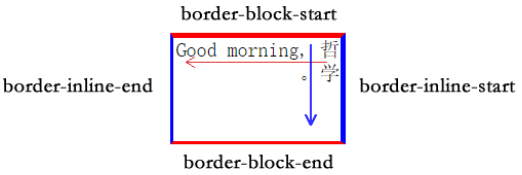
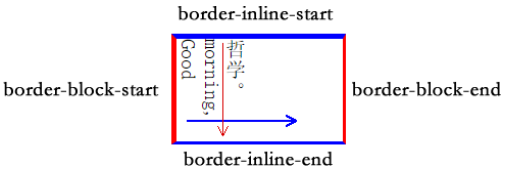
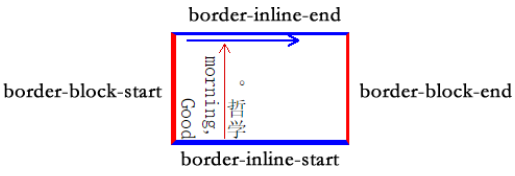
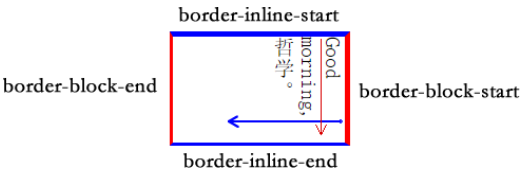
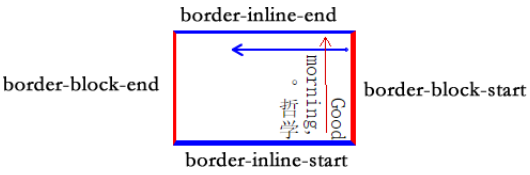
图 5.77 凹下去的圆

5.7.1.5 逻辑边框

为了适应人类的各种语言环境，CSS3 新增了逻辑边框属性，即 border-block*和 border-inline*，浏览器在渲染页面时会根据语言的书写方向把逻辑边框属性映射成非逻辑边框属性，比如可能把 border-block-start-width 映射为 border-top-width，也可能映射为 border-left-width。

人类语言的书写规则由 writing-mode、direction 和 text-orientation 三个属性确定，具体参考“§ 5.4.2 文本”小节。其中 writing-mode 和 direction 两个属性会影响逻辑边框的映射，参见表 5.11。

表 5.11 逻辑边框映射

writing-mode	direction	映射
horizontal-tb	ltr	
horizontal-tb	rtl	
vertical-lr	ltr	
vertical-lr	rtl	
vertical-rl	ltr	
vertical-rl	rtl	

由表可知，多行的行进方向就是 border-block*，见表中的蓝色箭头，行内书写方向就是 border-inline*，见表中红色箭头。对于具体的 border-*-start-width 属性，用法参考 border-width 属性，其他逻辑边框属性用法也是如此。看一个例子：

```
div {
    font: 16px monospace;
    width: 130px;
    height:80px;
    writing-mode: horizontal-tb;
    direction: rtl;
    border-block-start: 4px solid red;
    border-block-end: double green;
    border-inline-start: 4px groove blue;
    border-inline-end: ridge;
}
……
<div>Good morning，哲学。</div>
```

5.7.2 背景

给元素添加背景，主要采用“CSS3 样式速查表 • 元素外观 • 背景”中的样式。

1、background/background-* — 设置背景

采用属性 background 给元素定义若干背景层，各层之间用逗号分隔，左侧的背景层叠加在右侧背景层的上面（离我们眼睛最“近”），所以如果要设置背景色层，那么背景色层必须放在最右侧。例如对于图 5.58 所示的效果，可以这样设计背景层：

```
background: 狼，月亮，天蓝色;
```

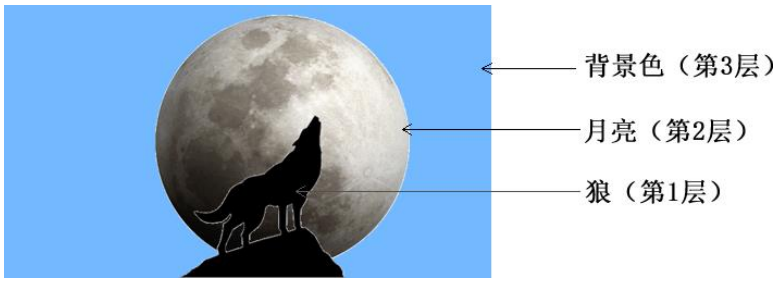


图 5.58 多个背景层

background 是复合属性，它对应的分属性有 background-clip、background-color、background-image、background-origin 、 background-position 、 background-repeat 、 background-size 和 background-attachment，这些都是多值样式属性，值与值之间用逗号分隔。背景可以出现在边框下、padding 下或元素内容下，参考图 5.59。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

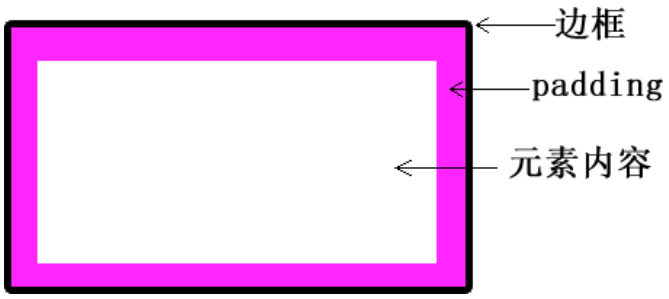


图 5.59 背景样式说明

background 属性的使用语法如下：

```
background: [ <bg-layer>, ]* <final-bg-layer>
其中:
<bg-layer> = <'background-image'>||<'background-position'>[/<'background-size'>]?
||<'background-repeat'>||<'background-attachment'>||<'background-origin'>||
<'background-clip'>
<final-bg-layer>=<'background-color'>||<'background-image'>||<'background-
position'>[/<'background-size'>]?||<'background-repeat'>||<'background-
attachment'>||<'background-origin'>||<'background-clip'>
```

因此为了搞懂 background 属性的用法，我们先要搞懂各个分属性的用法。

background-image

以 Z 轴堆叠方式指定若干背景层，background-image 属性的使用语法如下：

```
background-image: none | <image>[,<image>,...];
```

左侧的背景层堆叠在上面（离眼睛更“近”），所以为了能看到全部的背景图片，一般左侧的图片要小于右侧的图片，默认值为 none。如：

```
background-image: url("images/catfront.png");
```

语法中的<image>表示的“图片”可为下列类型之一：

(1)、**url()函数**。函数的参数可为图片的绝对路径 URL、相对 URL 或者数据 URL，例如 url(https://example.com/images/myImg.jpg)、url(pig.png)、url(……)。其中“data:image/……”表示图片的数据 URL，即图片被编码为 base64 格式的数据后直接附加在后面(相当于嵌入在 HTML5 页面中，这样就减少了资源请求次数)，例子中的“PHN2ZyBmaWxsPSIj……”就是对下面的 SVG 绘图代码作 base64 编码的：

```
<svg fill="#000000" height="24" viewBox="0 0 24 24" width="24" xmlns="http://ww
w.w3.org/2000/svg">
    <path d="M8 5v14l11-7z" />
    <path d="M0 0h24v24H0z" fill="none" />
</svg>
```

此 SVG 代码绘制一个播放图标“▶”。可以直接在浏览器的控制台上输入 JS 代码 btoa(unescape(encodeURIComponent('hello,世界！'))) 对 'hello,世界！' 作 base64 编码为 "aGVsbG8s5LiW55WM77yB"，再敲入代码 decodeURIComponent(escape(atob('aGVsbG8s5LiW55WM77yB'))) 可解码为'hello,世界！'。我们一般

把小图片或者 svg 代码编码成 base64 格式，然后通过数据 URL 引入到页面文件中，可作为元素的背景或边框图形。可以通过下面的方法把图片编码 base64 个数的数据：

用浏览器 firefox 打开一个图片文件，按 Ctrl+Shift+i 打开开发者工具，在控制台中输入代码：

```
    function imgToBase64(img) {
        const canvas = document.createElement('canvas');
        const ctx = canvas.getContext('2d');
        canvas.width = img.width;
        canvas.height = img.height;
        img.crossOrigin = 'anonymous';
        ctx.drawImage(img, 0, 0);
        return canvas.toDataURL();
    }
```

再按 Ctrl+Shift+c，然后用鼠标点击浏览器中显示的图片，这样变量$0 指到了图片的 DOM 对象了，再在控制台上敲入语句 copy(imgToBase64($0))把图片编码后的数据 URL 拷贝到了剪贴板中，最后按 Ctrl+v 粘贴到页面文件中的 url()中即可。数据 URL 的格式如下：

```
url(data:image/<type>base64,<base64 的数据>)。
```

其中<type>定义图片类型，常见的有：jpeg、png、gif、x-icon、bmp、svg+xml 等，如：url(iVBORw0KGgoAAAANSUhEUgAAAA4AAAASC…)。

(2)、**渐变图形**。如 linear-gradient(red,blue)，关于渐变请参考后面的“§5.6.4 渐变”小节。

(3). **由 cross-fade()函数混合两个图片**。firefox 不支持，chrome 浏览器需要加前缀-webkit-，其语法为：

```
-webkit-cross-fade(<image1>,<image2>,<percentage>)
```

<percentage>取值范围为[0,1]或用百分数，它有两个作用：①<image2>的透明度：(1-<percentage>)为<image1>的透明度，<percentage>为<image2>的透明度；②参与计算混合后的图片尺寸，设宽为 width，高为 height，则有计算公式：

width=<image1>.width*(1-<percentage>)+<image2>.width*percentage

height=<image1>. height*(1-<percentage>)+<image2>. height*percentage

例如：

```
div {
    width: 551px;
    height: 551px;
    background-image: -webkit-cross-fade(url('moon.png'), url('wolf.png'), 40%);
}
```

浏览的效果见图 5.60 右一。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

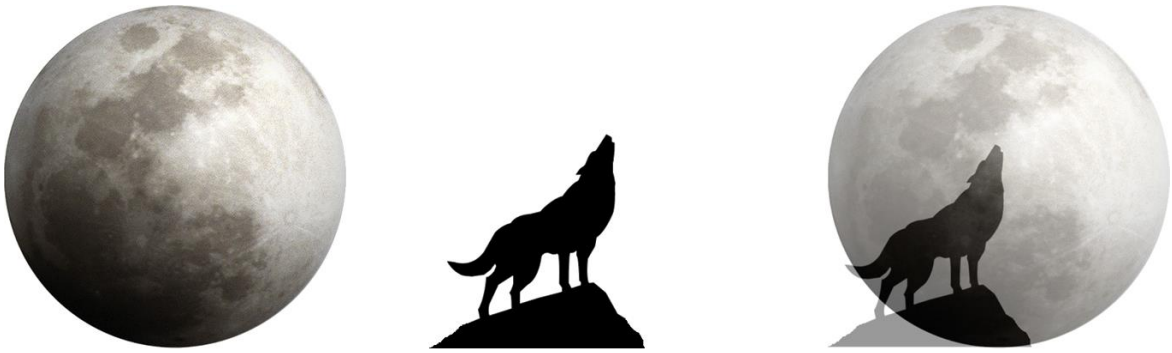


图 5.61 混合两张图片

⑷、**由 image-set()函数指定的一组图片集**。image-set()函数的语法如下：

```
image-set(picture1, picture2,…)
```

其中 pictureN 的格式为<image>|<string> [<resolution>]，<image>|<string>表示必须二选一，如果选择<image>，那么此<image>就不能是 image-set()，否则就是循环嵌套了，一般会选择使用<string>，即用字符串表示的图片文件，如"cat-2x.png"。<resolution>的单位有 x（等同 dppx，即设备像素比）、dpi（每英寸物理像素点个数）等，如果省略<resolution>，默认就是 1x。关于设备像素比，请参考“§2.1 屏幕与长度单位”小节。注意，不能出现多个 picture 具备相同<resolution>的现象。例如：

```
background-image: -webkit-image-set(url(cat.png) 1x, url(cat-2x.png) 2x,
  url(cat-print.png) 600dpi);
```

background-origin

定义图片摆放的区域，其使用语法如下：

```
background-origin: border-box | padding-box | content-box[, border-box | padding-box | content-box, …]
```

对各个属性值的理解请参考图 5.62。

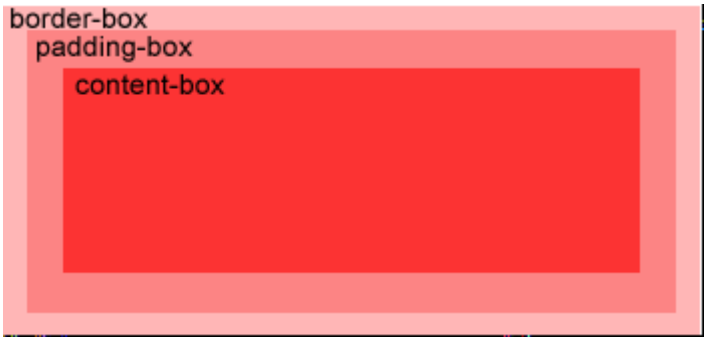


图 5.62 background-origin 属性语法说明

即 border-box 区域最广，背景允许覆盖边框在内的整个元素，padding-box 不包括边框，这是默认值，content-box 连元素的 padding 都不包括。默认值是 padding-box。如果要为 background-image 指定的每幅背景图片单独定义区域，则给 background-origin 多个值，中间用逗号隔开，比如：

```
background-origin: padding-box, border-box
```

表示第 1 幅背景图允许出现在 padding-box 及内部区域，第 2 幅背景图片允许出现在边框及内部的区域，图片的位置最终由 background-position 属性确定。

background-origin 属性的多个值按照相同的次序应用于 background-image 属性的各个图片上，如果这两个属性的值的个数不相等，那么 background-origin 多余的值被丢弃或者最后一个值被重复应用。参考示意图 5.63。

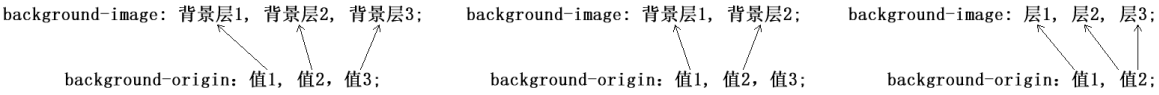


图 5.63 background-blend-mode 用法说明

background-position

为每一幅背景图片在 background-origin 属性规定的范围内定义位置，其使用语法参考图 5.64。

```
background-position:
        ① [ left | center | right | top | bottom | <length-percentage> ] |
        ② [ left | center | right | <length-percentage> ] [ top | center | bottom | <length-percentage> ] |
        ③ [ center | [ left | right ] <length-percentage>? ] && [ center | [ top | bottom ] <length-percentage>? ]
```

图 5.64 background-position 属性的使用语法

值有三种格式：①、②、③，但每次只能选择一种格式，如下面四个例子：

```
background-position: center;            /*水平垂直居中。采用格式①*/
background-position: 5%;                /*离左边和上边都是 5%。采用格式①*/
background-position: right bottom;      /*靠右下角。采用格式②*/
background-position: left 10px top 5%; /*离左边 10px 上边 5%。采用格式③*/
background-position: 0 0;               /*左上角。采用格式②*/
background-position: -1px -1000px;      /*左上角坐标为(-1,-1000)。采用格式②*/
```

background-position 是 background-position-x 和 background-position-y 属性的符合属性，background-position-x 定义水平方向的位置，另一个定义垂直方向的坐标。它们的使用语法如下：

```
background-position: center | [ left | right ]? <length-percentage>?
background-position: center | [ top | bottom ]? <length-percentage>?
```

background-position 属性的多个值按照相同的次序应用于 background-image 定义的各个图片上。

background-position 结合元素的 width 和 height 属性，我们可以实现只显示一个大图片的一部分。比如我们把全世界国家的国旗放在一个图片中，在需要显示一个国家的国旗时，指定 background-position 属性的值即可，参考下图（只显示英国的国旗），一个完整的例子参考 chapter-5/flags.html。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk




```
<div>…</div>
……
<style>
    div{
        background-image: url(images/flags.png);
        background-repeat: no-repeat;
        background-position: 0px -171px;
        width: 48px;
        height: 24px;
    }
</style>
```

background-clip

定义背景图片的可见区域（相当于在纸片上开孔去看背景图片），语法为：

```
background-clip: border-box | padding-box | content-box | text [, border-box |
padding-box | content-box | text, … ]
```

解释见图 5.65。

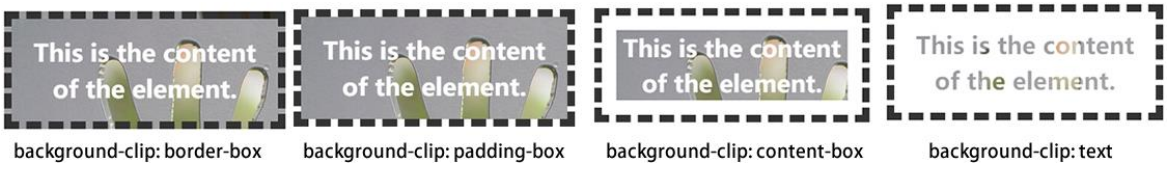


图 5.65 background-clip 属性使用说明

上面第四张图的字体要设成透明才能看到效果，即：

```
    color: transparent;
    background-clip: text;
    -webkit-background-clip: text;  /*针对 chrome 浏览器*/
```

background-clip 属性的多个值按照相同的次序应用于 background-image 定义的各个图片上。

实操练习

把你的名字显示为色彩斑斓的样子，类似图 5.66。



图 5.66 色彩斑斓的文本

background-attachment

定义背景图片与元素粘粘，其使用语法如下：

```
background-attachment: scroll | fixed | local [,scroll | fixed | local, … ]
```

scroll 表示背景图片与元素固定，不会随目标元素的滚动条滚动，但会随着父元素的滚动条滚动，fixed 与目标元素固定，同时不会随任何滚动条滚动，local 与目标元素的内容固定。演示 demo 参考：https://developer.mozilla.org/zh-CN/docs/Web/CSS/background-attachment

background-attachment 属性的多个值按照相同的次序应用于 background-image 定义的各个图片上。

backgound-color

定义背景色，使用语法如下：

```
background-color: <color> | transparent
```

transparent 为透明色，<color>定义具体颜色，如：#D40FBD、red、rgb(100, 150,50)、rgba(56, 200,120,0.8)等。本属性只能取单值。

background-size

定义背景图片大小，语法为：

```
background-size: [ <length-percentage> | auto ]{1,2} | cover | contain[, … ]
```

其中 cover 和 contain 都是按同比率缩放图片，cover 会让背景图片全部覆盖目标元素（图片可能被截掉部分），而 contain 让图片整个显示（元素多余部分用 background-color 定义的颜色显示），注意：如果采用百分数，表示 background-origin 的占比。例如:

```
background-size:100px, 200px 150%;
```

表示第一幅图片的宽度 100px，高度默认为 auto（同比例计算得到），第二幅图片宽度 200px，高度为 background-origin 所定义区域的 150%。下面的例子：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
background-size: cover;
```

表示背景图片同比例缩放使得图片的宽和高大于等于目标元素的宽和高，超出部分被截掉（当元素和图片的长宽比例不同时）。

background-size 属性的多个值按照相同的次序应用于 background-image 定义的各个图片上。

background-repeat

定义图片重复方式，语法为：

```
background-repeat: repeat-x | repeat-y | [ repeat | space | round | no-repeat ]{1,2}[, … ]
```

定义横向和纵向的平铺方式，如果是单值，则对应表 5.12 中的双值，双值当中的第一个值定义横向屏幕模式，第二个值定义纵向平铺模式。

表 5.12 单值/双值对应关系

单值	等价于双值
repeat-x	repeat no-repeat
repeat-y	no-repeat repeat
repeat	repeat repeat
space	space space
round	round round
no-repeat	no-repeat no-repeat

其中 repeat-x 表示水平方向重复垂直方向不重复，repeat-y 表示垂直方向重复水平方向不重复，repeat 两个方向都重复，no-repeat 都不重复，space 平铺尽可能多的图片，最后多余的空间平均分配到各个间隙。round 平铺尽量多的图片后的剩余空间：（1）如果大于图片的一半，则再插入一个图片，然后全部平铺的图片缩小；（2）如果小于图片的一半，则放大全部已经平铺的图片以填满剩余空间。见图 5.67 所示。



图 5.67 background-repeat 属性用法举例

例如:

```
width: 480px;
height: 210px;
margin: 100px;
background-color: blueviolet;
background-image:url("earth.jpg");
background-repeat: space round;
```

浏览效果参见图 5.68。



图 5.68 横向 space 纵向 round

即水平风向上的剩余空间平均分配到各个间隙（space），垂直方向上拉伸图片以填满剩余空间（round）。

background-repeat 属性的多个值按照相同的次序应用于 background-image 定义的各个图片上。

综合例子：把下面的图 5.69 和图 5.70 合并在一起。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 5.69 小鸟



图 5.70 山水画

合并后的效果图见图 5.71。



图 5.71 两张背景图

代码如下：

```
div {
    width: 900px;
    height: 600px;
    background-image: url("bird.png"), url("back.jpg");
    background-repeat: no-repeat;
    background-position: top 70px left 300px, center;
    background-size: 13%, auto;
}
……
<div></div>
```

参照 background 属性的语法，我们也可以采用这个复合属性达到相同的效果：

```
background: url("bird.png")  top 70px left 300px /13%  no-repeat, url("back.jpg")
center /auto  no-repeat;
```

2、background-blend-mode — 设置混合模式

定义背景层中的图片如何混合在一起，它的取值和介绍请参考“§4.5.4.3 绘画上下文”小节中的“表 4.26 <feBlend>标签的 mode 属性的取值”。background-blend-mode 是一个多值属性，值之间用逗号分隔，按照同样的次序应用于 background-image 属性定义的各层背景。

当处理第 *i* 背景层时只混合第 1、2、…、i 背景层，而第 i+1、i+2、…层不参与。第 1 背景层就是元素的背景色（默认是全透明）。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

实操练习

参考上面的例子合并成如图 5.72 所示的效果（提醒：采用 background-size 指定大小，background-position 定义位置）。



图 5.72 多张背景图

3、opacity — 设置不透明度

元素的不透明度影响被元素覆盖区域的可见性，越不透明就越看不清背后的内容，此属性的用法如下：

```
opacity: <number> | <percentage>
```

取值范围为[0,1]或者[0%,100%]，0 为全透明，1 为不透明，0.5 半透明。由 opacity 属性定义的不透明度对元素整体施加影响，包括元素的子元素，另外子元素不能继承父元素的 opacity。所以设置在子元素上的 opacity 属性的值如果大于父元素的 opacity 的值，是没有效果的，起作用的是父元素的不透明度。

如果只想影响本元素的不透明度，就用 background-color 属性，如 background-color: #FFF8。

```
div {
    width: 900px;
    height: 600px;
    background-image: url("bird.png"),
}
span{
    opacity: 0.5;
```

```
    writing-mode: vertical-rl;
    font: 40px 华文行楷,serif;
}
……
<div><span>窗前明月光，疑是地上霜。<img src="moon.png" /></span></div>
```

请尝试给<img>施加不同值的 opacity 属性，看看效果如何变化。

5.7.3 元素轮廓

轮廓不同于边框，轮廓绘于边框周围，且不占页面空间，你可以想象为轮廓漂浮在页面之上，目的是突出页面上的某个元素（比如获得输入焦点的元素），从而引起用户的注意力。默认情况下，除<input>外的元素都没有轮廓，看这个例子：

```
<label>输入姓名 <input style="border:none;border-bottom-style: solid;"></label>
```

刚开始输入框没有轮廓，见图 5.73 左一所示，当它获得输入焦点时显示轮廓，见图 5.73 右一。



图 5.73 轮廓示意图

“CSS3 样式速查表 • 元素外观 • 轮廓”中列出了设置轮廓特性的全部样式。

outline 是复合属性，它对应三个分属性：outline-color、outline-style 和 outline-width，它是多值样式，分别指定颜色、线型和线宽，且这些值的先后次序没有严格规定。outline-style 属性取值 auto 时，线型是浏览器默认的，一般是黑色细实线，此时 outline-color 和 outline-width 失效。outline-style 取值 none 时，没有轮廓。

outline-offset 属性定义轮廓线于边框之间的距离，它的使用语法如下：

```
outline-offset: <length>
```

<length>可为正数、0 和负数，正数表示在边框外面，0 就于变宽重合，负数在边框里面。

当鼠标指针移到元素上方时，由 cursor 属性定义鼠标指针的形状，此属性可取的值参见表，其中值<url>指定图片的 URL，用这个图片做鼠标指针。其他值对应的指针参见图 5.74。



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

图 5.74 cursor 属性各个值对应的图标

caret-color 属性定义光标的颜色，默认是黑色，它的取值同 border-color 属性。给本小节开头的那个例子中的<input>元素添加下面的样式，再看看效果如何？

```
…solid; outline:none; cursor:crosshair; caret-color:red;">
```

5.8 列表

“CSS3 样式速查表 • 列表”列出了专门用于列表元素的样式，列表元素是这样的元素：应用了 display: list-item 样式的元素、<ol>和<ul>。

list-style 是一个复合的多值属性，允许一个值、两个值和三个值，而且可以以任何次序出现，但如果同时给出了<'list-style-type'>和<'list-style-image'>，那么前者无效。list-type 属性对应的分属性为 list-style-type、list-style-position 和 list-style-image。

list-style-type 属性采用字符做列表项目的标记，标记的颜色可以采用::marker 定义。它的使用语法如下：

```
list-style-type: none | <string> | <custom-ident> | symbols()
```

none 没有标记，<string>用字符串做项目标记，如"★"。<custom-ident>自定义标识符，必须等于@counter-style 的值或者预定义标识符中的一个，比较常用的预定义标识符有：disc（实心原点：•，这是默认值）、circle（空心原点：○）、square（实心正方形：■）、decimal（从 1.开始的数字：1,2,3,…）、cjk-decimal（中日韩语言的数字：一,二,三,…）、decimal-leading-zero（以 0 作前导的数字：01,02,03,…）、lower-roman（小写罗马数字：i, ii, iii, iv, v,…）、upper-roman（大写让罗马数字：I, II, III, IV, V,…）、lower-greek（小写希腊字母：α, β, γ,…）、lower-alpha 或 lower-latin（小写字母：a, b, c, …, z）、upper-alpha 或 upper-latin（大写字母：A, B, C, …, Z）、cjk-heavenly-stem（中日韩的天干计数：甲,乙,丙,……,癸）、cjk-earthly-branch（中日韩的地支计数：子,丑,寅,…,亥）、simp-chinese-formal（简体中文的大写数字：壹,贰,叁,…,拾）等，例如 list-style-type: square。

symbols()目前只有 firefox 浏览器支持，具体介绍请参考“§5.16 CSS 函数”小节，看例子：

```
<ol style="list-style: symbols(cyclic '*' '†' '‡');">
  <li>这个时代太二——</li>
  <li>小时候说话词不达意</li>
  <li>长大后说话言不由衷</li>
  <li>年轻时活动身不由已</li>
  <li>年老后活动力不从心</li>
</ol>
```

本例子循环使用'*'、'†'、'‡'作标记。把 symbols(cyclic '*' '†' '‡')改为 decimal，看看效果有何变化？

list-style-image 属性采用图形作项目标记，默认值为 none，<url>为图片文件的 URL，图片不要太大，最好采用图标。例如：

```
list-style-image: url("/images/square.jpg");
```

list-style-position 属性定义标记相对于项目内容的位置，默认是 outside，即在项目内容的外面，另一个值 inside 表明标记在项目内容里面，即作为项目内容的第一个字符。参见如图 5.75 所示。

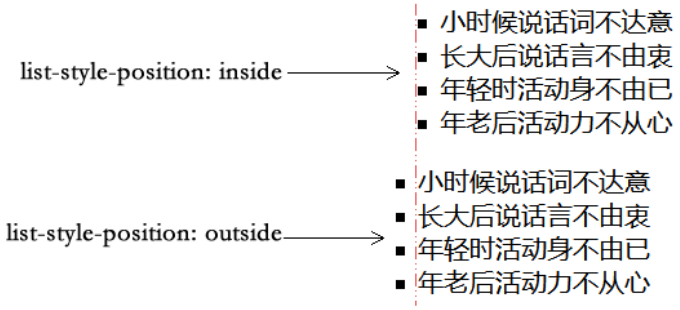


图 5.75 list-style-position 属性用法说明

5.9 表格

“CSS3 样式速查表 •表格”中的样式用于表格元素<table>、<thead>、<tbody>、<tr>、<td>等。

table-layout 属性应用于<table>元素，告诉浏览器如何安排表的行、列和单元格，默认值 auto 表示由浏览器自行决定，目前主流浏览器都是根据单元格内容来调整单元格的宽度。如果 table-layout 取值 fixed，那么表/列宽度由各自的 width 属性确定或者由首行中的单元格确定各自所在列的列宽。为了防止溢出内容“侵占”相邻单元格，一般都会施加 overflow 属性来隐藏或剪除溢出的内容（即取值 hidden 或 clip）。看例子：

```
table {
     table-layout: fixed;
     width: 260px;
     border: 1px solid;
}

td {
     border: 1px solid blue;
     overflow: hidden;
     text-overflow: ellipsis;
     white-space: nowrap;
}
……
<table>
    <caption>影响人类历史进程的 20 名人</caption>
    <tr><td>默罕默德</td><td>牛顿</td><td>耶稣</td><td>孔子</td></tr>
    <tr><td>圣保罗</td><td>蔡伦</td><td>释迦牟尼</td><td>古腾堡</td></tr>
    <tr><td>哥伦布</td><td>爱因斯坦</td><td>马克思</td><td>巴斯德</td></tr>
</table>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

浏览的效果见图右侧，把代码中的 fixed 改为 auto，这时的效果见图 5.76 左侧。

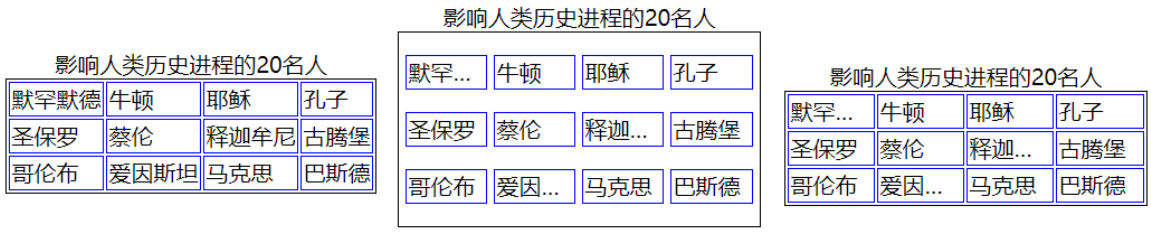


图 5.76 table-layout 属性用法举例

border-collapse 属性适应于<table>元素，取值 collapse 表示相邻单元格共享边线，取值 separate 表示独占边线且<table>元素没有边框，默认值为 separate。

当 border-collapse 取值 separate 时，由 border-spacing 属性定义相邻边线之间的距离，它可取 1 个值或 2 个值，1 个值表示横向和纵向距离一样，取 2 个值时，第一个值代表一行之内的单元格左右两侧边线间的距离，第二个值代表一列之内的单元格上下边线间的距离。在上面的例子中添加下面的一条样式代码，浏览效果见图中间部分。

```
table {
     border-spacing: 5px 1em;
```

caption-side 属性定义表题的位置，它的语法如下：

```
caption-side: top | bottom | block-start | block-end | inline-start | inline-end
```

这些值分表表示在表的上方、下方、行进方向的起点、行进方向的终点、书写方向的起点和书写方向的终点。行进方向由 writing-mode 属性定义，书写方向由 direction 属性定义，具体参考“§5.4.2 文本”小节。默认情况下，block-start 就是 top，block-end 就是 bottom。把样式“caption-side: block-start;”加到上面的例子中，看看效果如何？

5.10 网页布局

针对网页的布局问题，html 先后发展了多种方法，从最早的 table 布局，到后来的 float 布局、position 定位、inline-block，再到最近很火的 flex 布局（弹性布局）和 grid 布局（网格布局）。grid 结合 flex 布局是最流行的手法。“CSS3 样式速查表 • 网页布局”分类中列举了多列、弹性和网格布局中用到的全部样式属性。

5.10.1 多列布局

如果版面比较大（比如报纸），人们倾向于采用多列布局，内容从第 1 列、第 2 列、……依次填充。“CSS3 样式速查表 • 网页布局 • 多列布局”中的样式专门用于多列布局，这些样式只适用于块元素。

columns 属性是 column-width 和 column-count 属性的复合属性，指名理想的列宽和列数，可以是一个值也可以是两个值，而且可以是任何次序。如果只一个值且带长度单位，则是定义列宽，列数自动计算；如果是一个纯数字，则定义列数，而列宽自动计算。

column-width 属性定义理想的列宽，最终浏览器会这样处理：创建尽可能多的列数且列宽不小于 column-width 属性指定的值，但如果元素的总宽度小于 column-width 的值，那只有一列，且列宽就是元素的总宽度。例如：元素宽度 width 等于 300px，而 column-width 等于 120px，则总共 2 列，列宽 150px。

column-count 属性定义理想的列数，如果 column-width 属性不是 auto，那么 column-count 就是定义最大的列数。

看例子：

```
<div style="column-count: 3;">
看看大学里那些拥有“知识”的人，看看华尔街那些拥有“财富”的人，看看鬼谷那些拥有“技术”的人，看看看DC 那些拥有“权力”的人，再看看那些掌握主流媒体拥有“话语权”的人，好莱坞、NBA、还有那个鞋匠，他们的所作所为，已经表明今日的美国已经驶上通往地狱的单行道了，过些年月，就可以在地狱的入口处看到它掠过的身影。</div>
```

调整浏览器视口的宽度，看到如图 5.77 所示的效果。

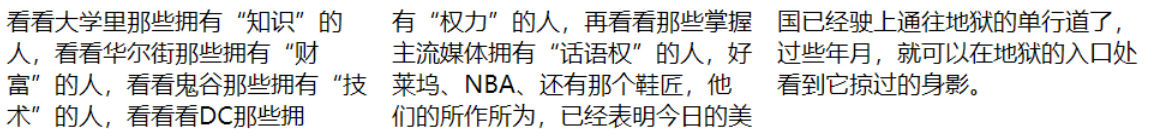


图 5.77 column-count 属性用法举例

column-gap 属性定义列与列之间的间距，如果取值 normal，则由浏览器确定一个默认值，一般为 1em，还可以为非负长度（如 10px）值，或者元素宽度的百分数。在上面的例子中增加样式 column-gap: 5%，浏览时不断调整视口宽度，看看效果如何变化？

column-rule 是复合属性，对应的几个分属性为 column-rule-width、column-rule-style 和 column-rule-color，这几个属性用来列边线。用法与 border-right 一样，请参考“§5.7.1.1 边框线”小节。举例：在上面的例子中增加如下样式：

```
column-rule: 5px solid rgba(255,0,0,0.6);
```

column-span 属性应用在子元素上，表明本子元素横跨多少列，而父元素施加了 columns 样式。默认值 none，即不跨域多列，另一个值为 all，表示横跨所有列。举例：对于上面的例子，我们在 元素中添加一个子元素：

```
<h2 style="column-span:all; text-align:center;">对 2020 美国大选有感</h2>
```

column-fill 属性定义如何平衡各列中包含的内容。默认值 balance 表明内容在各列之间平均分配，在片段化的排版中，只有最后一个片段是各列平衡的，例如在分页媒体中，只有最后一页是平衡的。如果 column-fill 取值 balance-all，那么内容在各列之间平均分配，片段化排版也不例外。此属性的 auto 值表明先排满第一列、然后第二列、…，直到把内容排完为止，因此有可能一列没排满，若干列全空白。举例：在前面的代码中给 元素增加下面的样式，看看效果如何？

```
height:200px; column-fill:auto;
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

orphans 和 widows 两个属性的理解参考图 5.78，有三列和三段文本内容，第 2 段（图中字体加粗部分）跨了三列，留在第 1 列中的部分内容称为 orphans，分配在第 3 列中的部分内容称为 widows。orphans 属性就是定义一段内容留在首次分配的列中的最少行数，默认是 2，假如本例中，列布局容器的高度减少差不多 1 行文字的高度，这样第 2 段留在第 1 列中的文字只能是 1 行，但这违背了不能少于 2 行的规定，因此浏览器干脆从第 2 列开始分配第 2 段了，第 1 列中宁愿浪费空间。widows 属性的理解类似，默认值也是 2。

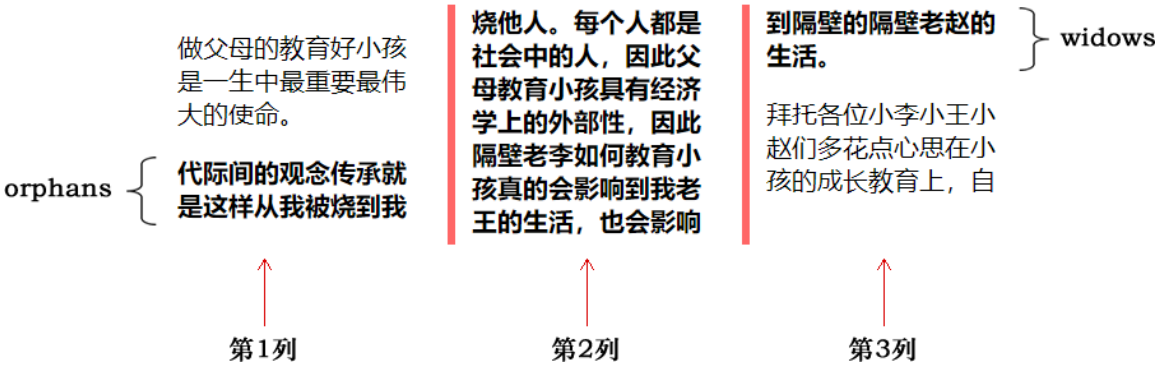


图 5.78orphans 和 widows 属性的说明

5.10.2 弹性布局

Flex 是 Flexible Box 的缩写，意为"弹性布局"，用来为盒状模型提供最大的灵活性。任何一个容器都可以指定为 Flex 布局，即给元素附加样式 display: flex 就开启了弹性布局。代表未来布局趋势的是 grid 布局，我们在下小节详细介绍它。

采用 Flex 布局的元素，称为 Flex 容器（flex container），简称"容器"，它的所有子元素自动成为容器成员，称为 Flex 项目（flex item），简称"项目"，对孙子辈元素不起作用。

容器默认存在两根轴：水平的主轴（main axis）和垂直的交叉轴（cross axis）。主轴的开始位置（与边框的交叉点）叫做主轴起点（main start），结束位置叫做主轴终点(main end)；交叉轴的开始位置叫做交叉轴起点（cross start），结束位置叫做交叉轴终端（cross end）。参考图 5.79。

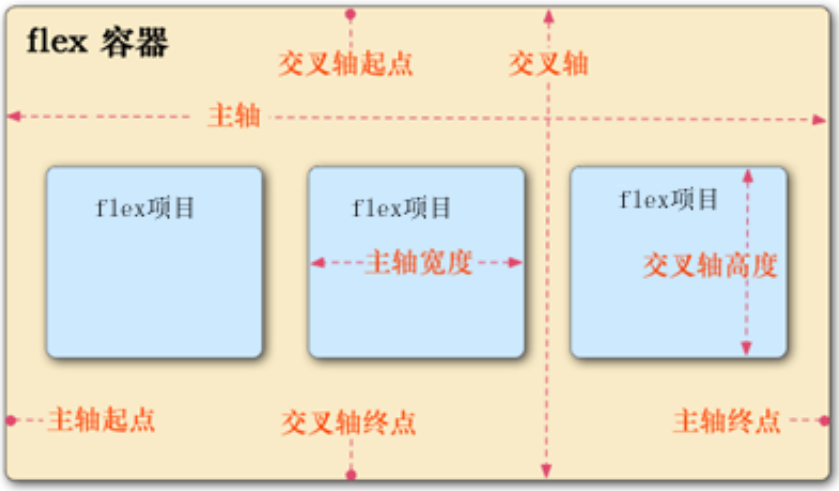


图 5.79 flex 布局说明

项目默认沿主轴排列。单个项目占据的主轴空间叫做主轴宽度（main size），占据的交叉轴空间叫做交叉轴高度（cross size）。

5.10.2.1 应用在容器上的样式

有以下六个属性只在 display:flex 的情况下才有效：

flex-direction、flex-wrap、flex-flow、justify-content、align-items、align-content

注：后续的例子都是以下面的网页代码为基础。

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>弹性布局</title>
    <style>
        div{
            border: 1px solid grey;
            padding: 2px;
        }
        #container{
            width: 400px;
            height: 130px;
            display: flex;
        }
        .item{
            background-color:#DDD;
        }
    </style>
</head>

<body>
    <div id="container">
        <div class="item">11</div>
        <div class="item">2222</div>
        <div class="item">333333</div>
    </div>
</body>

</html>
```

浏览的效果见图 5.80。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 5.80 flex 布局用法举例

flex-direction 属性决定主轴的方向，最终影响项目的排列方向，它有 4 个取值：flex-direction: row | row-reverse | column | column-reverse，各个值的含义见图 5.81。

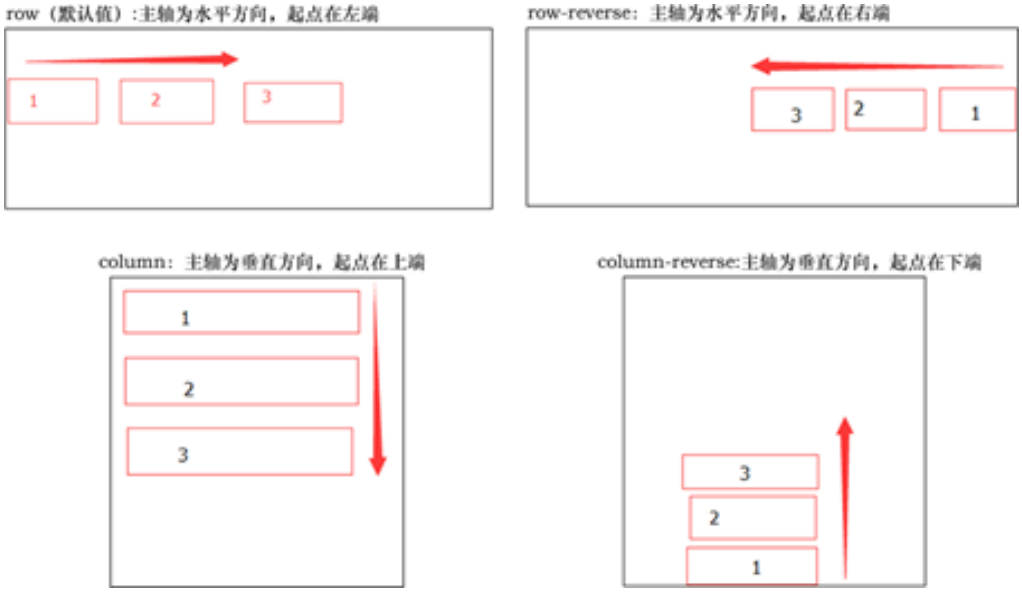


图 5.81 flex-direction 属性说明

实操练习 1：产生如图 5.82 所示的效果。给 flex-direction 赋不同的值，观察排列情况。

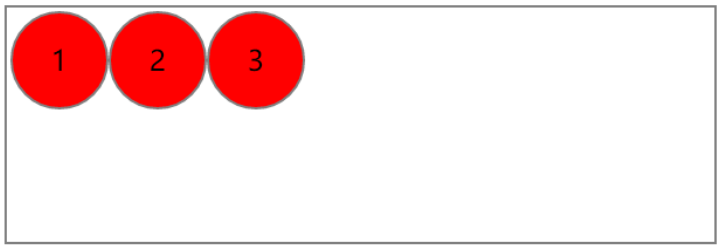


图 5.82 flex-direction 属性实验

flex-wrap 定义项目是否折行（当一行排不下时），有三个值：flex-wrap: nowrap | wrap | wrap-reverse，各个值的含义参见图 5.83。



图 5.83 flex-wrap 属性说明

其中 nowrap 是默认值。

flex-flow 属性是 flex-direction 属性和 flex-wrap 属性的简写形式，默认值为 row nowrap。

justify-content 属性定义了项目在主轴上的对齐方式。可能取值有：justify-content: flex-start | flex-end | center | space-between | space-around，含义参见图 5.84。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

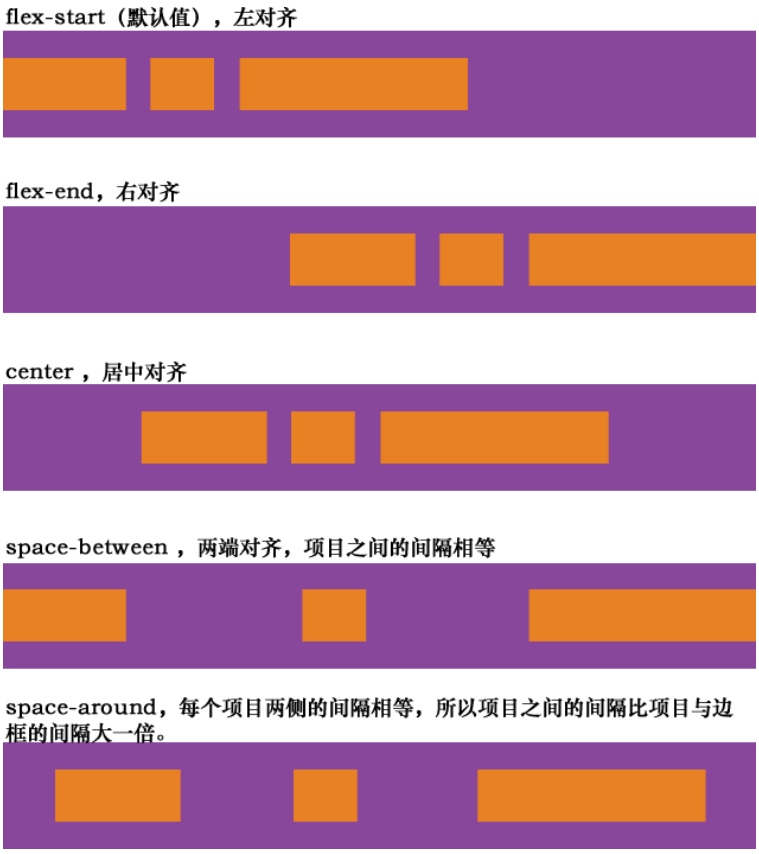


图 5.84 justify-content 属性说明

align-items 属性定义项目在交叉轴上的对齐方式，可能的取值有：align-items: flex-start | flex-end | center | baseline | stretch，具体含义参见图 5.85。

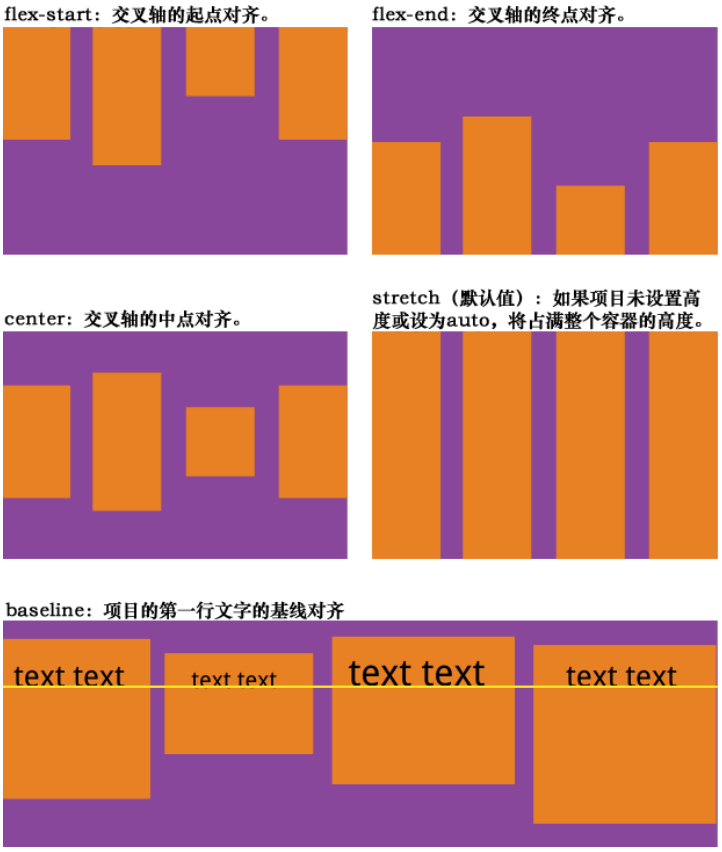


图 5.85 align-items 属性说明

align-content 属性定义了多根主轴线的对齐方式（当 flex-wrap:wrap 且发生折行时就产生多根主轴）。如果项目只有一根主轴线，该属性不起作用（这时要采用 align-items 属性了）。其六个属性值是：align-content: flex-start | flex-end | center | space-between | space-around | stretch，具体含义参见图 5.86。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

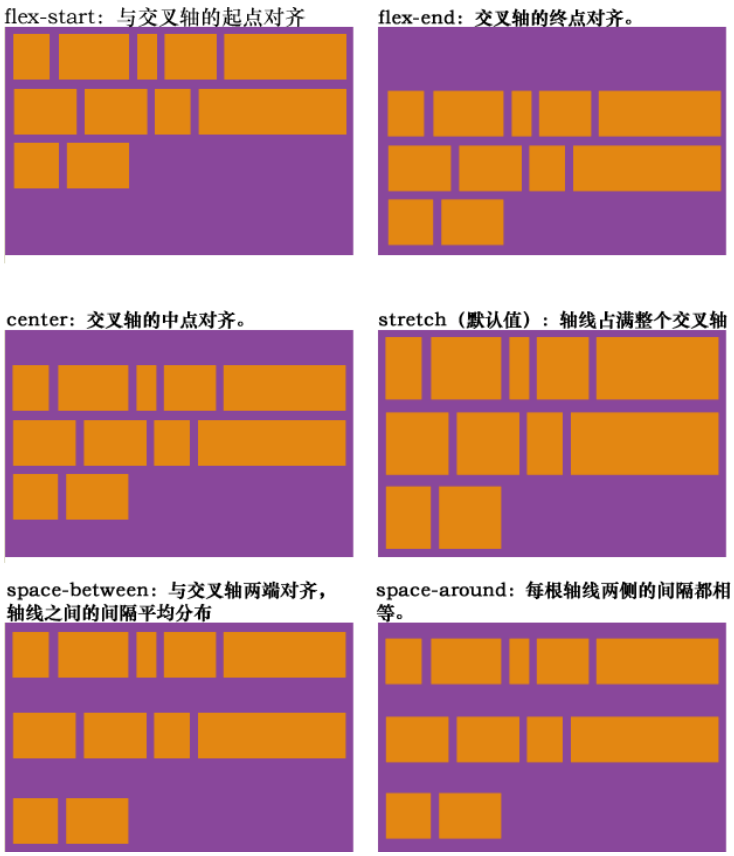


图 5.86 align-content 属性说明

align-items 与 align-content 的区别是前者适应于单行的 flex 容器，后者只适用于多行的 flex 容器，每行的行高就是交叉轴的高度。

5.10.2.2 应用在项目上的样式

有以下六个属性：order、flex-grow、flex-shrink、flex-basis、flex、align-self，只有在父元素添加了 display:flex 时才有效。

order 属性定义项目的排列顺序，格式为 order:<整数>，可为负数，如 order: 5。数值越小，排列越靠前，默认为 0。具有相同 order 值的项目就安装文档流次序。

flex-grow 属性定义项目的放大比例，即如果存在剩余空间，本项目在这剩余空间中占了多大的比率用于扩充项目宽度，默认为 0，即不放大。格式为 flex-grow:<非负数>，不能负数，可为小数。如果所有项目的 flex-grow 属性都为 1，则它们将等分剩余空间（如果有的话）。如果一个项目的 flex-grow 属性为 2，其他项目都为 1，则前者占据的剩余空间将比其他项多一倍，见图 5.87 和图 5.88。

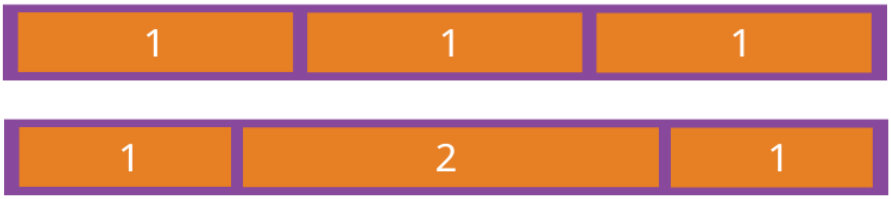


图 5.87 flex-grow 属性说明之一

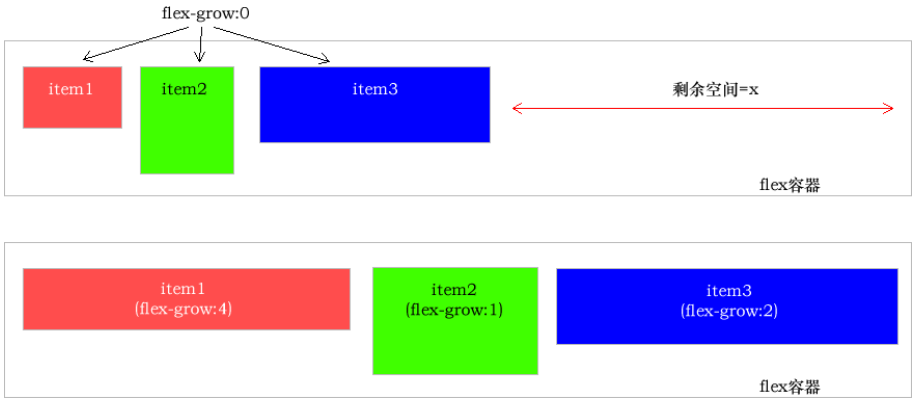


图 5.88 flex-grow 属性说明之二

即 item1 的宽度在原来的基础上增加 x*4/(4+1+2)=x*4/7，4 就是 item1 的 flex-grow 属性的值，类似地，item2 增加 x*1/7，item3 增加 x*2/7。

flex-shrink 属性定义了项目的缩小比例，即如果 flex 容器的空间不足（假如全部项目的宽度加起来减去容器的宽度等于 x），该项目缩小的宽度占 x 的比率。语法是 flex-shrink:<非负数>，可为小数，默认为 1。还是图 5.89 来说明更直观。

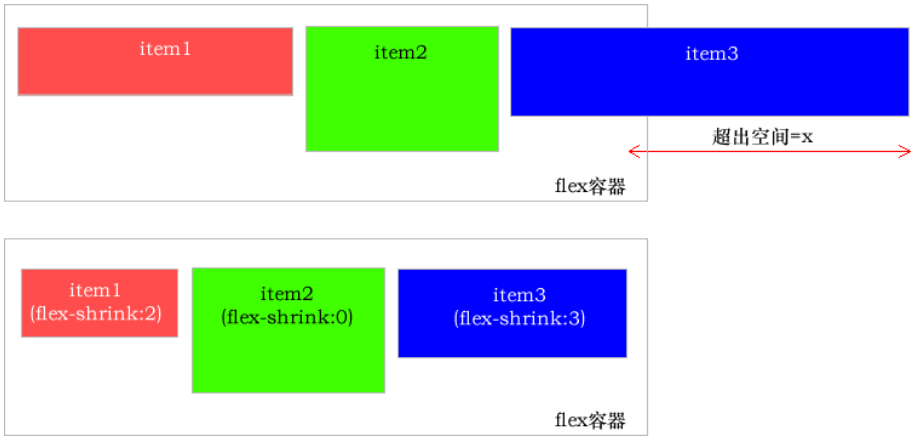


图 5.89 flex-shrink 属性说明

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

总共三个项目 flex-shrink 属性的值加起来就是 2+0+3=5，三个项目宽度加起来减去容器的宽度等于 x，所以项目 1 最终要缩小的量等于 x*2/5，项目 2 不缩小（x*0/5=0），项目 3 缩小量为 x*3/5。

flex-basis 属性定义了在分配多余空间之前，项目占据的主轴空间（main size）。浏览器根据这个属性，计算主轴是否有多余空间。它的默认值为 auto，即项目的本来大小。语法是 flex-basis:<长度>或 auto，比如 flex-basis:300px、 flex-basis:20rem 等。

flex 属性是 flex-grow、flex-shrink 和 flex-basis 的简写，默认值为 0 1 auto，后两个属性可选。语法为：

```
flex: auto| none | [ <'flex-grow'> <'flex-shrink'>? || <'flex-basis'> ]
```

flex:auto 相当于 flex:1 1 auto，flex:none 相当于 flex:0 0 auto。如果用到三个值，一定要记得顺序：flex-grow flex-shrink flex-basis，好记 gsb（哥傻逼），比如 flex: 2 1 300px，表示 flex-grow:2，flex-shrink:1 和 flex-basis:300px。

align-self 属性允许单个项目有与其他项目不一样的对齐方式，可覆盖 align-items 属性。默认值为 auto，表示继承父元素的 align-items 属性，如果没有父元素，则等同于 stretch。语法如下：

```
align-self: auto | flex-start | flex-end | center | baseline | stretch;
```

各个值的含义与 align-items 一样。

最后我们把众多的与 flex 布局有关的属性汇总成表 5.17，便于大家记忆。

表 5.17 flex 布局的样式汇总

适用点	属性名	作用	取值范围	默认值
容器	flex-direction	主轴方向	row，row-reverse，column，column-reverse	row 从左往右
	flex-wrap	是否换行	nowrap, wrap, wrap-reverse	nowrap 排在一行上
	flex-flow	flex-direction 和 flex-wrap 的简写	<flex-direction> || <flex-wrap>	row nowrap
	justify-content	项目在主轴上的对齐方式	flex-start,flex-end,center,space-between,space-around	flex-start 左对齐
	align-items	项目在交叉轴上如何对齐	flex-start, flex-end, center, baseline, stretch	stretch 当项目未设置高度或设为 auto，将占满整个容器的高度
	align-content	多根主轴线的对齐方式	flex-start, flex-end, center, space-between, space-around, stretch	stretch 轴线占满整个交叉轴
项目	order	排列序号	整数	0
	flex-grow	放大比例	非负数	0 不放大
	flex-shrink	缩小比例	非负数	1 缩小
	flex-basis	占据主轴的宽度	长度（如 100px）	auto 项目本来大小
	flex	flex-grow, flex-shrink 和 flex-basis 的简写	auto | none | [<'flex-grow'> <'flex-shrink'>? || <'flex-basis'>]	0 1 auto
	align-self	对齐方式，覆盖 align-items 属性	auto, flex-start, flex-end, center, baseline, stretch	auto 继承父元素的 align-items 属性

下面我们举一个例子来进一步说明 flex 布局的用法。最终效果见图 5.90。

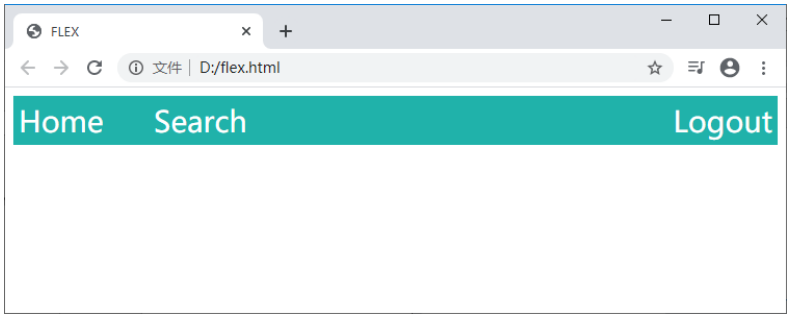


图 5.90 flex 布局综合实例

第 1 步：设计网页结构

```
<!DOCTYPE html>
<html>

<head>
    <title>FLEX</title>
</head>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
<body>
    <header>
        <div>Home</div>
        <div>Search</div>
        <div>Logout</div>
    </header>
</body>

</html>
```

浏览效果见图 5.91。

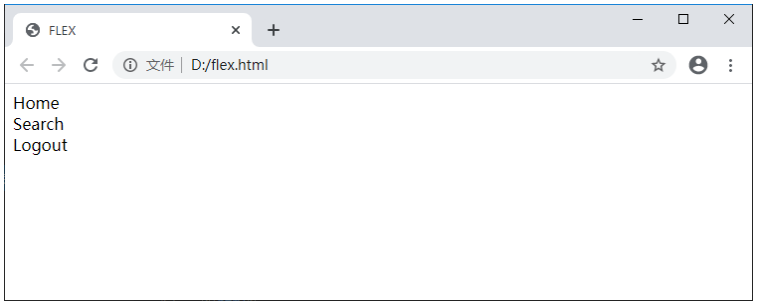


图 5.91 网页结构

第 2 步：添加样式——在<head>元素内添加

```
<style>
      header{
          display: flex;
      }
</style>
```

浏览效果见图 5.92。

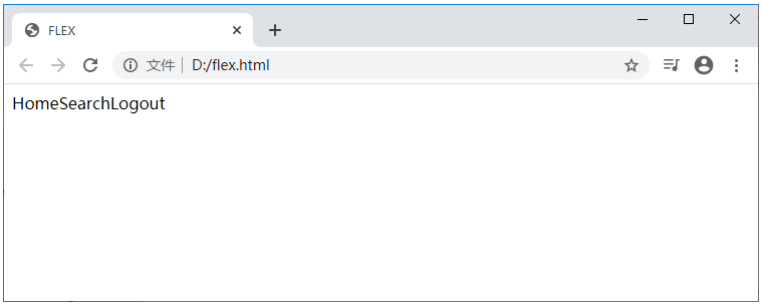


图 5.92 添加样式

注意：这时其他样式属性全部是默认值，参见表 5.1 的最后一列。再添加样式使得 Logout 移到最右边（这是因为默认 margin-right:0px）：

```
header>div:nth-child(3) {
     margin-left: auto;
}
```

浏览效果图 5.93。

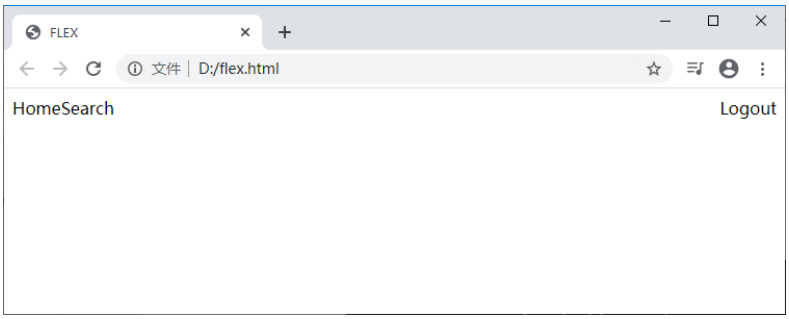


图 5.93 Logout 右移

再给项目添加样式 flex-basis:130px:

```
div{
    flex-basis: 130px;
}
```

浏览效果见图 5.94。

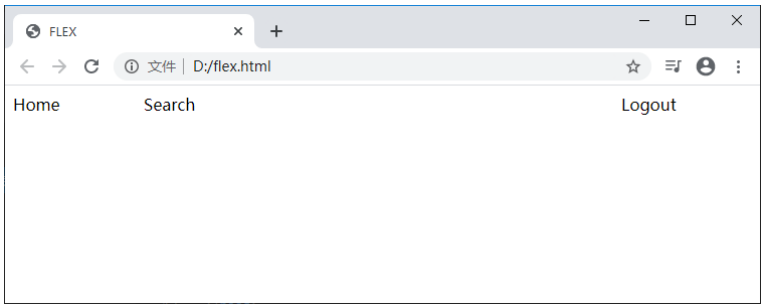


图 5.94 增加间距

最后调整字号，背景色，最终的样式表为：

```
<style>
    header {
        display: flex;
        background-color: lightseagreen;
        padding: 5px;
    }
    div {
        flex-basis: 130px;
        color: white;
        font-size: 28px;
    }
    header>div:nth-child(3) {
        margin-left: auto;
        text-align: right;
    }
</style>
```

本例中，真正与弹性布局有关的样式属性就两个：一个是容器上的 display:flex，另一个是项目上的 flex-basis:130px，由此可知，大部分取默认值就能满足我们的需要。这样就可以看到如图 5.90

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

所示的页面了。打开开发者工具，参照图 5.95 所示的操作，就可以发现 flex 布局是自动适应各种屏幕了。

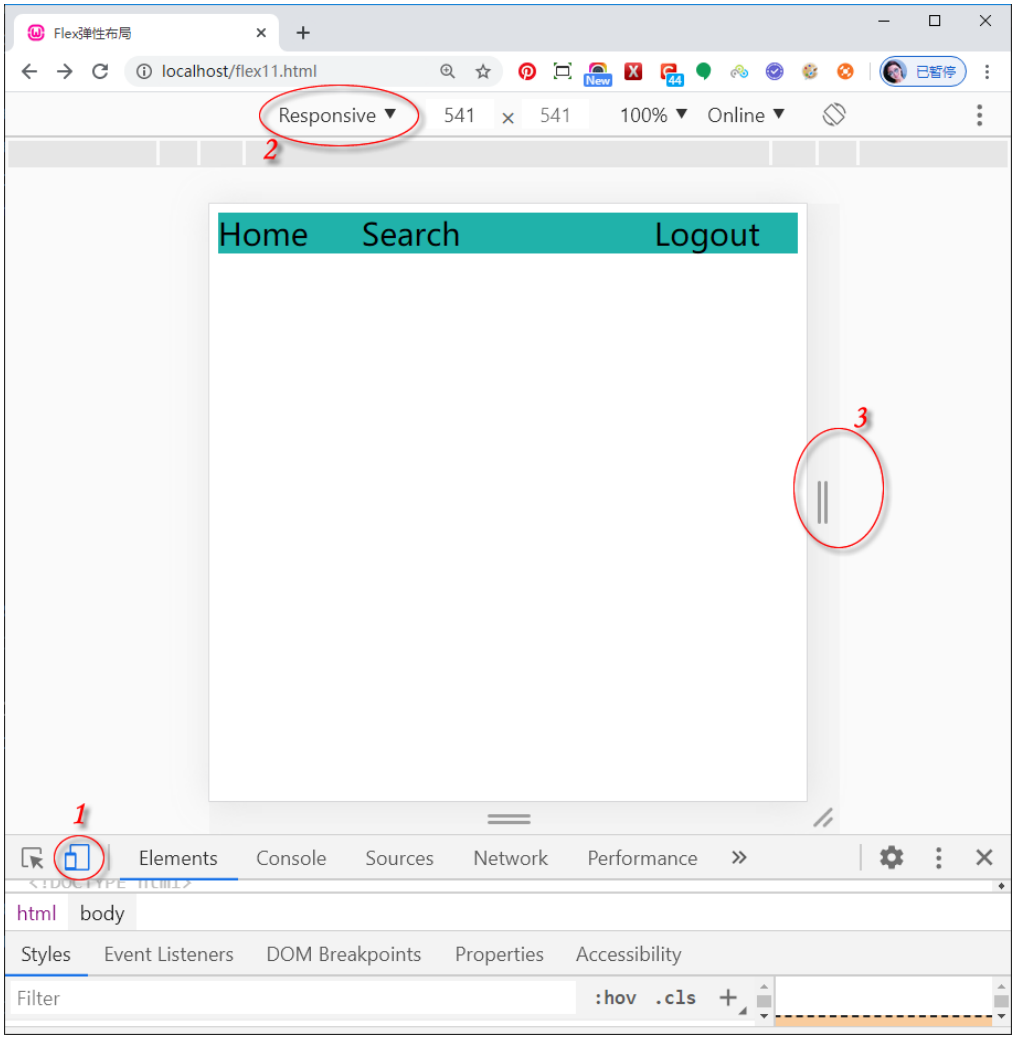


图 5.95 flex 自适应布局

实操练习 2：完成图 5.96 所示的布局（提醒：下图对角线上的三点——对第 2、第 3 个 flex 项目单独设置 align-self 属性，最后一个图——flex 容器里再套三个 flex 容器）。

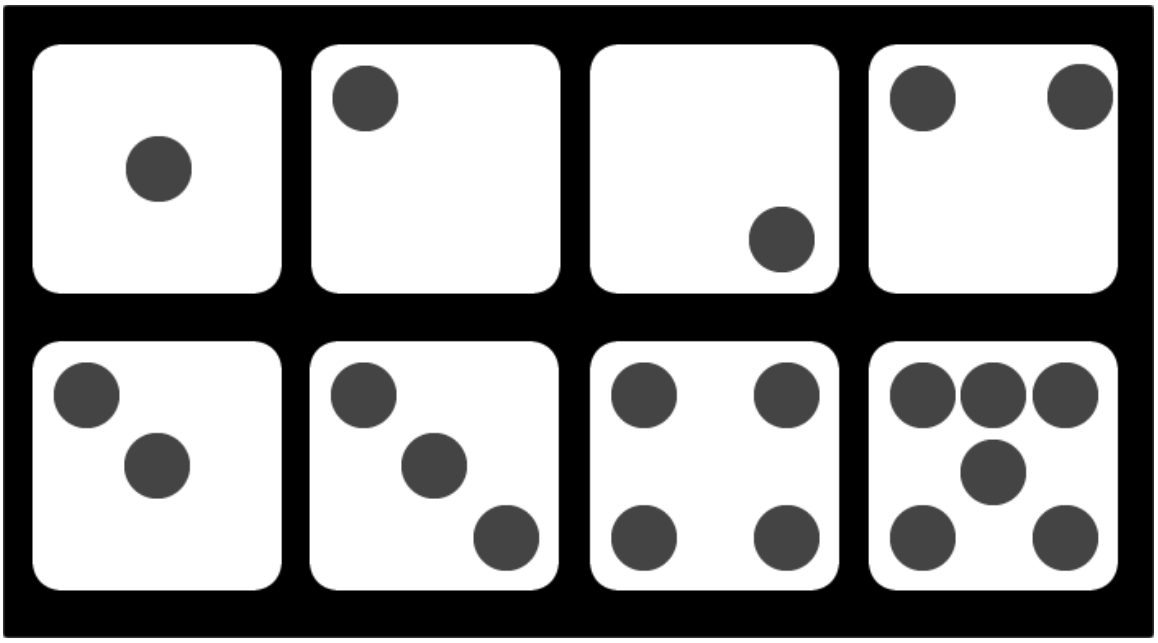


图 5.96 flex 布局实验

5.10.3 网格布局

flex 只是很好的一维布局，最近出现的 grid 算是完美地解决了二维布局。预计在 grid 中套用 flex 将是一个最流行的网页布局方法。

Grid 布局的示意图见下面这张图，首先我们先要搞清楚几个概念。参考图 5.97 所示。

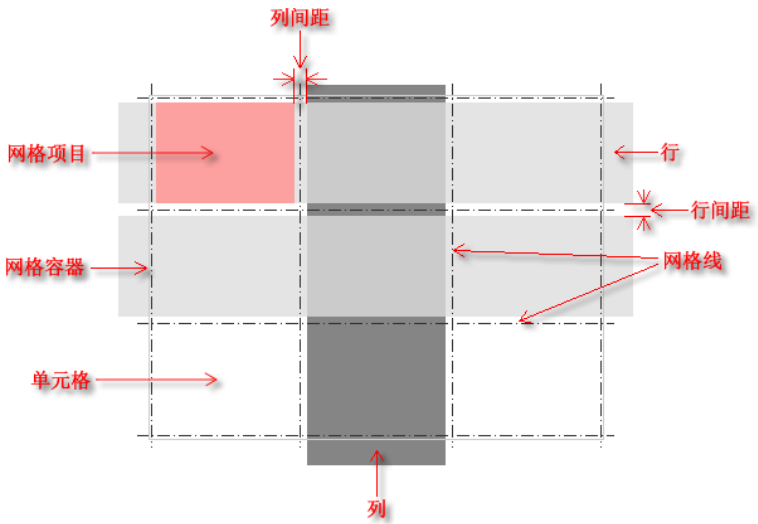


图 5.97 grid 布局的几个概念

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

1、 基本概念

1) 容器和项目

采用网格布局的区域，称为"容器"（container）。容器内部采用网格定位的一级子元素，称为"项目"（item）。注意：孙辈元素不叫项目，Grid 布局只对项目生效。请看如下代码：

```
<div class="container">
     <div class="b">1</div>
     <div class="b">2</div>
     <div class="b"><p>3</p></div>
     <div class="b">4</div>
</div>
```

最外层 元素就是容器，里层的 元素就是项目，而<p>元素不能称为容器的项目，因为它是孙辈了。任何容器元素都可作为 grid 容器使用，比如 、<header>、<section>、<footer>、<nav>、<ul>等等，任何元素都可作为项目。

2) 行、列和单元格

容器里面水平方向划分为若干"行"（row），垂直方向划分为若干"列"（column）。一行与一列的交叉部分称为一个单元格（cell）。正常情况下，n 行和 m 列会产生n × m个单元格，比如，3 行 3 列会产生 9 个单元格。“行”和“列”又统称为“轨道”或“网格轨道”。

3) 网格线

划分网格单元的线，称为"网格线"（grid line）。水平网格线划分行，垂直网格线划分列。正常情况下，n 行就有 n + 1 根水平网格线，m 列就有 m + 1 根垂直网格线，比如三行就有四根水平网格线。网格线允许命名，方便给某个具体的项目指定单元格（比如摆放在垂直网格线 2 和 4 之间）。

4) 网格区域

若干个网格单元组成一个区域，组合成区域的目的是为了便于组织网页内容。

涉及 grid 布局的样式属性有很多，有的设置在容器上，有的设置在项目上，下面一一作介绍。

5.10.3.1 应用在容器上的样式

使用于容器的样式属性有：display、grid-template-columns、grid-template-rows、gap、grid-template-areas、grid-auto-flow、grid-auto-rows、grid-auto-columns、justify-items、justify-content、align-items、align-content。

1) display: grid | inline-grid

grid 定义一个块级容器，inline grid 定义一个内联容器，参见示意图 5.98。

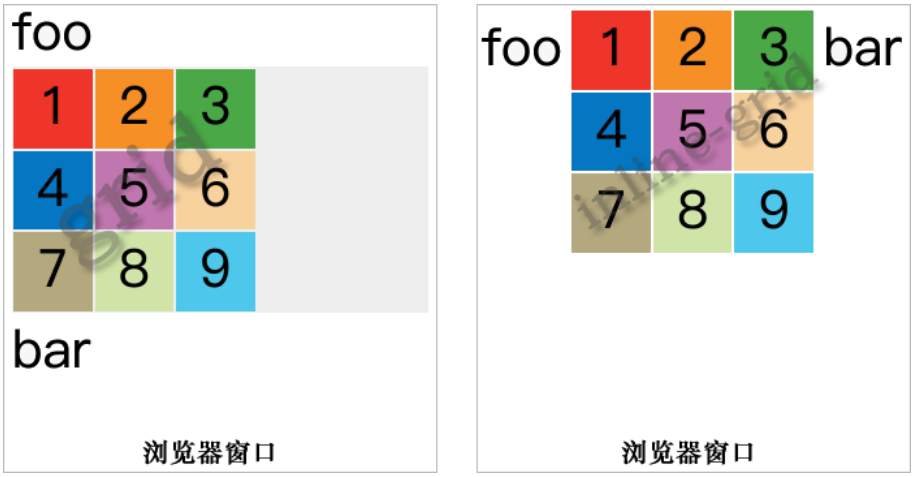


图 5.98 grid 和 inline-grid 布局

注意，容器一旦设为网格布局后，子元素（项目）的 float、display: inline-block、display: table-cell、vertical-align 和 column-*等设置都将失效。

2) grid-template-columns 、grid-template-rows 属性

定义了网格容器之后，接着就要划分行和列。grid-template-columns 属性定义垂直网格线和每一列的列宽，grid-template-rows 属性定义水平网格线和每一行的行高，参考图 5.99。

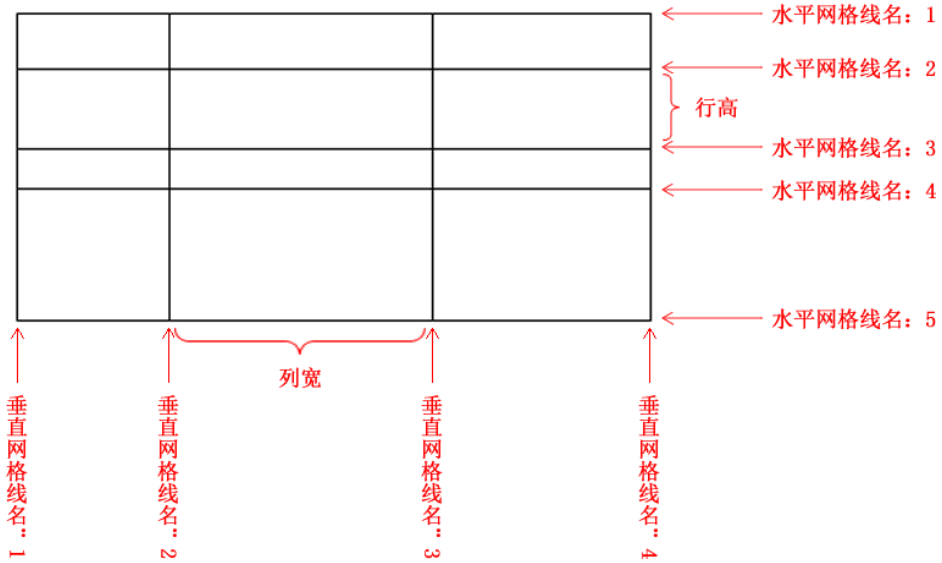


图 5.99 行和列说明

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

水平网格线线名从上到下默认为 1、2、3、4、5、……，垂直网格线线名从左到右默认为 1、2、3、4、5、……，网格线线名可以改变，并且同一条线允许取多个名字。

grid-template-rows 属性用于行数、行高和水平网格线的名称，其语法图 100 所示。

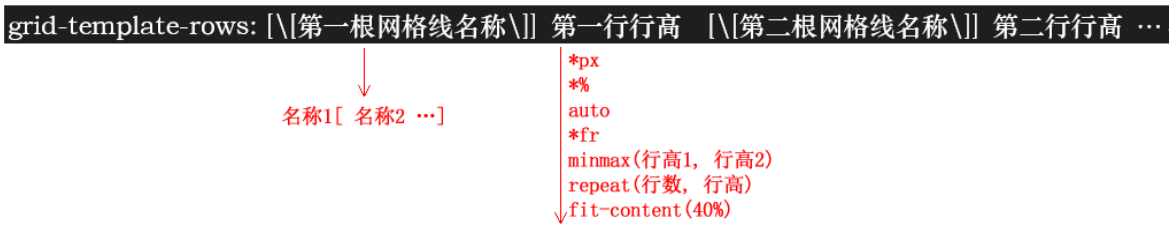


图 5.100 grid-template-rows 属性语法

同一根网格线可以取多个名字，允许采用多种方法来定义一行的行高。先看下面的例子：

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>网格布局</title>
    <style>
        #container {
            display: grid;
            grid-template-rows: 50px 80px;
            grid-template-columns: 300px;
        }
    </style>
</head>
<body>
    <div id="container">
    </div>
</body>
</html>
```

定了两行，行号分别为 50px 和 80px。浏览效果如图 5.101。

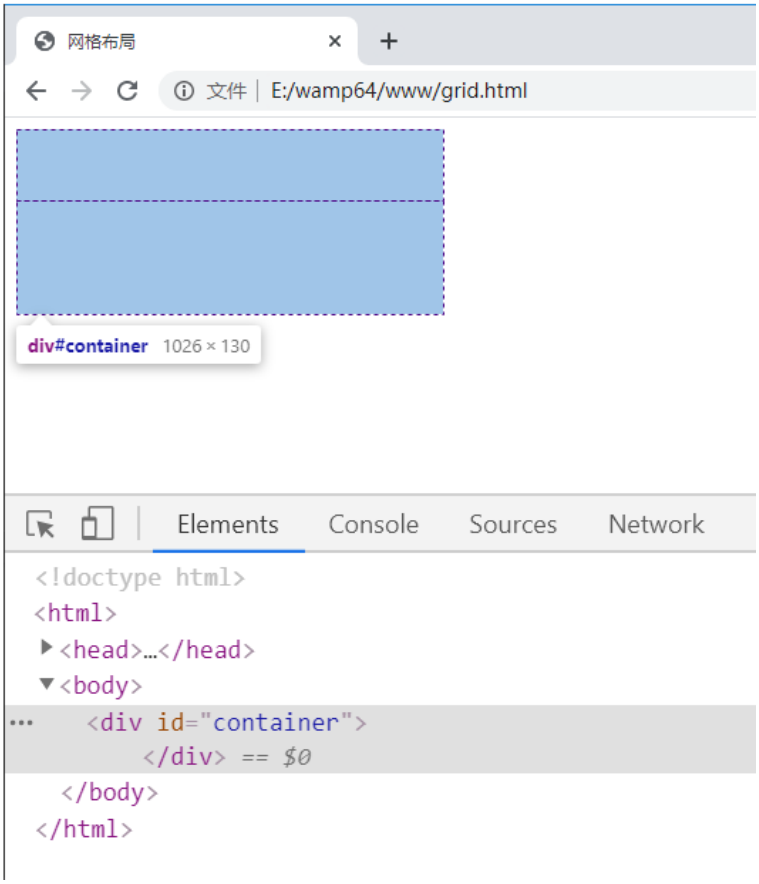


图 5.101 grid 布局举例——定义行/列

打开 chrome 浏览器的开发者工具，进入 Elements 页，鼠标指到网格容器，就可以看到网格样子。

```
    grid-template-rows: 100px 200px 30%;
```

定义三行，行高分别为 100px、200px、容器高度的 30%。注意：如果容器本身没有定义高度，那么就参照 grid-template-rows:100px auto 先计算出容器高度，最后再用此高度乘以 30%得出第二行的高度。关于 auto 请参照后续内容。

```
    grid-template-rows: [line1] 100px;
```

定义一行高为 100px，第一根水平网格线叫 line1，第二根网格线默认为 2。

```
    grid-template-rows: [linename1] 200px [linename2 linename3];
```

第二根网格线有两个名字 linename2 和 linename3。

```
    grid-template-rows: [L1] 1fr [L2] 10rem [L3] 3fr [L4];
```

定义三行，第一行行高等于行内最高的项目的高度，第二行行高为 10rem，第三行行高为第一行的三倍，并且给全部的水平网格线分别命名为 L1、L2、L3、L4。注意：如果容器本身定义了高度 x，那么 1fr=(x-10rem)/(1fr+3fr)。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

实操练习

开启 grid 布局，有 4 行 1 列，容器高度为 300px，第一行行高为 50px，第三行行高是第二行的 2 倍，第四行行高是第二行的 3 倍，给第三根水平网格线取名 thirdLine。

```
    grid-template-rows: minmax(100px, 1fr);
```

只有一行，行高最少 100px，最大等于行内最高的项目的高度。如果容器本身定义了高度，那么行高就是容器的高度。

```
    grid-template-rows: fit-content(40%);
```

具体解释见后续部分。

```
    grid-template-rows: repeat(3, 200px);
```

定义了三行，行高都是 200px。再看一个定义了 7 行的综合的例子：

```
grid-template-rows: [first] 200px 2fr minmax(100px,1fr) fit-content(20%)
[firth] repeat(3,5rem);
```

实操练习

看看下面的代码，浏览你效果是什么？

```
……
height: 100vh;
grid-template-rows: 50px 1fr 50px;
……
<div id="container">
     <div>header</div>
     <div>main</div>
     <div>footer</div>
</div>
……
```

注意：1vh 等于浏览器窗口高度的百分之一。

继续看下面这些例子：

```
grid-template-rows: 200px repeat(auto-fill, 100px) auto;
grid-template-rows: minmax(100px, max-content) repeat(auto-fill, 200px) 20%;
grid-template-rows: [line1] 100px [line2] repeat(auto-fit, [line3 line4] 300px)
 100px;
grid-template-rows: [line1 line2] 100px repeat(auto-fit, [line1] 300px) [line3];
```

要理解上面这几个例子，我们首先要搞清楚这几个概念：auto-fill、auto-fit、auto、max-content、min-content 以及上文提到的 fit-content()函数。

我们以前对一个元素的大小设置一个百分数，那是指占据父元素空间的百分比，例如 width:40% 表示本元素的宽度是父元素宽度的百分之四十。父元素大小改变会导致本元素大小也随着改变。下面要讲解的诸如 max-content 等都是与本元素的内容相关的，而不是与父元素相关。

max-content：是指本轨道上最大的那个项目的尺寸。比如网格的一列中有一个单元放了一张图片（假设宽度为 X），另一个单元是一行介绍图片的文字（假设不换行的宽度为 Y），则 max-content=max(X,Y)。

min-content：先计算本轨道上每一个项目的最小尺寸，那么这些最小尺寸当中最大的那个就等于 min-content。还是拿上面这个图片和描述文字举例，图片的最小宽度还是图片的宽度，不能改变。一行文字的最小宽度是这样计算的：在不截断单词的情况下尽可能多地回车换行。因此很容易得出结论：一行文字的最小宽度就是里面最长的那个单词的宽度，如果全是中文，则等于一个中文的宽度。假设图片描述文字的最小宽度为 Z，那么 min-content=max(X,Z)。参考图 5.102 所示。



图 5.102 取值 min-content 的说明

auto：在定义行高时，如果容器设置了高度 x，那么一个 auto 等于 x 减去其他已经确定高度的行高之和再除以 auto 的个数，比如：

```
    grid-template-rows:100px auto auto;
    height: 360px;
```

一个 auto 等于(360-100)/2=130px，即定义了三行，行高分别为 100px、130px、130px。如果容器没有定义高度，auto 定义的行高就是该轨道内全部项目当中的最高项目的高度。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

fit-content(arg)：这个函数的返回值就是 min(max-content, max(min-content, arg))表达式的值。例如 fit-content(200px)表示：当内容小于 200px 时，以内容为长度，当内容大于 200px 时，以 200px 为长度。

auto-fill 和 auto-fit：只用于 repeat()函数的第一个参数，即到底产生多少高度一样的行。如果容器没有设置高度，那么 auto-fill 和 auto-fit 的 repeat()只产生一行。

如果容器设置了高度 x，那么 repeat(auto-fill,y)产生的行数等于 floor((x-z)/y)，其中 z 就是确定高度的行高之和，函数 floor(x)就是取整，比如 floor(4.6)=4。例如：

```
grid-template-rows: 100px repeat(auto-fill,80px) 50px;
height: 600px;
```

那么最终会产生 7 行，第一行行高 100px，最后一行行高为 50px，中间 5 行行高都是 80px，因为 floor((600px-150px)/80px)=5。而 repeat(auto-fit,y)产生的行数还与项目个数有关，0≦行数≦floor((x-z)/y)，即产生的轨道数加上前面的轨到数刚好能放下全部的项目。看下面这个例子:

```
grid-template-rows: 100px repeat(auto-fit, 80px) 50px;
grid-template-columns: 100px auto;
height: 600px;
```

第一行行高 100px，有两列，假设有 2 个项目，显然第一行就可以放下这两个项目，所以 repeat(auto-fit, 80px)不产生行数；如果有 5 个项目，那么 repeat(auto-fit, 80px)需要产生 2 行，加上第一行才能放下这 5 个项目，尽管存在第四行（行高 50px），第四行上也不放项目；如果有 16 个项目，至少得 8 行才能放下这些项目，这时 repeat(auto-fit, 80px)产生的行数就是 floor((600px-150px)/80px)=5，见图 5.103。

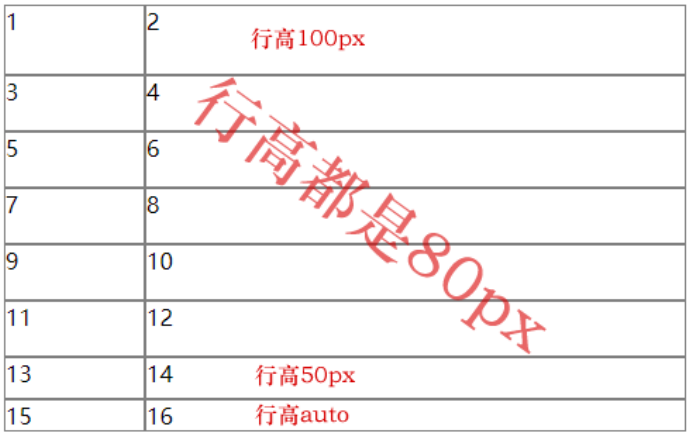


图 5.103 repeat()函数举例

接下来我们再来看看 grid-template-columns 属性，它用来定义垂直网格线和每一列的列宽，用法与 grid-template-rows 差不多，由于浏览器窗口都存在宽度，所以理解起来相对简单。其语法参照图 5.104 和图 5.105。

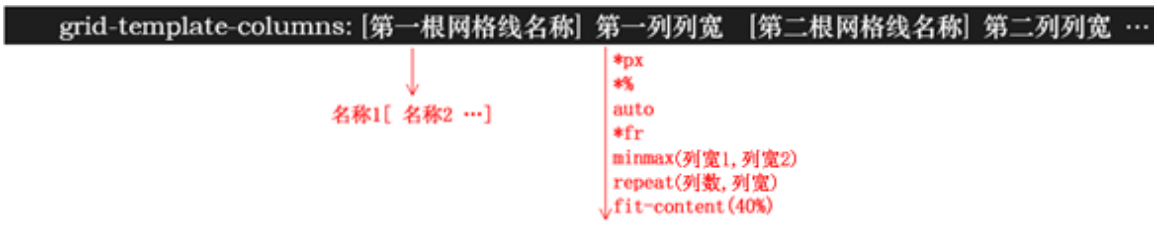


图 5.104 grid-template-columns 语法

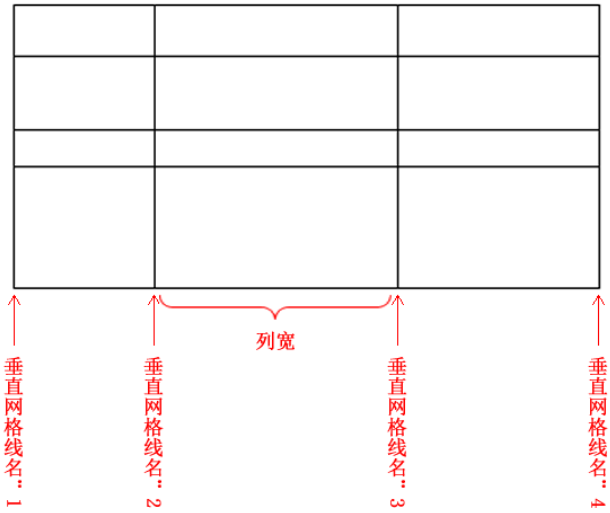


图 5.105 grid-template-columns 属性用法说明

先看几个例子：

```
grid-template-columns: 100px 1fr;
```

定义两列，第一列宽度 100px，第二列充满剩余空间。

```
grid-template-columns: [linename] 100px;
```

第一根垂直网格线叫 linename，只一列，列宽 100px。

```
grid-template-columns: [linename1] 100px [linename2 linename3];
```

定义一列，第一根网格线叫 linename1，第二根网格线叫 linename2 和 linename3。

```
grid-template-columns: minmax(100px, 1fr);
```

定义一列，列宽不能小于 100px。

```
grid-template-columns: fit-content(40%);
```

只一列，列宽等于 min(max-content, max(auto, 40%))。

```
grid-template-columns: repeat(3, 200px);
```

产生三列，列宽都是 200px。

```
grid-template-columns: 100px 1fr 3fr 200px;
```

定义四列，第一列列宽 100px，最后一列列宽为 200px，第二列占剩余空间的四分之一，第三列占剩余空间的四分之三。

fr 就是 fraction 的缩写，意为“份数”，1fr=剩余空间/总份数。比如上面的例子，总份数=1fr+3fr=4fr。

再来看几个复杂些的例子：

```
grid-template-columns: 200px repeat(auto-fill, 100px) 300px;
grid-template-columns: minmax(100px, max-content) repeat(auto-fill, 200px) 20%;
grid-template-columns: [col1] 100px [col2] repeat(auto-fit, [col3 col4] 300px)
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
 100px;
grid-template-columns: [colname1 colname2] 100px
 repeat(auto-fit, [colname1] 300px) [colname3];
```

由于浏览器窗口宽度都是确定的，所以理解 auto-fill、auto-fit 相对简单，可参考 grid-template-rows 介绍去理解。下面来看一个综合的例子，要求如图 5.106 所示的布局。

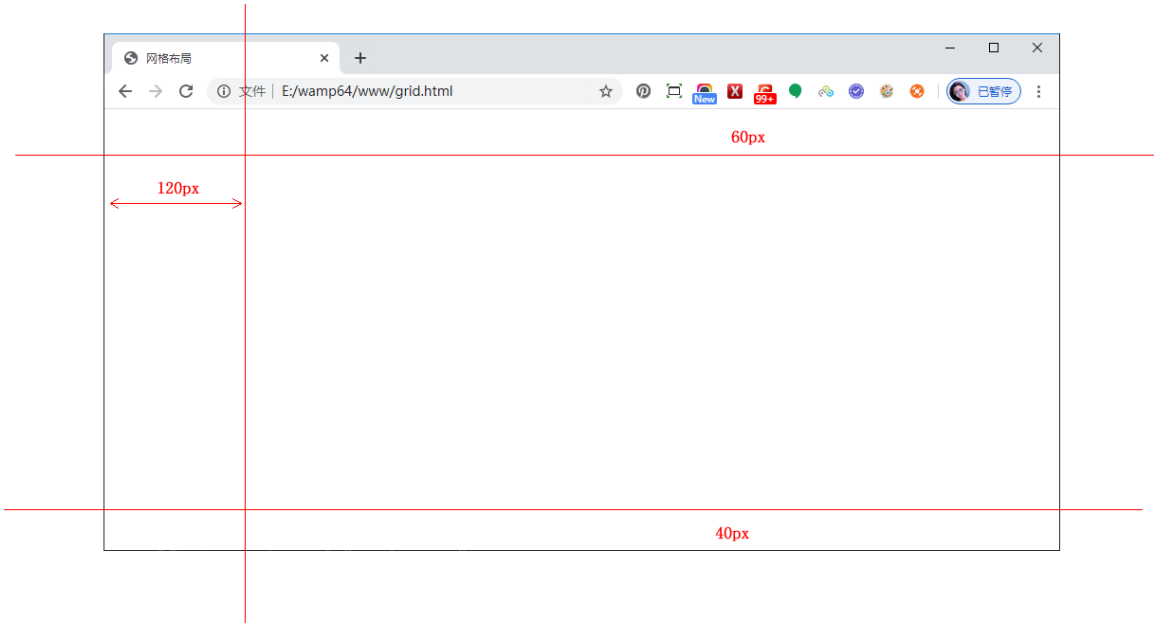


图 5.106 grid 布局综合实例

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Grid 综合例子</title>
    <style>
        #container{
            height: 100vh;
            display: grid;
            grid-template-rows: 60px 1fr 40px;
            grid-template-columns: 120px 1fr;
        }
    </style>
</head>
<body>
    <div id="container"></div>
</body>
</html>
```

实操练习：完成如图 5.107 所示的布局（提醒：采用 vh 和 vw 单位）

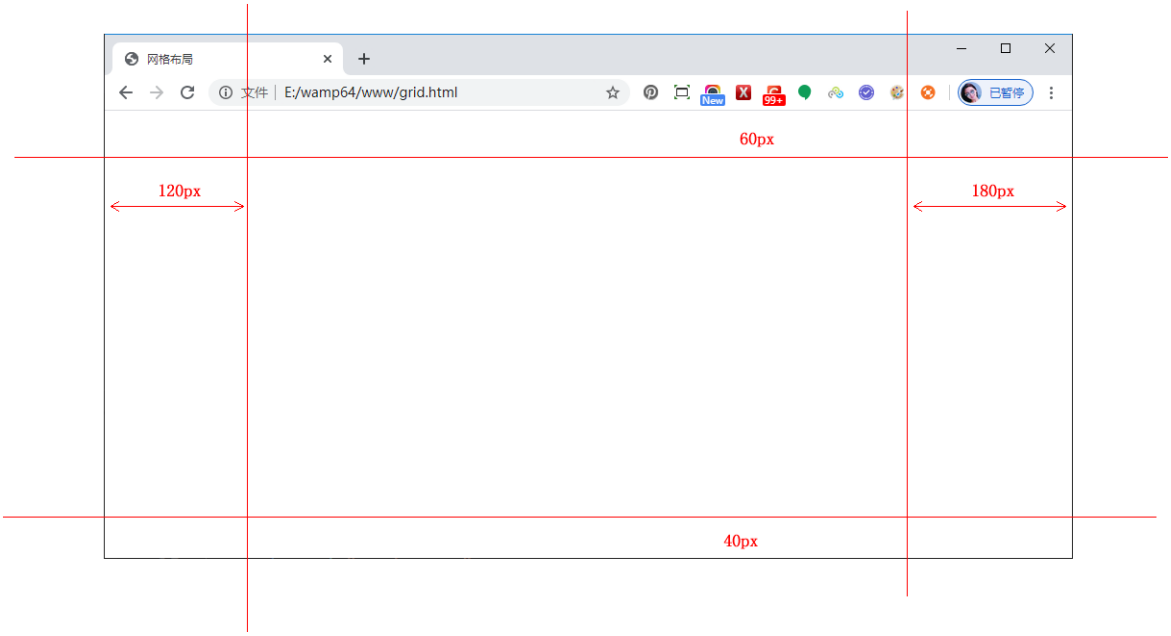


图 5.107 grid 布局实验

3) gap 属性

定义网格线的宽度，语法参考图 5.108 和图 5.109。

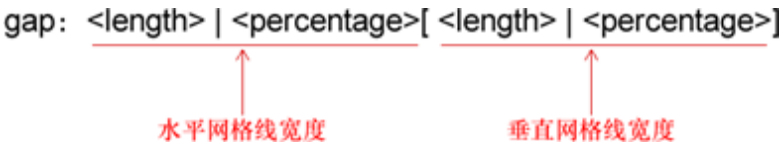


图 5.108 gap 属性语法

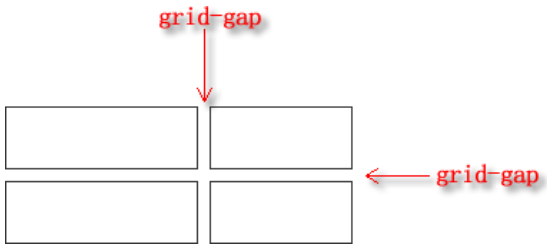


图 5.109 gap 属性说明

例如:

gap: 10px; 横向和纵向网格线宽度都是 10px。

gap: 2% 1rem; 水平网格线宽度为容器高度的 2%，垂直网格线宽度为 1rem。

gap: calc(10px+2%);

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

4) grid-template-areas 属性

定义网格区域，也就是把网格单元划分为不同区域，从而形成不同的功能区，以便后续放置项目，分区有点类似与 excel 表中的合并单元格，见示意图 5.110。

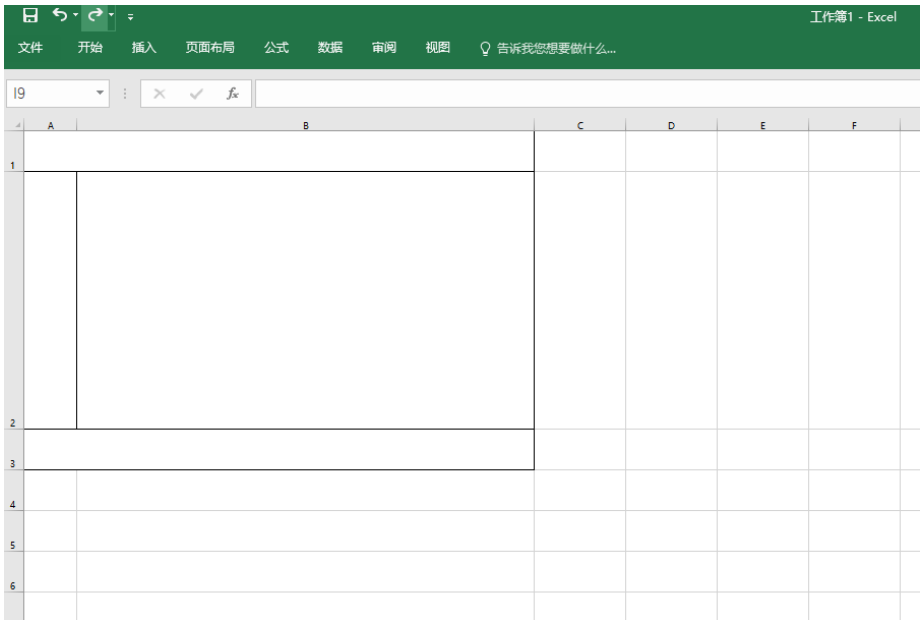


图 5.110 excel 表合并单元格

grid-template-areas 属性的语法参见图 5.111。

```
grid-template-areas: none | <string>+
```

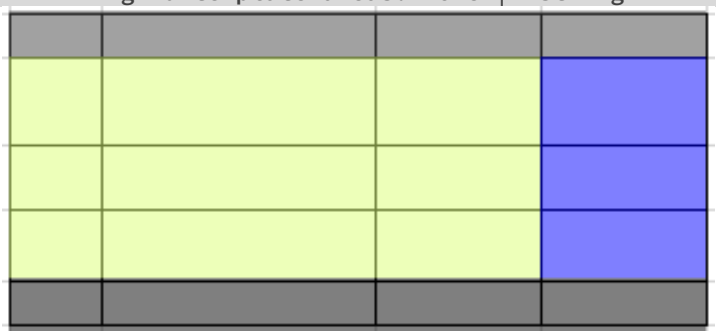


图 5.111 分区示意图

参照示意图 5.111 所示，一个字符串对应网格的一行，字符串中一个单词对应一个单元，单词相同的单元归到一个区域。看代码:

```
grid-template-rows: 50px 1fr 30px;
grid-template-columns: 150px 1fr;
grid-template-areas: "head head"
                     "nav main"
                     "foot foot";
```

上面最后一条代码也可以写在一行上，但是行列对应关系不那么直观。

```
grid-template-areas: "head head" "nav main" "foot foot";
```

网格一共三行两列，第一行归到 head 区域，最后一行归到 foot 区域，中间一行左边的单元命名为 nav 区域，右侧命名为 main 区域。最终实现如图 5.112 所示的布局:

head	head
nav	main
foot	foot

图 5.112 分区举例之一

如果某些单元不归到任何区域，就用“.”命名。比如:

```
grid-template-areas: "a a ."
                     "a a ."
                     ". b c";
```

最终形成如图 5.113 所示的布局。

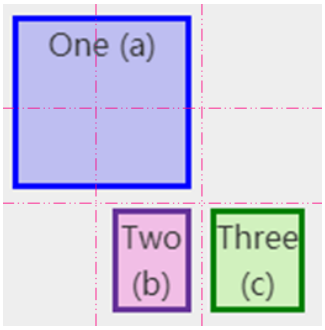


图 5.113 分区举例之二

归属于同一个区域的单元必须能组成一个矩形，否则无效。比如

```
grid-template-areas: "a a ."
                     "a c ."
                     ". b a";
```

就是无效的，因为组成 a 区域的单元不成矩形。

实操练习：采用 grid 布局，要求 4 行 4 列，网格线宽度 8px，分区要求：第一列分成一个区，第一行的 2、3、4 列为一个区，剩余的其他单元格组成一个区。

grid-template-rows、grid-template-columns 和 grid-template-areas 有一个综合属性 grid-template，这个属性可以同时定义行、列和区域，看例子:

```
grid-template: "a a a" 40px
               "b c c" 40px
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
            "b c c" 40px / 1fr 1fr 1fr;
```

本例子定义了三行三列，行高都是 40px，列宽为容器宽度的三分之一，一共分成三个区。再看一个例子：

```
grid-template: repeat(4, 1fr) / repeat(2, 1fr);
```

此例子划分为四行，行高为容器高度的四分之一，划分为两列，列宽为容器宽度的二分之一。

5) grid-auto-flow 属性

指定项目放置算法，即定义把项目摆放在单元格中的方法，项目有大有小，单元格也有大小之分。该属性的语法:

```
grid-auto-flow: [ row | column ] || [ dense ]
```

那么最终有以下几种可能的组合：

row：先行后列，且保持项目的先后次序不变。

column：先列后行，且保持项目的先后次序不变。

dense：等价于 row dense。

row dense：先行后列，但是每次都要从最前面的空白单元开始查找摆放的位置，这样项目的顺序可能就打乱了。

column dense：先列后行，但是每次都要从最前面的空白单元开始查找摆放的位置，这样项目的顺序可能就打乱了。看例子：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>网格布局</title>
    <style>
        #container {
            background-color: #eee;
            border: .75em solid;
            padding: .75em;
            width: 220px;
            display: grid;
            grid-template-columns: 1fr 1fr 1fr;
            grid-template-rows: repeat(3, minmax(40px, auto));
            grid-gap: 10px;
            grid-auto-flow: column;
        }

        #container>div {
            background-color: rgba(0, 0, 255, .2);
            border: 3px solid #00f;
            text-align: center;
        }

        #container>div:nth-child(1) {
```

```
            grid-column: auto/span 2;
        }

        #container>div:nth-child(2) {
            grid-column: auto/span 2;
        }
    </style>
</head>

<body>
    <div id="container">
        <div>One</div>
        <div>Two</div>
        <div>Three</div>
        <div>Four</div>
        <div>Five</div>
    </div>
</body>

</html>
```

通过改变 grid-auto-flow 属性的值，浏览的效果如图 5.114。

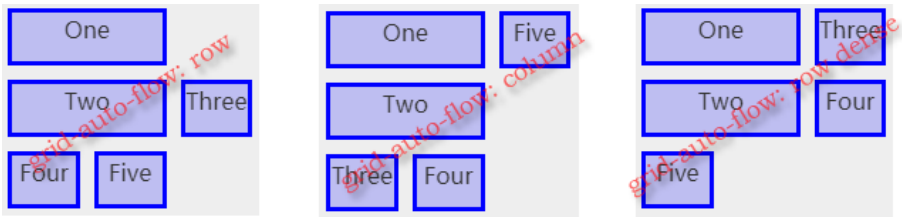


图 5.114　　grid-auto-flow 属性用法举例

6)　grid-auto-rows 属性

如果项目数量多于用 grid-template-rows 和 grid-template-columns 定义的网格数量，或者一些项目的指定位置落在现有网格的外部，浏览器就会自动创建更多的网格单元以便放下全部项目。grid-auto-rows 属性就是定义自动创建的网格行的高度。grid-auto-rows 是语法与 grid-template-rows 一样，当出现多个值时，就循环起作用，例如：

```
grid-auto-rows: 50px 100px;
```

即如果自动创建一行，行高就是 50px，如果创建两行，行高分别是 50px,100px，如果创建五行，行高分别是 50px,100px,50px,100px, 50px。

7)　grid-auto-columns 属性

定义浏览器自动创建的列的列宽。有两种可能会导致自动创建列：第一种情况是项目指定的位置在现有网格之外，比如网格为 3 行 3 列，而某个项目指定位置为 2 行 5 列；第二种情况是采

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

用 column 项目摆放算法（grid-auto-flow:column） 而现有网格单元放不下全部项目。grid-auto-columns 语法与 grid-template-columns 一样。

8) justify-items\align-items\place-items 属性

这三个属性定义容器内的全部项目在网格内（或区域内）的默认对齐方式，另外在项目上还可以单独指定对齐方式（具体参见后续内容）。

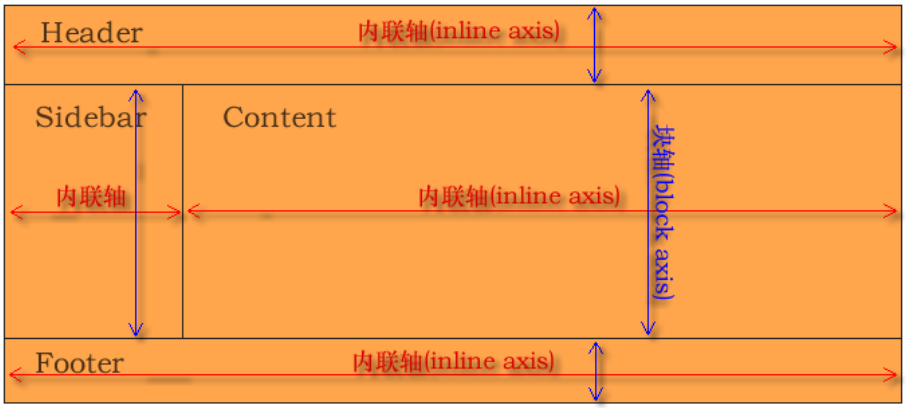


图 5.115 项目对其方式说明

参考图 5.115,在网格内的一个区域中存在两根轴线,沿着文字排列方向的轴线称为内联轴(inline axis),如果语言文字的排列方向是水平的（比如中文等大部分语言），那么内联轴就是水平线，如果语言文字的排列是垂直的（这种语言很少），那么内联轴就是垂直的。与内联轴垂直的轴线称为块轴（block axis）。

justify-items 属性定义项目在内联轴上的对齐方式,比如起点对齐、终点对齐和中点对齐等。align-items 属性定义项目在块轴上的对齐方式。这两个属性的值相同，都是：

```
start | end | center | stretch
```

分别表示起点对齐、终点对齐、中点对齐和拉伸，默认是 stretch（拉伸），即项目沾满整个区域。

place-items 属性是 align-items 属性和 justify-items 属性的合并简写形式,语法是 place-items: <align-items> <justify-items>。看下面的例子：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>网格布局</title>
    <style>
        #container {
            border: .75em solid;
            padding: .75em;
            width: 400px;

            display: grid;
```

```
            grid-template-columns: 1fr 2fr;
            grid-auto-rows: 40px;
            grid-gap: 10px;
            justify-items: end;
        }

        #container>span {
            border: 3px solid #00f
        }
    </style>
</head>

<body>
    <div id="container">
        <span>One</span>
        <span>Two</span>
        <span>Three</span>
    </div>
</body>

</html>
```

通过改变 justify-items 属性的值，浏览的效果如图 5.116。

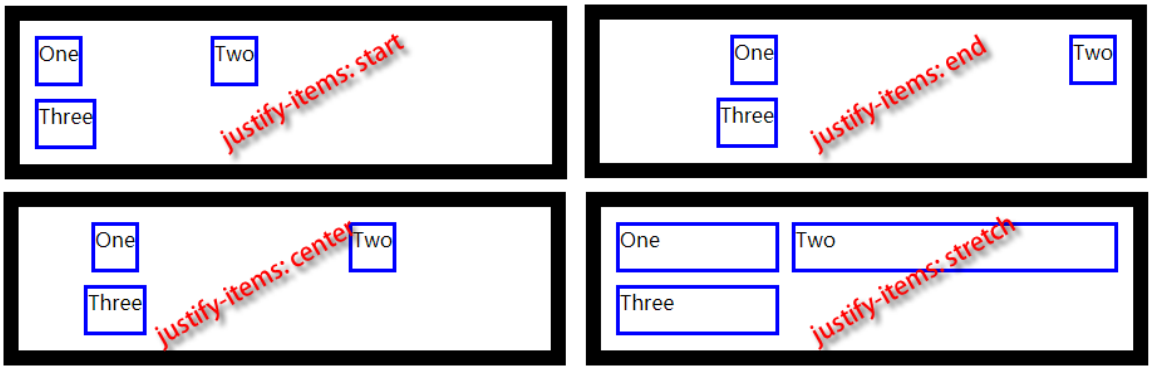


图 5.116 justify-items 属性说明

实操练习

参考上面的例子，给 align-items 属性设置不同的值，观察浏览的效果。

9) justify-content\align-content\place-content 属性

如果容器还有富余空间，这三个属性定义富余空间如何在轨道间分配，换句话说也就是定义轨道水平和垂直方向上的对齐方式。如果没有富余空间，这些属性无效。参考图 5.117。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

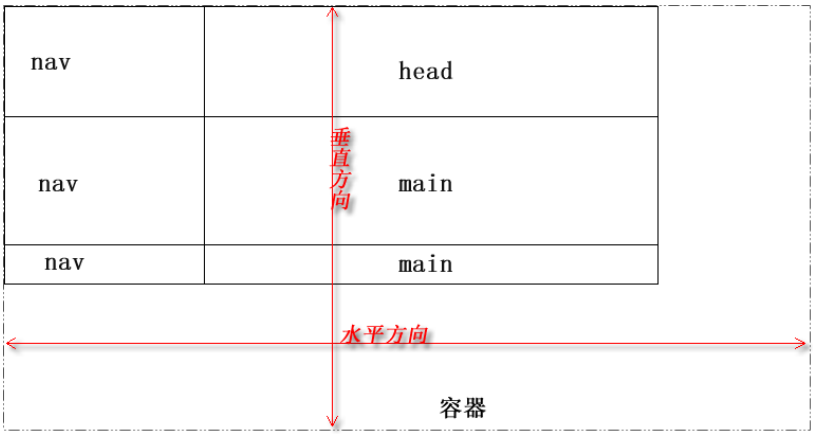


图 5.117 轨道对齐方式说明

justify-content 定义垂直轨道对齐方式，align-content 定义水平轨道对齐方式，这两个属性具有相同的值:

```
start | end | center | stretch | space-around | space-between | space-evenly
```

前四个值分别表示所有轨道紧靠起点、紧靠终点、居中、同比例拉伸沾满容器空间。

space-around：轨道两侧空间相同；space-between：轨道间空间相同，轨道与容器间无空间；space-evenly：轨道之间的间隔相等，轨道与容器边框之间也是相同的间隔。

示意图参见图 5.118。

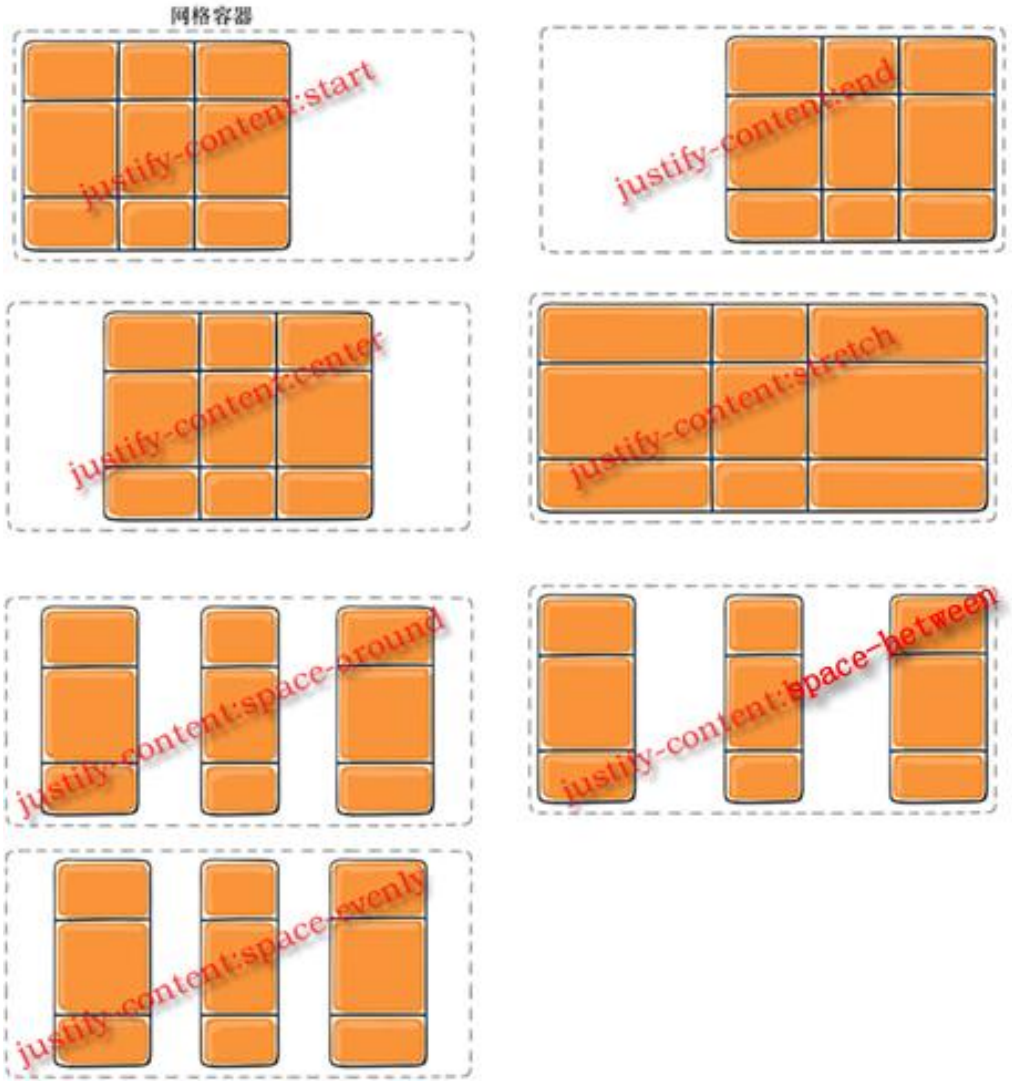


图 5.118 justify-content 属性说明

place-content 属性是 align-content 属性和 justify-content 属性的合并简写形式，语法是 place-content: <align-content> <justify-content>

实操练习

参考上面的例子，把容器的宽度和高度分别设为 400px 和 150px，grid-template-columns 属性设为 100px 150px，然后分别给 justify-content 和 align-content 设置不同的值，观察浏览的效果。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

5.10.3.2 应用在项目上的样式

适用于 grid 项目的样式属性有这些：grid-column-start、grid-column-end、grid-row-start、grid-row-end、grid-column、grid-row、grid-area、justify-self、align-self、place-self ，前 7 个属性用于控制项目的摆放位置，后 3 个属性用于项目的对齐方式。

1) grid-column-start、grid-column-end、grid-row-start、grid-row-end

定义项目摆放位置的起始和终止网格线。前面已经讲过，网格线默认名字是 1、2、3、4、…，水平方向上最左侧的网格线名称为 1，垂直方向上最上面的网格线编号为 1，参见图 5.119。

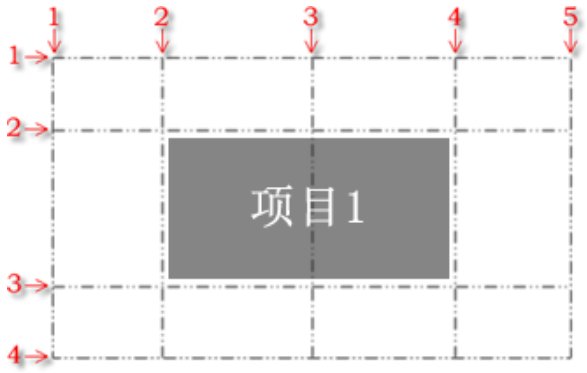


图 5.119 网格线默认命名

另外可以在 grid-template-rows 和 grid-template-columns 属性中定义网格线的名称，具体参考前面的内容。

这四个属性的语法一样，都是：

网格线名称 | span [整数]

“span [整数]”表示跨越几个轨道，如果省略了“整数”，则表示 1。看例子：

```
<head>
    <meta charset="UTF-8">
    <title>网格布局之项目放置</title>
    <style>
        #container {
            border: 1px solid;
            width: 156px;
            height: 210px;
            padding: 2px;
            display: grid;
            grid-auto-rows: repeat(4, 50px);
            grid-template-columns: repeat(3, 50px);
            gap: 2px;
        }

        #container>div {
            font-size: 40px;
            line-height: 50px;
```

```
            text-align: center;
        }

        #container>div:nth-child(1) {
            background-color: red;
            grid-column-start: 2;
            grid-column-end: 4;
        }

        #container>div:nth-child(2) {
            background-color: orange;
        }

        #container>div:nth-child(3) {
            background-color: yellow;
        }

        #container>div:nth-child(4) {
            background-color: green;
        }

        #container>div:nth-child(5) {
            background-color: cyan;
        }

        #container>div:nth-child(6) {
            background-color: blue;
        }

        #container>div:nth-child(7) {
            background-color: purple;
        }

        #container>div:nth-child(8) {
            background-color: grey;
        }

        #container>div:nth-child(9) {
            background-color: blueviolet;
        }
    </style>
</head>

<body>
    <div id="container">
        <div>1</div>
        <div>2</div>
        <div>3</div>
        <div>4</div>
        <div>5</div>
        <div>6</div>
        <div>7</div>
        <div>8</div>
        <div>9</div>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    </div>
</body>
```

项目 1 摆放的水平轨道自动分配（由容器上的 grid-auto-flow 决定），但指定占据 2、3 垂直轨道，示意图如图 5.120。

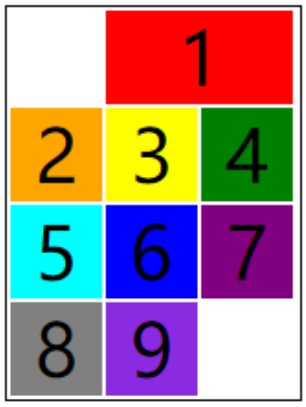


图 5.120 项目摆放举例之一

再比如：

```
#container>div:nth-child(1) {
    grid-column-start: 1;
    grid-column-end: 3;
    grid-row-start: 2;
    grid-row-end: 4;
}
```

浏览效果如图 5.121。



图 5.121 项目摆放举例之二

从上面的例子可以看出，除了特别指定摆放位置的项目外，其他项目的摆放位置还是由容器上的属性 grid-auto-flow 决定，即 grid-auto-flow 把单独定义了摆放位置的项目排除在外。

再看代码：

```
.item-1 {
    grid-column-start: header-start;
    grid-column-end: header-end;
```

```
}
```

要求：网格线 header-start 和 header-end 已经在 grid-tenplate-columns 属性中定义。

代码：

```
#container>div:nth-child(1) {
    grid-column-start: span 2;
}
```

表示跨越两列，示意图如图 5.122。



图 5.122 项目摆放举例之三

grid-column 属性是 grid-column-start 和 grid-column-end 的合并简写形式，grid-row 属性是 grid-row-start 属性和 grid-row-end 的合并简写形式。语法：

```
grid-column: <grid-column-start>/<grid-column-end>
grid-row: <grid-row-start>/<grid-row-end>
```

如：

```
grid-row: 2/4;
grid-column: 1/3;
```

等价于：

```
grid-row-start: 2;
grid-row-end: 4;
grid-column-start: 1;
grid-column-end: 3;
```

2) grid-area 属性

指定项目放在哪一个区域。其语法为：

```
grid-area: 区域名 | <row-start>/<column-start>/<row-end>/<column-end>
```

有两种选择：一是直接指明区域名（由容器属性 grid-template-area 定义），二是指定摆放位置的左上角网格线和右下角网格线名称。如：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
grid-area: Header;
grid-area: 2/3/4/6;
```

3) justify-self、align-self、place-self 属性

容器上的 justify-items 和 align-items 定义所有项目的对齐方式，如果某个项目需要不同的对齐方式，就可以在此项目上运用 justify-self、align-self 属性。其中 justify-self 定义水平对齐，align-self 定义垂直对齐，两个属性的语法和取值一样：

```
justify-self: start | end | center | stretch;
 align-self: start | end | center | stretch;
```

四个值分别表示起点对齐、终点对齐、居中对齐和拉伸。

place-self 属性是 align-self 属性和 justify-self 属性的合并简写形式。格式为：

```
place-self: <align-self>[ <justify-self>];
```

如果只有一个值，那么浏览器认为水平和垂直方向上的对齐方式一样。

看例子：

```
.item-a  {
    justify-self: start;
}
```

示意图如图 5.123。

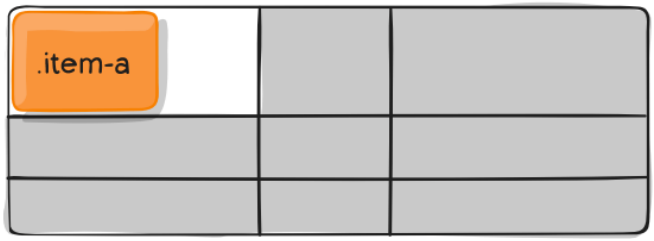


图 5.123 justify-self 属性用法举例

5.10.4 布局实战：结合 grid 和 flex

前面已经讲过，flex 为一维布局（通常是水平方向），以内容优先，而 grid 为二维布局（水平和垂直方向），以布局优先。我们在涉及网页时一般是在二维布局中嵌套一维布局，所以结合 grid 和 flex 是一个非常流行的网页布局方法：先用 grid 规划整个网页布局，然后在具体的块内采用 flex 做一维布局。本实验实现如图 5.124 所示的网页。

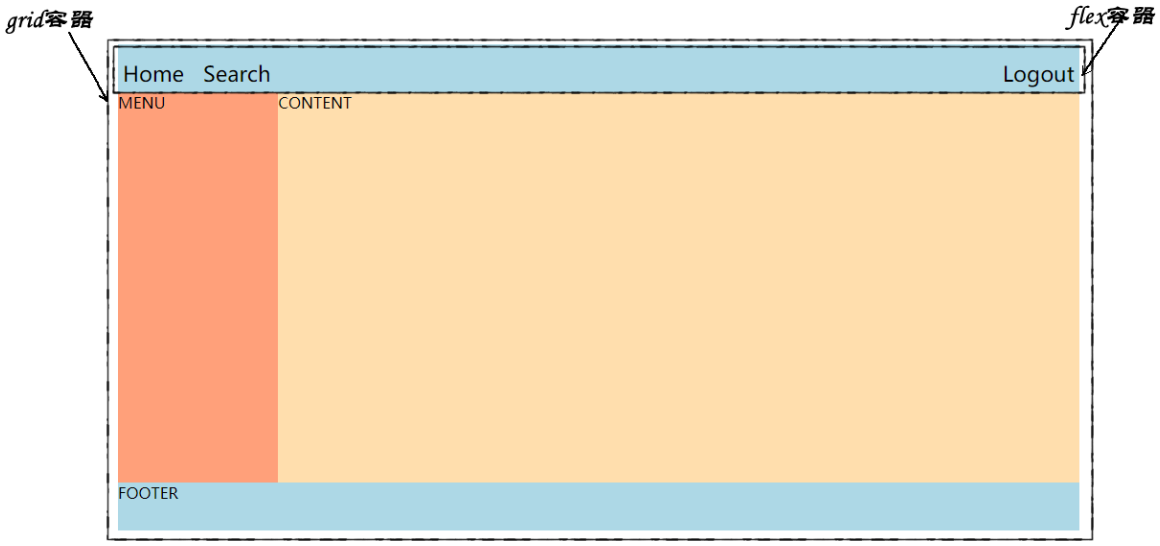


图 5.124 结合 grid 和 flex 的综合布局例子

第 1 步：先用 grid 完成如图 5.125 所示的布局。



图 5.125 grid 布局

网页代码如下：

```
<head>
    <title>Grid-Flex 布局实验</title>
    <style>
        .container {
            height: 97vh;
            display: grid;
            grid-template-columns: repeat(12, 1fr);
            grid-template-rows: 50px minmax(300px,auto) 50px;
        }

        header {
            grid-column: span 12;
            background-color: lightblue;
        }
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
        aside {
            grid-column: span 2;
            background-color: lightsalmon;
        }

        main {
            grid-column: span 10;
            background-color: navajowhite;
        }

        footer {
            grid-column: span 12;
            background-color: lightblue;
        }
    </style>
</head>

<body>
    <div class="container">
        <header>HEADER</header>
        <aside>MENU</aside>
        <main>CONTENT</main>
        <footer>FOOTER</footer>
    </div>
</body>
```

第 2 步：在 HEADER 部分再采用 flex 布局

html 结构代码改为：

```
<body>
    <div class="container">
        <header>
            <div>Home</div>
            <div>Search</div>
            <div>Logout</div>
        </header>
        <aside>MENU</aside>
        <main>CONTENT</main>
        <footer>FOOTER</footer>
    </div>
</body>
```

css 样式代码修改为：

```
        header {
            grid-column: span 12;
            background-color: lightblue;
            display: flex;
        }

        header>div {
            align-self: flex-end;
            font-size: 22px;
        }
```

```
header>div:nth-child(2) {
    margin-left: 20px;
}

header>div:nth-child(3) {
    margin-left: auto;
}
```

最终看到的效果见图 5.126 所示，完整的代码参见 chapter-5\index.html。

实操练习

分别完成下面图 5.127 和图 5.128 所示的布局。

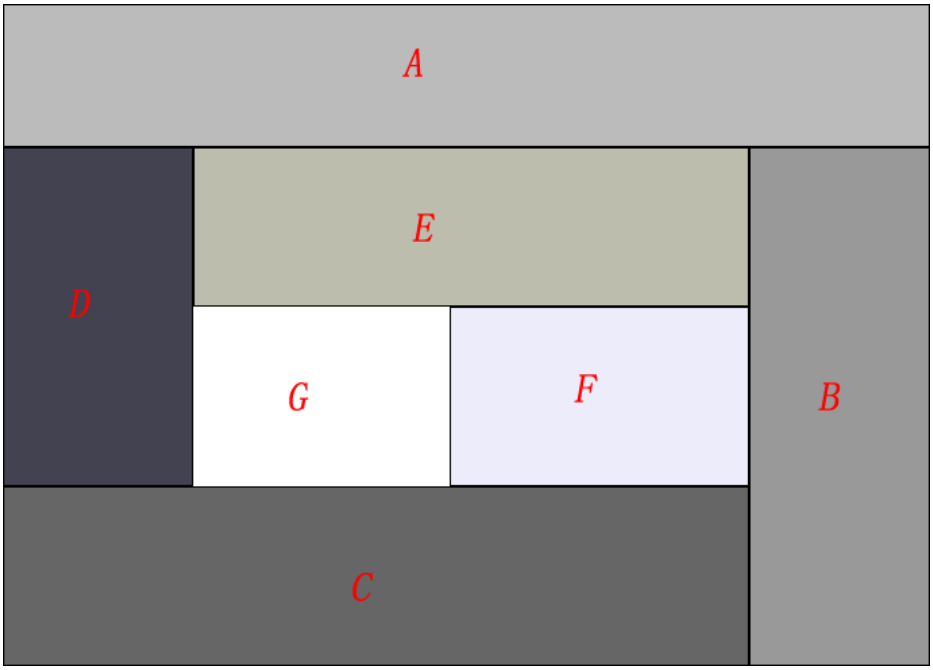
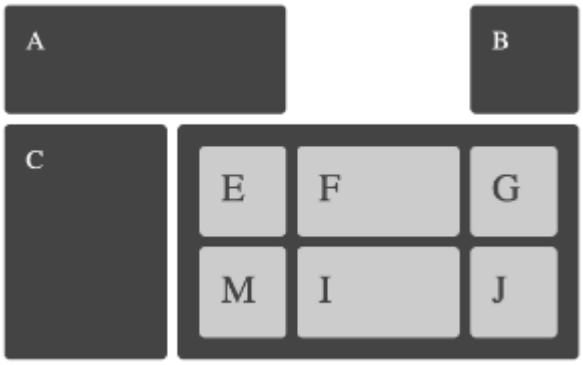


图 5.127 实验一



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

图 5.128 实验二

5.11 变换

可以对非 inline 元素进行移动、旋转、伸缩、切变、矩阵变换等操作，变换的结果不会导致周围元素移动，本小节分别讲述元素在 2D 和 3D 空间中的变换。

5.11.1 2D 变换

采用样式属性 transform 做 2d 变换，它的值是一系列的函数，具体语法如图 5.129。

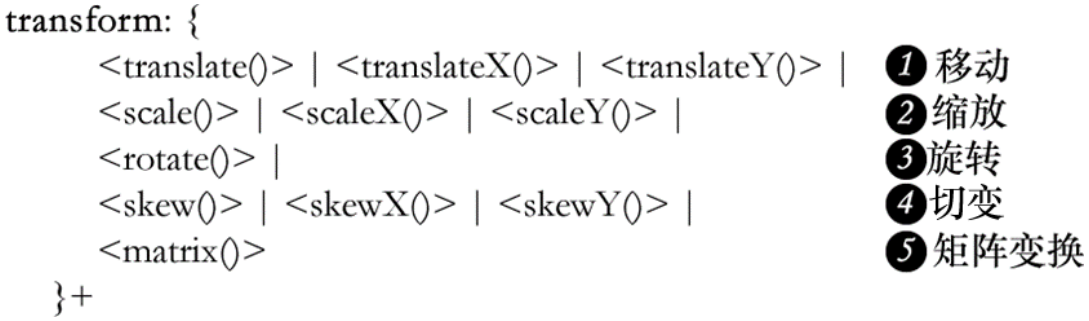


图 5.129 transform 属性的使用语法

默认的值为 none，即没有变换，允许出现多个值，多个值之间用空格分隔，从右至左执行变换，比如：

```
transform: translateX(10px) rotate(-45deg);
```

元素在文档流中的原始位置的中心就是变换的源点（即坐标原点），变换操作不会改变元素的源点，除非采用 transform-origin 样式修改源点，参考图 5.130 所示。

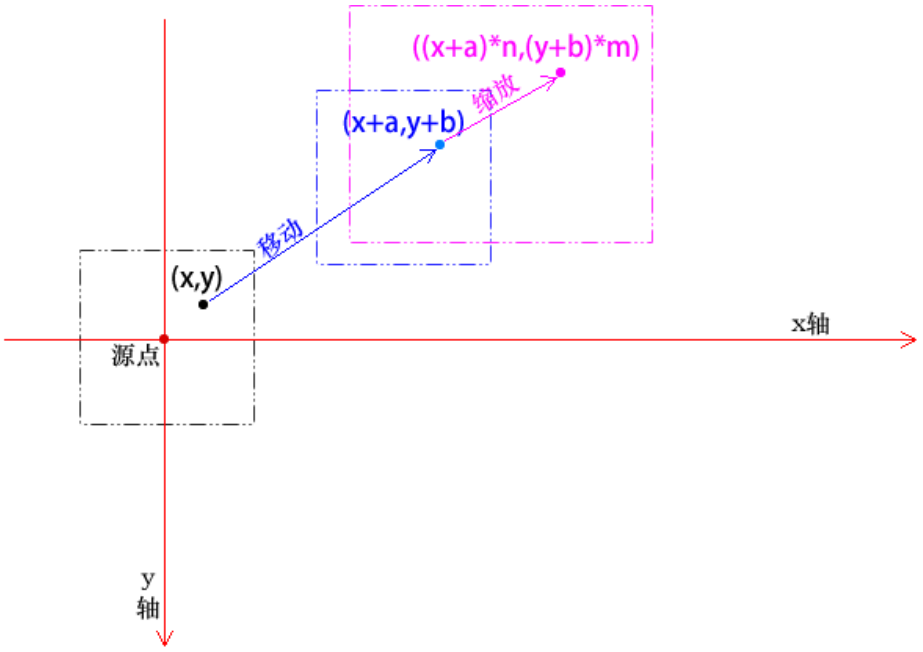


图 5.130 源点没变

移动函数 translate()的语法如下：

```
translate( <length-percentage> [, <length-percentage> ]? )
```

此函数允许带一个或两个参数，如果是一个参数，则表示水平移动，如果是两个值，则表示水平和垂直方向上都移动。注意：参数既可用长度单位（length）也可以用百分数（percentage）表示，可为正数或负数，比如-100px、30%。当采用百分数时表示移动的距离是当前元素参考区域（具体由 transform-box 定义）的宽或高的百分比，例如 translate(40%)就是水平方向上向右移动的距离是参考区域宽度的 40%。看例子：

```
<style>
    span {
        background-color: yellow;
        display: block;
        transform: translate(10px, 30px);
    }
</style>
……
<div>
    <p>111111</p>
    <span>201801234</span>
    <p>222222</p>
</div>
```

使 span 元素右移 10px，下移 30px，浏览的效果如图 5.131。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

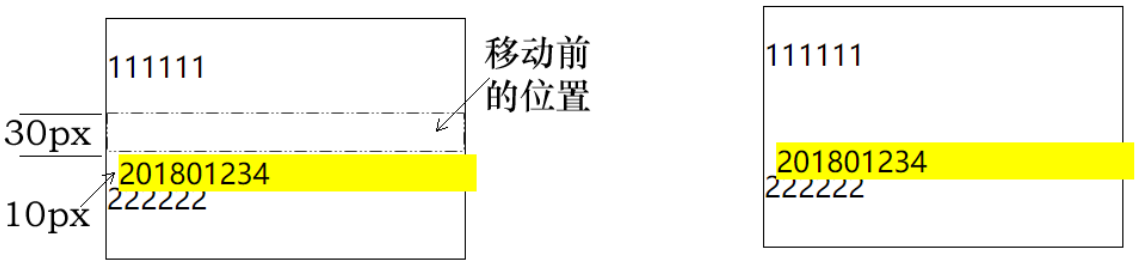


图 5.131 移动变换

函数 translateX()和 translateY()只带一个参数，分别表示水平移动和垂直移动。比如 translateX(-10px)表示左移 10px。看例子：

```
transform: translateX(40%);
```

浏览效果如图 5.132。

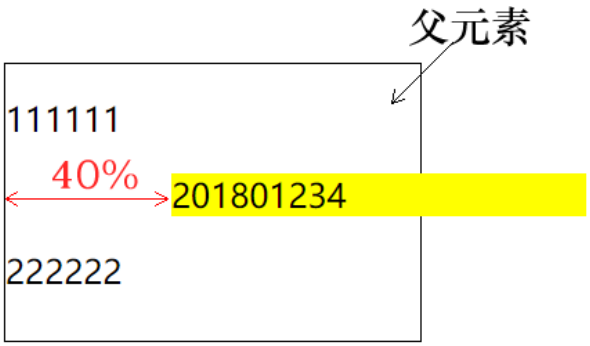
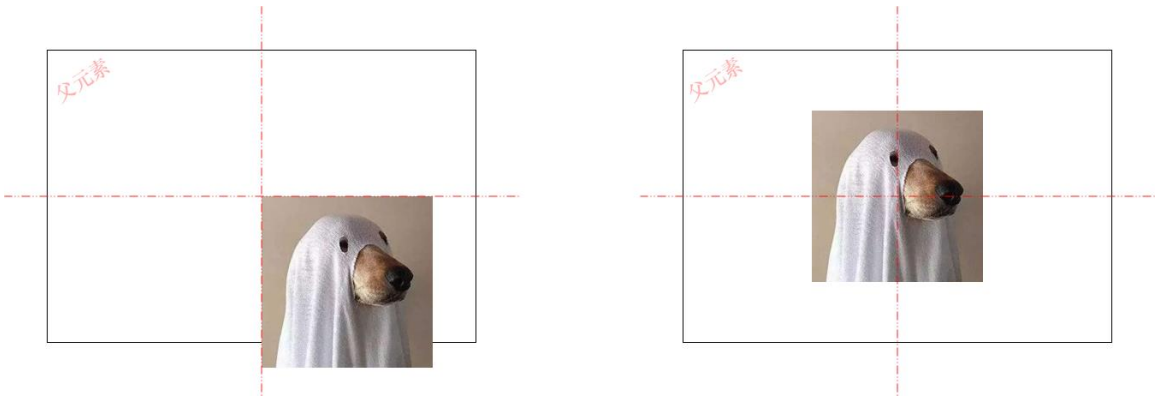


图 5.132 translateX()用法举例

结合 trnasform: translate()、top 和 left 可以非常方便实现在一个容器元素的中央显式图片，参考下图所示，即先把图片的左上角定位在父元素的中心，然后对图片施行 transform: translate(-50%, -50%)变换，从而使得图片往上往左各移动图片的一半。



参考代码如下：

```
<body>
   <div style="width: 800px; height: 300px;border: 1px solid; top: 200px; left: 200px;
position: relative;">
```

```
      <img src="photo.png" style="display: block; position: absolute; left: 50%; top: 50%;
transform: translate(-50%, -50%);">
   </div>
</body>
```

实操练习

完成如图 5.133 所示的效果，图中由三个元素，移动那个蓝色的圆。

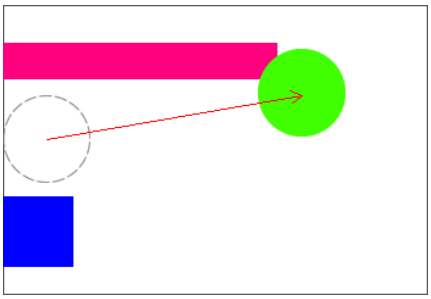


图 5.133 移动变换实验

缩放函数 scale()的语法如下：

```
scale( <number> [, <number>]? )
```

number 为正实数，元素的尺寸乘以 number 就是缩放后的大小，因此当 number 大于 1 时就是放大，小于 1 时就是缩小，等于 1 时不变。当 scale()函数只有一个参数时表示元素的水平和垂直方向同比例缩放，有两个参数时，分别定义水平方向和垂直方向的缩放比率。看例子：

```
      transform: scale(2);    /* 水平和垂直方向都放大 2 倍 */
      transform: scale(0.5, 1.5);  /* 水平方向缩小一半，垂直方向放大 1.5 倍 */
```

函数 scaleX()只缩放元素的水平方向的尺寸，scaleY()只缩放元素垂直方向的尺寸。

实操练习

在设备像素比（DPR）为 2 的屏幕上如何得到线宽为 1 个物理像素的横线（提醒：对<hr>元素做 scaleY(0.5)的变换）。DPR 的概念参考“§2.1 了解显示屏幕”小节。

需要注意的是水平方向不一定就是元素的宽度，类似地，垂直方向不一定就是元素的高度，比如图 5.134 中的右边的矩形，由于之前旋转了它，所以水平方向就不是元素的宽度了。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

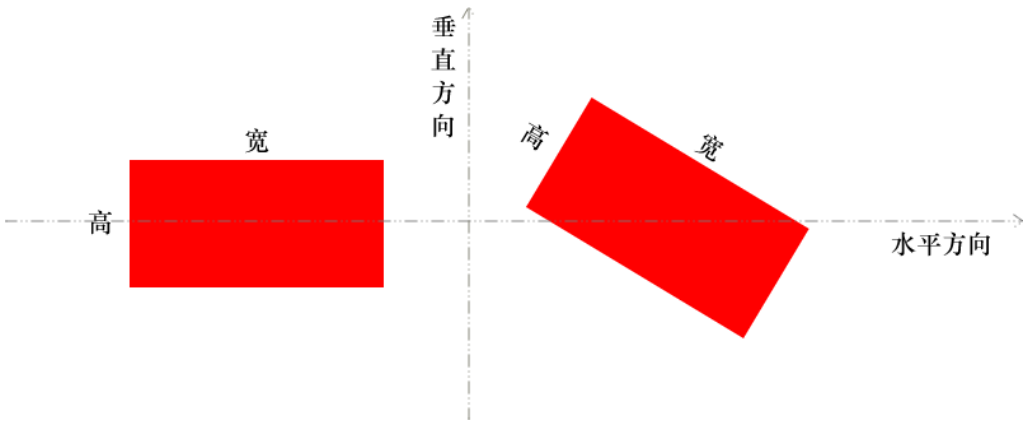


图 5.134 元素的宽度不一定就是水平方向

实操练习

对图 5.135 所示的红色圆做缩放操作。

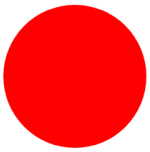


图 5.135 缩放红色圆

函数 rotate()旋转元素，其语法如下：

```
rotate( <angle> );
```

以元素的源点为转轴，可顺时针（参数为正数）或逆时针（参数为负数）旋转元素，需要注意的是源点不一定就是元素的中心，如图 5.136 所示，尽管旋转的角度一样，但是源点不同，最终的效果也不同。

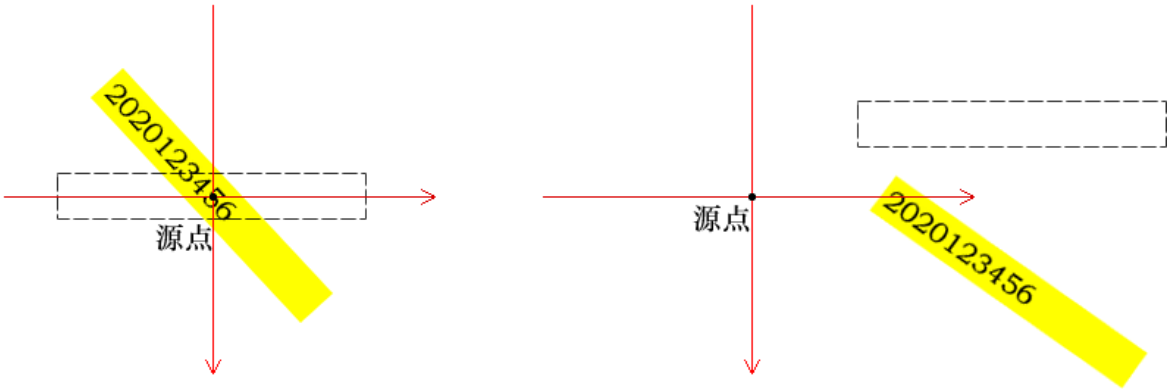


图 5.136 源点影响旋转效果

再看例子：

```
transform: rotate(30deg);
```

浏览的效果如图 5.137 所示。

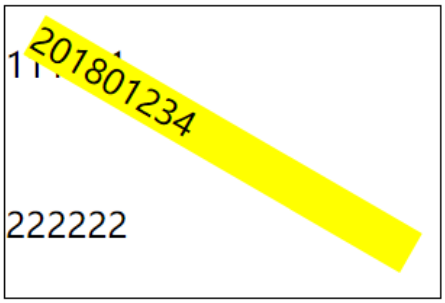


图 5.137 旋转变换举例

实操练习

对一个红色的正方形做移动、旋转和缩放操作。然后交换变换函数的先后次序，观察浏览效果有何不同。



采用切变函数 skew()可以使得元素在水平和垂直方向上产生切变变化，它的语法如下：

```
skew( <angle> [, <angle> ]? )
```

如果只有一个参数，表示只在水平方向上切变，第二个参数表示在垂直方向上切变。看例子：

```
transform: skew(-35deg);
```

浏览效果如图 5.138。

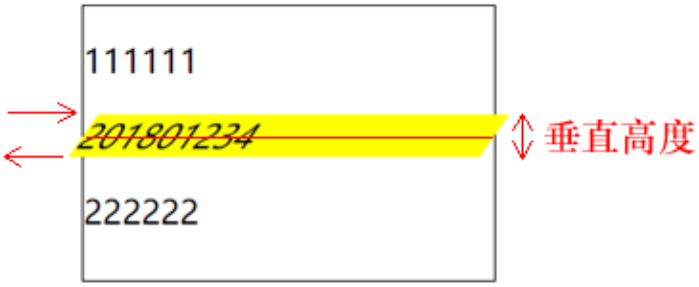


图 5.138 切变变换举例（一）

水平方向切变保持元素的垂直高度不变，水平中间线也不变。再看例子：

```
transform: skew(0, -35deg);
```

浏览效果如图 5.139：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

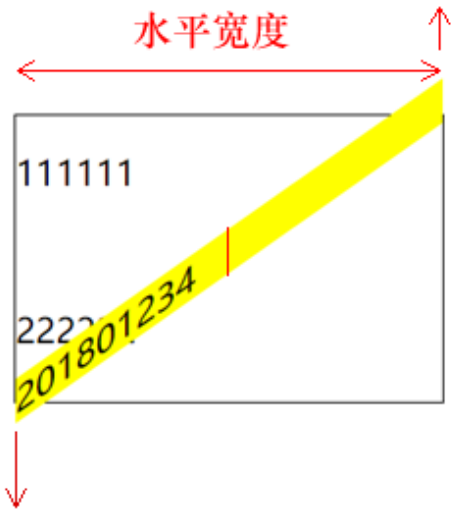


图 5.139 切变变换举例（二）

垂直方向切变保持元素的水平宽度不变，垂直中间线也不变。函数 skewX()和 skewY()分别只在水平方面和垂直方向上切变，参考图 5.140 所示。

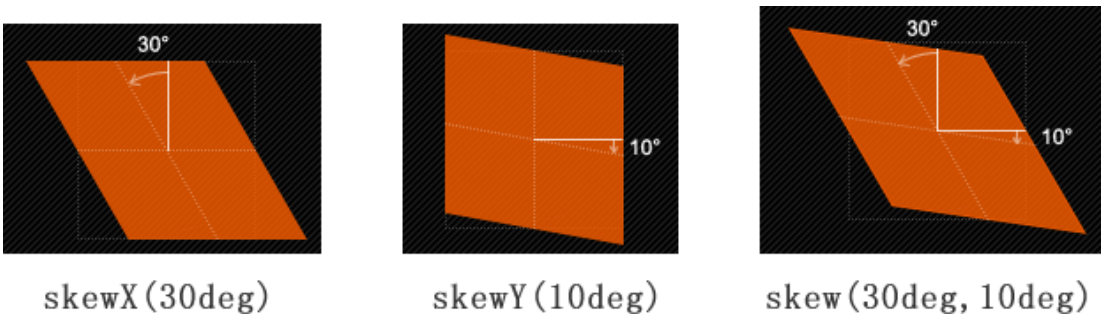


图 5.140 skewX()和 skewY()用法说明

矩阵变换函数 matrix(a,b,c,d,e,f)可以一次性完成复杂的变换，它的六个参数 a,b,c,d,e,f 在变换矩阵中的位置如下：

$$\begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix}$$

假设(x,y)对应矩阵变换后的坐标为(x_1,y_1)，那么这两个坐标之间存在如下关系：

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \xrightarrow{可得} \begin{cases} x_1 = a \times x + c \times y + e \\ y_1 = b \times x + d \times y + f \end{cases}$$

因此 matrix(1,0,0,1,e,f)等价于 translate(e,f)，把 a=1,b=0,c=0,d=1 代入上面的公式得$\begin{cases} x_1 = x + e \\ y_1 = y + f \end{cases}$，这就是平移函数 translate(e,f)完成的操作，因此 translate(x,y)对应的变换矩阵为$\begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 0 & 0 & 1 \end{bmatrix}$，即 translate(x,y)等价于 matrix(1,0,0,1,x,y)，同理可以得出缩放函数 scale(x,y)的等价变化矩阵为$\begin{bmatrix} x & 0 & 0 \\ 0 & y & 0 \\ 0 & 0 & 1 \end{bmatrix}$，即 scale(x,y)等价于 matrix(x,0,0,y,0,0)，旋转函数 rotate(x)的等价矩阵为$\begin{bmatrix} \cos x & -\sin x & 0 \\ \sin x & \cos x & 0 \\ 0 & 0 & 1 \end{bmatrix}$，即 rotate(x)等价于 matrix$(\cos x, \sin x, -\sin x, \cos x, 0, 0)$。

关于矩阵变换更详细的介绍本书前面章节的 canvas 画布元素的 tranform()函数（§4.5.4.10 小节的 transform()函数）。

可以给 transform 属性指定多个 2d 变换函数，以便一次性完成多种变换，此时变换的顺序是从右至左，看例子：

```
div {
     width: 200px;
     height: 200px;
     border: 1px solid black;
     margin: 50px;
     font-size: 20px;
     transform: translate(100px, 40px) scale(1, 1.5);
 }
```

先缩放再移动，本质上就是各个变换函数对应的矩阵相乘，即平移矩阵乘以缩放矩阵：

$$\begin{bmatrix}1 & 0 & 100\\0 & 1 & 40\\0 & 0 & 1\end{bmatrix}\times\begin{bmatrix}1 & 0 & 0\\0 & 1.5 & 0\\0 & 0 & 1\end{bmatrix}=\begin{bmatrix}1 & 0 & 100\\0 & 1.5 & 40\\0 & 0 & 1\end{bmatrix}$$

因此这两个变换最终等价的矩阵变换函数为 matrix(1,0,0,1.5,100,40)，浏览效果参见下图左侧部分。由于矩阵乘法不满足交换律，所以如果我们把上面例子中的两个变换函数交换顺序，改为：

```
transform: scale(1, 1.5) translate(100px, 40px);
```

即先移动再缩放，对应的矩阵运算公式为：

$$\begin{bmatrix}1 & 0 & 0\\0 & 1.5 & 0\\0 & 0 & 1\end{bmatrix}\times\begin{bmatrix}1 & 0 & 100\\0 & 1 & 40\\0 & 0 & 1\end{bmatrix}=\begin{bmatrix}1 & 0 & 100\\0 & 1.5 & 60\\0 & 0 & 1\end{bmatrix}$$

等价的矩阵变换函数为 matrix(1,0,0,1.5,100,60)，显然与上面的矩阵变换函数不同，最后一个参数由 40 改变为 60，那么最终浏览的效果也就不同了，参见图 5.141 右侧部分。

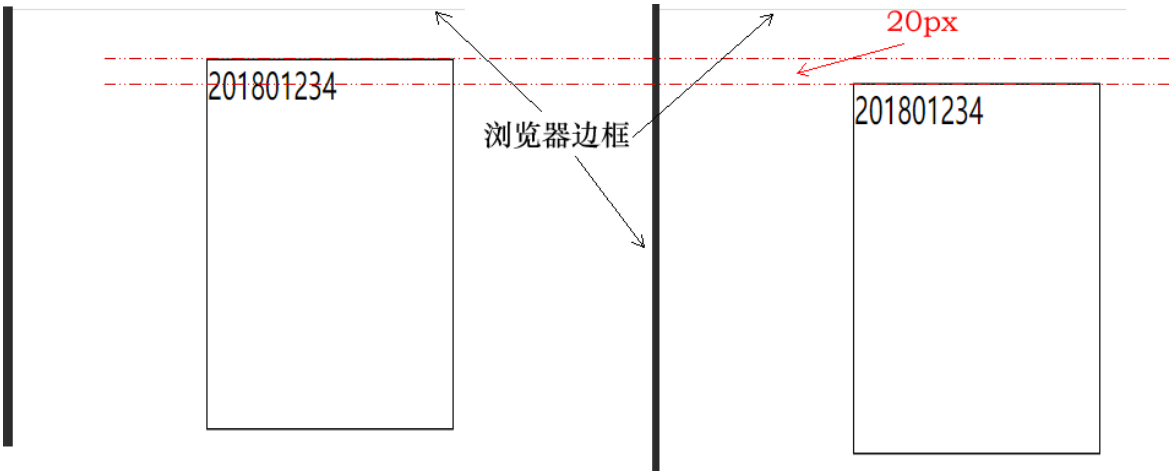


图 5.141 变换的先后次序影响最终效果

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

为什么会出现这种情况呢？解释过程请参考示意图 5.142，图中红色框为变换前的样子，蓝色框为首次变换后的样子（左侧是放大，右侧是移动），黑色框是第二次变换后的样子（左侧是移动，右侧是放大），大家可以对 a 点的坐标（以源点为坐标原点）进行放大计算。

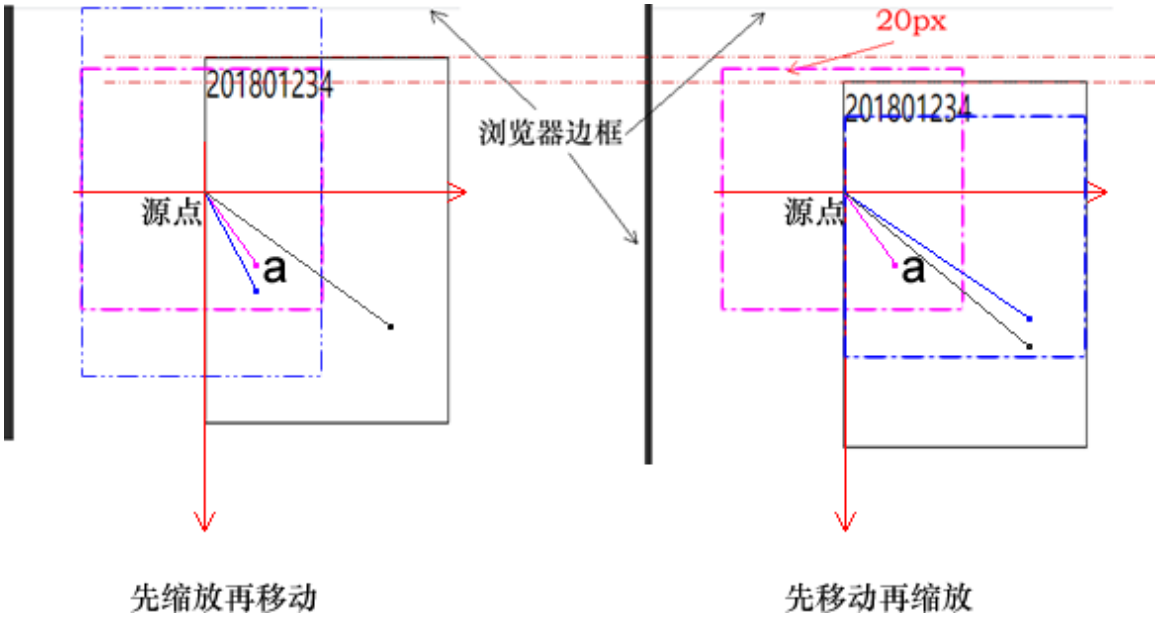


图 5.142 变换次序影响结果说明

因为缩放和旋转与元素的源点紧密相关。因此我们可以得出结论：transform 样式属性的多个值的顺序会影响变换的最终效果。

我们再来看一个例子：

```
transform: scaleX(0.6) rotate(45deg);
```

交换变换函数的顺序为：

```
transform: rotate(45deg) scaleX(0.6);
```

最终的效果不同，见图 5.143。

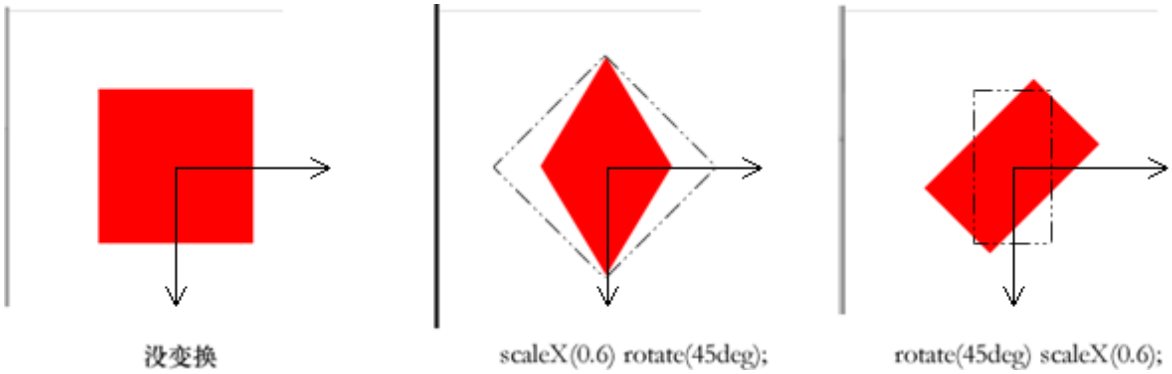


图 5.143 变换次序影响结果举例

图 5.143 中间和右边部分，虚线代表做第一次变换后的效果，即中间部分做旋转，右侧部分做缩放。

实操练习

完成如图 5.144 所示的黑桃方块（提醒：先旋转后缩小）。

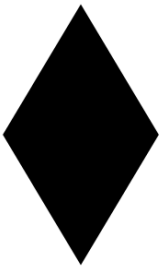


图 5.144 变换实验

5.11.2 3D 变换

三维空间内的坐标是这样的：x 轴为元素的宽度方向，y 轴为高度方向，而 z 轴始终垂直于元素表面，坐标原点位于元素的中央，见图 5.145 所示。牢记这个坐标系很重要，它就是一个左手坐标系统，左手大拇指朝 Z 轴方向，其他四指朝 x 轴方向，手臂就是 y 轴方向了。图 5.145 中左侧部分元素内没有子元素，右侧部分变换的元素里面有六个子元素，分别表示体力放的六个面。

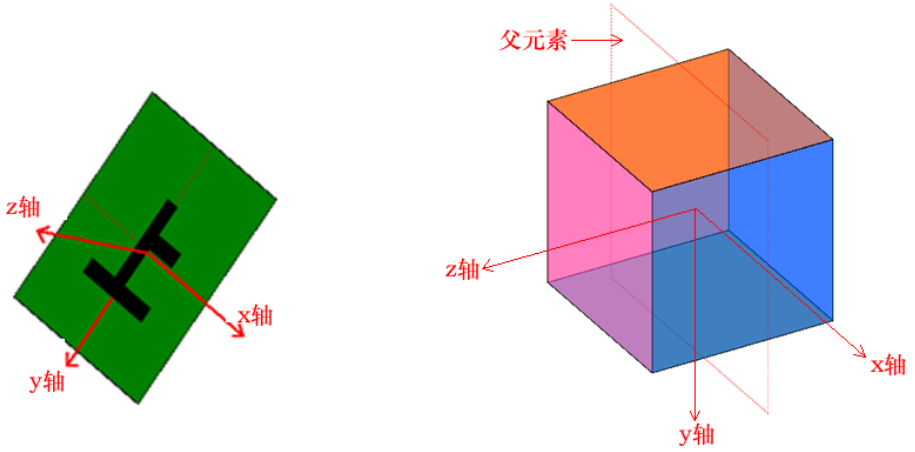


图 5.145 三维坐标系

我们都知道，在三维空间中观察一个物体的效果与很多因素有关，比如观察者与物体的距离（称为视距）、观察者站立的位置以及物体本身的形状等，相应地 3D 变换涉及的样式有 perspective（定义视距）、perspective-origin（定义观察者的位置）、transform-origin、transform 和 transform-style 等。

transform-style 的值为 flat 或者 preserve-3d，默认值是 flat(即 2d 变换)，在 3d 中的值只能是 preserve-3d，表示子元素在 3d 空间中呈现，注意：本属性只对子元素起作用。

transform-origin 属性定义变换源点，默认是元素的中心。其语法如下：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

transform-origin:

❶ [<length-percentage> | left | center | right | top | bottom] |

❷ [<length-percentage> | left | center | right] && [<length-percentage> | top | center | bottom] <length>?

格式①为单值，如果取<length-percentage> 、left、center、right 中的一个则代表 x 轴方向上的位置，y 轴方向上就是 center，如果取 top 或者 bottom，则是 y 轴方向上元素的顶部或底部，x 轴方向上为 center。格式②可能为双值或三值（加上 z 轴方向上的<length>），双值则是明确定义 x 轴和 y 轴上的位置，三值就连 z 轴上的位置都确定了，例如：

```
transform-origin: left top 100px;
```

x 轴上靠左，y 轴上靠上，即左上角，z 轴上 100px 处。再看图 5.146 所示的一些例子。

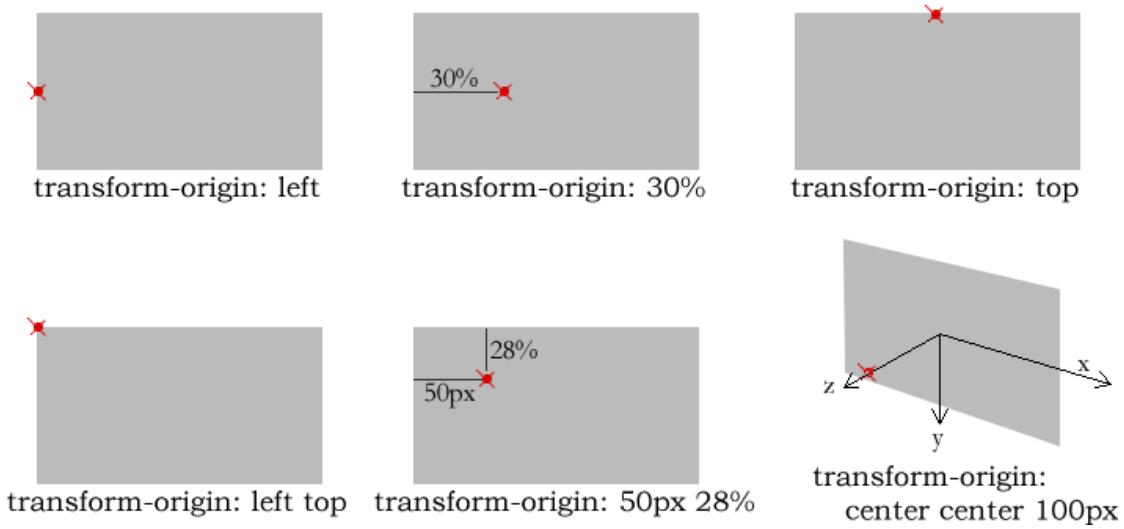


图 5.146 transform-origin 属性用法举例

perspective 属性定义视距，即观察者与(x,y)坐标平面的垂直距离，它的值为长度单位，如 400px、10rem 等，默认值是 none，可为 0 或负数，值越小视觉越深。本属性只对子元素有效。但是存在一个同名函数 perspective()，可做 transform 样式属性的值，此函数对本元素和子元素都有效。注意：凡是涉及 z 轴方向上的变换，必须结合使用 perspective 属性或函数。请参考图 5.147。

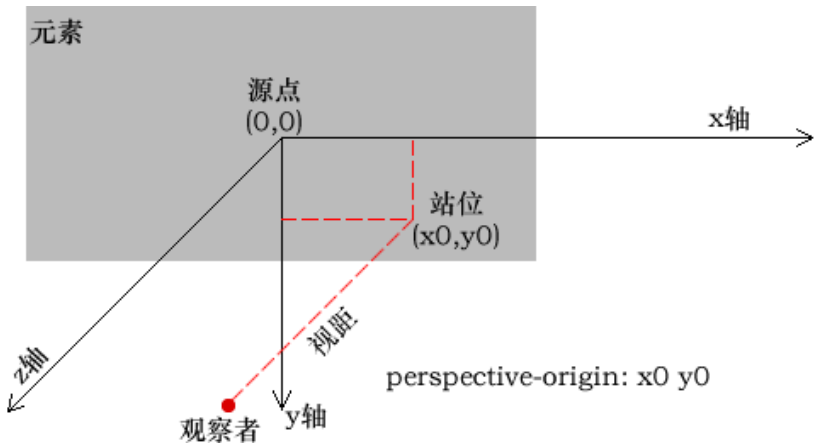


图 5.147 定义视距

perspective-origin 属性定义观察者的位置，即镜头在 x/y 坐标平面上的投影的坐标值(x0，y0)（即站位），本属性需要结合 perspective 属性（定义视距）使用，见图 5.147 所示。perspective-origin 只对子元素有效。此属性的语法如下：

perspective-origin:

❶ [left | center | right] || [top | center | bottom] |
❷ [left | center | right | <length-percentage>] [top | center | bottom | <length-percentage>]? |
❸ [[left | right] <length-percentage>] && [[top | bottom] <length-percentage>]

值有三种格式，格式①中的“||”两边可选一边的一个值或者两边各选一个值，比如 perspective-origin: left、perspective-origin: bottom、perspective-origin: right top，如果是两个值，第一个值定义水平方向的位置，第二个值定义垂直方向的位置。格式②是在格式①的基础上增加了长度值或相对于元素宽度的百分比值，可为负值，例如 perspective-origin: 100px 25%表示里元素左侧 100px、距离元素顶部为元素高度的 25%。格式③必须两个值，分别表示离左侧或右侧多远，离顶部或底部多远，例如 perspective-origin: right 30% top 200px 表示离元素右侧距离为元素宽度的 30%，离元素顶部的距离为 200px。

transform 样式属性的语法如下：

transform: {

<translate3d()> | <translateX()> | <translateY()> | <translateZ()> | ❶ 移动
<scale3d()> | <scaleX()> | <scaleY()> | <scaleZ()> | ❷ 缩放
<rotate3d()> | <rotateX()> | <rotateY()> | <rotateZ()> | ❸ 旋转
<perspective()> | ❹ 视距
<matrix3d()> ❺ 矩阵变换

}+

移动函数 translate3d()的语法如下：

```
translate3d( <length-percentage>, <length-percentage> , <length> )
```

三个参数分别定义了相对于元素从文档中原始的位置向 x、y、z 轴方向上移动的距离，可正可负。由于第三个参数代表 z 轴方向上的移动距离，所以不能采用百分数，因为元素没有厚度。如果涉及到在 z 轴上的移动，必须结合使用 perspective 属性或者 perspective()函数。

```
……
    p {
        width: 100px;
        height: 50px;
        background-color: #777;
        color: white;
        font-size: 26px;
        text-align: center;
        line-height: 50px;
    }

    #change3d {
        margin-left: 50px;
        transform: translate3d(100px, -30px, 0px);
        background-color: crimson;
    }
……
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    <p>1</p>
    <p id="change3d">2</p>
    <p>3</p>
……
```

左移 100px，上移 30px，z 轴方向没有移动，浏览效果如图 5.149 左侧部分。

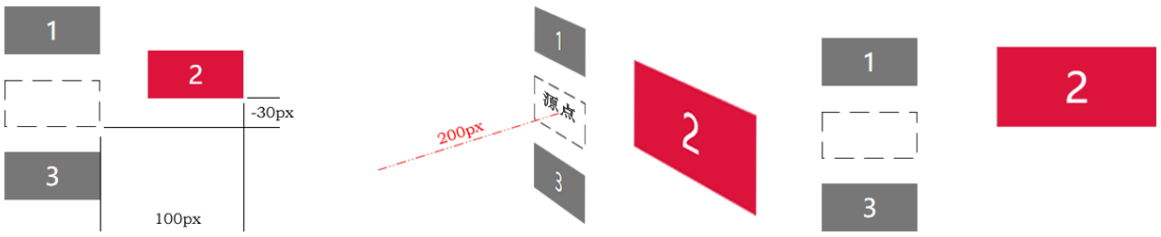


图 5.148 translate3d()函数用法举例

为了能在 z 轴上移动，我们需要增加 perspective()函数，变换的代码如下：

```
transform: perspective(200px) translate3d(100px, -30px, 80px);
```

浏览效果见图 5.152 右一，上图中间部分是视距和观察者位置示意图，观察者位置在 x\y 坐标平面上的投影仍然在元素变换前的中心，所以随着视距的改变，不但元素大小会改变，而且元素的位置也会改变。视距越近，看到的元素越大。其他三个移动函数 translateX()、translateY 和 translateZ()分别沿着 x 轴、y 轴和 z 轴移动，但是 translateZ()需要结合 perspective()才有效果。

实操练习

在 3d 空间中完成图 5.149 所示的变换。

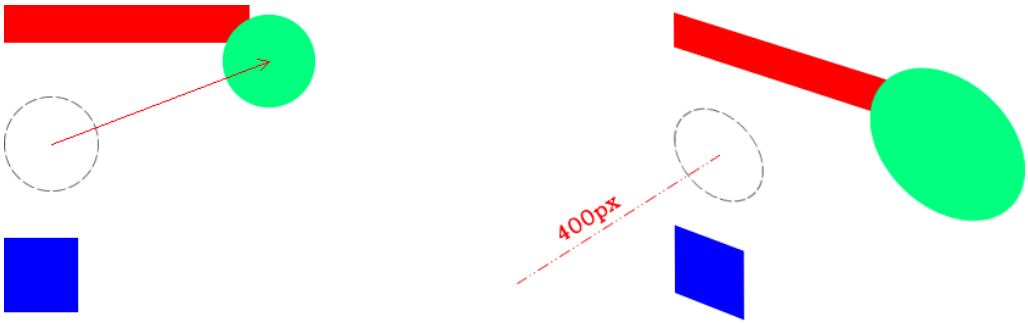


图 5.149 3D 移动实验

函数 scale3d(x, y, z)实现缩放，参数 x, y, z 都为实数，分别代表 x 坐标、y 坐标和 z 坐标上的缩放比率。当为负数时反射为一个点，当大于 0 小于 1 时缩小，等于 1 时不变，大于 1 时放大。

我们修改上面的例子的变换代码如下：

```
transform: perspective(200px) translate3d(100px, -30px, 80px) scale3d(0.7, 1.5,
 0.2);
```

由于元素没有厚度，所以尽管在 z 轴上缩小比率为 0.2，但是没有效果。最终看到的效果如图 5.150。

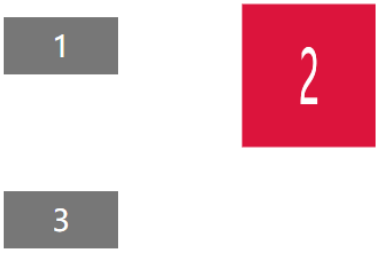


图 5.150 scale3d()函数用法举例

其他三个函数 scaleX()、scaleY()和 scaleZ()只是在相应的坐标方向上缩放。

函数 rotate3d(x, y, z, a)表示以过(0, 0, 0)和(x, y, z)两点的线为转轴旋转 a 角度，如果参数 a 为正数表示顺时针旋转，否则表示逆时针旋转。参考图 5.151，红色虚线即为转轴。x, y, z 的取值范围是 [0, 1]。注意：(0, 0, 0)是元素的变换源点，一般是元素的中心，除非用 perspective-origin 重新定义过。

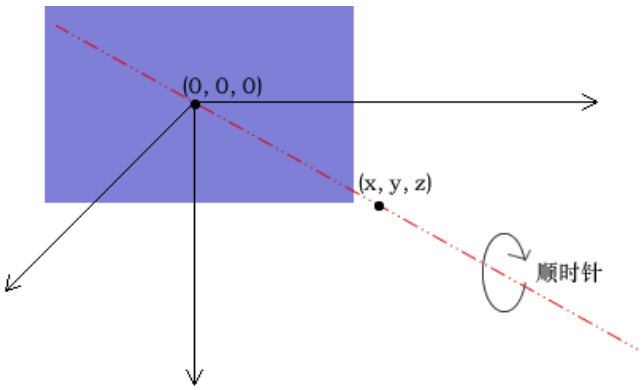


图 5.151 rotate3d()函数说明

看例子：

```
……
<head>
    <style>
        p {
            text-align: center;
            line-height: 80px;
            font-size: 60px;
            color: white;

            height: 80px;
            width: 150px;
            background-color: black;
            margin-left: 100px;

            transform: perspective(200px) rotate3d(1, 0, 0,60deg);
        }
    </style>
</head>
……
<body>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    <hr>
    <p>上</p>
    <hr>
</body>
```

分别修改旋转参数，可以看到如图 152 所示的各种效果。

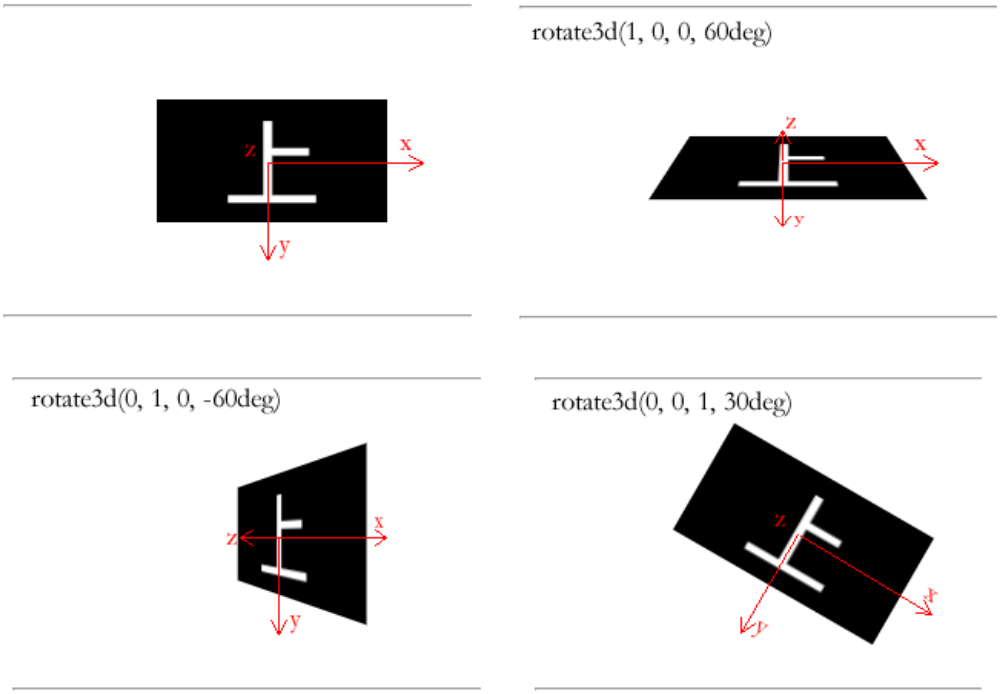


图 5.152 rotate3d()函数举例

其他三个旋转函数 rotateX()、rotateY()和 rotateZ()分别在 x、y、z 轴上旋转。再看一个例子：

```
        #parent{
            width: 120px;
            height: 200px;
            background-color: yellow;
            transform: rotateY(60deg);
            transform-style: preserve-3d;
        }
        #child{
            width: 60px;
            height: 160px;
            margin-left: 30px;
            background-color: red;
            transform: rotateX(130deg);
        }
……
<div id="parent">
    <div id="child"></div>
</div>
```

浏览的效果如图 5.153 右侧部分，如果把父元素的 transform-style 属性的值改为 flat，就可以看到图 5.153 左侧部分。

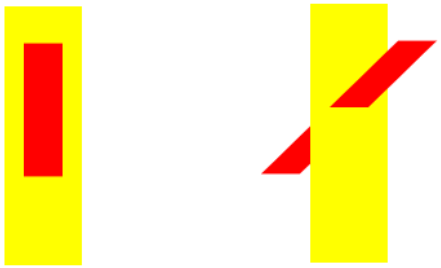


图 5.153 rotateX()、rotateY()用法举例

实操练习

实现如图 5.154 所示的十字架效果（提醒：父元素做变换 rotateY(45deg) rotateX(45deg)）。



图 5.154 倾斜的十字架

再来看一个绘制立方体的例子：

```
<!doctype html>
<html>

<head>
    <meta charset="utf-8">
    <title>立方体</title>
    <style>
        .box {
            position: relative;
            width: 200px;
            height: 200px;
            margin: 200px auto;
            transform-style: preserve-3d;
            transform: rotateX(-30deg) rotateY(-60deg);
        }

        .face {
            position: absolute;
            left: 0;
            top: 0;
            display: inline-block;
            width: 200px;
            height: 200px;
            border: solid 1px black;
        }

        .face:nth-child(1) {
            transform: translateY(-100px) rotateX(90deg) ;
            background-color: rgba(255,0,0,0.5);
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
        }

        .face:nth-child(2) {
            transform: translateY(100px) rotateX(90deg) ;
            background-color: rgba(0,255,0,0.5);
        }
        .face:nth-child(3) {
            transform: translateX(100px) rotateY(90deg) ;
            background-color: rgba(0,0,255,0.5);
        }

        .face:nth-child(4) {
            transform: translateX(-100px) rotateY(90deg);
            background-color: rgba(255,255,0,0.5);
        }

        .face:nth-child(5) {
            transform: translateZ(100px);
            background-color: rgba(255,0,255,0.5);
        }

        .face:nth-child(6) {
            transform: translateZ(-100px);
            background-color: rgba(0,255,255,0.5);
        }
    </style>
</head>

<body>
    <div class="box">
        <span class="face"></span>
        <span class="face"></span>
        <span class="face"></span>
        <span class="face"></span>
        <span class="face"></span>
        <span class="face"></span>
    </div>
</body>

</html>
```

浏览的效果如图 5.155 所示。

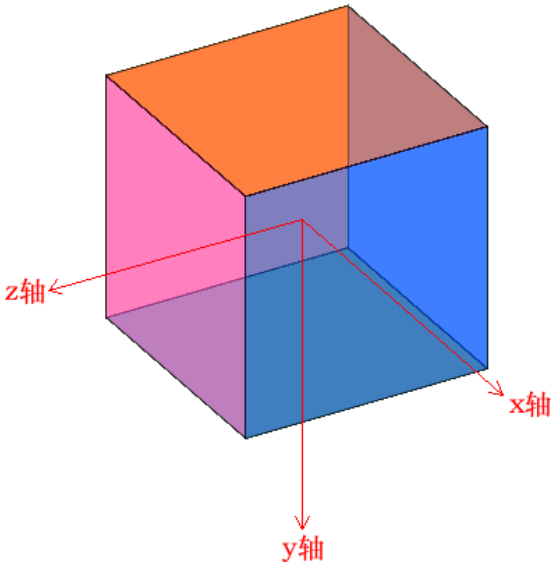
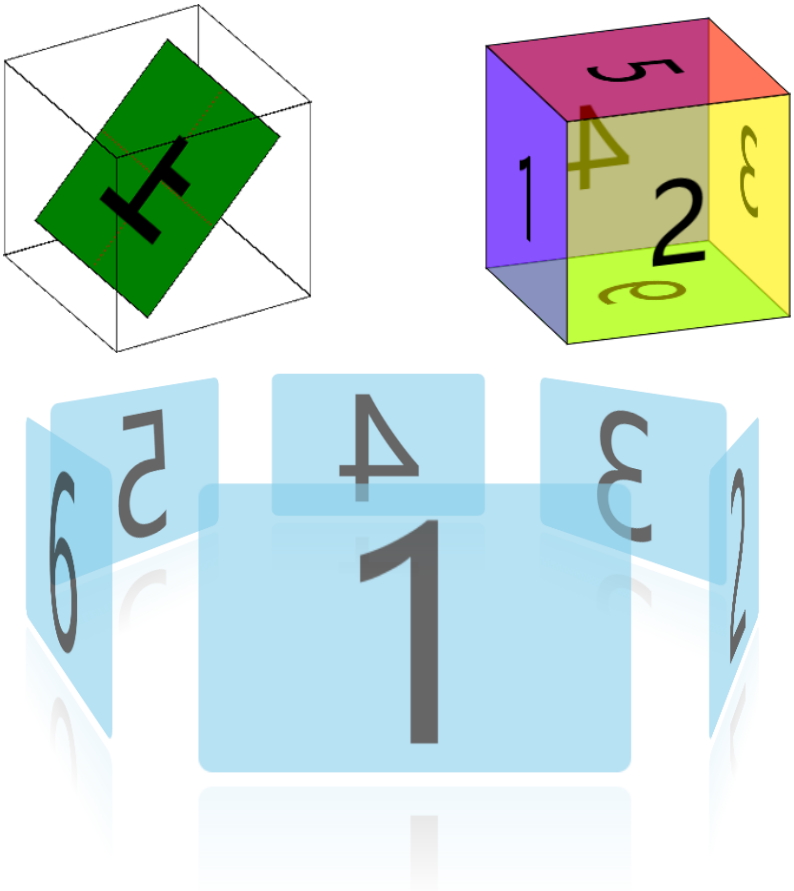


图 5.155 3D 变换实现立方体效果

实操练习

实现如图 5.156 所示的各个效果。



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

图 5.156 3D 变换实验

矩阵变换函数 matrix3d(a_1, b_1, c_1, d_1, a_2, b_2, c_2, d_2, a_3, b_3, c_3, d_3, a_4, b_4, c_4, d_4)有 16 个参数，先列后行分别对应 4╳4 变换矩阵中的元素。假设变换前的坐标(x_0, y_0, z_0)对应变换后的坐标为(x_1, y_1, z_1)，有公式：

$$\begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = \begin{bmatrix} a_1 & a_2 & a_3 & a_4 \\ b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix} \times \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} = \begin{bmatrix} a_1x_0 + a_2y_0 + a_3z_0 + a_4 \\ b_1x_0 + b_2y_0 + b_3z_0 + b_4 \\ c_1x_0 + c_2y_0 + c_3z_0 + c_4 \\ d_1x_0 + d_2y_0 + d_3z_0 + d_4 \end{bmatrix}$$

因此移动为 maxtrix(1,0,0,0,0,1,0,0,0,0,1,0,a_4,b_4,c_4,1)，对应的矩阵为：

$$\begin{bmatrix} 1 & 0 & 0 & a_4 \\ 0 & 1 & 0 & b_4 \\ 0 & 0 & 1 & c_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

因此缩放为 maxtrix(a_1,0,0,0,0,b_2,0,0,0,0,c_3,0,0,0,0,1)，对应的矩阵为：

$$\begin{bmatrix} a_1 & 0 & 0 & 0 \\ 0 & b_2 & 0 & 0 \\ 0 & 0 & c_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

5.12 过渡

在发生一个事件时，以动画效果完成样式的过渡，比如元素的原始宽度为 100px，鼠标停靠其上时宽度改为 300px。如果对宽度属性定义了过渡，此时就能看到元素的宽度逐渐加宽到 300px。其实过渡就是一段动画，给出了开始和结束的关键帧，中间的帧自动补齐。请参考图 5.157。

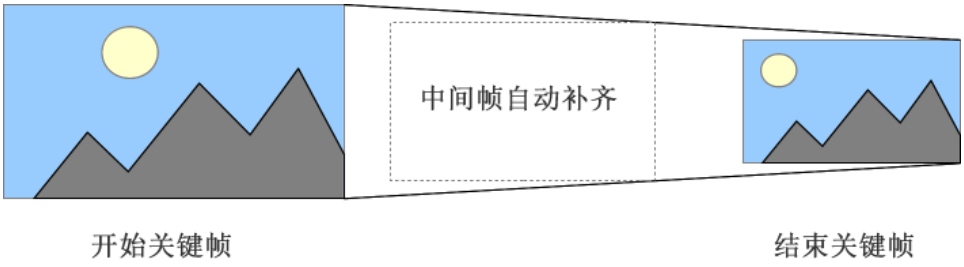


图 5.157 过渡说明

过渡样式采用 transition，它的使用语法如图 5.158。

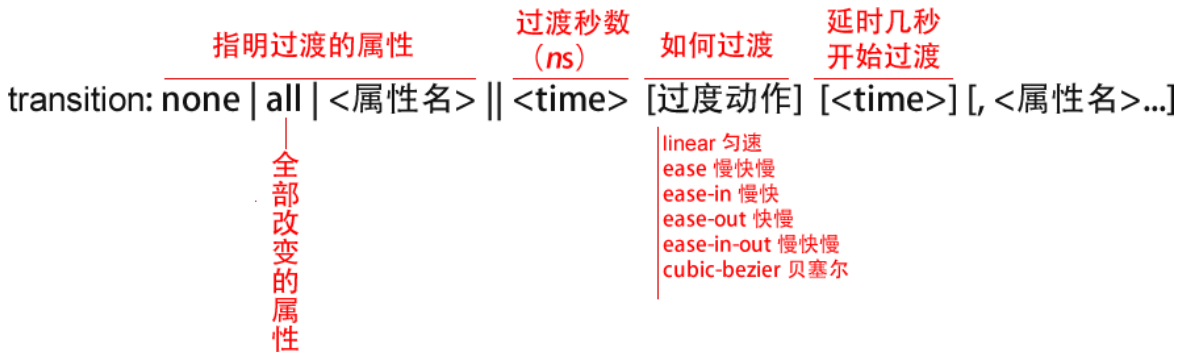


图 5.158 transition 属性说明

transition 对应几个分属性:transition-property, transition-duration, transition-timing-function 和 transition-delay，分别定义需要过渡的样式属性、过渡延续的时间长度、过渡的动作和过渡开始前端延时秒数。时间允许使用小数，比如半秒钟 0.5s。过渡动作就是过渡产生的效果，比如 ease-in-out，即慢快慢，就是刚开始时慢，中途加快，最后又慢下来。transition 属性的默认值为：

```
transition: all 0s ease 0s
```

由于过渡时长为 0 秒，所以默认是没有过渡效果的。下面看几个例子：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>过渡</title>
    <style>
        div {
            width: 100px;
            height: 100px;
            background-color: red;
            transition: width 3s;
        }

        div:hover {
            width: 200px;
            background-color: blue;
        }
    </style>
</head>

<body>
    <div></div>
</body>

</html>
```

当鼠标停靠在元素上时，元素的宽度和背景色都发生变化，但是由于过渡属性只指明了 width，所以最终会发现，只有元素的宽度慢慢变化（3 秒内完成），颜色瞬间改变，无过渡效果。为了让颜色的改变也有过渡效果，把上面例中的样式代码改为：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
transition: all 3s;
```

或者：

```
transition: width 3s, background-color 3s;
```

再看下面的例子：

```
div {
     width: 100px;
     height: 100px;
     background-color: red;
     transition: transform 5s ease-in-out 2s;
 }

div:hover {
      transform: rotate(360deg);
}
```

鼠标在元素上停靠 2 秒后开始旋转（5 秒内完成）。其他常用的过渡样式还有：坐标、圆角、阴影、字号、padding、margin、透明度、颜色等。

过渡的分属性都是多值，最好是各个分属性的值的个数相同，例如：

```
transition-property: opacity, left, top, height;
transition-duration: 3s, 5s, 2s, 4s;
transition-timing-function: ease-in-out;
```

其中 transition-timing-function 只有一个值，等于四个属性的过渡遵循一样的效果。

随书资源 chapter-5\transition.html 是一个很实用的例子。浏览的效果如图 5.159 所示，点击右上角的“？”，内容从右侧展开或缩回。



图 5.159 transition 综合例子

实操练习

点击“滚动”按钮，球从左边滚到右边（提醒：点击按钮时修改样式属性 transform:rotate(360deg) 和 left）。参考图 5.160。



图 5.160 过渡样式实验

5.13 动画

动画不像过渡，不需要事件来触发，也没有两个关键帧的限制。我们采用 animation 样式属性和@keyframes 规则完成动画效果。animation 是一个合并样式属性，它对应几个分属性：animation-duration(以秒为单位的动画时长)、animation-delay(以秒为单位的动画开始时间)、animation-timing-function、animation-iteration-count、animation-direction、animation-fill-mode 和 animation-play-state。

animation 的语法：

```
animation: { <time> || <timing-function> || <time> || <animation-iteration-count>
|| <animation-direction> || <animation-fill-mode> || <animation-play-state> ||
[ none | <keyframes-name> ] } +
```

第一个<time>为动画时长（对应 animation-duration 分属性）；第二个<time>为动画开始时间（对应 animation-delay 分属性）；动画动作<timing-function>取值参考 transition 的过渡动作（对应 animation-timing-function 分属性）；动画循环次数<animation-iteration-count>（可为整数、小数和 infinite）（对应 animation-iteration-count 分属性），infinite 表示无限循环播放；动画方向<animation-direction>（对应 animation-direction 分属性）取值为：normal（默认值，每次循环都是从头开始）、reverse（每次循环都是从尾开始）、alternate（头->尾->头->…）、alternate-reverse（尾->头->尾->…）；<animation-fill-mode>定义动画开始前和结束后的元素样式，取值 none（默认值，样式不变）、forwards（动画结束后保留最后关键帧的样式）、backwards（开始前的元素设置第一个关键帧的样式）、both（结合 forwards 和 backwards 的作用）;<animation-play-state>控制动画暂停/播放，可取值 running 播放、paused 暂停；<keyframes-name>指定@keyframes 规则的名字。如：

```
div {
    width: 100px;
    background-color: yellow;
    animation: 5s ease-in-out 2s 3 alternate both myGun;
}
```

@keyframes 规则定义动画中的关键帧，关键帧之间形成过渡的动画效果，它的语法如下：

```
@keyframes  <名字>  {
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    from | x% {
        样式表
    }
    [y% {
    样式表
}]
    to | z% {
        样式表
    }
}
```

其中的<名字>要与 animation 样式中的一致。其他参数参考下图，假如动画总时长为 L，现在要在时长 k 处插入一个关键帧，那么有公式：$k \div L \times 100 = n\%$。用 from 或 0%表示第一帧，用 to 或 100%表示最后一帧，显然 50%表示播放到一半的帧，每隔关键帧都可以单独设置样式表，这样相邻关键帧的样式就会产生过渡。示意图参考图 5.161。

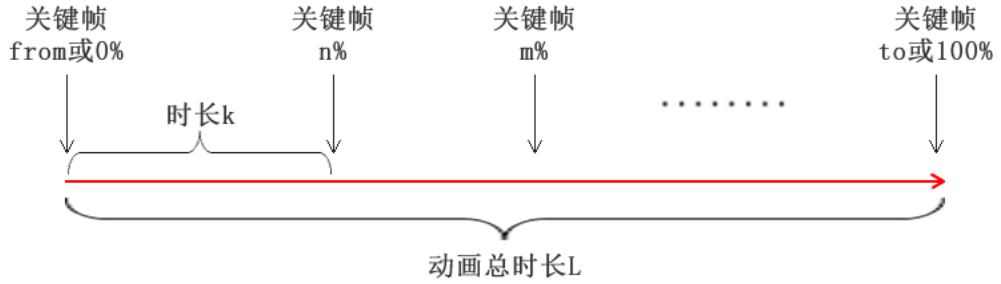


图 5.161 @keyframes 用法说明

如下面的例子：

```
@keyframes myGun {
    from {
        background-color: red;
        width: 200px;
    }
    40% {
        background-color: blue;
        width: 150px;
        left: 300px;
    }
    100% {
        background-color: black;
        width: 500px;
        top: 500px;
    }
}
```

本例中一共有三个关键帧：开始、结尾和播放到 40%的位置。允许插入任意多个关键帧，相邻的关键帧之间就会发生过渡效果。

5.14 滚动条

“CSS3 样式速查表·滚动条”中列出了专门用于滚动条的样式。

1、overscroll-behavior* —— 定义浏览器滚动到内容的边界时的表现

不断滚动内容到达边界时，由 overscroll-behavior*系列属性告诉浏览器如何操作，比如弹回、刷新页面、移动父元素的滚动条等。overscroll-behavior 是 overscroll-behavior-x 和 overscroll-behavior-y 属性的复合属性，单值时表明水平和垂直两个方向遵循相同的行为，双值时表示前一个值定义水平方向的行为，后一个值定义垂直方向的行为。overscroll-behavior-x 属性定义 x 轴（一般为水平方向）上的行为，它的语法如下：

```
overscroll-behavior-x: auto | contain | none
```

auto 是默认值，由浏览器自行决定。contain 不会触发周边元素（包括父元素）的滚动事件而保留本元素的默认滚动行为，如反弹、刷新页面等。none 禁止本元素的默认滚动行为且不触发周边元素的滚动事件。overscroll-behavior-y 属性定义 y 轴上的行为，语法与 overscroll-behavior-x 一样。看例子：

```
p {
    block-size: 100px;
    border: solid;
    font-size: 18px;
    overflow: auto;
    overscroll-behavior: auto;
}

div {
    inline-size: 330px;
    block-size: 250px;
    overflow: auto;
    border: thin solid;
}
……
<div>
   <p>真是洗涤灵魂的对白！生活中的人啦，被各种藩篱分割，财富的藩篱、权力的藩篱、知识的藩篱、更有那铜墙铁壁般的观念的藩篱……，处于不同藩篱中的人对话压根尿不到一个壶里！
   </p>
   <img src="shackles.jpg">
</div>
```

浏览的效果如图 5.162 所示，鼠标进入文字部分然后往下滚动，会发现滚到底后继续下滚，这时就会滚动外面的滚动条。修改代码中的 auto 为 contain 或 none，再试试效果如何？

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

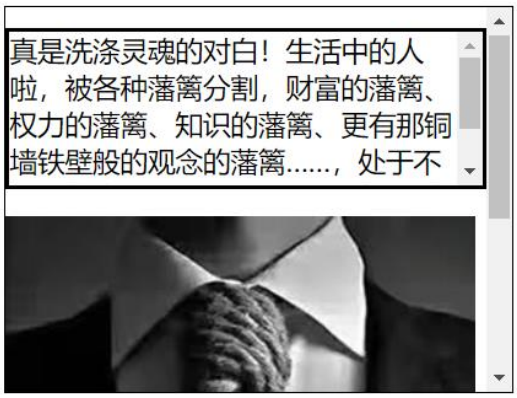


图 5.162 overscroll-behavior 属性用法举例

overscroll-behavior-block 和 overscroll-behavior-inline 属性是逻辑样式，前者代表行进方式上滚到底如何处理，后者表示书写方向上滚到底又如何处理。取值与 overscroll-behavior-x 属性一样。

2、scroll-snap-*/scroll-padding*/scroll-margin* — 定义滚动条的捕捉特性

带滚动条的元素相当于一个放大镜，移动滚动条就相当于用放大镜在地图上移动。假如你的桌子上平铺了很多照片，放大镜每次最多能看到一幅照片，你希望每次移动放大镜都能准确框住一张相片，这样你停止移动放大镜就能仔细观察整张相片，参考示意图 5.163。

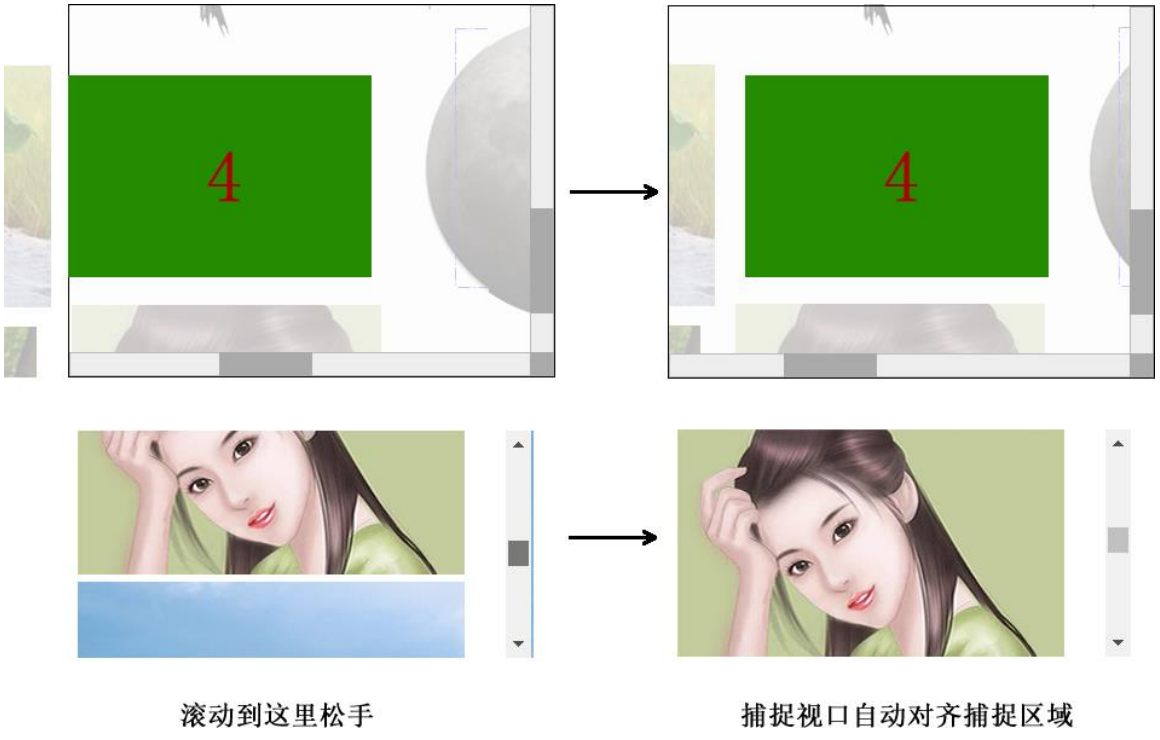


图 5.163 滚动条捕捉说明

放大镜就是捕捉视口，而一幅照片就是捕捉区域。定义捕捉视口和捕捉区域，以及捕捉视口和捕捉区域如何对齐，这就是 scroll-snap-*/scroll-padding*/scroll-margin*属性的功能，参考图 1.164。

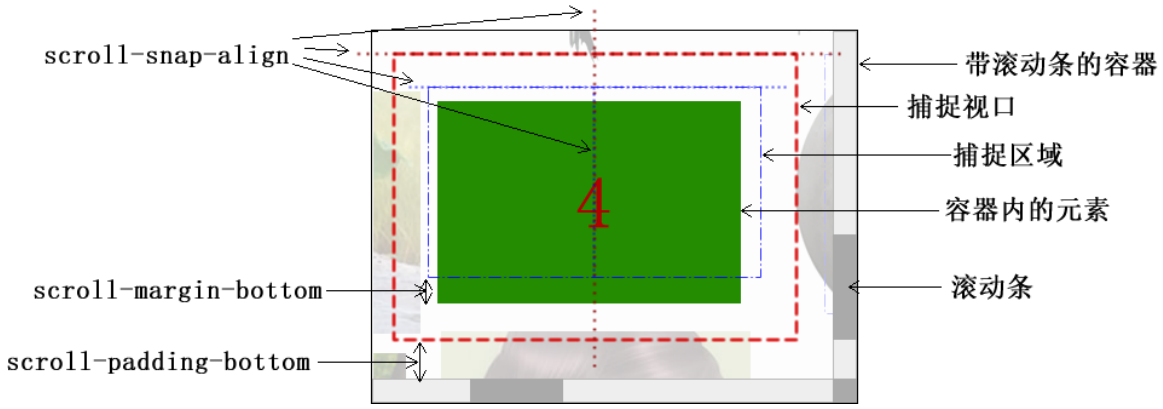


图 5.164 滚动条各个属性说明

scroll-snap-type 和 scroll-padding*定义捕捉视口（放大镜），即应用在带滚动条的容器元素上，而 scroll-snap-align、scroll-snap-stop 和 scroll-margin*属性定义捕捉区域（照片）和对齐方式，应用在子元素上。

scroll-snap-type 属性设置捕捉视口和捕捉区域对齐的严格程度，它的使用语法如下：

```
scroll-snap-type: none | [ x | y | block | inline | both ] [ mandatory | proximity ]?
```

none 无捕捉效果，x、y、block、inline 和 both 五个值至多只能选一个，指名哪个方向要捕捉对齐，mandatory 和 proximity 只能选一个或者都不选，mandatory 严格对齐，而 proximity 大约对齐。看例子：

```
#scroll-box {
    scroll-snap-type: x mandatory;
    inline-size: 250px;
    block-size: 250px;
    overflow-x: scroll;
    display: flex;
    box-sizing: border-box;
    border: 1px solid #000;
}

#scroll-box>div {
    scroll-snap-align: start;
    flex: 0 0 250px;
    inline-size: 250px;
    font-size: 40px;
    display: flex;
    align-items: center;
    justify-content: center;
    background: #EEE;
}

#scroll-box>div:nth-child(even) {
    background-color: black;
    color: white;
}
……
<div id="scroll-box">
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    <div>1</div><div>2</div><div>3</div><div>4</div>
</div>
```

即在滚动容器上施加样式 scroll-snap-type: x mandatory，在子元素上施加样式 scroll-snap-align: start 即可。浏览的效果如图 5.165 左侧。

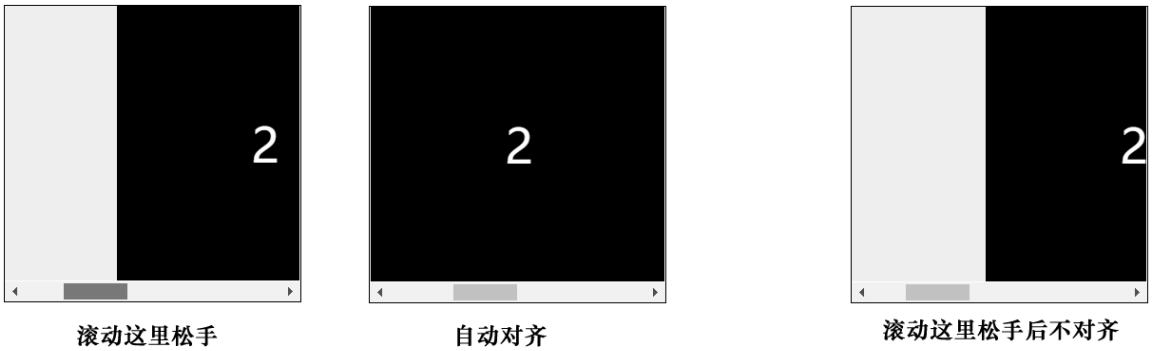


图 5.165 捕捉视口与捕捉区域对齐举例

把代码中的 mandatory 改为 proximity，效果见图右侧。mandatory 规定哪个子元素出现在捕捉视口的面积大，松手后就完全对齐它，而 proximity 规定捕捉视口和捕捉区域的对齐点相差较远，松手后就不对齐。

捕捉视口为滚动容器中的一个矩形，这个矩形到滚动容器的边界由 scroll-padding*属性定义，scroll-padding*属性的用法参考“§5.6.3 元素的内边距”小节中的 padding*属性。

scroll-snap-align 属性定义捕捉视口与捕捉区域的对齐方式，可能的值有四个：none（默认值，表示由浏览器自行决定）、start（起始位置对齐，水平滚动就是左侧对齐）、end（终止位置对齐）、center（中心位置对齐）。scroll-snap-align 属性还可以取两个值，第一个值定义行进方向的对齐方式，第二个值定义书写方向的对齐方式，如 scroll-snap-align: start center。在上面的例子中，把代码 flex: 0 0 250px 改为 flex: 0 0 200px 后看看浏览效果如何？然后增加样式 scroll-snap-align: center，在看看效果如何？

scroll-snap-stop 属性定义滚动时是否允许越过一些可能的捕捉点，比如对于一些重要的子元素，希望用户一次滚动不要匆匆越过去，我们可以在此子元素上设置 scroll-snap-stop: aways。不过此属性目前支持的浏览器不多。

scroll-margin*属性定义捕捉区域到子元素边界的距离，用法与 margin*属性一样，请参考“§5.6.2 元素的外边距”。

5.16 渐变

线性渐变函数是三个渐变函数中的一个，另外两个分别是径向渐变和圆锥渐变函数。线型渐变就是多种颜色沿着直线发生过渡，径向渐变就是多种颜色呈放射状由里向外发生过渡，而圆锥渐变就是多种颜色顺时针围绕一点发生过渡。参考示意图 5.166。

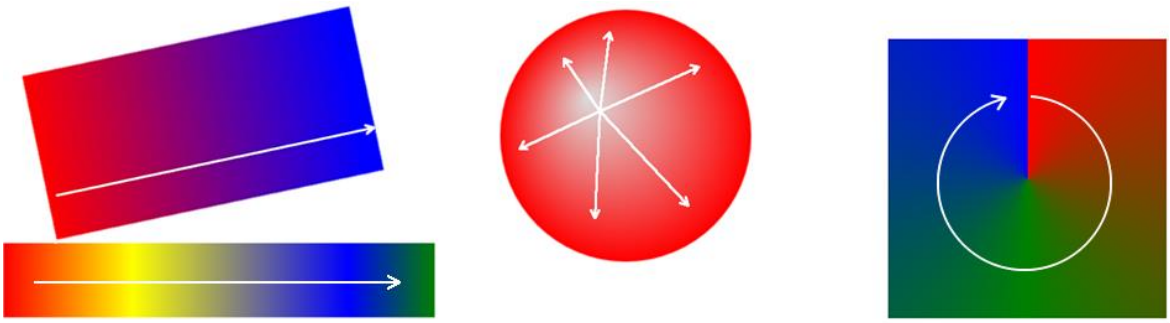


图 5.166 三种渐变示意图

渐变作为一种图形，不能单独渲染呈现，它只能作为元素的背景图或者边框图形才能显示出来。

5.16.1 线性浅变

线性渐变就是若干种颜色沿着直线方向发生过渡，采用函数 linear-gradient()定义线性渐变，它的语法是：

```
linear-gradient( [ <angle> | to <side-or-corner>,] <color-stop> [ [,<color-hint>],<color-stop>]+)
```

定义渐变线　　定义颜色点

我们结合图 5.112 来理解这个语法。渐变线过元素的中心，可用两种方法之一确定方向：(1)与垂直向上的线条夹角为 angle，(2)采用朝向参数 to <side-or-corner>。参数<angle>的单位为 deg(度)，比如 90 度角就写成 90deg，参数<side-or-corner>=[left|right]||[top|bottom]，因此有八个朝向：left（朝左）、right（朝右）、top（朝上）、left top（朝左上角）、……，具体参考图 5.167。如果不定义渐变线，默认就是朝下，即 180deg。

从离渐变线最近的两个角引出两根垂直线，与渐变线的交叉点就是渐变的起点和终点，各个颜色点就定义在从起点到终点的线段上。

参与过度的颜色至少两个，一个 color-stop 定义一个颜色，其语法是：

```
color-stop = <color>[ <length>|<percentage>]
```

<color>指定颜色，如 red、#000、rgba(255,0,255,0.8)、hsl(120, 100%, 25%)，<length>或<percentage>定义颜色在渐变线段上的位置，即离起点的距离，可以用两种方式表示距离，一是单位为 px 的绝对长度值，二是百分比（即离起点的距离除以线段的长度再乘以 100%），一般最好采用百分比，显然 0%代表渐变起点，100%位于渐变终点，50%位于起点与终点的中间位置。如果省略了距离，那参与渐变的颜色之间的间距相等，且第一个颜色在起点，最后一个颜色在终点。

<color-hint>定义相邻两种颜色的中间色（相加再除以 2 得到的颜色）离起点的距离，采用长度单位或者百分数均可。如果省略 color-hint 参数，中间色就在相邻颜色的中间位置。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

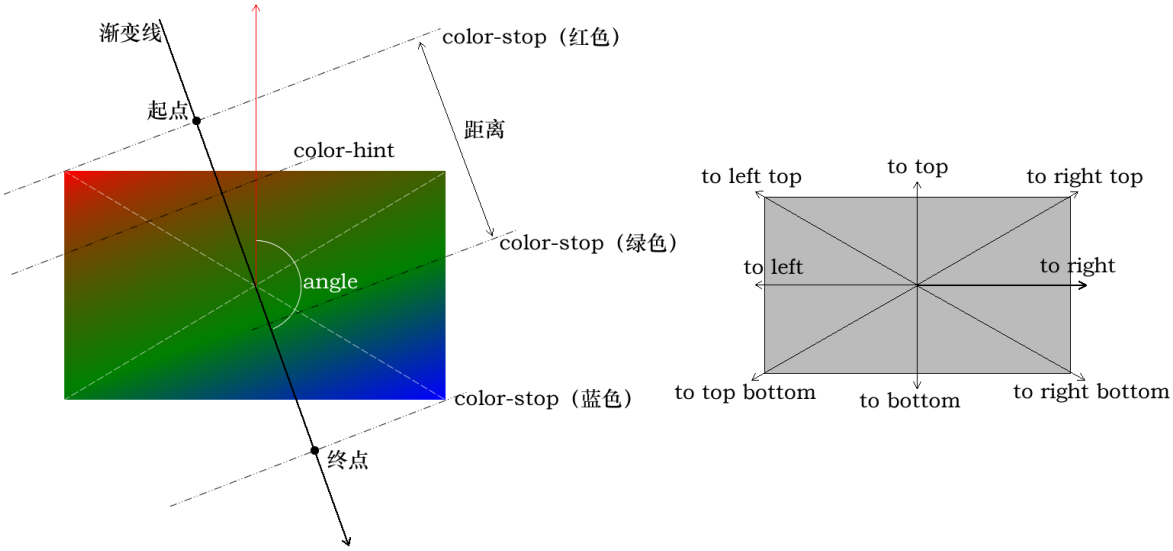


图 5.167 线性渐变语法说明

看例子：

```
<head>
    <style>
        div {
            width: 500px;
            height:300px;
            background-image: linear-gradient(120deg, red, blue);
        }
    </style>
</head>

<body>
    <div></div>
</body>
```

即红色位于起点，蓝色位于终点，浏览的效果见图 5.168 右侧部分。

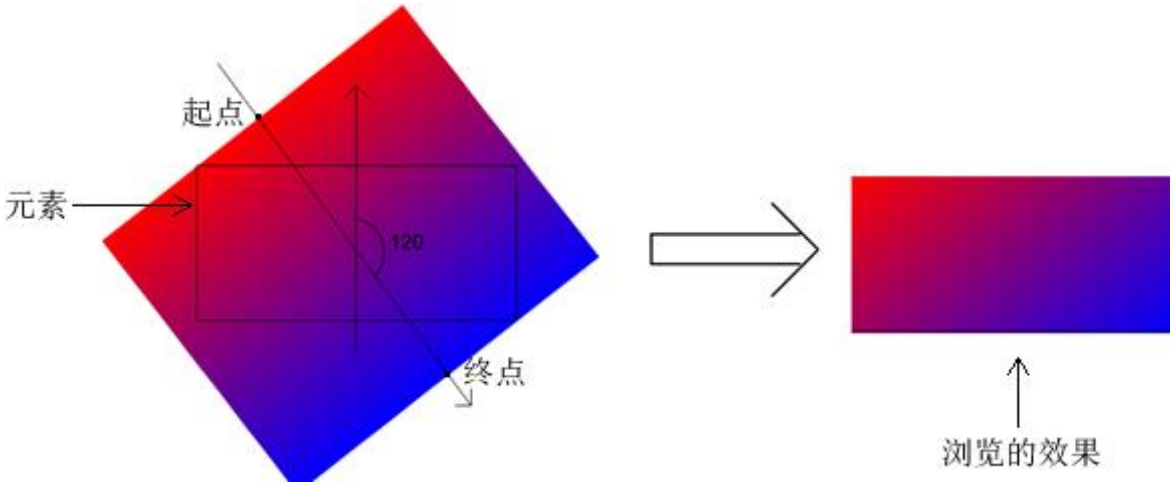


图 5.168 线性渐变用法举例之一

再看例子：

```
background-image: linear-gradient(to right, red 0%, blue 80%, yellow 100%);
```

即红色在起点，黄色在终点，而蓝色的位置与起点的距离占整个渐变线段长度的 80%，浏览的效果如图 5.169 左侧部分。

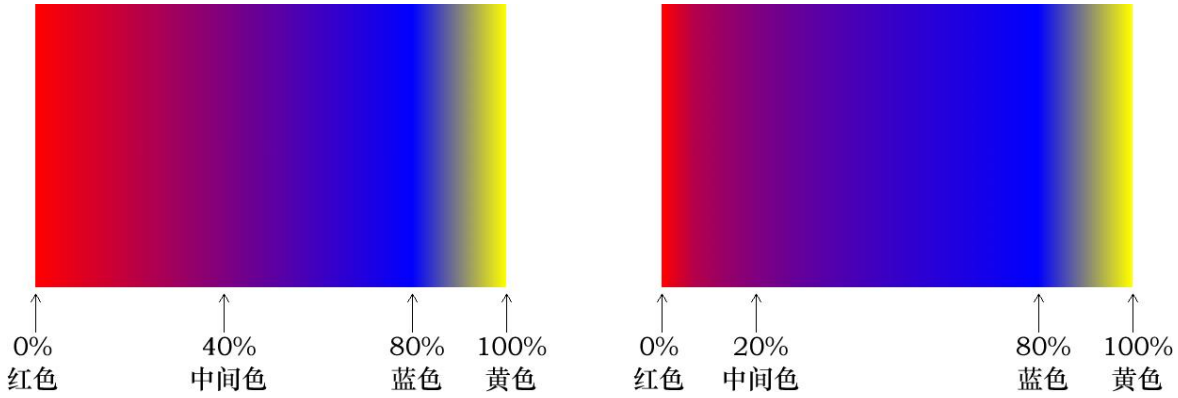


图 5.169 线性渐变用法举例之二

而下面的代码浏览的效果见图 5.169 右侧部分。

```
background-image: linear-gradient(to right, red 0%, 20%, blue 80%, yellow 100%);
```

如果第一个颜色不在起点上，那么从起点到第一个颜色的位置之间是纯色（即第一个颜色），类似地，如果最后一个颜色不在终点上，那么从最后一个颜色的位置到终点之间为纯色（即最后一个颜色）。如果相邻两个颜色的位置重合或者后一个颜色的位置还在前一个颜色位置的前面，那么这两个颜色紧挨在一起，不会产生过渡效果。

实操练习

分别实现如图 5.170 所示的各种效果

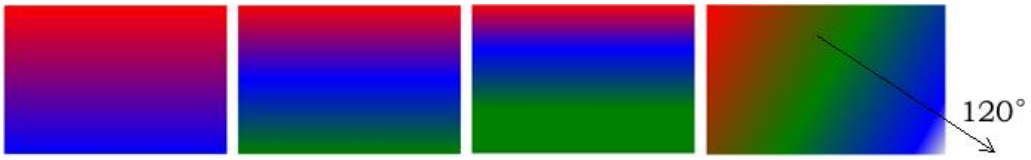


图 5.170 线性渐变实验之一

再看例子：

```
width:300px;
height: 300px;
background-image:
linear-gradient(to top,transparent 50%,rgba(200,0,0,.5) 50%,rgba(200,0,0,.5)),
linear-gradient( to left,transparent 50%,rgba(200,0,0,.5) 50%,rgba(200,0,0,.5));
background-size: 40px 40px, 40px 40px;
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

两个渐变叠加在 40px*40px 的小方框内（参见下图蓝色方框），由于 background-repeat 默认是 repeat，所以 40*40 的小图片会铺满整个元素，浏览的效果见图 5.171。

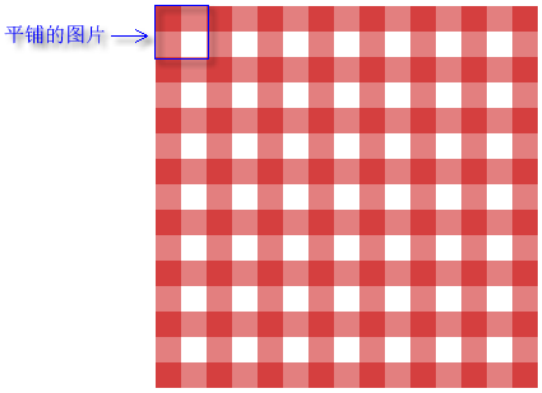


图 5.171 线性渐变用法举例之三

再看下面的例子：

```
background-size: 100px 100px;
background-image: linear-gradient(45deg, black 25%, transparent 25%,
 transparent 75%, black 75%, black),
linear-gradient(45deg, black 25%, transparent 25%, transparent 75%, black 75%,
 black);
background-position: 0 0, 50px 50px;
```

浏览的效果如图 5.172。

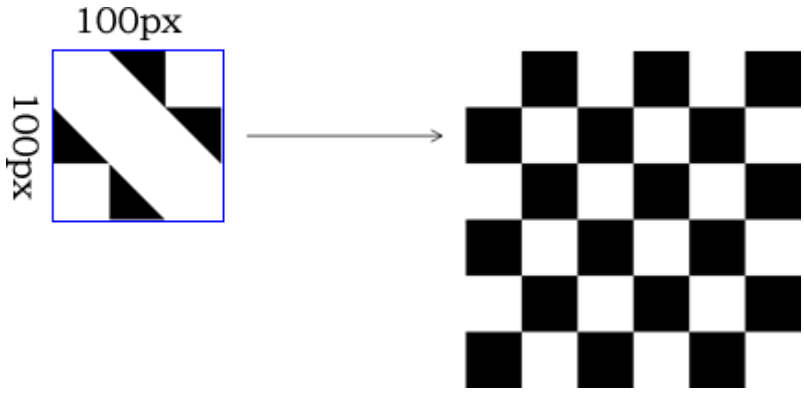
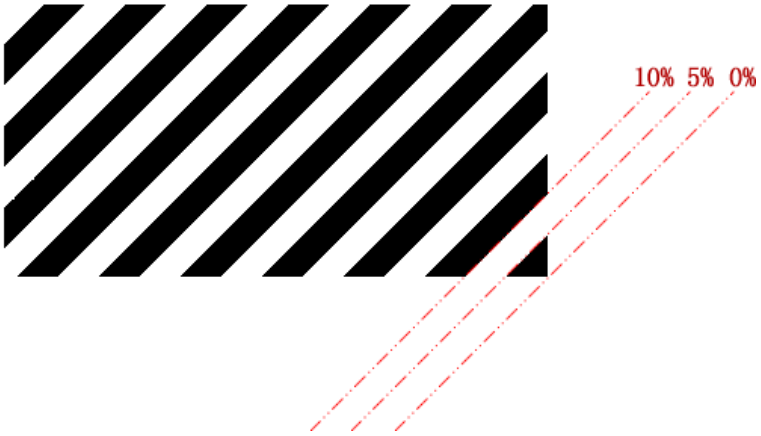


图 5.172 线性渐变用法举例之四

函数 repeating-linear-gradient()的语法与 linear-gradient()一样，但是会不断重复浅变。如：

```
background: repeating-linear-gradient(-45deg,black, black 5%,transparent 5%,
 transparent 10%);
```

前 5%为黑色，接下来的 5%为透明色，然后从 10%至 100%不断重复前 10%的浅变，浏览效果如图 5.173。



5.173 重复渐变

下面的例子产生百叶窗帘的效果：

```
background: repeating-linear-gradient(0deg,rgb(26,198,204),rgb(26,198,204) 7%,
  rgb(100,100,100) 10%);
```

浏览的效果如图 5.174。

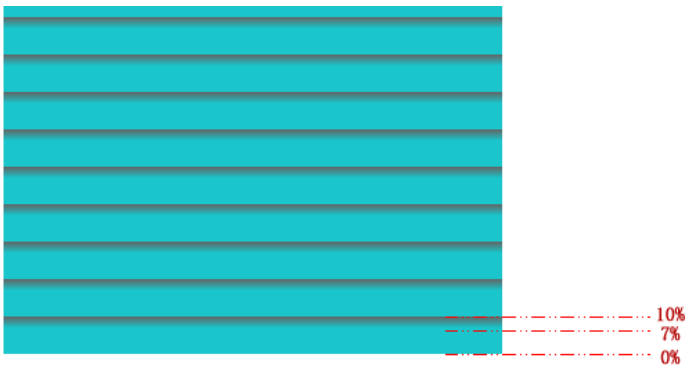


图 5.174 百叶窗

实操练习

完成图 5.175 所示的效果（提醒：border-image-source、border-image-slice）。



图 5.175 线性渐变之二

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

5.16.2 径向浅变

径向渐变就是若干种颜色从中心点沿着半径方向产生过渡效果，采用 radial-gradient()函数完成，如果是 firefox 浏览器则为-moz-radial-gradient()，它的语法参见图 5.176。

```
radial-gradient(
    [
    ① [ circle || <length> ] [ at <position> ]? , |
    ② [ ellipse || [ <length> | <percentage> ]{2} ] [ at <position> ]? , |
    ③ [ [ circle | ellipse ] || <extent-keyword> ] [at <position> ]? , |
    ④ at <position> ,
    ]?
    <color-stop> [ [,<color-hint>], <color-stop> ]+
)
```

终止线轮廓 定义起点和

颜色点 定义

图 5.176 径向渐变语法

语法中的①、②、③、④只能选一个或者都不选，<length>、<percentage>表示渐变终止轮廓线的半径，circle 表示终止轮廓线为圆形，ellipse 为椭圆形。<position>为渐变起点的坐标（对应④）或者圆和椭圆中心点的坐标（对用①、②、③），它的格式如图 5.177。

```
[
① [ left | center | right | top | bottom | <length-percentage> ] |
② [ left | center | right | <length-percentage> ] [ top | center | bottom | <length-percentage> ] |
③ [ center | [ left | right ] <length-percentage>? ] && [ center | [ top | bottom ] <length-percentage>? ]
]
```

图 5.177 <position>的格式

语法中的①、②、③只能选一个，如“left 20px”、“bottom 10%”都是选①的例子，“right 10px top 5%”是选②的例子，“center top 20%”就是选③的例子。<extent-keyword>取值见表 5.19。

表 5.3 <extent-keyword>取值

取值	描述
closest-side	终止轮廓线与容器距离渐变中心点最近的一边相切（圆形）或者至少与距离渐变中心点最近的垂直和水平边相切（椭圆）。
closest-corner	渐变的边缘形状与容器距离渐变中心点最近的一个角相交。
farthest-side	与 closest-side 相反，边缘形状与容器距离渐变中心点最远的一边相切（或最远的垂直和水平边）。
farthest-corner	终止轮廓线与容器距离渐变中心点最远的一个角相交（默认）

如果不定义起点和终止轮廓线，默认是这样：起点位于元素的中心，终止轮廓线过元素的四个角。

颜色点的定义与线性渐变一样。对于 radial-gradient()语法的进一步理解参考图 5.178。

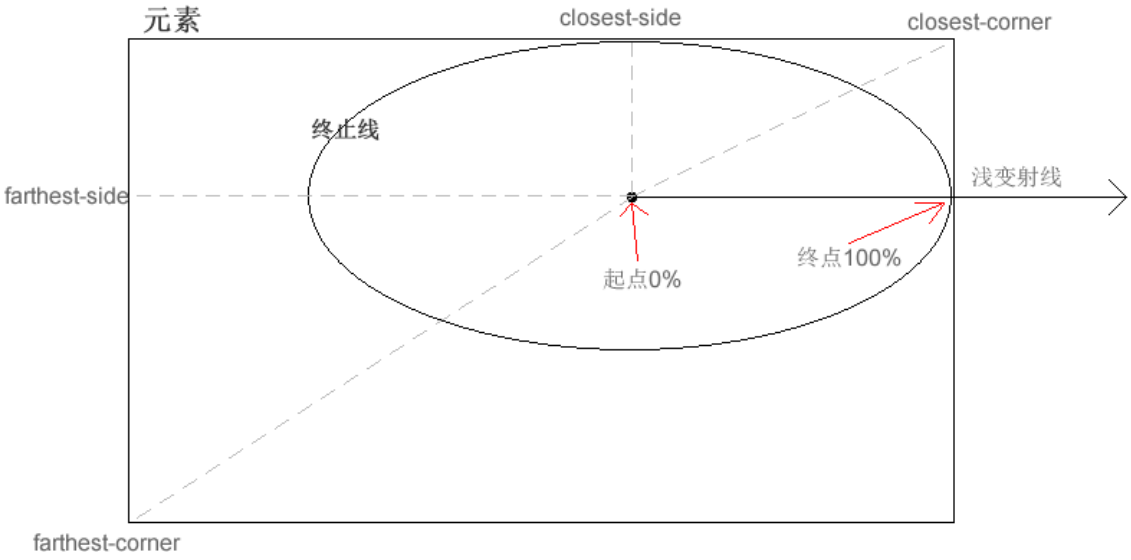


图 5.178 径向渐变语法说明

下面的例子从简单到复杂：

```
background: radial-gradient(white, red);
```

即起点在元素的中央，终止轮廓线与元素四个角相交，浏览的效果如图 5.179 中的左一。



图 5.179 径向渐变用法举例之一

```
background: radial-gradient(circle, white, red);
```

即终止轮廓线为圆，且与元素四个角相交，效果见图 5.179 中的左二。

```
background: radial-gradient(circle 50px, white, red);
```

终止轮廓线是半径为 50px 的圆，效果见图 5.170 中的左三。

```
background: radial-gradient(ellipse 50px 70px at left 30px top 40%, white, red);
```

起点在“left 30px top 40%”，终止轮廓线为椭圆型，它的水平半径为 50px，垂直半径为 70px，浏览效果见图 5.179 右二。

```
background: radial-gradient(circle closest-corner at left 50px top 30%, white,
 red);
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

起点在“left 50px top 30%”，终止轮廓线为圆形且与元素最近的角相交，浏览效果图见图 5.179 右一。

```
background: radial-gradient(circle 100px at left top, red 60%, blue 30%, green);
```

起点在左上角，终止轮廓线是半径为 100px 的圆，红色位于半径 60%的位置，蓝色位于半径上 30%的位置，绿色位于终止轮廓线上，浏览效果如图 5.180 左一。

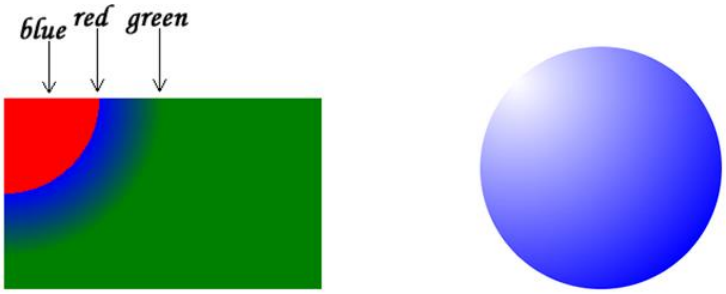


图 5.180 径向渐变用法举例之二

```
div {
     width: 200px;
     height: 200px;
     border-radius: 50%;
     background-image: radial-gradient(circle 200px at left 30px top 30px,
 white, blue);
}
```

起点在“left 30px top 30px”，终止轮廓线是半径为 200px 的圆，浏览效果见图 5.125 右一。

```
background: radial-gradient(ellipse 250px 200px at left 50% bottom -30px,
white 80%, red, orange, yellow, green, cyan, blue, purple, white 95%);
```

看到了如图 5.181 所示的彩虹。



图 5.181 彩虹

repeating-radial-gradient()语法与 radial-gradient()一样，如果最后一个颜色的位置小于 100%，那么就会不断重复之前的浅变。例子：

```
width: 300px;
height: 300px;
background-image: repeating-radial-gradient(circle, white, white 10px,
 black 50px);
```

浏览的效果如图 5.182 左一。

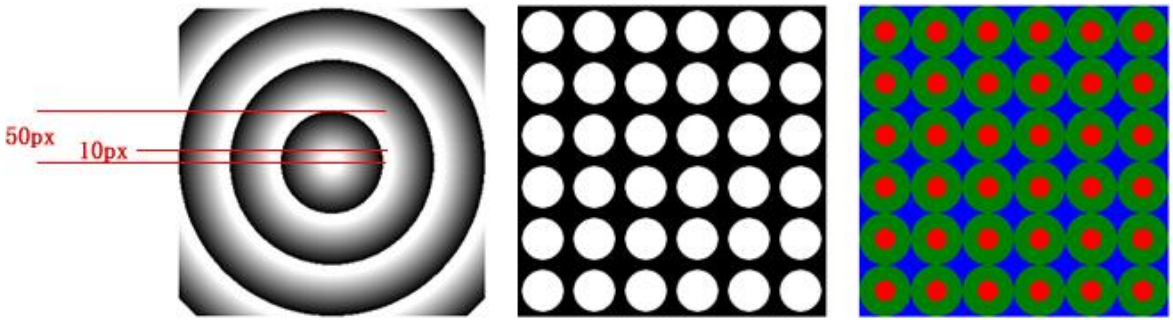


图 5.182 重复径向渐变举例

```
background-image: radial-gradient(#fff 20px, #000 20px);
background-size: 50px 50px;
```

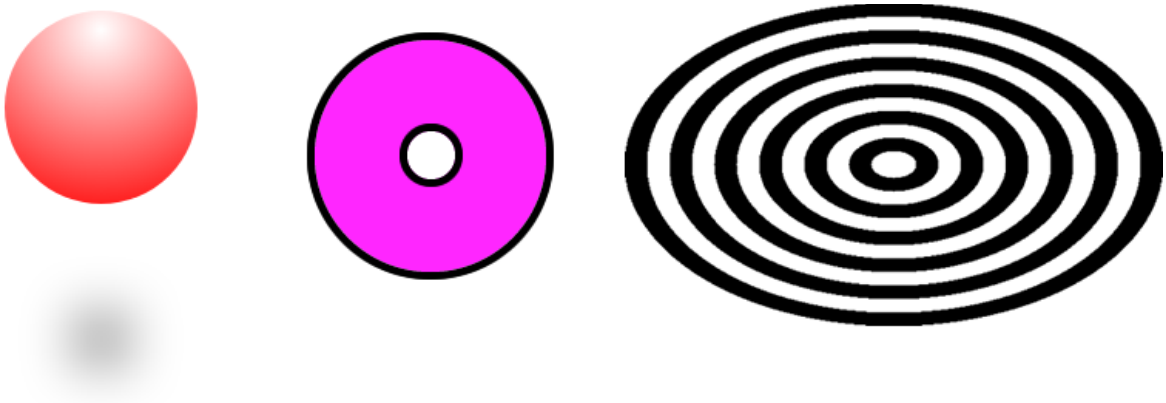
因为 background-repeat 样式的默认值就是 repeat，而在 50px*50px 见方的方位内径向渐变效果就是  ，最后水平和垂直方向都平铺它，看到的效果见图 5.182 左二。

```
background-image: radial-gradient(red 10px, green 10px, green 25px, blue 25px);
background-size: 50px 50px;
```

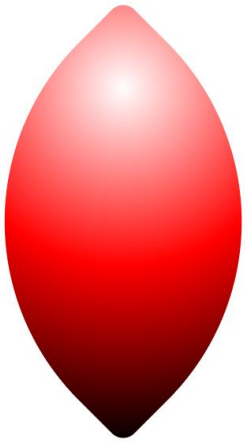
浏览效果见图 5.182 右一。

实操练习

实现如下图所示的各种效果（提醒：第四张图，径向渐变涉及三种颜色：白、红、黑）。



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



拓展综合题：

```
<head>
    <meta charset="utf-8">
    <title>光盘</title>
    <style>
        .radial-gradient {
            position: relative;
            width: 262px; height: 262px;
            border-radius: 50%;
            background:
linear-gradient(30deg, transparent 40%, rgba(42, 41, 40, .85) 40%) no-
repeat 100% 0,
linear-gradient(60deg, rgba(42, 41, 40, .85) 60%, transparent 60%) no-
repeat 0 100%,
repeating-radial-gradient(#2a2928, #2a2928 4px, #ada9a0 5px, #2a2928 6px);
            background-size: 50% 100%, 100% 50%, 100% 100%;
        }
        .radial-gradient:after {
            position: absolute;
            top: 50%; left: 50%;
            margin: -35px;
            border: solid 1px #d9a388;
            width: 68px; height: 68px;
            border-radius: 50%;
            box-shadow: 0 0 0 4px #da5b33, inset 0 0 0 27px #da5b33;
            background: #b5ac9a;
            content: '';
        }
    </style>
</head>

<body>
    <div class="radial-gradient"></div>
</body>
```

浏览效果如图 5.183。



图 5.183 光盘效果

5.16.3 圆锥渐变

圆锥渐变就是若干种颜色顺时针围绕一个中心点产生过渡效果，采用 conic-gradient()函数实现圆锥渐变，它的语法如下：

conic-gradient([from <angle>]? [at <position>]?, <color-stop> [[,<color-hint>],<color-stop>]+)

定义开始角度和中心点　　定义颜色点

结合图 5.184 来理解此语法，<angle>定义开始角度，即与垂直线的夹角，如果不定义开始角度，默认就为 0 度。<position>定义中心点的位置，具体说明参考径向渐变的同名参数，如果不定义中心点，默认就是元素的中心。

<color-stop>定义一个颜色点，它的完整的语法为<color>[[angle1] angle2]，即必须指定一种颜色，但是可选择性地指定起始角度和终止角度，角度单位既可以是 deg 也可以采用百分数。如果省略 angle1 和 angle2，那么颜色位置就在平均分配的位置上，比如一共三种颜色，第一种颜色在 0deg，第二种颜色的默认位置就是 180deg，第三种颜色在 360deg。如果提供了 angle2 而省略了 angle1，那么 angle2 就是颜色的位置，如果同时提供了 angle1 和 angle2，则从 angle1 到 angle2 之间为纯色。后一种颜色的位置允许在前一种颜色的位置之前，这样就不会产生过渡效果。不能少于两个颜色点，否则没有效果。

至于<color-hint>，请参考前两种渐变介绍，只不过单位是 deg 或者百分数。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

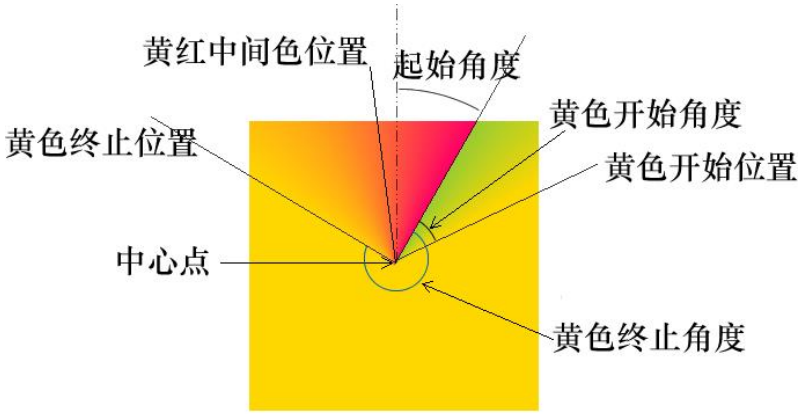


图 5.184 圆锥渐变说明

下面举的例子从简单到复杂。

```
    div {
        width: 200px;
        height: 200px;
        border-radius: 50%;
        background: conic-gradient(red, green);
    }
```

浏览的效果见图 5.184 左一。

```
    background: conic-gradient(from -45deg, red, black);
```

浏览的效果见图 5.184 左二（改变起始位置为-45 度），而下面的例子的效果见图 5.184 右二。

```
    background: conic-gradient(from -45deg, red, black, red);
```

下面的例子的效果参见图 5.184 最右侧部分（改变了中心点）。

```
    background: conic-gradient(from -45deg at 30% 20%,red, black, red);
```

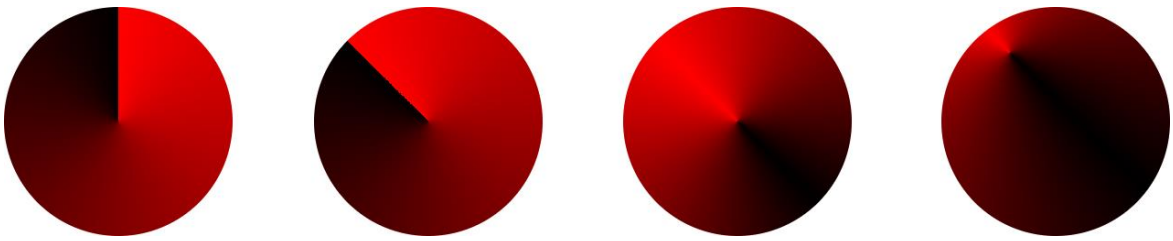


图 5.184 圆锥渐变用法举例之一

下面的例子产生圆饼效果图（见图 5.185 左一）。

```
    background: conic-gradient(yellowgreen 40%, gold 0deg 75%, #f06 0deg);
```

下面的例子产生折纸风车的效果（见图 5.185 左二）。

```
background: conic-radient(
        #222, #eee 20deg 60deg, #222 60deg, #eee 80deg 120deg,
        #222 120deg, #eee 140deg 180deg, #222 180deg, #eee 200deg 240deg,
```

```
        #222 240deg, #eee 260deg 300deg,  #222 300deg, #eee 320deg 360deg );
```

下面的例子产生国际象棋的棋盘效果（参见图 5.185 右一）。

```
background: conic-gradient(from -45deg, #000 90deg, #FFF 90deg 180deg,
     #000 180deg 270deg, #FFF 270deg 360deg);
background-size: 40px 40px;
```

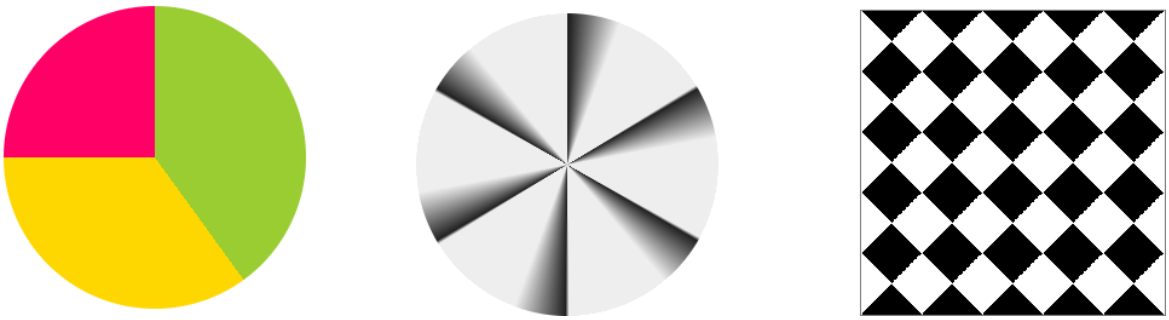


图 5.185 圆锥渐变用法举例之二

repeating-conic-gradient()函数不断重复已经定义的过渡，所以为了体验重复效果，一般我们明确定义开头部分（比如前 20%），剩下的 80%不断重复前面的 20%。看例子：

```
  background: repeating-conic-gradient(deeppink 0 15deg, yellowgreen 0 30deg);
```

浏览的效果见图 5.186 左侧部分，而下面的代码产生的效果见图 5.186 右侧部分。

```
background: repeating-conic-gradient( rgb(26,198,204),rgb(26,198,204) 7%,
  rgb(100,100,100) 10%);
```

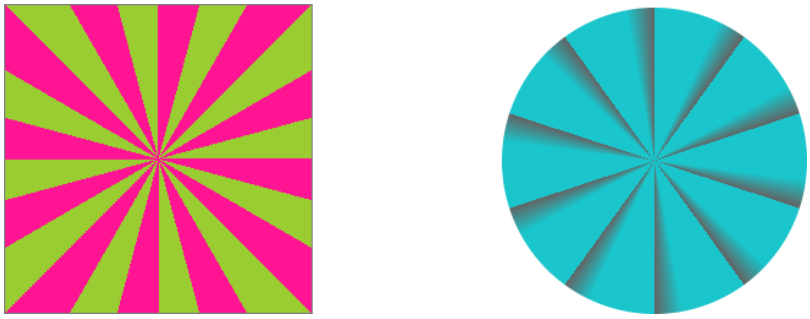


图 5.186 重复圆锥渐变举例之一

下面的例子产生舞台旋转灯光的效果（参见图 5.187 左一），关于动画请参考“§5.13 动画”小节。

```
<style>
     #out {
          width: 500px;
          height: 200px;
          overflow: hidden;
     }

     #in {
          width: 600px;
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
          height: 600px;
          margin-top: -200px;
          margin-left: -50px;
          background: conic-
gradient(#cf1639, #58928d, #46d9ba, #2ccb68, #934249, #694c38, #73ddae, #dcd16f
, #a66fdf, #fe04a8, #c236e4, #fa7808, #b6d4c7, #d938ec, #b71f92, #cf1639, #5892
8d, #46d9ba, #2ccb68, #934249, #694c38, #73ddae, #dcd16f, #a66fdf, #fe04a8, #c2
36e4, #fa7808, #b6d4c7, #d938ec);
          animation: rotate 6s ease-in-out infinite;
     }

     @keyframes rotate {
          to {
              transform: rotate(1turn);
          }
     }
</style>
……
<div id="out">
    <div id="in"></div>
</div>
```

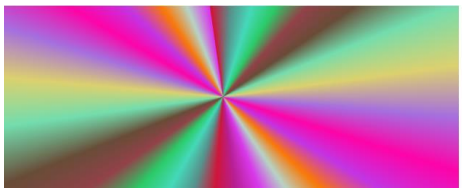
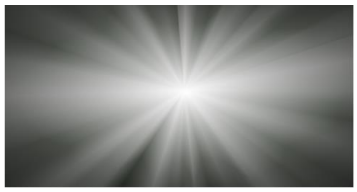




图 5.186 重复圆锥渐变举例之二

而下面的例子产生的舞台效果更逼真（参考图 5.186 中间部分）。

```
body {
    overflow: hidden;
}

div {
    width: 100vw;
    height: 100vh;
    margin: 0;
    background: radial-gradient(white, black);
}

div:before,
    div:after {
    position: absolute;
    top: 50%;
    left: 50%;
    margin: -100vmax;
    width: 200vmax;
    height: 200vmax;
    opacity: 0.5;
    mix-blend-mode: overlay;
    animation: rotate 12s ease-in-out infinite;
    content: "";
}
```

```
div:before {
    background: conic-
gradient(rgba(3, 10, 0, 0.29), rgba(4, 10, 1, 0.09), rgba(3, 10, 0, 0.29), rgba
(3, 10, 0, 0.49), rgba(3, 10, 0, 0.37), rgba(3, 10, 0, 0.49), rgba(3, 10, 0, 0.
67), rgba(3, 10, 0, 0.25), rgba(3, 10, 0, 0.19), rgba(3, 10, 0, 0.08), rgba(3,
10, 0, 0.66), rgba(3, 10, 0, 0.92), rgba(3, 10, 0, 0.96), rgba(3, 10, 0, 0.06),
 rgba(3, 10, 0, 0.86), rgba(3, 10, 0, 0.81), rgba(3, 10, 0, 0.01), rgba(3, 10,
0, 0.46), rgba(3, 10, 0, 0.94), rgba(3, 10, 0, 0.26), rgba(3, 10, 0, 0.51), rgb
a(3, 10, 0, 0.16), rgba(3, 10, 0, 0.43), rgba(3, 10, 0, 0.24), rgba(3, 10, 0, 0
.29), rgba(4, 10, 1, 0.09), rgba(3, 10, 0, 0.29), rgba(3, 10, 0, 0.49), rgba(3,
 10, 0, 0.37), rgba(3, 10, 0, 0.49), rgba(3, 10, 0, 0.67), rgba(3, 10, 0, 0.25)
, rgba(3, 10, 0, 0.19), rgba(3, 10, 0, 0.08), rgba(3, 10, 0, 0.66), rgba(3, 10,
 0, 0.92), rgba(3, 10, 0, 0.96), rgba(3, 10, 0, 0.06), rgba(3, 10, 0, 0.86), rg
ba(3, 10, 0, 0.81), rgba(3, 10, 0, 0.01), rgba(3, 10, 0, 0.46), rgba(3, 10, 0,
0.94), rgba(3, 10, 0, 0.26), rgba(3, 10, 0, 0.51), rgba(3, 10, 0, 0.16), rgba(3
, 10, 0, 0.43));
}

div:after {
     background: conic-
gradient(rgba(3, 10, 0, 0.17), rgba(3, 10, 0, 0.33), rgba(3, 10, 0, 0.64), rgba
(3, 10, 0, 0.94), rgba(3, 10, 0, 0.42), rgba(3, 10, 0, 0.25), rgba(3, 10, 0, 0.
8), rgba(3, 10, 0, 0.91), rgba(3, 10, 0, 0.38), rgba(3, 10, 0, 0.24), rgba(3, 1
0, 0, 0.31), rgba(3, 10, 0, 0.92), rgba(3, 10, 0, 0.49), rgba(3, 10, 0, 0.7), r
gba(3, 10, 0, 0.94), rgba(3, 10, 0, 0.53), rgba(3, 10, 0, 0.74), rgba(3, 10, 0,
 0.28), rgba(3, 10, 0, 0.61), rgba(3, 10, 0, 0.75), rgba(3, 10, 0, 0.17), rgba(
3, 10, 0, 0.33), rgba(3, 10, 0, 0.64), rgba(3, 10, 0, 0.94), rgba(3, 10, 0, 0.4
2), rgba(3, 10, 0, 0.25), rgba(3, 10, 0, 0.8), rgba(3, 10, 0, 0.91), rgba(3, 10
, 0, 0.38), rgba(3, 10, 0, 0.24), rgba(3, 10, 0, 0.31), rgba(3, 10, 0, 0.92), r
gba(3, 10, 0, 0.49), rgba(3, 10, 0, 0.7), rgba(3, 10, 0, 0.94), rgba(3, 10, 0,
0.53), rgba(3, 10, 0, 0.74), rgba(3, 10, 0, 0.28), rgba(3, 10, 0, 0.61));
     animation-duration: 5.1s;
     animation-direction: reverse;
}

@keyframes rotate {
     to {
          transform: rotate(1turn);
     }
}
```

至于图 5.186 右侧的表盘，请参考随书资源中的代码文件 chapter-5\compus.html。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

5.17 伪元素

伪元素可以设置样式，但是样式属性集合是真实元素样式集合的子集——真实元素的某些样式属性对伪元素无效。

::before

CSS 中，::before 创建一个伪元素，并成为被选中元素的第一个子元素，所以它不适用于没有元素内容的元素上（比如<img>,<hr>,
等）。常通过样式属性 content 来为一个元素添加修饰性的内容。::before 默认为内联元素。参考下图 5.187。



图 5.187 ::before 和::after 伪元素

我们都直到元素内容中可能包含更多的子元素，而::before 是第一个子元素，::after 是最后一个子元素。我们来看一个例子：

```
<!DOCTYPE html>
<html>

<head>
    <style>
        #photo::before,#photo::after{
            content: "❤";
            font-size: 100px;
            color:red;
        }

    </style>
</head>

<body>
    <div id="photo">
        <p>我的相册</p>
        <figure>
            <img src="me.jpg" />
            <figcaption>2019-8-10 杭州</figcaption>
        </figure>
    </div>
</body>

</html>
```

被选中的元素为 id 等于”photo”的 div，因此::before 和::after 就是这个 div 的两个伪元素，代码中给伪元素设置了样式：指明了内容、字号和颜色。浏览的效果如图 5.188。

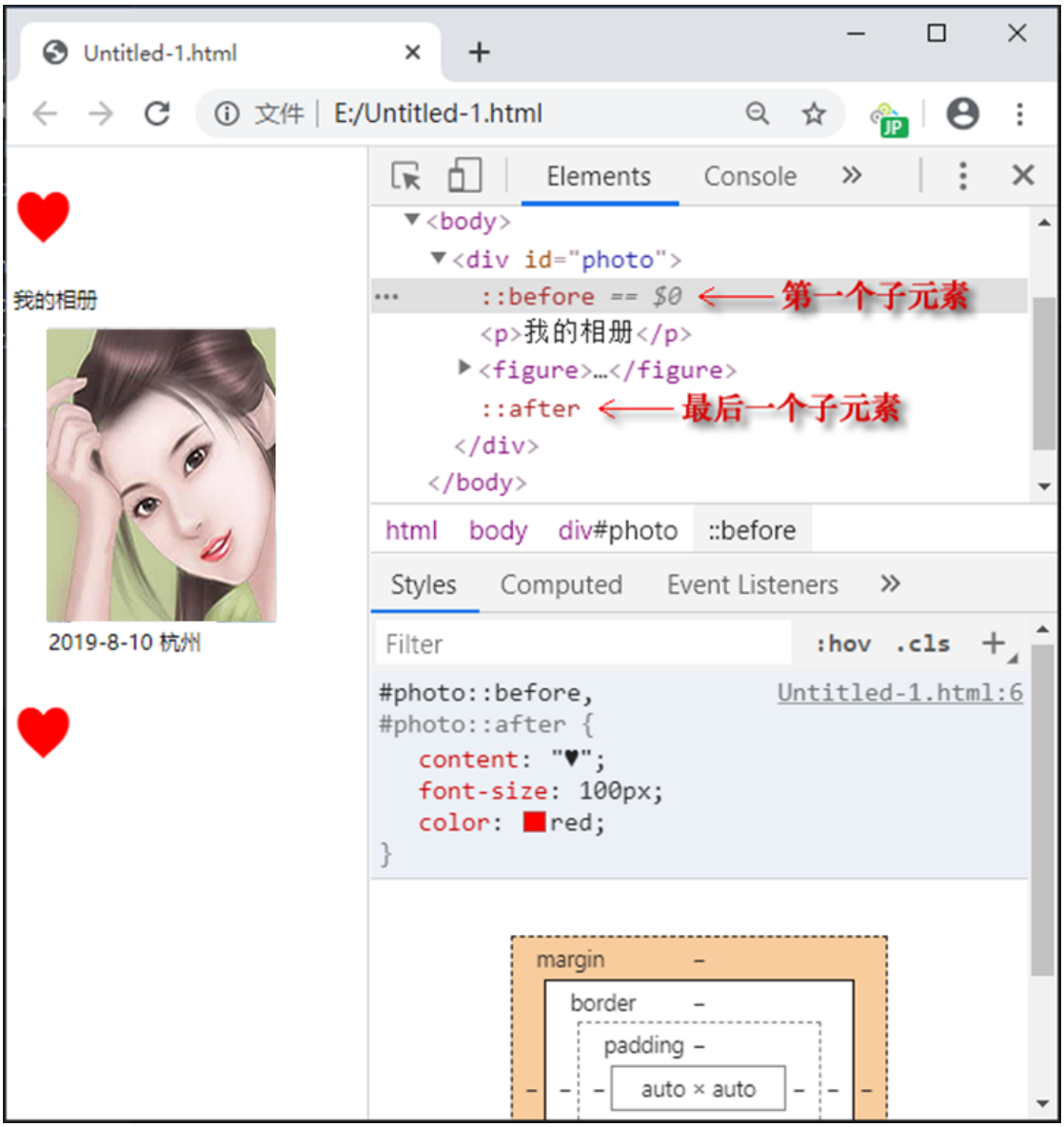


图 5.188 ::before 为元素用法举例

给::before 伪元素设置样式的语法如下：

```
CSS 选择器::before{
     content: <string> || attr(n) || counter(name) || counters(name) || <icon>
|| url() || <quote> | none;
     其他 css 属性: 值;
}
```

如果没有 content 属性或者该属性的值等于 none，被选中的元素就不会插入::before 子元素。content 属性的值会被显示在网页上，它可能的取值范围列举在下面，除了 none 外，其他取值都允许同时使用，如 content: <string> attr(id);。

<string>：普通字符串，可为空，如 content: ""; 、content: "我说：";、content: "★";等。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

attr(n)：返回被选中元素的 n 属性的值（作为字符串）。如 content: attr(id);，即获取被选中元素的 id 的值并显示在网页上。再比如 content: attr(class);、content: attr(href);等。

counter(name)：获取 css 自动计数器的值，不过计数器必须事先要定义好，采用 counter-reset: countername，采用样式 counter-increment: countername 使计数器递增 1。看代码：

```
<head>
    <style>
        div {
            counter-reset: sect; /* 定义计数器 sect，初始值为 0 */
        }

        h3::before {
            counter-increment: sect; /* 递增计数器值 */
            content: "Section "counter(sect) ": "; /* 显示计数器的值 */
        }
    </style>
</head>

<body>
    <div>
        <h3>Introduction</h3>
        <h3>Body</h3>
        <h3>Conclusion</h3>
    </div>
</body>
```

浏览效果见图 5.189。

Section 1: Introduction

Section 2: Body

Section 3: Conclusion

图 5.189 利用::before 自动给章节编号

counters(name)：获取多个计数器的值并且格式化输出，这在输出书目章节时非常有用。我们看例子：

```
ol {
       counter-reset: section; /*每作用于一个 ol 元素就新定义一个计数器 section，计数器允许同名*/
       list-style-type: none;
}

li::before {
       counter-increment: section;
       content: counters(section, ".") " "; /*每作用于一个 li 元素，就显示嵌套的多个计数器，中间用.分隔*/
        color:tomato;
}
……
<ol>
     <li>item</li> <!-- 1     -->
     <li>item
```

```
        <!-- 2     -->
        <ol>
            <li>item</li> <!-- 2.1   -->
            <li>item</li> <!-- 2.2   -->
            <li>item
                <!-- 2.3   -->
                <ol>
                    <li>item</li> <!-- 2.3.1 -->
                    <li>item</li> <!-- 2.3.2 -->
                </ol>
                <ol>
                    <li>item</li> <!-- 2.3.1 -->
                    <li>item</li> <!-- 2.3.2 -->
                    <li>item</li> <!-- 2.3.3 -->
                </ol>
            </li>
            <li>item</li> <!-- 2.4   -->
        </ol>
    </li>
    <li>item</li> <!-- 3     -->
    <li>item</li> <!-- 4     -->
</ol>
<ol>
    <li>item</li> <!-- 1     -->
    <li>item</li> <!-- 2     -->
</ol>
```

浏览的效果见图 5.190。

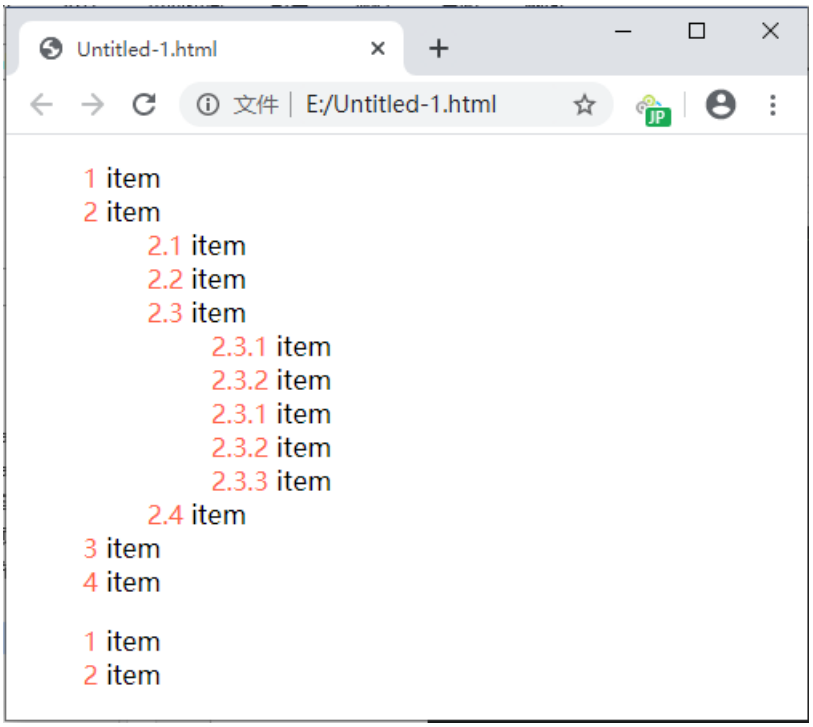


图 5.190 利用::before 产生多级大纲

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

<icon>：使用 unicode 字符集，采用 4 位 16 进制编码，但不同的浏览器显示存在差异。一般是显示图标。比如 content: '\260E'; 就是显示图标 ☎ 。

url()：引用外部资源（如图片或网上其他资源）。如 content: url("me.jpg");、content:url('http://jimyuan.github.io/blog/favicon.ico');。不但能通过 url 显示图片，而且还可以引用浅变，比如代码：

```
li::before {
     content: radial-gradient(circle at 35% 35%, white 10%, rgb(250, 173, 186)
70%);
     display: block;
     border-radius: 50%;
     width: 80px;
     height: 80px;
     overflow: hidden;
}
```

浏览的效果如图 5.191。



图 5.191 ::before 伪元素的 content 样式

<quote>：具体可取值 open-quote , close-quote , no-open-quote 和 no-close-quote，open-quote 代表某种语言的左引号，如中文默认是“，英文默认是"，法语默认是«。而 close-quote 代表某种语言的右引号，如中文默认是”，英文默认是"，法语默认是»。no-open-quote 不引入任何内容，只是递增一个层级，类似地，no-close-quote 也不引入内容，只是降低一个层级。对于中文，引号默认层级为“ ‘’ ”，比如张三说：“孟德斯鸠说‘共和政体的性质是品德。’我很赞同。”第一层级是“”，第二层级是 ‘’，引号层级可以通过 css 的 quotes 属性定义。看例子：

```
<!DOCTYPE html>
<html>
    <head>
        <style>
            div {
                quotes: "【" "】" "『" "』"; /*定义引号层级，第一层【】，第二层『』*/
            }

            p::before {
                content: open-quote;
            }

            p::after {
                content: close-quote;
            }
        </style>
    </head>

    <body>
        <div>
            <p>红太阳升起来</p>
```

```
        </div>
    </body>

</html>
```

浏览效果间图 5.192。

【红太阳升起来】

图 5.192 ::before 伪元素举例

如果要跳过第一层级，用第二层级，修改代码

```
   p::before {
           content: no-open-quote open-quote;
   }
```

::after

用来创建一个伪元素，作为被选中元素的最后一个子元素。用法与::before 完全一样。不过利用::after 可以轻松实现热提示，当鼠标悬停某个元素或者某个元素获得输入焦点时，往往要给用户提示一些帮助信息，样式选择器为 xxx:hover::after{…}、yyy:focus::after{…}。请看下面例子：

```
<head>
    <style>
        span[help-descr] {
            position: relative;
            text-decoration: underline;
            color: #00F;
            cursor: help;
        }

        span[help-descr]:hover::after {
            content: attr(help-descr);
            position: absolute;
            left: 0;
            top: 24px;
            min-width: 200px;
            border: 1px #aaaaaa solid;
            border-radius: 10px;
            background-color: #ffffcc;
            padding: 12px;
            color: #000000;
            font-size: 14px;
            z-index: 1;
        }
    </style>
</head>

<body>
    <p>这是上面代码的实现<br />
        我们有一些 <span help-descr="就是单词和标点符号的集合">文字</span> 有一些
        <span help-descr="点击时弹出的说明小窗口">提示</span>。<br />
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
        把鼠标放上去<span help-descr="有惊奇的发现! ">看看</span>.
    </p>
</body>
```

浏览效果图如图 5.193。

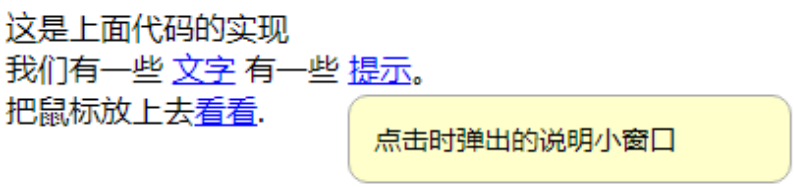


图 5.193 ::after 伪元素用法举例

下面的例子是利用::before 和::after 来画一个六角形，见图 5.194。



图 5.194 用::before/::after 画六角形

```
<head>
    <style>
        div {
            position: relative;
        }

        div::before {
            content: '';
            border-top: transparent;
            border-bottom: 100px solid red;
            border-left: 60px solid transparent;
            border-right: 60px solid transparent;
            width: 0px;
        }

        div::after {
            position: absolute;
            content: '';
            border-bottom: transparent;
            border-top: 100px solid red;
            border-left: 60px solid transparent;
            border-right: 60px solid transparent;
            width: 0px;
            left: 2px;
            top: 66px;
        }
    </style>
</head>

<body>
    <div></div>
```

```
</body>
```

类似的，利用圆角可以会出如图 5.195 所示的花瓣。



图 5.195 用::before/::after 画花瓣

::backdrop

当选择元素处于浏览器全屏模式下时，代表背景蒙版元素。看代码：

```
<head>
    <style>
        img::backdrop {
            background-color: rgba(63, 62, 62, 0.219);
        }
    </style>
</head>

<body>
    <img id="image" src="add-logo.png">

    <script>
        var image = document.getElementById("image");
        image.addEventListener('click', function () {
            var doc = document;
            if (doc.fullscreen) {
                doc.cancelFullscreen();
            } else {
                this.requestFullscreen();
            }
        });
    </script>
</body>
```

::selection

应用于文档中被用户选中的内容（比如使用鼠标或其他选择设备选中的部分）。看代码：

```
<head>
    <style>
        /* 选中的文本是红色背景，金黄色的字体 */
        ::selection {
            color: gold;
            background-color: red;
        }

        /*选中的是蓝色背景，白色的字体的段落*/
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
        p::selection {
            color: white;
            background-color: blue;
        }
    </style>
</head>

<body>
    This text has special styles when you highlight it.
    <p>Also try selecting text in this paragraph.</p>
</body>
```

浏览效果见图 5.196。

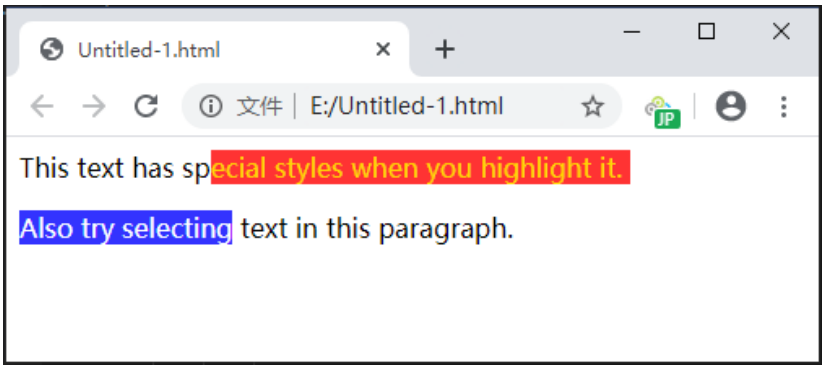


图 5.196 ::selection 伪元素用法举例

第 6 章

HTML5 高级应用

哈，终于学完了 HTML5 的基本工了，如果你止步于此，恭喜你，你已经算是一个网站开发人手了。“什么？连人才都不算？！”不算，充其量比码农好一点。“！@#￥%^&* 那好吧，我继续学。喂，小卖部吧，给我再送两箱方便面……。”

时至今日，HTML5 的功能已经非常强大了，几乎能实现任何应用，包括拖放、地理位置定位、本地存储、全双工网络通讯、多线程、媒体处理、点到点语音视频、物联网等等（具体参考 https://www.w3.org/TR/）。利用 HTML5、CSS3 和 JavaScript 这三件套能实现任何 SaaS 应用，比如在线文字处理（office 365、googe docs 等）、在线图片处理、在线音乐制作、在线工程制图、在线游戏等等。目前 adobe 公司已经把全套产品都移植到了基于 Web 的 SaaS，再也不发布需要安装在电脑上的产品了。另外越来越多的人采用 HTML5 开发移动 app，这种 app 之前唯一欠缺的是没有原生 app 的性能，现在这块短板逐渐被消除了。可以预计，HTML5+CSS3+JavaScript 将迎来爆发期，其他编程语言的命运怎一个“惨”字了得！

本章开始介绍 HTML5 的高级应用，高级应用严重仰赖 JavaScript 语言，如果有了扎实的 JS 基础，那学习本章内容就容易了，否则刚开始有点吃力，但只要你从遇到的第一个例子开始，搞懂每一条语句，这样越到后面就会越轻松。

来吧，跟我一起进入 HTML5 的深水区，趟过去了，你就是人才！一不小心还成为了天才也难说呢，以后的事情谁预料到呢？

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.1 拖放

把网页上的一个元素拖到另一个元素中，并作为其子元素永久存在，也允许把硬盘上的对象（比如文件）拖到网页上。拖放的操作过程是这样的：鼠标指向被拖动的元素并按住鼠标左键不放，然后移动鼠标，经过若干个中间元素后到达目标元素，最后释放鼠标左键，完成拖放操作，参见图 6.1 所示。

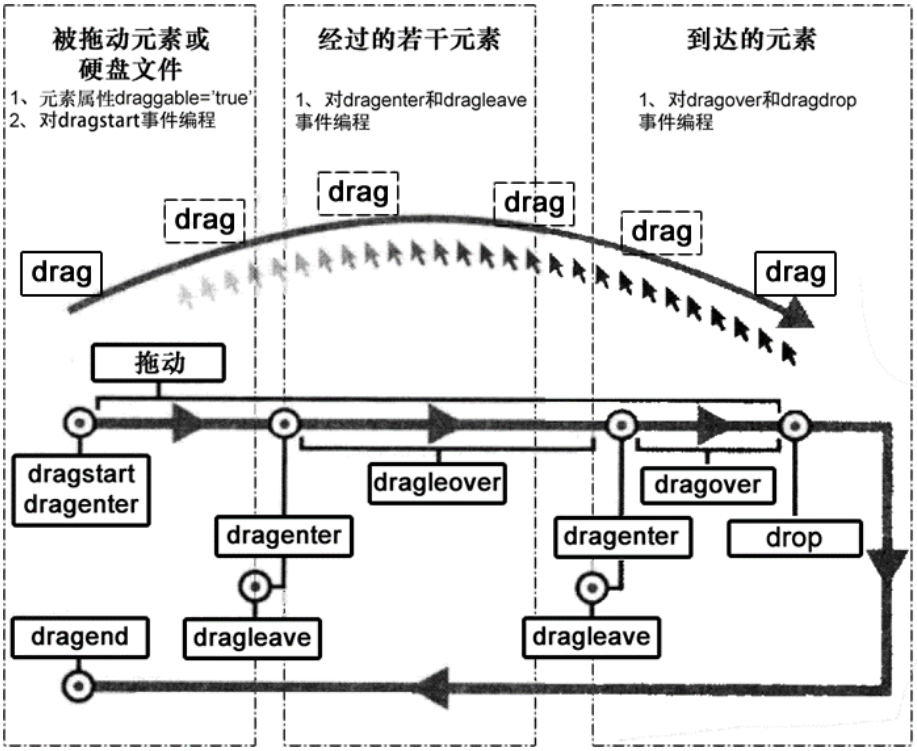


图 6.1 拖放说明

图中带字的方框表示事件，箭头指名这些事件发生的先后次序，其中发生在被拖动元素上的 drag 事件从移动鼠标开始到松开鼠标结束的这段事件内，每隔几十毫秒触发一次，chrome 浏览器为 60 毫秒。我们在编程时不一定要对全部事件编程，根据需要而定，但是被拖动元素的 dragstart、目的元素的 dragover 和 drop 这三个事件一般要要编程处理函数。

最简单的拖放编程思路就是不管经过的中间元素，被拖放的元素加上属性 draggable="true"，并对 dragstart 事件编程，定义一些携带的数据，再对目标元素的 dragover 和 drop 两个事件编程即可。先看一个简单的例子，以使我们有个感性认识。

```
<head>
    <meta charset="utf-8">
    <style type="text/css">
        #div1 {
            width: 198px;
            height: 66px;
            padding: 10px;
            border: 1px solid #aaaaaa;
        }
    </style>
```

```
    <script type="text/JavaScript">
        function allowDrop(e) {
            e.preventDefault();
        }

        function drag(e) {
            e.dataTransfer.setData("Text", e.target.id);
        }

        function drop(e) {
            e.preventDefault();
            var data = e.dataTransfer.getData("Text");
            e.target.appendChild(document.getElementById(data));
        }
    </script>
</head>

<body>
    <div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
    <p draggable="true" id="me" ondragstart="drag(event)">把我拖到上面的方框</p>
</body>
```

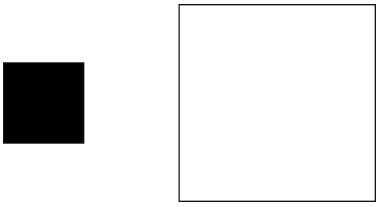
被拖动元素<p>一定要加上属性 draggable="true"，拖放时先后触发 ondragstart 事件、ondragover 事件和 ondrop 事件，因此执行函数的顺序就是 drag(event)→ allowDrop(event)→ drop(event)。参数 event 为 DragEven 对象类型，后面我们重点介绍 DragEven，搞清楚了它，拖放编程就简单了。本例浏览的效果如图 6.2。



图 6.2 拖放用法举例

实操练习

参考上面的例子把左侧的实心正方形拖放到右侧的空心正方形中。



拖放的事件对象 DragEven 继承 MouseEvent 和 Event，DragEven 对象中最重要的属性和方法有这些：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

currentTarget 和 target 属性 - 前者代表与事件关联的当前元素对象，后者代表最开始触发事件的那个对象。由于事件会向父元素冒泡，所以 currentTarget 的值是实时变化的，而 target 始终代表最初发生事件的那个元素，不会变化。比如<p>元素是 元素的子元素，当点击<p>元素时，click 事件冒泡到父元素 时，在 的点击事件处理函数中，currentTarget 属性的值等于 元素对象，而 target 属性的值仍然是<p>元素对象。在事件处理函数中，this 等于 currentTarget，所以尽量采用 this 以及相对获取其他元素节点的方法（参见§3.2.2 小节）。

dataTransfer 属性 – 值为 DataTransfer 对象，包含了拖放事件的数据，详细介绍见下文。此对象包括拖放事件的状态，如拖放的类型（如复制或移动）、拖放的数据（一个或多个条目），以及每个拖放条目的 MIME 类型。另外 dataTransfer 对象还包含添加或删除数据条目的方法。注意：dragover 事件对象中没有 dataTransfer 对象。

type 属性 – 类型为字符串，表示事件名称，如"dragstart"、"dragover"、"drop"等。

preventDefault()方法 - 阻止浏览器执行默认的事件处理函数（在拖放事件中，默认处理函数是不允许停放的）。实际上是把 DragEven.defaultPrevented 属性置为 true。不管程序员有没有给事件编写处理函数，当事件发生时，浏览器最后都会执行默认的事件处理函数，除非你明确告诉它：“喂，你不要执行默认的事件处理函数啊！”。

stopPropagation()方法 - 阻止事件继续链式触发，比如在冒泡机制中，调用本方法后，事件就不会继续冒泡了。关于事件冒泡和捕获机制请参考“§3.3 页面事件”小节。另外一个方法 stopImmediatePropagation()除了阻止事件继续链式触发外，还取消本元素上注册的剩下的未执行的处理函数，也就是说执行完本处理函数后不再执行任何处理函数了。

DragEven.dataTransfer 属性的值为 DataTransfer 对象，此属性只在 dragstart 和 drop 事件中才存在值，代表被拖拽的数据项。DataTransfer 对象中最重要的属性和方法有这些：

files 属性 - 包含被拖放的文件对象数组（从硬盘拖放文件），或者为空数组（拖放网页元素），文件对象 File 包括这些属性：name（文件名，不含目录）、size（以字节为单位的文件大小）、type（文件类型，如"text/plain"、"image/png"）、lastModified（最近修改时间，从 1970 年 1 月 1 如 0 时以来的毫秒数）。例如：

```
function dragstart(evt) {
     const date = new Date(evt.dataTransfer.files[0].lastModified);
     console.log(evt.dataTransfer.files[0].name+"修改时间是"+date);
}
```

effectAllowed 属性和 dropEffect 属性 – 前者规定允许的拖放操作类型（集合），而后者在前者规定的集合中选择具体的一个操作类型，即 effectAllowed 规定了 dropEffect 的取值范围。通常在 dragstart 事件处理函数中设置 effectAllowed 属性的值，而主要在 dragenter 或 dragover 事件处理函数中设置 dropEffect 属性的值。dropEffect 定义的操作类型不同，用户拖放过程中的鼠标指针也不同，参见图 6.3 所示。



图 6.3 拖放时鼠标指针

effectAllowed 属性可能的取值有："none"（不能拖放）、"copy"、"copyLink"、"copyMove"、"link"、"linkMove"、"move"、"all"和"uninitialized"（默认值，相当于"all"），copy 就是复制，move 就是移动，link 就是建立链接（相当于 windows 的桌面快捷方式）。比如 effectAllowed="copy"，那么只能 dropEffect="copy"，如果赋予其他值，那么拖放无效果。再比如 effectAllowed="copyMove"，此时 dropEffect="copy"，也可以 dropEffect="move"。但是最终的拖放成效还是由 drop 事件的处理函数决定，比如虽然 dropEffect="copy"，但是我编程时偏要实现移动操作。

items 属性和 types 属性 – 只读属性，types 为字符串数组，存储了全部的拖拽数据项类型，结合 getData()方法可以遍历全部的拖拽数据项。这两个属性提供了另外一种管理拖拽数据项的方法，有的浏览器不支持，但全部浏览器都支持 dataTransfer，所以建议使用 dataTransfer 来管理被拖拽的数据项。

setData(type, data)方法 – 增加或修改被拖拽的数据项，每个数据项就是“数据/类型”对，而已存在的数据项中，type 不能相同，可以把 type 看作“键”。参数 type 定义数据项类型（即 mimeType），最常见的类型有"text/plain"（纯文本，也可以写成"text"）、"text/html"（HTML 文档）、"text/uri-list"（URI 地址，也可以写成"url"）、text/css（css 代码）、"image/jpeg"（JPEG 图像）、image/svg+xml（svg 矢量图）、application/javascript（js 代码）、application/json（JSON 对象）等。参数 data 是用字符串表示的数据。如果不存在 type 类型，那么本方法就增加一个数据项，否则替换已有数据项中的数据。本方法返回 void。看例子：

```
    evt.dataTransfer.setData("Text", e.target.id);
    evt.dataTransfer.setData("text/html", "<h1>Hello World!</h1>");
    evt.dataTransfer.setData("text/uri-list",
"https://www.youduo.me/index.html");
```

setDragImage(img, xOffset, yOffset)方法 – 设置跟随鼠标指针移动的缩略图，默认由浏览器根据被拖拽对象产生跟随缩略图。参数 img 为 Image 对象，xOffset 和 yOffset 为鼠标指针左上角与缩略图左上角的偏移。看例子：

```
   var img = new Image();
   img.src = "shock.jpg";
   function dragstart(e) {
        e.dataTransfer.setDragImage(img, -20, -10);
        …
    }
```

拖拽时的效果见图 6.4 中间部分。



图 6.4 设置跟随的鼠标指针

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

getData(type)方法 - 获取数据项，参数 type 取值与 setData()方法的同名参数一样，返回代表数据的字符串或者空串。下面的代码输出全部的数据项：

```
<p>拖动本文字或下面图片：</p>
<p><img src="./shock.jpg"></p>
<input>
<script defer>
 document.addEventListener('dragstart', function (event) {
   let data = event.dataTransfer;
   data.types.forEach(type => console.log(type + " <-
-> " + data.getData(type)));
 });
 document.querySelector("input").addEventListener("drop", event => {
   let data = event.dataTransfer;
   data.types.forEach(type => console.log(type + " <==> " + data.getData(type))
);
 });
</script>
```

浏览的效果参见下图：



图 6.5 getData()方法举例

clearData([type])方法 - 清除 DataTransfer 对象中 type 类型的数据项，如果省略 type 参数，则清除 DataTransfer 对象中的全部数据项。

下面再看一个例子，把硬盘上的文件拖放到网页上。

```
<!DOCTYPE HTML>
<html>

<head>
    <meta charset="utf-8">
    <script type="text/JavaScript">
        function dragover(e) {
            e.preventDefault();
            e.dataTransfer.dropEffect = 'copy';
        }

        function drop(e) {
            e.preventDefault();
            var t = document.getElementById("text");
            t.value = '';
            var files = e.dataTransfer.files;
            for (i = 0; i < files.length; i++) {
```

```
                t.value += "\n" + files[i].name + " --" + files[i].size;
            }
        }
    </script>
</head>

<body>

    <textarea id="text" rows="4" cols="56" readonly ondrop="drop(event)" ondrag
over="dragover(event);"></textarea>
    <p>把硬盘上的文件拖放到上面的框中</p>

</body>

</html>
```

从硬盘拖过来的文件对象都放在 e.dataTransfer.files 数组中，这个数组的元素类型为 File 对象。因为不存在被拖拽页面元素，所以只需要对目的元素的 drop 和 dragover 事件编程即可。

再来看看一个较复的拖放例子：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <style>
        #div1 {
            width: 200px;
            height: 200px;
            padding: 10px;
            border: 1px solid #aaaaaa;
            margin-bottom: 50px;
        }

        #drag1 {
            cursor: pointer;
        }
    </style>
    <script>
        function allowDrop(ev) {
            ev.preventDefault();
            ev.dataTransfer.dropEffect = "copy"

        }

        function drag(ev) {
            ev.dataTransfer.setData("Text", ev.target.id);
            ev.dataTransfer.effectAllowed = "copyMove";

        }

        function drop(ev) {
            ev.preventDefault();
            var data = ev.dataTransfer.getData("Text");
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
            var item = document.getElementById(data).cloneNode();
            ev.target.appendChild(item);
        }
    </script>
</head>

<body>
    <p>请把图片拖放到矩形中：</p>
    <div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
    <img id="drag1" src="brother.jpg" draggable="true" ondragstart="drag(event)
"/>
</body>

</html>
```

实现思路就是克隆被拖拽的元素对象（调用元素对象的 cloneNode()完成克隆），然后把克隆元素 appendChild()到指定位置。效果图 6.6（左侧为刚开始的页面，右侧为拖放完毕的效果）：



图 6.6 复制拖放

实操练习

1、刷新网页时看到的是图 6.7 左侧的圆和正方形，然后把硬盘上的一个文件拖入到圆中，最后再把圆拖放到正方形中，最终的效果如图 6.7 右侧部分。

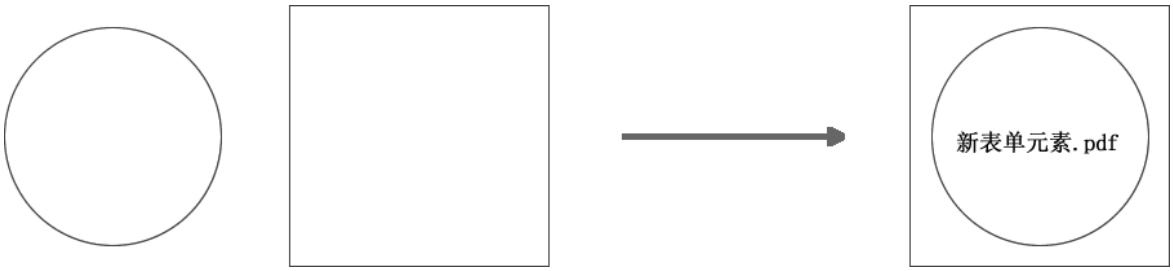


图 6.7 拖放实验 1

2、把喜欢的项目拖放到收藏夹，把不要的项目拖到垃圾桶（被删除）。【提醒：收藏项目采用复制（调用元素对象的 cloneNode()方法产生克隆对象，然后再把它加入），移到垃圾桶采用移动】，参考图 6.8。

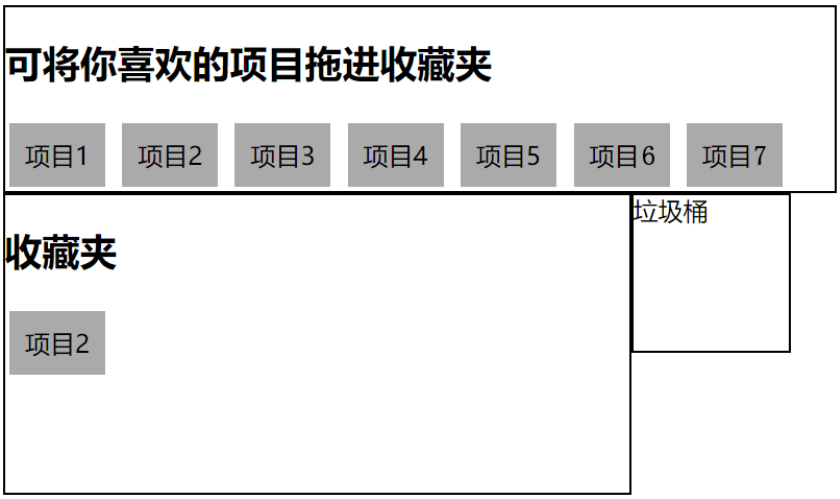


图 6.8 拖放实验 2

6.2 系统剪贴板

操作系统有一个系统剪贴板，通过它我们可以方便拷贝、剪贴和粘贴内容。今天，我们开发的 Web 页面也能轻松访问系统剪贴板——通过 window.navigator.clipboard 接口，只不过出于安全考虑，只能通过 https 浏览页面才行，且用户同意授权给浏览器"clipboard-read"权限——读取剪贴板前浏览器会询问用户是否同意，示意图如图 6.9。



图 6.9 剪贴板授权

window.navigator.clipboard 属性的值就是 Clipboard 对象实例，此对象继承 EventTarget 对象，因此除了 EventTarget 提供方法外，Clipboard 还提供了访问剪贴板的如下方法：

read() – 从剪贴板中读取任何类型的数据。返回 Promise，成功完成后确值为 ClipboardItem 对象的数组，读取的数据保存这个数组中，如果没权限访问剪贴板，Promise 会 rejects。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

readText() – 从剪贴板中读取文本数据。返回 Promise，成功完成后确值为字符串，如果没权限，Promise 会 rejects。readText()方法就是 read()方法的特例。例如：

```
navigator.clipboard.readText()
.then( readStr => document.getElementById("outbox").innerText = readStr);
```

write(data) – 向剪贴板写入任意类型的数据，包括 png 图形。参数 data 为 ClipboardItem 对象的数组，其中存储了要写入剪贴板的数据。返回 Promise，成功写入时确值，失败时拒绝。下面的函数把 canvas 画布上的图形拷贝到剪贴板：

```
function copyCanvasContentsToClipboard(canvas) {
    canvas.toBlob(function (blob) {
        let data = [ new ClipboardItem({ [blob.type]: blob }) ];
        navigator.clipboard.write(data).then(     () => console.log("    成    功
"), err => console.log("出错: " + err) );
    });
}
```

writeText(writeStr) – 向剪贴板写入字符串，参数 writeStr 为被写入剪贴板的字符串，可以看出 writeText()方法是 write()方法特例。返回 Promise，成功写入时确值，失败时拒绝。

read()和 write()方法都涉及 ClipboardItem 对象的数组，可采用类似“var clip = new ClipboardItem(data)”的语句实例化 ClipboardItem 对象，其中参数 data 为包含若干“键:值”的对象，“键”指明 MIME 类型，“值”为 Blob 类型的数据。常见的 MIME 类型参见上一小节的 DataTransfer.setData()方法介绍，但是图形目前只支持'image/png'。例如：

```
   async function saveCanvas(canvasObj) {
       const text = new Blob(['作者: 齐白石'], { type: 'text/plain' });
       const image = await canvasObj.toBlob();
       const item = new ClipboardItem({ 'text/plain': text,
'image/png': image });
   }
```

ClipboardItem 对象的属性 types 返回 MIME 类型的数组，而 getType(type)方法返回 Promise，成功执行后确值为 type 这个“键”对应的 Blob 这个“值”，如对于上面的例子，调用 item.getType('text/plain').then(value=>{…})，value 就是 text 变量的值。

随书配套资源 chapter-6\clipboard-1.html 例子通过拖放页面上的图片实现拷贝，然后你就可以在其他绘图软件中粘贴了。里面最核心的代码如下：

```
    function copyClipboard(imgObj) {
        const canvas = document.createElement("canvas");
        canvas.width = imgObj.width;
        canvas.height = imgObj.height;
        const ctx = canvas.getContext("2d");
        ctx.drawImage(imgObj, 0, 0);
        canvas.toBlob(imageBlob => {
            const item = new ClipboardItem({ [imageBlob.type]: imageBlob });
            navigator.clipboard.write([item]).then(
                () => alert("拷贝成功，以后你可以在绘图软件中粘贴！"),
                (err) => alert("拷贝失败: " + err)
            );
        });
    }
```

imgObj 就是被拖动图片的元素对象，把它导入到 canvas 画布上，然后调用画布元素的 toBlob() 方法把画布输出 Blob 对象，最后写到系统剪贴板中，完成拷贝任务。

由于剪贴板支持 png 图形,所以代码中统一采用<canvas>元素对象把其他格式的图形转化为 png 格式。

下面的代码把图片粘贴到页面上，然后上传图片文件：

```
<img src="default.jpg" id="profile" contenteditable="true">
……

window.addEventListener("load", () => {
    navigator.permissions.query({ name: 'clipboard-read' }).then(
        permissionStatus => {
            if (permissionStatus.state == 'denied') {
                alert("用户不同意浏览器访问系统剪贴板");
            }else{
                doWorks();
            }
        }
    );
});

function doWorks() {
    const imgElem = document.getElementById("profile");
    document.addEventListener("paste", pasteImg);
    function pasteImg(evt) {
        navigator.clipboard.read().then(data => {
            for (var item of data) {
                for (var type of item.types) {
                    if (type.indexOf("image/") != -1) {
                        item.getType(type).then(blob => {
                            imgElem.src = URL.createObjectURL(blob);
                            fetch('upload.php', { method: "POST", body: blob })
                                .then(response => console.log(response.text()))
                        });
                        return;
                    }
                }
                console.log(type);
            }
        }).catch(err => console.log(err));
        evt.preventDefault();
    }
}
```

upload.php 的代码如下：

```
<?php
    // 获取 POST 的 body
    $data = file_get_contents('php://input');
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    $fp = fopen("profile.png", "wb");
    fwrite($fp, $data);
    fclose($fp);
?>
```

6.2 地理定位

打开网页，就能看到你当前所处的位置，这就是 HTML5 引入的地理定位的功劳——通过 Geolocation 对象，但是要求页面必须走 https 协议。属性 navicator.geolocation 的值为 Geolocation 对象实例，此对象只有三个方法，没有属性和事件。

getCurrentPosition(successHandler[, errorHandler[, [options]]) - 获取设备的当前位置。成功获取位置时调用 successHandler(GeolocationPositio)函数，其他两个参数是可选的，出错时调用 errorHandler(GeolocationPositionError)函数，参数 options 为 PositionOptions 对象类型，针对获取位置信息，提供额外的要求。

watchPosition(successHandler[, errorHandler[, [options]]) – 注册一个回调函数，当设备所在的位置发生变化时调用它。参数与上 getCurrentPosition()方法一样。本方法返回一个整数，clearWatch()方法通过此整数注销回调函数。

clearWatch(id) – 注销由 watchPosition()方法注册的回调函数，参数 id 为整型。

调用 getCurrentPosition()方法成功时会把 GeolocationPositio 对象实例作为实参传给被调用函数，GeolocationPositio 对象存储了获取的位置信息，它只有两个属性：

coords – 只读属性，值为 GeolocationCoordinates 对象实例——定义了位置的经纬度等数据。GeolocationCoordinates 对象有这些只读属性：latitude（表示纬度的双精度浮点数）、longitude（表示经度的双精度浮点数）、altitude（表示海拔的双精度浮点数或 null，单位为米）、accuracy（表示经纬度的误差范围，单位为米）、altitudeAccuracy（海拔的精度，单位是米）、heading（表示设备朝向的双精度浮点数，单位为度——与北方的顺时针夹角，因此 90 表示正东方向）、speed（双精度浮点数，表示速度，单位为“米/秒”）。

timestamp – 只读属性，存储了生成位置信息时的邮戳，即从 1970 年 1 月 1 日 0 时以来的毫秒数。对于一个移动的设备，位置与时间形成函数关系。

在获取位置信息时，可通过 PositionOptions 对象提供额外的要求，它有三个属性：

enableHighAccuracy– 布尔类型，置为 true 时则会利用设备更精确的定位部件，从而获得更为准确的位置信息，但是也会消耗更多的电能和时间，默认值为 false。

enableHighAccuracy– 设置超时，单位为毫秒，规定在多长的时间内完成获取位置信息的任务，默认为 Infinity，即没限制时间。

maximumAge– 设置缓存的期限（单位为毫秒），即保存在缓存中的位置信息的有效期。在有效期内，不再获取真实的位置信息，而是直接反馈缓存中的位置信息。如果为 0，那么相当于关闭缓存区，每次都得获取真实的位置信息，相反如果为 Infinity，除了第一次外，以后都直接反馈缓存中的位置信息。本属性的值默认为 0。

获取位置信息出错时会把错误信息保存在 GeolocationPositionError 对象中，此对象只有两个属性：

code– 错误代码，1（PERMISSION_DENIED）- 用户没有授权、2（POSITION_UNAVAILABLE）- 至少缺失一个重要的位置数据据而导致获取地理位置失败、3（TIMEOUT）- 超时。小括号中的内容代表对应的常量（宏定义）。

message– 错误信息。一般用于调试，最好不要显示在页面上。

下面的例子获取当前的经纬度和海拔高度。

```
<!doctype html>
<html>

<head>
    <meta charset="utf-8">
    <title>获取我的位置信息</title>
</head>

<body>
    <p id='position'>我的位置: </p>
    <button>获取位置信息</button>
    <script>
        window.addEventListener("load", () => {
            document.querySelector("button").onclick = myPosition;
            function myPosition() {
                if (navigator.geolocation) {
                    var options = {
                        enableHighAccuracy: true,
                        timeout: 5000,
                        maximumAge: 0
                    };
                    navigator.geolocation.getCurrentPosition(ok, err, options);
                } else {
                    alert('浏览器不支持地理位置');
                }
            }

            function ok(po) {
                document.getElementById("position").innerHTML =
                    "我的位置:  经度=" + po.coords.latitude +
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
                    ", 纬度=" + po.coords.longitude +
                    ", 海拔=" + po.coords.altitude +
                    ", 时间=" + (new Date(po.timestamp)).toLocaleString();
            }

            function err(error) {
                alert('获取位置信息失败:' + error.message);
            }
        });
    </script>
</body>

</html>
```

如果要在地图上显示具体的位置，那么再用获得的经纬度去调用地图 api 接口，当然国内使用百度地图比较方便，也可以采用谷歌地图。下面的例子是采用百度地图。

1、获取百度地图 ak 码。

打开 http://lbsyun.baidu.com/网站，注册一个账号，注册成功后会发送一封邮件，我们需要查看邮箱并点击里面的激活链接，返回到百度会员中心，会提示激活成功。然后点击控制台》应用管理》我的应用，在这里创建应用，“应用名称”可以自己随意编辑，既然是百度地图，建议大家就写“地图”“IP 白名单”处建议大家就填*，方便在不同的电脑上操作，然后点击“提交”。参考图 6.10。



图 6.10 百度地图 AK 码

最后拷贝 ak 码“5v7pYU1VMDugFAhteb4DdLUpGsZ8IDIy”，共后续使用。

2、编写页面文件。

具体代码见随书资源文件 chapter-6\geolocation-baidu.html，通过 https 协议浏览（在 firefox 浏览器上测试成功）。基本思路就是调用 navigator.geolocation.getCurrentPosition()方法获取当前位置的经纬度，然后调用百度地图 api，核心代码如下，其中红色下划线部分就是从百度获得的 AK 码：

```
<script src="//api.map.baidu.com/api?type=webgl&v=1.0&ak=5y7pYU1VMDugFAhteb4DdLUpGsZ8IDIy"></script>
……
function showPosition(position) {
    var y = position.coords.longitude;//纬度
    var x = position.coords.latitude;//经度
    var map = new BMapGL.Map('container'); // 创建 Map 实例
    // 初始化地图,设置中心点坐标和地图级别
    map.centerAndZoom(new BMapGL.Point(y, x), 12);
    map.enableScrollWheelZoom(true); // 开启鼠标滚轮缩放
}
```

6.3 本地存储

HTML5 新增了 window.Storage 对象，用于在本地保存数据。不同于离线应用，web 存储专门用于保存数据（比如用户名和密码），而离线应用是保存页面资源（比如图片）。早期常用 cookie 保存数据，容量不能超过 4KB，HTML5 引入了 web 存储，这样就大大增加了存储容量。web 存储分为两类：一类是 localStorage，保存的内容永久有效，除非用户主动删除，数据可被同源的每个浏览器窗口或者元素页共享；另一类是 sessionStorage，存储时间就是会话期，页面一关闭，内容就被清空，而且不能跨浏览器标签页共享。

web 存储的格式为“键/值”对，即“键名=值”，“键”和“值”都是字符串。存储的容量上限 5-10MB，具体的容量各个浏览器不同。现代浏览器另外引入的 IndexedDB 相当于 noSQL 数据库，它是基于事务的异步操作，也是存储“键/值”，不过这里的“值”为 JS 对象，存储容量上限为空闲磁盘的 50%，对于 IndexedDB 的讲解请参考后续章节。

window 对象的 localStorage 和 sessionStorage 属性都返回 Storage 对象实例，Storage 对象用于对数据的“增/删/改/查”操作，它的属性和方法如下：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

length – 只读属性。返回存储的记录条数，一个“键/值”对称为一条记录。

key(index) – 返回第 index 个记录的“键”，index 取值为 0、1、2、…。

getItem(keyName) – 返回指定键的值，keyName 参数就是键名，如 getItem('age')。

setItem(keyName, keyValue) – 插入一条新记录或修改现有的记录。

removeItem(keyName) – 删除指定键名的记录。

clear() – 删除所有记录。

看例子：

```
<body>
    <p id="info"></p>
    <script>
        if (Storage) {
            localStorage.name = "张三";
            sessionStorage.age = '20';
            document.getElementById("info").innerHTML = localStorage.name;
        }
    </script>
</body>
```

本例子中，我们把姓名保存在 localStorage 的 name 键名中，把年龄保存在 sessionStorage 的 age 键名中。在开发者工具中查看，见图 6.11 所示。

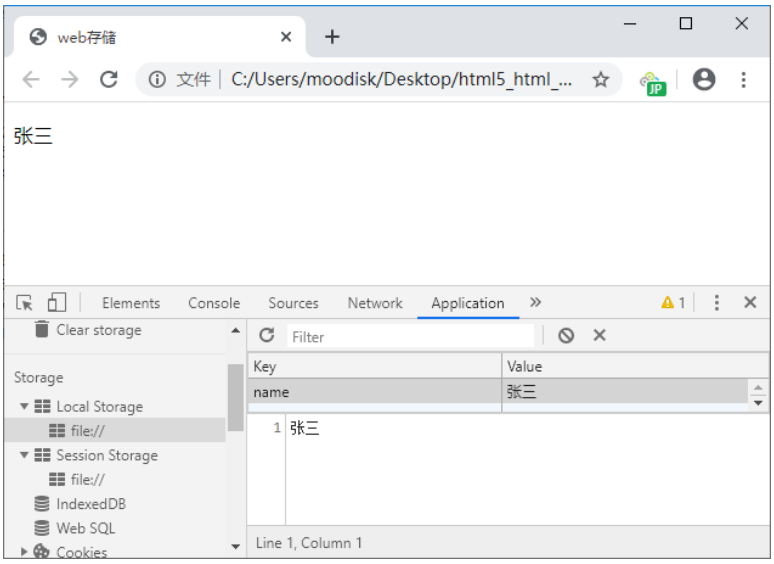


图 6.11 查看本地存储的数据

也可以采用下面的方式存储和修改数据：

```
    localStorage.key=value
    sessionStorage.key=value
```

也可以采用下面的方式读取数据：

```
    localStorage.key
    sessionStorage.key
```

再看一个例子：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>web 存储</title>
</head>

<body>
    <p id="info"></p>
    <script>
        if (Storage) {
            localStorage.name = "zhangsan";
            var n = localStorage.getItem('name');
            localStorage.setItem('age','20');//增加键/值对;
            localStorage.setItem('phone','13670055530');//增加键/值对;
            console.log(localStorage.length);
            console.log(localStorage['name']);//显示键 name 的值;
            console.log(localStorage.key(2));//显示第 3 个元素的键;
            localStorage.removeItem('name');
            localStorage.clear();//全部删除。
        }
    </script>
</body>


</html>
```

实操练习

分别在 localStorage 和 seesionStorage 中保存信息：name="张三",age="20", sex="male"。然后关闭浏览器，再打开网页读取上次保存的信息。

6.4 全双工通信

AJAX 技术使得前端能主动从后端获取数据，在不知道后端何时数据准备就绪时，前端只有轮询了，就比如你每隔一个小时跑一趟邮局询问信件到了没有。每一次轮询都要消耗网络带宽，尤其是对于当下流行的“订阅/推送”应用，AJAX的缺点非常明显。为此HTML5引入了WebSocket

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

对象类，此对象类在前端和后端之间建立双向通信管道，实现类似 c 语言的 socket 网络编程。这下好了，你不用总是跑邮局了，邮局会主动打电话过来告诉你信件到了。

WEB 全双工编程主要采用 WebSocket、CloseEvent 和 MessageEvent 三个对象，其中 WebSocket 是最重要的对象，可采用如下所示的代码实例化一个 WebSocket 对象：

```
var aWebSocket = new WebSocket(url [, protocols]);
```

参数 url 为 WebSocket 服务器的 URL，协议为 ws 或者 wss，其中 wss 是安全协议，类似 https。例如"ws://www.youduo.me:5905/echo"。 可选参数 protocols 为字符串或者字符串数组，指定更多的自定义子协议，比如["protocolOne", "protocolTwo"]。

1、 WebSocket 对象的属性

binaryType – 传输的二进制数据类型，可能的值有"blob"（Blob 对象）、"arraybuffer"（ArrayBuffer 对象）。

bufferedAmount – 只读属性，发送队列中的字节数。

protocol – 服务器选中的子协议名称，只能是实例化时指定的若干子协议中的一个。

readyState – 为无符号短整型，表示链接状态。可能的值有（小括号内是对应的宏）：

0(WebSocket.CONNECTING) - 表示连接尚未建立。

1(WebSocket.OPEN) - 表示连接已建立，可以进行通信。

2(WebSocket.CLOSING) - 表示连接正在关闭。

3(WebSocket.CLOSED) - 表示连接已经关闭或者连接不能打开。

url – WebSocket 服务器的完整的 URL。

2、 WebSocket 对象的方法

send(data) – 把数据放入发送队列中，参数 data 的类型可为字符串、ArrayBuffer 对象、Blob 对象或任何 JavaScript 数组（如 Int8Array、Uint8Array、Int32Array 等）。例如：

```
aWebSocket.send("How are you?");
var msg = {
      type: "message",
      text: document.getElementById("text").value,
      id: clientID,
      date: Date.now()
};
aWebSocket.send(JSON.stringify(msg));
```

close([code, reason]) – 关闭 WebScoket 连接，两个参数都是可选的，code 为才错误代码，默认是 1005，reason 为人类语言描述的关闭原因。

3、 WebSocket 对象的事件

open/onopen – 链接建立成功时触发。一般在本事件的处理函数中向对方发送消息。

message/onmessage – 消息到达时触发。消息内容在事件参数 event 的 data 属性中，即 event.data。

close/onclose – 关闭 WebSocket 连接时触发。

error/onerror – 因出错而关闭连接时触发。

请看下面的例子：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="utf-8">
    <title>WebSocket</title>
</head>

<body>
    <p id="answer"></p>
    <script>
        if ("WebSocket" in window) {
            var ws = new WebSocket("ws://www.veryopen.org:5905/echo");
            ws.onopen = function () {
                ws.send("你好吗？");
            }
            ws.onmessage = function (evt) {
                document.getElementById("answer").innerText = evt.data;
            }
            ws.onclose = function () {
                alert("对方关闭连接");
            }
        } else {
            alert("浏览器不支持 WebScoket！");
        }
    </script>
</body>

</html>
```

作者采用 pywebsocket 项目（https://github.com/google/pywebsocket）在 www.veryopen.org 机器上搭建的 WebSocket 后端，读者们可以直接采用 ws://www.veryopen.org:5905/echo 做测试，这个后台只是简单地把你发送的消息反弹回来。具体的搭建过程参见“§1.5.2 开发测试”小节。

socket.io.js 为 Web Socket 的框架开发库，它包含基于 Node.js 的服务端和 JavaScript 客户端库，采用它能非常方便地开发浏览器和 Web 服务器之间的实时双向通讯的应用，比如网页版的即时聊天工具、WebRTC 的信令服务等。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

实操练习

发开如图 6.12 所示的简单聊天网页。

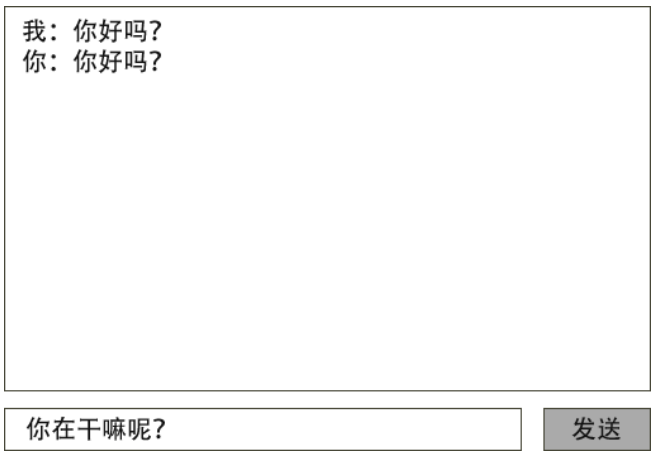


图 6.12 聊天页面

6.5 服务器发送事件

HTML5 引入的服务器发送事件（Server-Sent Event）对象 window.EventSource，允许网页被动获得来自服务器的更新。其实现原理是一个数据流，会连续不断地发送过来，这时，网页前端不会关闭连接，一直等着网页后端发过来的新的数据流，就像看视频那样。通过 SSE 机制，很容易实现订阅服务，前端一旦订阅后端的一些消息，那么以后只要后端发布了消息，前端就能及时显示在网页上。SSE 比 WebSocket 更轻量级，而且是单向的（后端→前端）。

一般采用如下所示代码实例化一个 EventSource 对象：

```
var eventSource = new EventSource(url[, config]);
```

参数 url 为 Web 服务器上资源的 URL，该资源会推送消息，可选参数 config 对象用于配置新建的连接，目前只有一个属性 withCredentials，访问跨源资源时可能要用到，指示是否要一起发送 cookie，值为 true 或 false，默认是 false。

1、 EventSource 对象的属性

readyState– 表示连接的状态码，0-正在建立连接，1-连接成功，2-连接已关闭。

url –代表资源的 URL。

withCredentials –指示跨源时是否一起发送 cookie。

2、 EventSource 对象的方法

close() – 关闭连接。

3、 EventSource 对象的事件

open/onopen – 连接成功时触发。

message/onmessage – 消息到达时触发，消息内容在 event.data 中。

error/onerror – 建立连接失败时触发。

看一个例子：

```
<h1>获取后端的时间</h1>

<div id="result"></div>

……

<script>

    if ("EventSource" in window) {

        var source = new EventSource("sse.php");

        source.onmessage = function (event) {

            document.getElementById("result").innerHTML += event.data + "<br>";

        };

    } else {

        alert("抱歉，你的浏览器不支持 server-sent 事件");

    }

</script>

</body>
```

后端 sse.php 代码：

```
<?php
header('Content-Type: text/event-stream');
header('Cache-Control: no-cache');

date_default_timezone_set('PRC');
$time = date('Y-m-d H:i:s');
echo "data: 现在的时间是: {$time}\n\n";
flush();
?>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

后端要求是：把报头 "Content-Type" 设置为 "text/event-stream"，通知浏览器不对页面进行缓存（Cache-Control 设为 no-cache），发出的消息包要以 "data: "开头"\n\n"结尾（一个消息包可以包含多行），flush()立即把缓存中的数据输出到前端。

因为 php 语言的 echo 或 print 语句把信息放到缓存区，当缓存区满的时候再输出到前端，而 flush()函数直接把缓存区的信息输出到前端，不管缓存区有没有满。

需要注意的是：浏览器一般在数秒之内没有收到消息就会重连，所以为了维持 sse 会话的可用性，Web 服务器大概每隔 5 秒就会执行 sse.php 一次，因此尽管 sse.php 中没有循环语句，你仍然能源源不断看到后端发送过来的消息。

实操练习

订阅 web 服务器的时钟，并显示在浏览器中的某个固定位置，格式如下：

现在时间：15:45:30

6.6 独立线程

window.Worker 是 HTML5 引入的网页线程对象，运用此对象并结合 JavaScript 语言可以轻松实现网页多线程，这些线程在“后台”运行，不会影响“前台”网页的渲染，这里的“后台”和“前台”都在浏览器中。一些计算重的任务或者订阅后台服务器的消息，最好交给另外的后台线程去做。但是后台线程不能访问浏览器的 window 对象和 DOM 树，所以在“后台”线程中不能再创建线程，因为它根本访问不了 window.Worker 对象。注意：以 file 协议测试 Web Workers 时不能用 chrome 浏览器，火狐浏览器可以。

可采用类似下面的语句实例化一个线程对象——Worker：

```
    var myWorker = new Worker(aURL, options);
```

参数 aURL 指名在线程中执行的 JavaScript 程序，比如与页面同源。可选参数 options 为对象，提供给新线程的一些配置项：

type：定义线程的类型，可能的值有"classic"（作为普通脚本执行）或"module"（作为模块执行），默认值为"classic"。

credentials：是否发送 cookie。当 type="module"时，本属性可取值"omit"（不发送）、"same-origin"（默认值。同源请求时发送，跨域请求时不发送）或"include"（发送）。当 type="classic"时，只能取值"omit"。

name：给线程的全局执行上下文命名，命名的目的主要用于调试。DedicatedWorkerGlobalScope 对象是线程的全局执行上下文。

看例子：

```
var worker = new Worker("worker.js", { type: "module", credentials: "include"});
```

Worker 对象的方法和事件介绍如下：

postMessage(message[, transfer]) - 向独立线程发消息，会触发独立线程的 DedicatedWorkerGlobalScope.message 事件，在独立线程中可以对 self.message 事件编程，收到的消息保存在事件函数的参数 event.data 中。message 参数不能省略，可以为任何数据类型，如果不重要可以采用 null 或 undefined。可选参数 transfer 为 Transferable 对象的数组，数组中的对象将被转移其所有权——在发出方的上下文中变得不可用，而在接收方的上下文中可用。目前只有 ArrayBuffer、MessagePort 和 ImageBitmap 对象的所有权允许被转移。

terminate() – 立即终止独立线程。

message/onmessage – 收到消息时触发，消息在 event.data 中，其中 event 是传递给事件处理函数的实参。

messageerror/onmessageerror – 收到不能反序列化的消息时触发。把 JS 对象转化为字符串称为序列化，相反，把字符串转化为 JS 对象称为反序列化。

onerror – 独立线程发生错误并冒泡到了主线程。

独立线程运行在 DedicatedWorkerGlobalScope 全局上下文中，DedicatedWorkerGlobalScope 对象继承 WorkerGlobalScope 对象，后者再继承 EventTarget，因此 DedicatedWorkerGlobalScope 对象的属性、方法和事件有这些（只能在线程中使用）：

self 属性 – 代表当前 DedicatedWorkerGlobalScope 对象的实例。

console 属性 – 浏览器控制台。因此再线程中继续可用 console.log()方法打印信息。

location 属性 – 表示与独立线程关联的 WorkerLocation 对象，一般为 Location 对象的子集。

navigator 属性– 表示与独立线程关联的 WorkerNavigator 对象，一般为 Navigator 对象的子集。

message/onmessage 事件– 线程收到消息时触发。

messageerror/onmessageerror 事件 - 解析同上。

postMessage(message[, transfer])方法 - 向主线程发消息，参数解析同上，只不过只有 MessagePort 和 ArrayBuffer 对象允许转移所有权。

close()方法 - 线程主动退出。

importScripts(str[str,str,...])方法 - 导入更多的外部 JS 文件，比如你在 foo.js 中编写了很多函数，可以这样导入 importScripts('foo.js')。

其他的方法还有 atob()、btoa()、clearInterval()、clearTimeout()、setInterval()和 setTimeout()，请参考网络或其他书籍，这里不再赘述。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

看例子：

```
<p id="result">10 的阶乘等于</p>
<button onclick="javascript:w.terminate();">终止独立线程</button>
……
<script>
   if (Worker) {//如果浏览器支持多线程
      var w = new Worker("factorial.js");
      w.onmessage = function (event) {
          document.getElementById("result").innerHTML = "10 的阶乘等于:
" + event.data;
      };
      w.onerror = function (evt) {
          console.log("出错"+event);
      }
      w.postMessage(10);
   }
</script>
```

代码中加了下划线的语句就是创建一个独立线程，新线程执行 factorial.js 程序计算一个数的阶乘，最后把计算结果反馈给主线程。factorial.js 文件的内容如下：

```
onmessage = function(event){
    postMessage(count(event.data));
}
function count(i){
    if(i<3) return i;
    else return(i*count(i-1));
}
```

两个线程之间通讯如图 6.13。

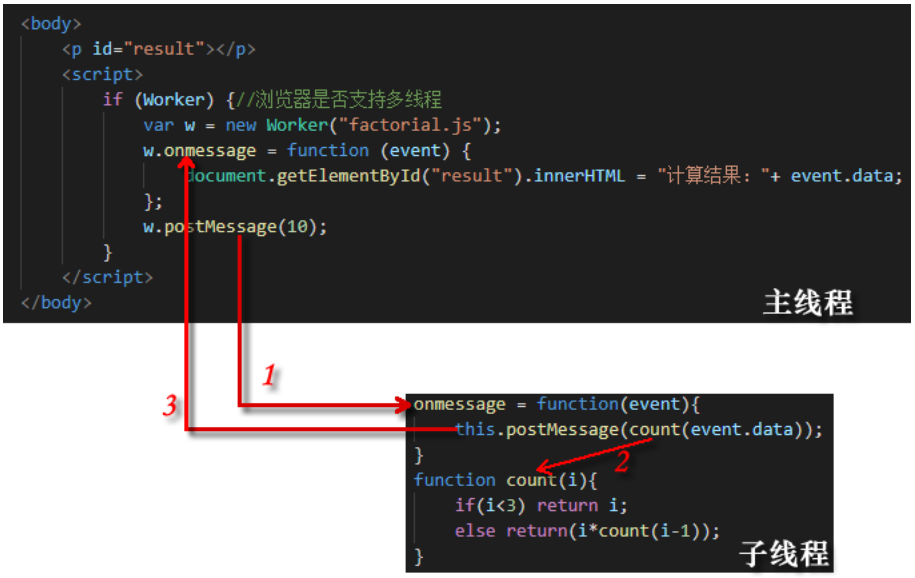


图 6.13 线程之间通讯

线程之间的通信采用 postMessages()方法，当收到对方发来的消息时会触发线程对象的 message 事件，消息内容被封装在 MessageEvent 对象中，该对象的 MessageEvent.data 属性存储了传递的消息内容。

再来看一个简单聊天的例子：

```
9   <body>
10      对方回答：<p></p>
11      <button type="button" onclick="ask()">询问对方</button>
12      <button type="button" onclick="terminal()">终止线程</button>
13
14      <script>
15          var w;
16          if (Worker) { //如果浏览器支持多线程
17              var w = new Worker("worker.js");
18              w.onmessage = function (event) {
19                  document.getElementsByTagName('p')[0].innerHTML = event.data;
20              };
21              w.onerror = function (e) {
22                  console.log(e.message, e);
23              }
24          } else {
25              alert("你的浏览器不支持多线程！ ");
26          }
27
28          function ask() {
29              w.postMessage("How are you?");
30          }
31
32          function terminal() {
33              w.terminate();
34          }
35      </script>
36  </body>
```

文件 worker.js 的代码如下：

```
onmessage = function(event){
    this.postMessage("I'm fine, Thank you.")
}
```

当发生错误时，Worker 对象会产生 onerror 事件，可以对此事件编程（参见上面的第 21 行代码），调用 Worker.terminate()方法终止“后台”线程（33 行代码）。

我们总结一下独立线程编程的一般步骤：

判断浏览器是否支持多线程（参见第 16 行代码）→创建线程对象（第 17 行代码）→给“收到对方消息”事件定义要执行的代码（第 18 到 20 行）→给“发生错误”事件定义要执行的代码（第 21 到 23 行）→给子线程发消息（第 29 行代码）→最后可能要终止子线程（第 33 行代码）。

主线程（前端 js 程序）允许实例化多个独立线程，这些独立线程之间可以通过管道（即 MessageChannel 接口）通信，其实任何线程之间都可以采用管道通信。

MessageChannel 接口有两个只读属性：port1（管道的一端口，类型为 MessagePort）、port2（管道的另一端口，类型也为 MessagePort）。由 port1 进的消息从 port2 出，反之亦然。

MessagePort 接口的方法有以下三个：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

postMessage() - 从本端口发送消息。

start() - 开始发送本端口队列中的消息（如果采用 MessagePort.onmessage 定义消息句柄，则隐式执行了 start()方法）。

close() - 关闭端口。

MessagePort 接口的事件有两个：message（有消息到达时触发）、messageerror（收到的消息无法反序列化时触发）。

参考图 6.14 所示的例子。

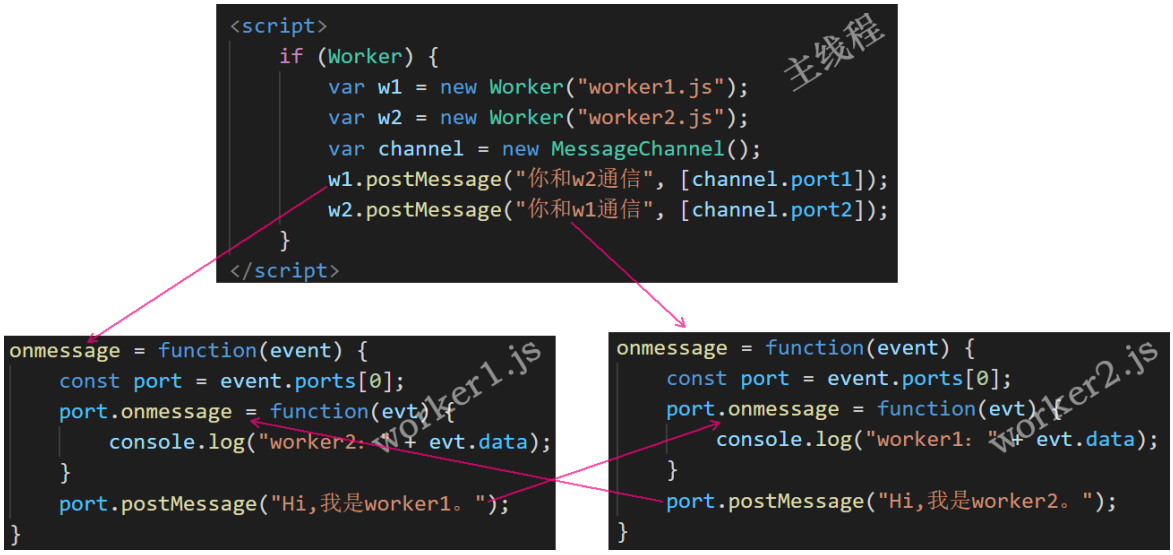


图 6.14 独立线程之间通过管道通信

window.ShareWorker 接口代表共享线程，意思是其他线程可以共享一个线程（目的是为了共享数据、计算、或者进行通信）。在实际的生产项目当中，共享线程常常用于同源的不同页面之间传递消息，而且这些页面只能在同一台电脑上的同一个浏览器打开才行，跨浏览器或者跨电脑的页面无法共享线程。 一个共享线程就像一个静态类，尽管多个主线程均采用类似“ws=new ShareWorker("myWorker.js")”这样的代码实例化同一个共享线程（共享线程里执行 myWorker.js），但是在浏览器的“后台”只有一个实例，每个主线程只是在共享线程实例上分别开一个类型为 MessagePort 的 port，在共享线程中可以对 port.onmessage 事件编程以便处理收到的消息。然后每个主线程均可通过 ws.port.postMessages("How are you?")方法向共享线程发消息，而共享线程通过每个主线程各自开设的 port 向主线程发消息（类似语句 port.postMessage("I'm fine, thank you.")），消息到达主线程时会触发 ws.port.onmessage 事件。请参考图 6.15 进一步理解。

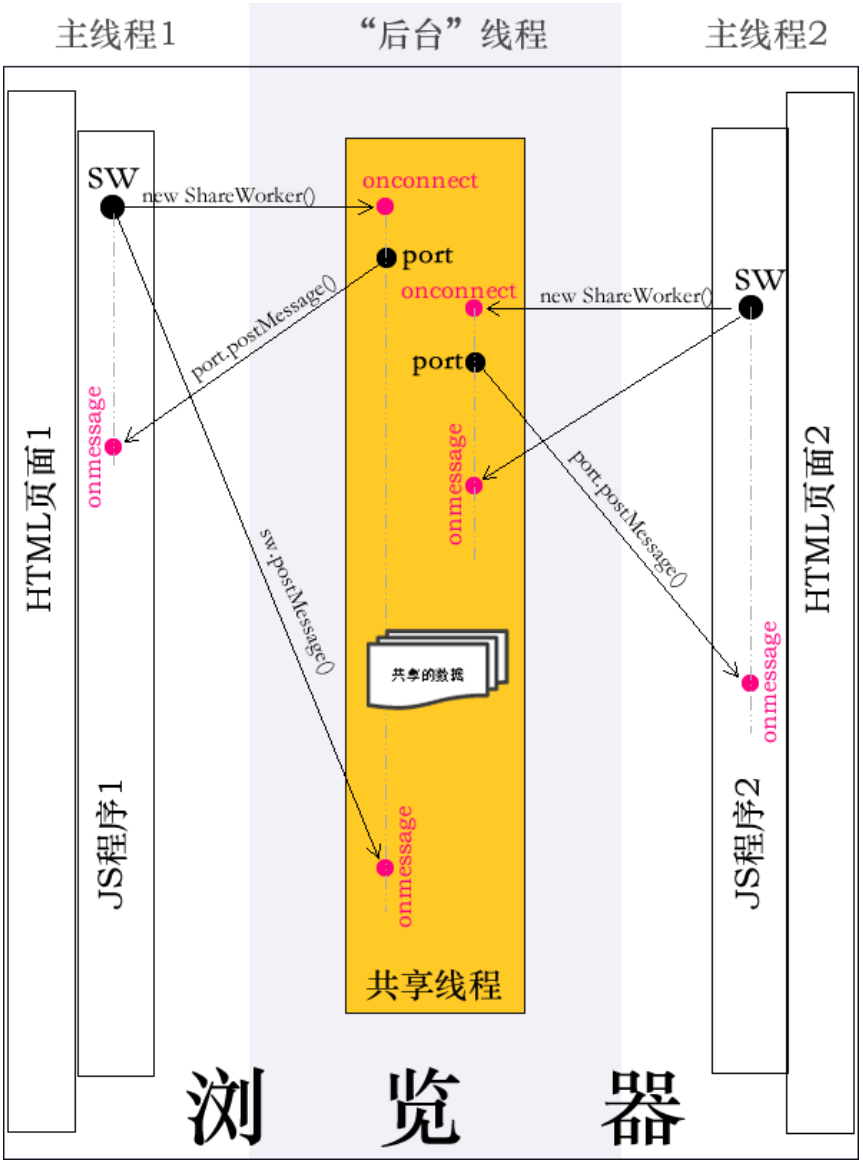


图 6.15 多个主线程共享一个 ShareWorker

SharedWorkerGlobalScope 对象为共享线程的全局执行上下文，因此共享线程的代码可以直接访问 SharedWorkerGlobalScope 对象的属性、方法和事件。此对象一个重要的属性是 console，这是只读属性，表示共享线程关联的控制台，类型为 console 对象。因此可以调用 console.log()方法在控制台上打印信息，但是此控制台非页面控制台，对于 chrome 浏览器，必须打开“chrome://inspect/#workers”才行，具体操作参见下文。SharedWorkerGlobalScope 对象的一个重要事件是 onconnect，当有主线程实例化本共享线程时触发，事件参数 event 对象的 ports 数组中保存了新开设的若干 port 对象（一般只有一个），因此 event.ports[0] 是有效的。SharedWorkerGlobalScope 对象的方法有 setTimeout()、setInterval()、clearTimeout()、clearInterval()、atob()、btoa()、importScripts()、close()等，这些方法与 window 对象的同名方法功能和用法一样。

我们接下来看一个完整例子，代码参见 chapter-6\main-worker1.html、chapter-6\main-worker2.html 和 chapter-6\shareworker.js 三个文件，然后用同一个浏览器打开 main-worker 1.html 和 main-worker

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

2.html 页面，然后两个页面之间可以收/发消息了。可以通过 chrome://inspect/#workers 调试共享线程，具体参见图 6.16。

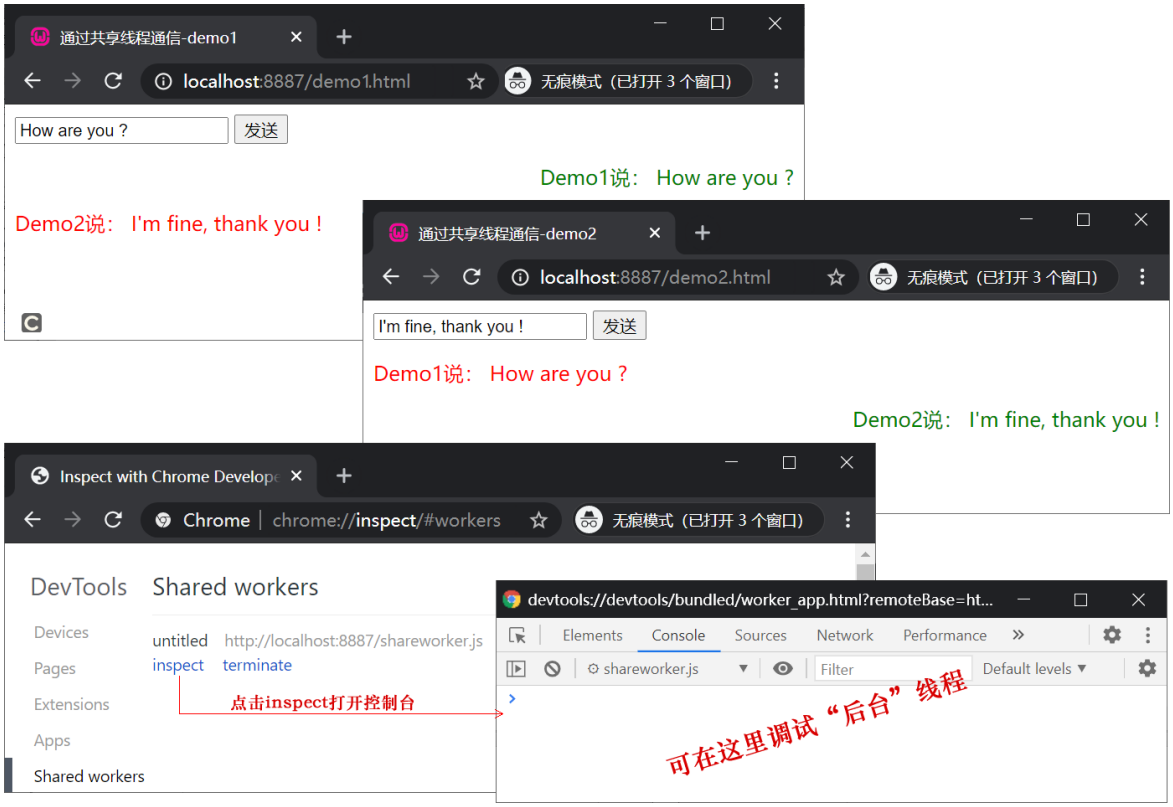


图 6.16 共享线程举例

实操练习

由后台线程计算第 n 个斐波拉契数，输入第几个斐波拉契数，由后台线程完成计算【提醒斐波拉契数：0、1、1、2、3、……，后面的每一个数等于前两个数的和】。

6.7 服务线程

主线程的 js 代码被直接嵌入在 HTML 文件中或者通过<script>元素的属性 src 引入外部 js 文件，主线程的执行会影响页面的渲染进度，其实浏览器渲染页面就是完成这三个任务：解析 HTML5 元素、应用 CSS 样式和执行主线程。而浏览器“后台”中的线程与主线程是并行执行的，常见的后台线程包括页面线程、共享线程和服务线程，页面线程和共享线程在上一小节已经介绍过，本小节详细介绍服务线程。

使用 window.navigator.serviceWorker 开发位于浏览器内部的服务线程——代理服务器，它能拦截客户端的请求、向客户端发送消息、向服务器发起请求等。实现离线资源缓存是服务线程最佳的应用场景，而早期实现类似功能的 window.applicationCache 技术即将被淘汰。之前讲过的 Web Worker 代表独立线程，但是此线程的生命期不会超过页面的生命期，当页面关闭时还未结束的线程被强行终止，而 Service Worker 也是独立线程，但是其生命期开始于注册，终于注销，中途与浏览器同命运——浏览器关闭服务线程就停止，浏览器启动服务线程就运行。一个源允许注册多个服务线程，但是各个服务线程的服务范围必须不同。window.navigator.serviceWorker 对象大量采用 JavaScript 语言的 Promise 对象，Promise 对象代表一个将来完成的异步操作，异步操作的结果要么成功要么失败，成功时会有一个结果值（简称为“确值”）。常见的用法为 promiseObject.then(sucesshandler, failHandler)，表示成功时调用函数 successHandler(result)，失败时调用函数 failHandler(error)函数。至于 Promise 对象详细的介绍请参阅网络或者其他书籍。参考图 6.17。

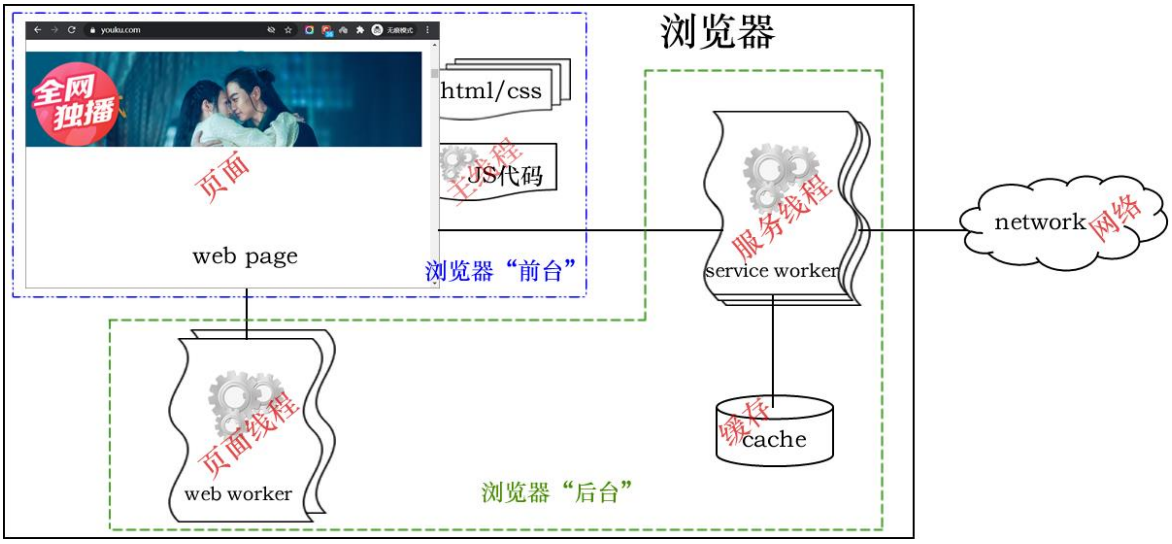


图 6.17 浏览器的“前/后台”

服务线程位于浏览器的渲染进程中，作为一种“后台”线程，在服务线程内不能访问页面 DOM 和浏览器的 Window 对象，而且同步机制也不能使用，只能使用异步方式。另外出于安全原因，只能在 https 协议中有效。

Service Worker 编程主要使用 ServiceWorkerContainer、ServiceWorkerRegistration、ServiceWorker 和 ServiceWorkerGlobalScope 对象，几个对象之间的关系见图 6.18 所示。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

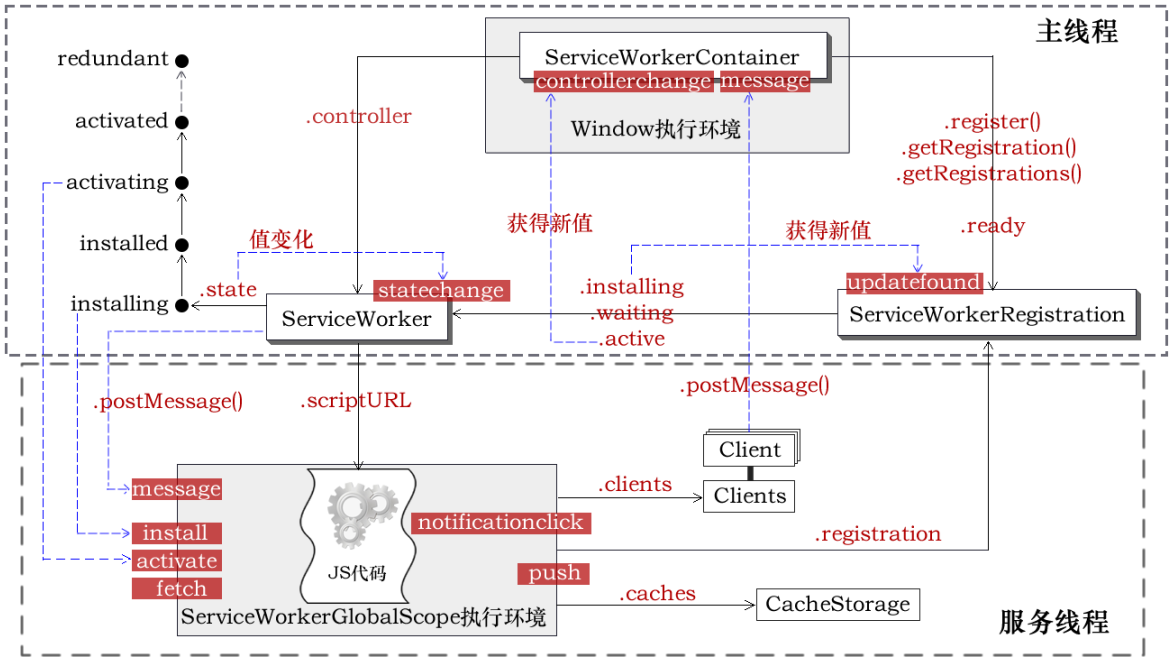


图 6.18 服务线程涉及的几个对象之间的关系

图 6.18 中红底白字部分为对象的事件，蓝色虚线箭头表示触发事件，比如 ServiceWorker.state 的值赋予"installing"时触发 ServiceWorkerGloblScope.install 事件，也触发 ServiceWorker.statechange 事件。对于各个对象的具体方法、属性和事件请参考后续相应章节。只有 ServiceWorkerGloblScope 对象在服务线程中使用，其他三个对象只在主线程中使用。

6.7.1 ServiceWorkerContainer

window.navigator.serviceWorker 属性返回一个 ServiceWorkerContainer 对象，此对象提供了若干个用于控制服务线程的方法，例如注册、更新和查询服务线程等。主线程采用 ServiceWorkerContainer 对象（通过 window.navigator.serviceWorker 属性）来操纵服务线程。

1、ServiceWorkerContainer 对象的方法

register(scriptURL[, options]) – 创建或更新由 scriptURL 程序所代表的服务线程，如果线程不存在就创建它，如果存在且代码不同就更新它，如果存在且代码没变就直接返回现存的 ServiceWorkerRegistration 实例。参数 scriptURL 指定具体的 js 代码文件，服务线程中就是执行此 js 程序。可选参数对象 options 定义了一些额外的注册选项，目前选项只有 scope，值为字符串，定义服务线程服务于页面的范围 URL。属性 scope 的值为相对于网站根目录的子目录，除非能修改 Web 服务器的配置 Service-Worker-Allowed，否则这个子目录只能是服务线程 js 代码文件（比如 serviceworker.js）所在的目录或其子目录，比如 register("/ws.js",{ scope: "/shop" })，但是 register("/shop/books/ws.js",{ scope: "/shop/food" })就不行，因为/shop/food 不是/shop/books，也不是/shop/books 的子目录，{ scope: "/shop/books" }也不行，因为"/shop/books"是"/shop/books/"的父目录。如果省略 options 参数，默认的服务范围就是服务线程 js 代码文件所

在的目录。register()方法返回一个 Promose 对象，成功时解析为一个 ServiceWorkerRegistration 接口实例，此后主线程通过它跟服务线程打交道。看例子：

```
    if ('serviceWorker' in navigator) {
        navigator.serviceWorker.register('/sw.js',{ scope: '/books' }).then(fun
ction (registration) {
            console.log('注册成功: ', registration);
        }, function (error) {
            console.log('注册失败: ', error);
        });
    } else {
        console.log('浏览器不支持服务线程! ');
    }
```

注册之后的服务线程会先后经历 download、insall 和 active 三个状态。当服务范围内的资源被访问时立即下载服务线程的 js 代码文件（downlod 状态），然后安装它（install 状态），但不一定马上激活它。如果是首次注册，则立即激活，如果是代码版本升级，则进入 waiting 状态，直到旧版本的服务线程不再服务任何资源（打开服务范围内的页面的浏览器 tab 都关闭）后，新版服务线程才进入激活状态（active 状态）。处于 active 状态的服务线程开始对外提供服务了。

getRegistration(scope) – 获取一个服务线程，scope 刚好落在此服务线程的服务范围内。本方法返回一个 Promise，成功时确值为代表服务线程的 ServiceWorkerRegistration 对象或者 undefined（没找到这样的服务线程）。

getRegistrations() – 返回与 ServiceWorkerContainer 对象关联的所有 ServiceWorkerRegistration 对象。

startMessages() – 开始接受服务线程发来的消息，以便尽早接受并处理各种消息。默认是页面渲染完毕或者给 onmessage 属性赋值后开始接受消息。在页面开始接受消息前，服务线程发来的消息排队等待。当采用 addEventListener("message",……)给 message 事件编程时，并没有隐式调用 startMessages()方法，这时如果你想今早接受消息，必须显式调用本方法。本方法无返回值。

2、ServiceWorkerContainer 对象的属性

controller – 值为 ServiceWorker 对象或者 null。如果值为 ServiceWorker 对象，则代表一个处于 activating 或 activated 状态的服务线程，并且本页面的 URL 落在此服务线程的服务范围之内。其他情况下，controller 属性的值都为 null，初始注册服务线程的页面，其 controller 也为 null。例如：

```
    if (navigator.serviceWorker.controller) {
        console.log(`控制本页面的服务线程是: ${navigator.serviceWorker.controller}`);
    } else {
        console.log('本页面没有被任何服务线程控制。');
    }
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

ready – 返回一个 Promise 对象，当本页面的服务线程的状态为 activating 或 activated 时，表示 Promise 对象所代表的异步操作成功完成，并确值为此服务线程对应的 ServiceWorkerRegistration 对象。如果本页面的 URL 没有落在服务线程的服务范围之内，那么 Promise 代表的异步操作永久等待。例如：

```
navigator.serviceWorker.ready.then(
    registration => console.log('我的服务线程是', registration.active)
);
```

3、ServiceWorkerContainer 对象的事件

message/onmessage – 当收到服务线程发来的消息时触发。

controllerchange/oncontrollerchange – 当本页面关联的 ServiceWorkerRegistration 对象获得了一个新的被激活的服务线程时触发，即 serviceWorkerRegistration.active 属性的值改变时触发，注意：由 null 改变成其他值不会触发（即第一次注册的 ServiceWorker 不会触发该事件）。

error/onerror – 当关联的服务线程出错时触发。

6.7.2 ServiceWorkerRegistration

主线程通过 navigator.serviceWorker.register()方法注册一个服务线程，该方法返回的 Promise 对象成功时确值为 ServiceWorkerRegistration 接口，服务线程的客户端通过 ServiceWorkerRegistration 对象与服务线程打交道。此对象的属性和方法介绍如下：

1、 ServiceWorkerRegistration 对象的属性

active - 返回一个状态为"activating"或"activated"的服务线程对象（ServiceWorker）。active 属性的初始值为 null，随着服务线程被安装和激活后，active 才正式指到一个服务线程对象，同时还会触发 ServiceWorkerGlobalScope.activate 事件（在服务线程中发生）和 ServiceWorkerContainer.controllerchange 事件（在主线程中发生）。

scope - 返回服务线程的服务范围 URL，参见前面有关 ServiceWorkerContainer.register()方法的内容。

installing - 返回一个状态为"installing"的服务线程对象，即 ServiceWorker.state= "installing"。本属性的初始值为 null，当服务线程的状态不再是"installing"时，本属性又被置为 null。

waiting - 返回一个状态为" installed"的服务线程对象，本属性的初始值为 null，当服务线程的状态不再是"waiting"时，本属性又被置为 null。。

pushManager - 引用一个 PushManager 接口对象，此对象提供了消息订阅/推送 API。

index – 引用一个 ContentIndex 接口对象，程序员可以使用此接口对象来索引离线资源。

2、 ServiceWorkerRegistration 对象的事件

updatefound - 当 ServiceWorkerRegistration.installing 属性被赋予一个服务线程对象时触发。刚开始 installing 属性的值为 null。

3、 ServiceWorkerRegistration 对象的方法

getNotifications([options]) - 返回一个 Promise，确值为 Notification 对象的列表——列表中的“通知”的顺序就是创建时的顺序。可选参数对象 options 定义更具体的过滤条件，目前只有一个属性 tag，值为字符串，例如 getNotification({ tag: 'user_alerts' })只返回包含'user_alerts'标记的通知。

showNotification(title[, options]) - 创建一个通知。参数 title 定义标题，可选参数对象 options 进一步配置通知的属性，允许的属性有：

actions：为 NotificationAction 对象类型的数组，当显示通知时允许用户执行的动作。一个 NotificationAction 对象包括三个属性：action（动作名称，类似<input>元素的 name 属性，用户点击的那个“动作”的名称会反馈给通知的主人——触发 ServiceWorkerGlobalScope.notificationclick 事件，事件对象 event.action 指明了用户点击的那个“动作”）、title（显示通知上的动作标题）、icon（动作的图标 URL）。目前只有 android 上的 chrome 和 opera 浏览器支持。

badge： 图片的 url，当空间不能显示整个通知时就显示此图片，目前只在 android 的 chrome 上有效果。

body：通知的具体内容，显示在标题下。

data： 通知上携带的任意类型的数据，比如用户点击通知后需要打开的页面 URL，data: { "url": "https://www.youduo.me" }。

dir： 定义文字方向，默认为 auto，可取值"ltr"或"rtl"。

icon： 通知上的图标的 url，一般是下图片，显示在标题和内容的侧边。

image： 通知上图片的 url。

lang： 定义语言。比如取值："en" - 英语、"zh" - 简体和繁体中文、"fr" - 法语、"ja" - 日语、"ru" - 俄语、"es" - 西班牙语、"ar" - 阿拉伯语。

renotify： 布尔值，true 表示被其他通知覆盖的通知需要再次通知，默认值为 false。

requireInteraction： 是否需要用户交互才关闭通知，默认值为 false，表示会自动关闭。

silent： 布尔值，true 表示无声音无振动，默认值为 false。

tag： 一个标记通知的字符串，标记通常用于区分不同类型的通知，通过标记可以查找、替换和删除相指定的通知。

timestamp： 时间戳，表示从 1970 年 1 月 1 日以来的毫秒数。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

vibrate： 定义物理设备的振动（比如手机振动），为整数类型的数组。比如[2000,1000,3000]表示振动 2 秒，静止 1 秒，再振动 3 秒，之后不再振动了。

showNotification()方法返回一个 Promise，执行完毕后确值为 undefined。

看例子：

```
var options = {
    body: "出大事了，美国总统川普的推特、脸谱、油管账户被永久封号！",
    requireInteraction: true,
    actions: [{
        action: 'shock',
        title: '震惊',
        icon: "/icons/warning.ico"
    }, {
        action: 'soso',
        title: '不奇怪'
    }],
    silent: true,
    icon: "/icons/shock.jpg"
};

Notification.requestPermission(function (result) {
    if (result === 'granted') {
        navigator.serviceWorker.ready.then(function (registration) {
            registration.showNotification('天下奇闻', options);
        });
    }
});
```

在 windows 的桌面上弹出的通知效果如图 6.19。



图 6.19 系统通知举例

update() - 尝试更新服务线程——从 Web 服务器上下载服务线程的 js 代码，然后按字节比较，如果不同，就安装它。本方法返回 Promise，成功后会确值为 ServiceWorkerRegistration 对象。

unregister() - 注销服务线程，返回 promise，结果确值为一个布尔类型的值，如果注销的服务线程不存在，那么解析的布尔类型为 false，否则为 true。但值为 true 并不表示服务线程已经被注销掉，服务线程需要完成所有正在进行的操作之后才被正式注销。

6.7.3 ServiceWorker

在主线程中，ServiceWorker 对象代表一个服务线程——相当于一个指针，指到具体的一个服务线程，允许多个主线程中的 ServiceWorker 对象指向同一个服务线程。

1、 ServiceWorker 对象的属性

scriptURL - 返回服务线程中执行的 js 文件的 URL，此 URL 与页面是同源的。例如 http://localhost:8887/serviceworker.js。

state - 服务线程的当前状态，可能的值有"installing"（正在安装）、"installed"（安装完毕）、"activating"（正在激活）、"activated"（激活完毕）或"redundant"（已被替换）。当 state 的值为"installing"时会触发 ServiceWorkerGlobalScope.install 事件，在服务线程中可以给 self.install 事件定义处理函数，只有当事件处理函数返回时，state 的值才会变为"installed"。当 state 的值为"activating"时会触发 ServiceWorkerGlobalScope.activate 事件，同样可以在服务线程中对 self.activate 事件编程，当事件函数返回后 state 的值变为"activated"。例子参见后面介绍 ServiceWorkerGlobalScope 对象的章节。

2、 ServiceWorker 对象的事件

statechange/onstatechange - 只要 state 属性的值发生变化就会触发本事件。

3、 ServiceWorker 对象的方法

postMessage(message[, transfer]) - 向服务线程发消息，会触发服务线程的 ServiceWorkerGlobalScope.message 事件，在服务线程中可以对 self.message 事件编程，收到的消息保存在事件函数的参数 event.data 中。message 参数不能省略，可以为任何数据类型，如果不重要可以采用 null 或 undefined。可选参数 transfer 为 Transferable 对象的数组。本方法无返回值。看例子：

```
var messageChannel = new MessageChannel();
messageChannel.port1.onmessage = event => console(event.data);
navigator.serviceWorker.controller.postMessage(" 我 们 通 过 管 道 交 流 吧
", [messageChannel.port2]);
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.7.4 ServiceWorkerGlobalScope

服务线程工作在 ServiceWorkerGlobalScope 全局执行上下文中，因此服务线程的 js 代码允许直接使用 ServiceWorkerGlobalScope 对象的属性、方法和事件，就像工做在 Window 全局执行上下文中的主线程那样，能直接使用 Window 对象的属性、事件和方法。ServiceWorkerGlobalScope 继承 WorkerGlobalScope 对象，因此除了父对象的属性、方法和事件之外，额外增加了如下一些属性、事件和方法。

1、ServiceWorkerGlobalScope 对象的属性

caches - 只读属性，返回一个 CacheStorage 类型的接口，此对象与本服务线程关联在一起，可用于存储离线资源。CacheStorage 接口提供了很多方法，比如打开缓存、查询缓存、删除缓存等，也就是说 CacheStorage 专门用来操纵一系列 Cache 对象，而 Cache 对象存储具体的数据（提供了增删改查方法）。CacheStorage 对象的具体方法参见下表 6.1。

表 6.1 CacheStorage 对象的方法

方法	说明
open(cacheName)	打开缓存。返回 Promise，如果成功则确值成匹配 cacheName 的 Cache 对象，参数 cacheName 为缓存的名字。如 caches.open("my1").then(cache=>{……});
delete(cacheName)	删除缓存。返回 Promise，如果成功则确值成 true，参数 cacheName 为缓存的名字。如 caches.delete("my1").then(result => {alert("success!"});
match(request[, howto])	以 request 参数为键搜索缓存中的 Response。返回 Promise，无错则确值为 Response（搜到）或 undefined（没搜到）。参数 request 为 Request 对象或 URL 字符串，可选参数 howto 是对象，定义如何搜索，此对象中允许出现的属性有：ignoreSearch（布尔值，是否忽略 URL 中“?”后面的字符串，默认值 false）、ignoreMethod（布尔值，是否忽略获取资源的方法，常用的方法是 GET，默认值为 false）、ignoreVary（布尔值，是否验证 Response 中的 VARY 头部，默认值为 false）、cacheName（在指定的缓存中搜索）。

	例 如 : caches.match(event.request,{cacheName: "my1"}).then(…)
has(cacheName)	是否存在指定名称的缓存。返回 Promise，如果存在则确值为 true。
key()	返回 Promose，无错则确值为：由全部缓存名字组成的数组。

Cache 对象专门存储“Request/Response”键值对，其提供的方法见表 6.2。

表 6.2 Cache 对象的方法

add(request)	获取对应的 Response 并存储 Request/Response 对。参数 request 为 Request 对象或者 URL 字符串，返回一个确值为 undefined 的 Promise 对象。 例如：cache.add('/sw-test/index.html')
addAll(request[])	获取对应的 Response 并存储若干 Request/Response 对。参数 request 为 Request 对象或者 URL 字符串的数组，返回一个确值为 undefined 的 Promise 对象。 例如：cache.addAll(['/index.html', '/js/sw.js'])
put(request, response)	直接存储 request/response 对。返回一个确值为 undefined 的 Promise 对象。
match(request[,howto])	参见 CacheStorage 对象的同名方法。
matchAll([request[, howto]])	返回一个 Promise，如果无误则确值为 Response 的数组。参数都可以省略。
delete(request[,howto])	删除 request/response 对。返回 Promise，如果删除成功确值 true，否则确值 false。例如： cache.delete('/images/image.png').then(…)
keys([request[,howto]])	返回 Promise，确值为 Request 类型的数组。在 Cache 中，键就是 Request。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

registration - 只读属性，返回本服务线程关联的 ServiceWorkerRegistration 对象。

clients - 只读属性，返回与服务线程关联的 Clients 对象，Clients 对象操纵若干个具体的 Client 对象，Client 对象就是服务线程的客户端，客户端有三种类型："window"（页面客户端）、"worker"（独立线程客户端）和"sharedworker"（共享线程客户端），页面客户端就是浏览器打开的一个页面，其 URL 落在服务线程的服务范围内。Clients 与 Client 的关系类似前面介绍的 Caches 与 Cache 的关系。Clients 对象有这些方法：

get(id)：获取指定 id 的 Client 对象。返回 Promise，确值为 Client 对象或者 undefined。参数 id 为字符串。

matchAll([howto])：返回 Promise，执行完毕后确值为 Client 对象数组。可选参数 howto 定义返回那些 Client，允许的属性有 includeUncontrolled（布尔值，true 则返回全部的同源 Client，false 只返回本服务线程的 Client。默认值为 false）、type（返回的 Client 的类型，可取值"window"（默认值）、"worker"、"sharedworker"或"all"）。

openWindow(url)：创建一个顶层 Window 客户端，并在其中打开 url 页面。成功时 Promise 确值为 WindowClient 对象。

claim()：一个处于激活状态的服务线程把自己设置为服务范围内所有客户端的控制者（服务者），并触发 serviceWorker. controllerchange 事件。刚开始浏览注册服务线程的页面时，页面并没有被服务线程控制，需要再次刷新页面后，服务线程才成为页面的控制者，但如果调用 claim()方法就立即成为控制者（页面不用刷新）。本方法返回 Promise，并确值为 undefined。看代码：

```
    self.addEventListener('activate', event => {
        event.waitUntil(clients.claim());
    });
```

event.waitUntil(promise)函数的参数为 Promise 对象类型，本例中此函数的目的是服务线程保持 activating 状态（即 ServiceWorker.state="activating"），直到 clients.claim()返回的 Promose 成功确值。

Client 对象的方法 postMessage(message[, transfer])向客户端发消息，可选参数 transfer 为自定义对象，会触发主线程的 navigator.serviceWorker.message 事件。Client 对象有三个只读属性：id（客户端的 UUID）、type（类型，可能的值有"window"、"worker"或"sharedworker"）、url（客户端对应的 URL）。WindowClient 对象表示页面客户端（类型为"window"的 Client），它继承 Client 对象，增加了下面的两个属性和两个方法：

focused 属性如果为 true，表示页面处于输入焦点；visibilityState 属性指明页面的可见性，可能的值有"hidden"（隐藏）、"visible"（可见）或"prerender"（渲染还没完成）。focus()方法使页面处于输入焦点；navigate(url)在客户端中加载 url 页面，这两个方法都返回 Promise，成功时确值为 WindowClient。

2、ServiceWorkerGlobalScope 对象的事件：

install/oninstall - 当 ServiceWorkerRegistration.installing 属性获得一个新服务线程对象（ServiceWorker）时触发，此时 ServerWorker.state="installing"。一般在本事件的处理函数中保存离线资源。看例子：

```
//sw.js:
self.addEventListener('install', (event) => {
     event.waitUntil(
          caches.open('v1').then(function (cache) {
              return cache.addAll([
                  '/sw-test/',
                  '/sw-test/index.html',
                  '/sw-test/style.css',
                  '/sw-test/app.js',
                  '/sw-test/star-wars-logo.jpg',
                  '/sw-test/gallery/',
                  '/sw-test/gallery/snowTroopers.jpg'
              ])
          })
     );
});
```

event.waitUntil()函数确保服务线程的状态一直为 installing，直到 caches.open()函数返回的 Promise 成功确值之后，状态才变为 activated。

activate/onactivate - 当 ServiceWorkerRegistration.active 属性被赋予一个新服务线程对象（ ServiceWorker ）时触发，此时 ServerWorker.state="activating"，刚开始 ServiceWorkerRegistration.active=null，随着服务线程状态的变换，最终 ServiceWorkerRegistration.active 指向了那个服务线程。在这里，您可以完成诸如释放缓存，注册事件处理程序等工作。

fetch/onfetch – 当向服务范围内的资源发起 fetch()时触发，比如浏览处于服务范围内的页面。在事件处理函数中，我们可以拦截服务范围内所有资源的请求，然后根据某种策略，使用缓存的内容或者向 Web 服务器请求相应资源，最后反馈给客户端。比如下面的例子就是直接反馈缓存的内容：

```
//sw.js:
self.addEventListener('fetch', function (event) {
     event.respondWith(caches.match(event.request));
});
```

而下面的例子是从 Web 服务器请求资源并反馈给客户端：

```
//sw.js:
self.addEventListener('fetch', function (event) {
     event.respondWith(fetch(event.request));
});
```

message/onmessage – 当收到消息时触发。客户端可通过 ServiceWorker.postMessage()向服务线程发消息，服务线程可以通过 event.source.postMessage()应答，或者通过 Client.postMessage()主动向某个客户端发消息。看例子：

```
    //客户端的代码：主动问候。
    navigator.serviceWorker.addEventListener('message', event => {
        console.log(`服务线程发来的消息: ${event.data}`);
    });
    navigator.serviceWorker.ready.then(registration => {
        registration.active.postMessage("您好，服务线程！");
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
    });

    //服务线程的代码：应答。
    self.addEventListener('message', event => {
        console.log(`客户端发来的消息：${event.data}`);
        event.source.postMessage("能为您服务是我的荣幸！");
    });
```

如果客户端直接用 navigator.serviceWorker.controller.postMessage()发消息，最好先判断 navigator.serviceWorker.controller 不为 null。例子中采用 navigator.serviceWorker.ready 更为稳妥，大家回顾一下前面对 ServiceWorkerContainer.ready 属性的解释，就能明白其中的道理了。

push/onpush – 当收到推送消息时触发。推送消息与普通的消息不同，推送消息需要事先订阅才行。

pushsubscriptionchange/onpushsubscriptionchange – 推送订阅发生变化时触发，比如订阅到期，或被取消。

contentdelete/oncontentdelete – 通过浏览器内建的界面删除一条 content index 记录时触发，注意：通过 ContentIndex.delete()方法删除记录不会触发本事件。

notificationclick/onnotificationclick - 当用户点击通知时触发，这个通知是由 ServiceWorkerRegistration.showNotification()方法发送的。

下面的例子完成任务：当用户点击通知时打开页面 notice-content.html。

```
self.addEventListener('notificationclick', function (event) {
     console.log('用户点击了通知：', event.notification.tag);
     event.notification.close();
     event.waitUntil(clients.matchAll({ type: "window" }).then( clientList => {
          for (var i = 0; i < clientList.length; i++) {
              var client = clientList[i];
              if (client.url == '/notice-content.html' && 'focus' in client)
                  return client.focus();
           }
           if (clients.openWindow)
              return clients.openWindow('/notice-content.html');
     }));
});
```

notificationclose/onnotificationclose – 当用户关闭通知时触发。

sync/onsync – 当服务线程的客户端调用 SyncManager.register()方法时触发。

3、ServiceWorkerGlobalScope 对象的方法：

skipWaiting() – 迫使处于 waiting 状态的服务线程转换为 active 状态，结合调用 Clients.claim()方法，能使一个新版服务线程马上起作用。返回 Promise，并确值为 undefined。看例子：

```
    self.addEventListener('install', event => {
        self.skipWaiting();
        //执行其他任务
    });
```

fetch(resource [, option]) – 从网络请求资源，不同于 XMLHttpRequest，fetch()方法返回一个 Promise，当收到响应时确值为 Response 对象——请求的响应，只有网络错误或者请求的资源不存在才会拒绝，所以成功确值并不代表一定获取资源成功，因此在 then 处理函数中最好判断 HTTP 是否错误（检查 Response.ok 和 Response.status）。收到响应之后就可以返回给客户端，或者缓存都行。参数 resource 为资源的 url 或者拦截到的客户端的 Request。可选参数 option 为对象，此对象可用的属性有：

method：定义资源请求方法，如 GET、HEAD、POST。默认值为 GET。

headers：请求的头部信息，包裹在 Headers 对象中。例如：

```
    headers: {
        "Content-type": "application/x-www-form-urlencoded; charset=UTF-8"
    }
```

body：携带的请求数据，比如 POST 请求时携带的登录表单数据。GET 和 HEAD 方法不需要 body。比如：body: 'login=zsan&pass=123456'

mode：定义请求的模式，比如"cors"（允许跨域请求。默认值）、"no-cors"（请求那些没有 CORS 头部的源，会导致要一个不确定的响应。很少使用）或"same-origin"（只能同源请求）。

credentials：是否发送 cookie，可取值"omit"（不发送）、"same-origin"（默认值。同源请求时发送，跨域请求时不发送）或"include"（发送）。

cache：定义缓存策略。可取值"default"（如果缓存中存在且是最新的，就直接从缓存中反馈，否则从网络请求并反馈，同时更新缓存）、"no-store"（直接从网络请求，也不更新缓存）、"reload"（直接从网络请求，并更新缓存）、"no-cache"（将服务器资源跟本地缓存进行比较，有新的版本才使用服务器资源并更新缓存，否则使用缓存）、"force-cache"（如果缓存命中，就使用缓存，否则从网络请求并更新缓存）、"only-if-cached"（只使用缓存，如果没命中，就反馈 504 错误代码。只能结合使用 mode="same-origin"）。例如：

```
fetch("some.json", { cache: "reload" }).then(function (response) { /* 进一步处理 response */ });
```

redirect：如何处理跳转，可能的值有"follow"（跟从跳转，这是默认值）、"error"（如果发生跳转，fetch()就报错）、"manual"（response.url 属性设置为新的 URL，response.redirected 属性置为 true，由程序员决定如何处理跳转）。

referrer：设置 HTTP Referer 头部。默认地，Referer 头部包含了页面的 URL，但有时出于安全考虑，不希望页面的 URL 出现头部中，这时你可以通过 referer 硬性设定头部内容。referer 可为任何字符串或空字符串，例如 fetch("/page",{ referer: "" })。

referrerPolicy：告诉浏览器设置 HTTP Referer 头部的策略，可能的策略有"no-referrer-when-downgrade"（默认值。发送完整的 Referer 头部，但从 https 到 http 时不发送）、"no-referrer"（不发送 Referer 头部）、"origin"（只包含源，而不是完整的 URL，比如 http://www.youduo.me，而不是 http://www.youduo.me/myshop.html）、"origin-when-cross-origin"（同源时发送完整 Referer 头部，跨域时只包含源）、"same-origin"（同源时发送完整 Referer 头部，跨域时不发送）、"strict-origin"（https 到 http 时不发送 Referer 头部，其他情况只包含源）、"strict-origin-when-cross-origin"

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

（同源时发送完整的 Referer 头部，跨域时只包含源，https 到 http 不发送）、"unsafe-url"（总是发送 wan 正的 Referer 头部）。最后用表 6.3 总结一下。

表 6.3 Referer 头部策略

策略	同源	跨域	HTTPS→HTTP
"no-referrer"	-	-	-
"no-referrer-when-downgrade" or "" (默认值)	完整	完整	-
"origin"	源	源	源
"origin-when-cross-origin"	完整	源	源
"same-origin"	完整	-	-
"strict-origin"	源	源	-
"strict-origin-when-cross-origin"	完整	源	-
"unsafe-url"	完整	完整	完整

integrity：完整性检查，支持的 hash 函数有 SHA-256、SHA-384 和 SHA-512。比如我们知道下载的文件的 SHA-256 的值为“abcdef”（实际上很长），这时的代码如下：

```
fetch('http://site.com/check.doc', { integrity: 'sha256-abcdef' });
```

这样 fetch()方法会对 check.doc 文件作 SHA-256 计算，计算的结果如果不等于“abcdef”就报错。

keepalive：默认为 false，如果为 true，那么调用 fetch()方法的页面关闭后允许请求任务继续直到完成。例如：

```
    window.onunload = function () {
        fetch('/analytics', {
            method: 'POST',
            body: "<用户浏览本页面的行为统计数据>",
            keepalive: true
        });
    };
```

用户关闭页面触发 onunload 事件，然后调用 fetch()方法向 Web 服务器提交统计信息，但在请求完成前页面已经关闭了，keepalive=true 告诉浏览器请求转入后台继续进行，直至完成。

signal：允许取消 fetch()方法发起的请求，signal 的值为 AbortSignal 接口对象，然后我们就可以通过 AbortController 对象执行“取消”动作。看例子:

```
    let controller = new AbortController();
    fetch('/verylargevideo.mp4', { signal: controller.signal });
```

fetch()方法监听 signal 上的 abort 事件。接下来可通过下面的代码触发 abort 事件，进而导致 fetch()方法取消资源请求，promise 会 rejects 错误 AbortError：

```
    controller.abort();
```

6.7.5 应用场景

在介绍完了与服务线程有关的几个对象之后，我们举几个实际的例子来贯穿上文介绍的知识点。服务线程调式方法：谷歌浏览器打开“chrome://inspect/#service-workers”，火狐浏览器打开“about:debugging#/runtime/this-firefox”，按 F5 刷新页面，按 Ctrl+F5 彻底刷新页面（绕开浏览器的缓存直接从网络请求资源）。

6.7.5.1 离线资源

网站文件如图 6.20 所示，我们注册一个服务线程，服务线程中执行 serviceworker.js，注册服务线程的页面由 serviceworker.html、serviceworker.css 和 mainclient.js 组成，服务线程的服务范围为"/subpage/"，即此目录中的文件要离线缓存。所有文件放在随书配套资源 chapter-6\offlines 目录中。



图 6.20 离线资源举例 1

把这些文件上传到 Web 服务器，然后参考下图先打开“chrome://inspect/#service-workers”，在浏览器的另外一个标签页上浏览 serviceworker.html，并且进入开发者工具，点击“注册服务线程”按钮注册一个服务线程。在浏览器的第三个标签页上打开 subpage/index.html 页面，多次刷新此页面看看由什么不同的效果，最后断开网络，再次按 Ctrl+F5 刷新，看看能否看到图片。注销服务线程后，要等到服务线程服务的所有页面关闭之后，服务线程才真正退出。参考图 6.21。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

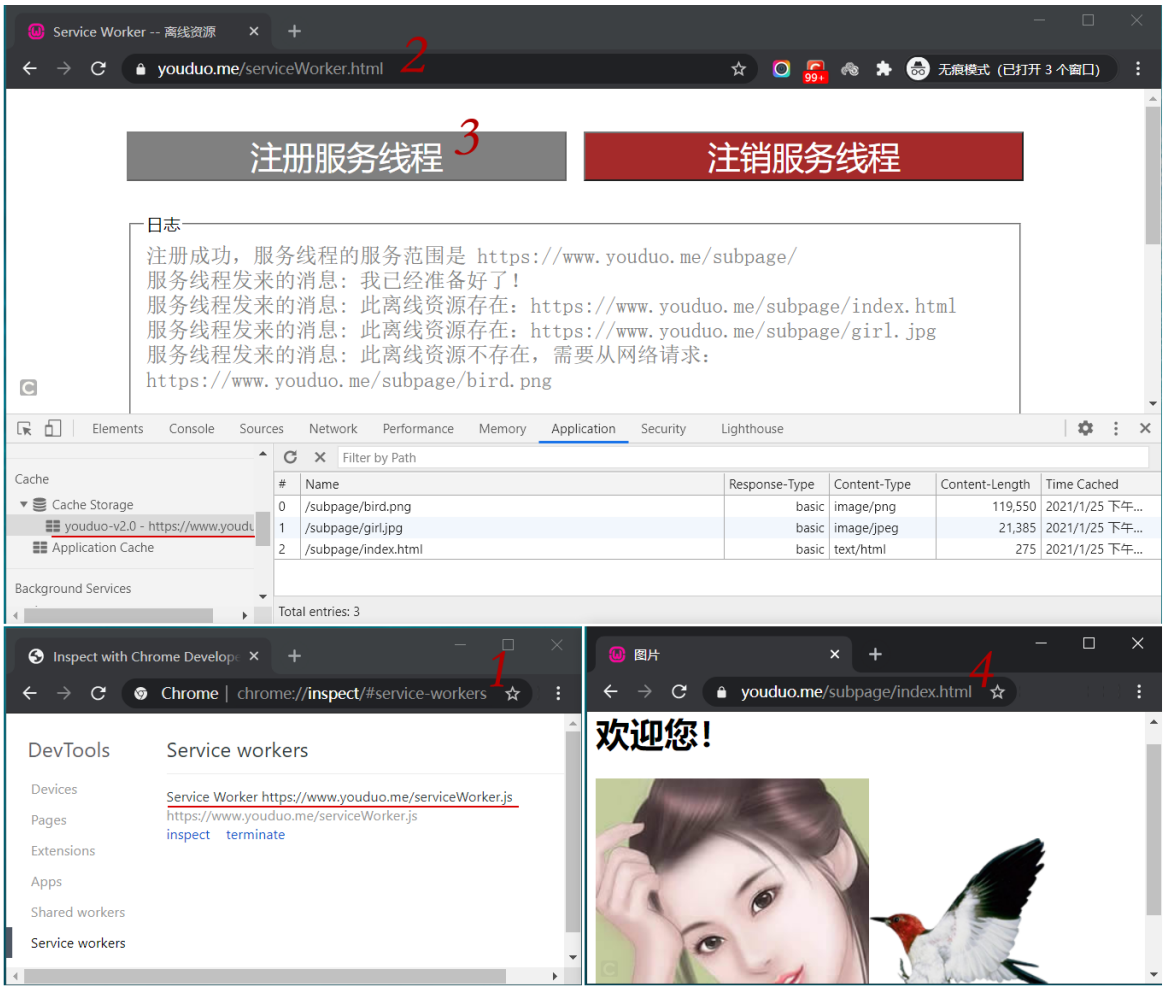


图 6.21 离线资源举例 2

serviceworker.html 文件比较简单，两个按钮，加上一个<textarea>元素，引入 mainclient.js 和 serviceworker.css。

点击“注册服务线程”后执行如下一行代码：

```
navigator.serviceWorker.register('/serviceWorker.js', { scope: '/subpage/' }).t
hen(successHandler, errorHandler);
```

如果注册成功就会调用 successHandler 函数：

```
function successHandler(registration) {
    ...
    setTimeout(() => {
        if (registration.active) {
            registration.active.postMessage("您准备好了吗？");
        }
    }, 200);
    navigator.serviceWorker.addEventListener('message', ... );
}
```

向服务线程发一条消息，这样以后服务线程就可以按照原路反馈消息。但是本例子不会触发 ServiceWorkContainer.controllerchange 事件，且 ServiceWorkContainer.ready 永远处于 pendding 状

态，因为/serviceWorker.html 不在服务线程的服务范围之内，ServiceWorkContainer.controller 永远等于 null。

注销时采用如下代码获取服务线程对应的 ServiceWorkerRegistration 对象：

```
navigator.serviceWorker.getRegistration("/subpage/").then(registration=>{...});
```

还是不能采用 ServiceWorkContainer.ready，原因一样，注册页面没有服务线程。

至于服务线程 serverWorker.js，在 self.install 事件处理函数中缓存资源（即/subpage/目录中的几个文件），在 self.activate 事件处理函数中删除旧版本的 cache。而针对 self.fetch 事件，本例采取的策略是：如果缓存命中，就直接把缓存资源反馈给客户端，否则从网络请求资源后一方面反馈给客户端，另一方面保存到缓存中。在 self.message 事件处理函数中，采用 event.source.postMessage(…)应答，而且把 event.source 保存到 client 变量中，这样以后就可以随时采用 client.postMessage()向主线程发送消息了。

```
self.addEventListener('message', event => {
    console.log(`客户端发来的消息: ${event.data}`);
    client = event.source;
    event.source.postMessage("我已经准备好了！");
});
```

6.7.5.2 消息推送

在被注销之前，服务线程随浏览器的启动而运行，而不管它服务的客户端是否存在。因此利用服务线程推送系统通知就是很好的应用场景，比如油管（youtube.com）会从电脑桌面右下角弹出一个通知，告知你订阅的某个频道发布了新视频。

消息推送涉及浏览器、主线程、服务线程、应用服务器（比如 Web 服务器）和推送服务器，整个流程见图 6.22 所示。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

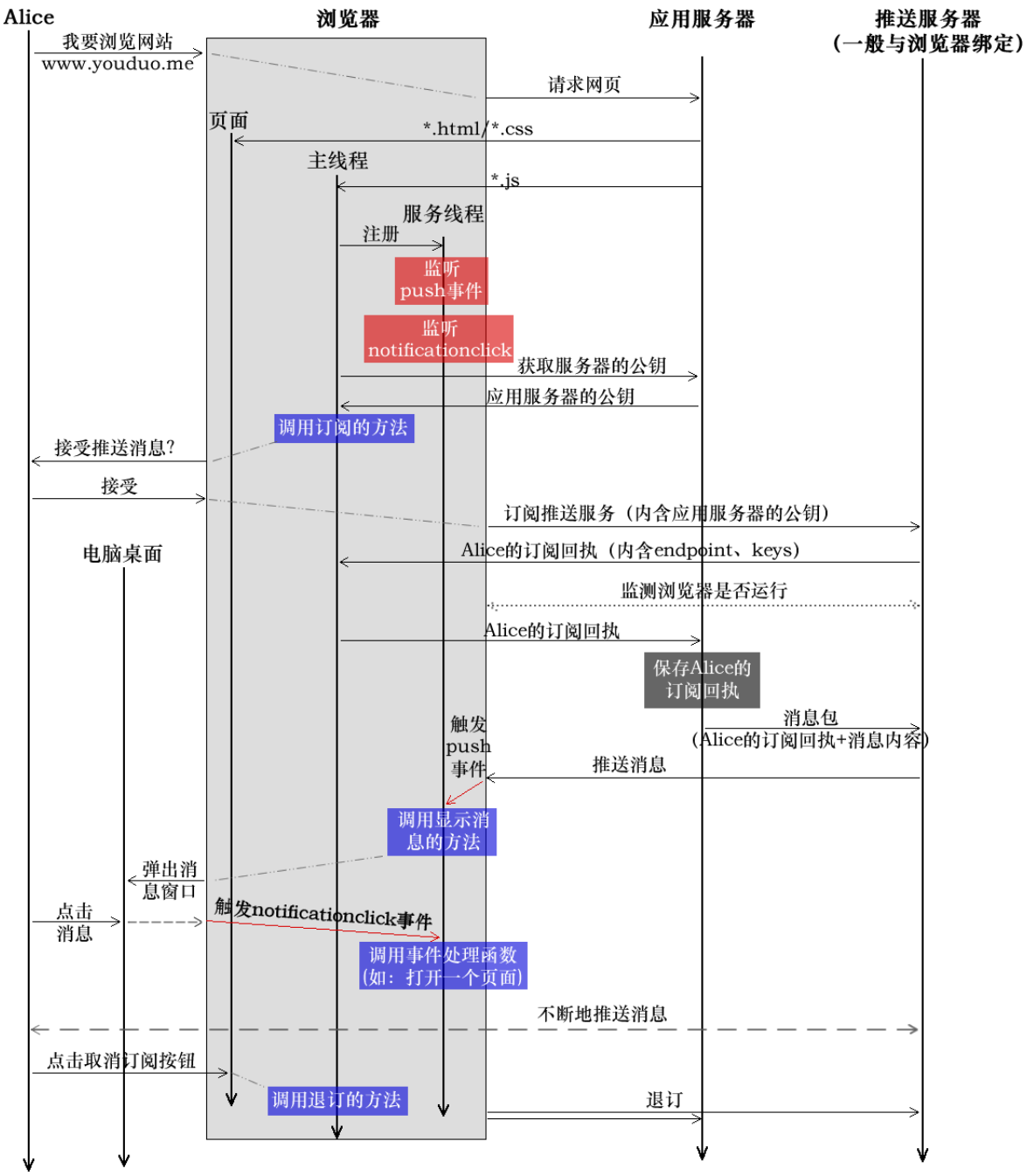


图 6.22 消息推送举例 1

用户选择什么样的浏览器，间接决定了采用哪个第三方的推送服务器，程序开发人员需要完成前端（页面、主线程、服务线程）和后端（运行在应用服务器上的程序）的编码工作。订阅之后，推送服务器时时监测浏览器是否运行，只有运行时才推送消息。

编程思路：在主线程中使用 ServiceWorkerRegistration.pushManger 属性订阅消息，在服务线程中的 self.push 事件处理函数中采用 self.registration.showNotification()方法显示推送过来的消息，后端可采用 node.js 的库 web-push 发送消息。

前面提到 ServiceWorkerRegistration.pushManager 属性返回一个 PushManager 对象，此对象用于消息推送——与消息推送服务器打交道。此对象的属性、方法介绍如下：

supportedContentEncodings 属性 – 字符编码的数组，只有数组中的字符编码可以用来编码消息内容。

getSubscription()方法 – 获取一个已经存在的订阅，返回 Promise 对象，如果订阅存在就确值为 PushSubscription 对象（内含具体的订阅信息），否则确值为 null。

subscribe([options])方法 – 浏览器向推送服务器订阅消息推送服务。可选参数 options 为对象，通过它可以指定更多的配置项：userVisibleOnly（布尔类型，true 指示返回的推送订阅将仅用于用户可见的消息，有的消息是用户不可见的）、applicationServerKey（应用服务器的公钥，推送服务器以此来识别应用服务器，非法的应用服务器不允许给推送服务器发消息）。由于目前的主流浏览器都与推送服务器绑定，比如 chrome 绑定 FCM，firefox 绑定火狐公司自己的推送服务器，所以参数 options 一般是不能省略的。本方法返回 Promise 对象，成功执行后确值为 PushSubscription 对象——订阅回执。

permissionState([options])方法 – 返回 Promise，确值的可能值有'prompt'（每一次消息到达时询问用户是否显示）、'denied'（不能显示）或'granted'（允许显示）。可选参数 options 与 subscribe()方法的一样。

看例子：

```
navigator.serviceWorker.register('serviceworker.js');
navigator.serviceWorker.ready.then(
  function (serviceWorkerRegistration) {
    var options = {
      userVisibleOnly: true,
      applicationServerKey: "xxxxxxxxxyyyyyyyyzzzzzzz"
    };
    serviceWorkerRegistration.pushManager.subscribe(options).then(
      (pushSubscription) => console.log(pushSubscription.endpoint)
    )
  }
);
```

订阅回执就是 PushSubscription 对象，下面是一个回执样例：

```
{
    endpoint: "https://updates.push.services.mozilla.com/wpush/v2/gAAAAABgE…",
    keys: {
          auth: "FrWJQ1iF0tAhrSJOaCHHLQ",
        p256dh: "BPpksV3sW51n3oaOVICvacL9BPEm9n5M-MKqkp7usfm993t…"
    }
}
```

此对象包含的属性和方法介绍如下：

endpoint – 为字符串类型的只读属性，与订阅关联的调用端点，推送服务器就是通过它向用户的服务线程推送消息的。

expirationTime 属性 – 只读，订阅的有效期，为 1970 年 1 月 1 日开始的毫秒数或者 null。

options 属性 – 只读，为订阅时指定的 options 对象值。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

unsubscribe()方法 – 取消订阅。返回 Promise 对象，执行完毕后确值为 true 或者 false 值，true 表示取消订阅成功。

getKey(name)方法 – 返回浏览器的公钥，参数 name 指定产生公钥的方法，可能的值有"p256dh"（基于 P-256 曲线法）、"auth"。在订阅回执中同时存在这两种方法产生的公钥。

toJSON()方法 – 序列化为 JSON 对象。

下面给出一个完整的例子，采用 firefox 浏览器（默认就是火狐公司的推送服务器），谷歌的推送服务器中国境内没法使用。

整个例子包含 6 个文件：index.html（页面文件，引入 index.js）、index.js（主线程）、serviceWorker.js（服务线程）、server-https.js（基于 node.js 的应用服务器，走 https 协议）、notice.html（用户点击通知后打开的页面）、shock.jpg（通知上的图标）。

大致的过程是这样：用户浏览页面 index.html，浏览器执行 index.js（完成任务：注册服务线程、订阅消息推送服务、把订阅回执发给应用服务器保存），应用服务器 server-https.js 定期向推送服务器发消息，推送服务器就给用户的浏览器推送通知，用户点击通知就打开 notice.html 页面。

1、 让应用服务器跑起来

安装最新的 node.js（可以到 nodejs.org 下载），然后执行下面的命令安装相关库：

npm install web-push express body-parser -g

进入网页文件所在的目录，参考本书的“1.5.1.1 WEB 服务”小节制作证书，分别保存在 server.key 和 server.crt 文件中。执行命令 web-push generate-vapid-keys 产生密钥对，拷贝公钥替换 server-https.js 文件中变量 publicVapidKey 的值，拷贝私钥替换 privateVapidKey 变量的值。修改 server-https.js 中的 192.168.0.50 为你自己电脑的 ip，然后执行下面的命令启动应用服务器：

```
node server-https
```

应用服务器监听 5000 端口，走 https 协议，注意 http 协议不支持消息推送。然后就可以打开网址“https://你电脑的 ip:5000”，点击页面上的“订阅消息”按钮后，过会儿就会收到推送通知了。

2、 代码简介

本实例的代码文件放在 chapter-6\ push-subscription 目录中，主要由以下几个文件。

index.html

这个比较简单，就一个“订阅消息”按钮，通过<script>引入了 index.js 文件。

index.js

首先注册服务线程，然后在 avigator.serviceWorker.ready.then((registra)=>{…})的函数中完成订阅消息推送服务（为了安全起见，最好先调用方法 registra.pushManager.getSubscription()看看是否已经订阅），然后把订阅回执发给应用服务器保存起来。不过在订阅前先从应用服务器获取公钥，订阅时要把公钥发给推送服务器。

serviceWorker.js

这是在服务线程中执行的程序，主要监听 push 和 notificationclick 事件。当收到消息时触发 push 事件，在事件处理函数中，我们调用 self.registration.showNotification()方法在电脑桌面上弹出通知。当用户点击通知时触发 notificationclick 事件，在本事件处理函数中，先判断/notice.html 页面是否已经打开，如果已经打开就让其获得输入焦点，如果没打开就调用 self.clients.openWindow()方法打开它。

server-https.js

这是应用服务器，即网站的后端。主要采用 node.js 上的 express 和 web-push 库，方法 app.get('/vapidPublicKey',(req,res)=>{…})响应用户'/vapidPublicKey'请求，即获取应用服务器的公钥，直接采用语句 res.send(publicVapidKey)把公钥发给用户。方法 app.post('/register',(req,res)=>{…})接受用户的订阅回执，然后保存在变量中。采用定时器 setInterval()每隔 40 秒向推送服务器发一条消息，在实际的项目中，一般是由事件（比如发布了一条爆炸新闻）决定消息内容的，例子中采用 web-push 库的 sendNotification()方法发消息。

6.8 IndexedDB

HTML4 引入的 cookie 技术，其设计初衷是方便后台验证用户的身份，容量被限制在 4KB 以内，而且不是持久化存储。HTML5 引入的 Web Storage 技术容量达到 5MB 至 10MB（各个浏览器支持的容量不同），而且实现了可持久存储，但是依然存在如下缺陷：容量还是不够大，存储格式“键/值”对，只能是字符串。所以最新引入了 IndexedDB 技术，这是一个存储在浏览器中的基于索引和事务的 noSQL 型数据库，容量能大到硬盘剩余空间的一半，能离线使用，可持久化存储。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.8.1 概述

IndexedDB 具备如下几个特征：

1、存储“键/值”对。“值”为结构对象，任何复杂的结构对象都可以作为“值”保存在 IndexedDB 中。“键”必须唯一，“键”可为结构对象中的属性，如果没有合适的属性作为键，可采用自增 id。另外二进制对象也可以作为键，因为一个复杂结构对象的属性可能还是一个结构对象。

2、支持事务。“增/删/改”操作必须在事务中完成。

3、异步操作。这一点与传统的关系数据库大不一样，每一个操作不会被阻塞，所以我们需要给相应的事件安排回调函数，告诉浏览器：如果操作成功就调用这个函数，如果失败就调用那个函数，等等。

4、不支持结构化查询语言（SQL）。只能调用 IndexedDB 提供的各种 JavaScript API 来完成数据库的操作。

5、不能跨源访问 IndexedDB。“源”由 URL 的协议、域名和端口定义，这三者当中只要存在一个不同，就不是同源，比如 http://www.veryopen.org 和 https://www.veryopen.org 不同源，因为前者的“源”（协议 http、域名 www.veryopen.org、端口 80）不同于后者的源（协议 https、域名 www.veryopen.org、端口 443）。

6、支持索引。除了键这个唯一索引外，还可以在更多的对象属性上创建索引。只有通过索引属性才可查询数据库。

7、支持游标。

IndexedDb 数据库是在浏览器里的，一个“源”（协议+域名+端口）允许创建多个数据库，每个数据库必须有一个名字和一个版本号，新建数据库时如果没有指定版本号，默认就是 1，此后允许不断升级版本号，但是在任何时刻，一个数据库只有一个版本号。数据库中可以创建多个对象库（类似关系数据库的表），“键/值”对就保存在对象库中，一个“键/值”对就是一条记录（相当于表中的行），一个对象库能存储不同对象的实例。一个对象库上允许创建多个索引，这些索引可以是唯一索引或不唯一索引，但是“键”对应的索引必须是唯一索引，只有通过索引列（对象里某个建立了索引的属性）才能查询对象库。对数据库的“增/删/改”都要通过事务完成，事务接口对象提供了 error、abort 和 complete 三个事件，可以给这些事件安排回调函数，以便事件发生时做出相应的处理。对于这些概念之间的关系，请看下面的示意图 6.23。

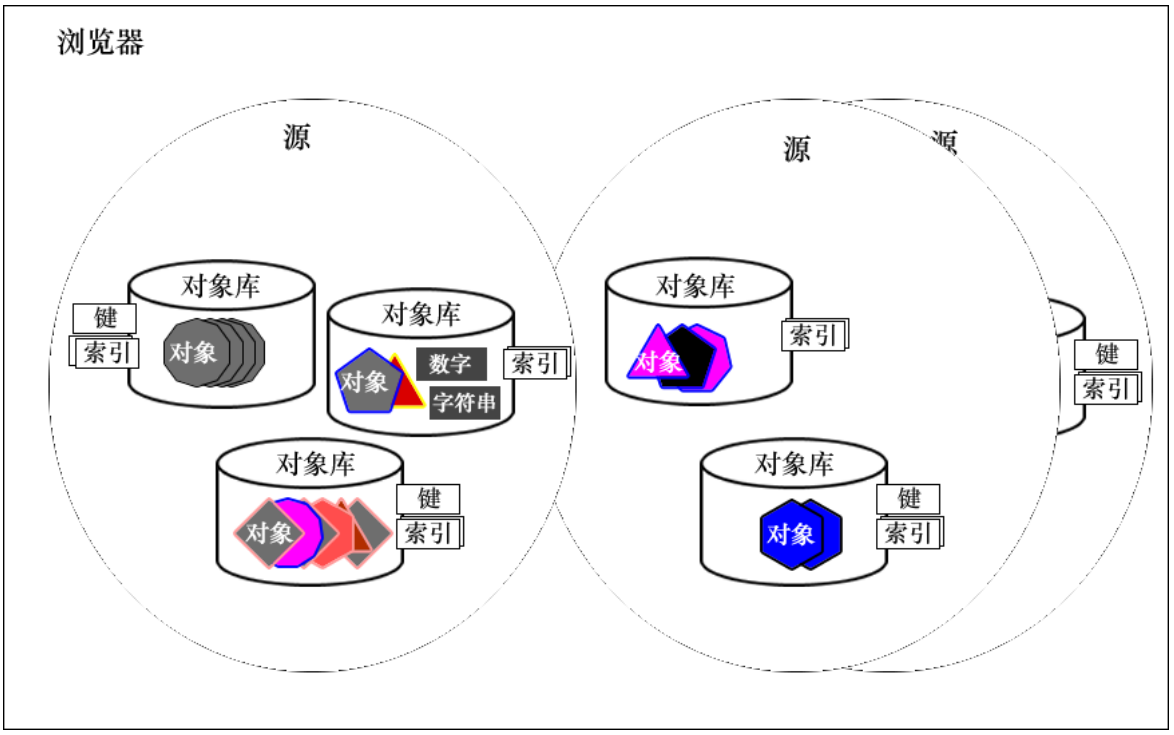


图 6.23 IndexDB 示意图

6.8.2 操作

一次使用 IndexedDB 的完整过程大致是这样：打开数据库→新建对象库→创建索引→开启事务→增/删/改/查→处理返回的数据（如果是查询）。

下面的代码是本书 IndexedDB 的页面框架，后续代码置于<script>元素内部。

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>IndexedDB</title>
</head>

<body>
    <h1>IndexedDB 数据库实验</h1>
    <p id="ret">结果：</p>
    <script>
        操作 IndexedDB 的 JS 代码
    </script>
</body>

</html>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.8.2.1 打开数据库

虽然目前绝的主流浏览器都支持 IndexedDB 数据库,但是在生产环境当中最好还是要判断一下,因为你不能排除有人还在使用古董浏览器。采用如下的代码:

```
var ret = document.getElementById("ret");
window.indexedDB = window.indexedDB || window.mozIndexedDB || window.webkitInde
xedDB || window.msIndexedDB;
if (!window.indexedDB) {
     ret.innerText = "抱歉，你的浏览器不支持 IndexedDB！ ";
     throw "exit";
}
window.IDBTransaction = window.IDBTransaction || window.webkitIDBTransaction ||
 window.msIDBTransaction || { READ_WRITE: "readwrite" };
window.IDBKeyRange = window.IDBKeyRange || window.webkitIDBKeyRange || window.m
sIDBKeyRange;
```

调用 window.indexedDB 对象的方法 open()打开数据库(如果不存在就创建),此方法的声明是:

```
window.indexedDB.open(name: string, version?: number): IDBOpenDBRequest
```

第一个参数 name 为数据库的名字。第二个参数为版本号，类型是无符号长整形，它是可选的参数，如果省略，要分两种情况：（1）如果数据库不存在，就创建一个新的数据库，并且指派版本号 1；（2）否则打开当前版本的数据库。如果没有省略版本号，那么版本号要大于等于当前版本号，而且必须大于 0。

open()方法返回一个 IDBOpenDBRequest 对象，该对象有四个事件:

success 事件 – 打开成功。一般在本事件处理函数中对对象库进行“增/删/改/查”操作。

error 事件 – 打开失败。

upgradeneeded 事件 – 新建数据库或版本升级成功。只要版本号发生改变（从无到有【新建】,或者从小到大【升级】）就会触发本事件，版本不变就不会触发。触发本事件并且事件处理函数成功返回后又会触发 success 事件,但是触发了 success 事件之前不一定就触发了 upgradeneeded 事件，这点要特别注意。只能在 upgradeneeded 事件处理函数中新建对象库、新建索引、删除索引、删除对象库，不允许直接修改对象库的键（只有通过先删除后再建对象库的方法来间接修改键）。

block 事件 – 数据库被阻塞在版本变更事务中。

看例子（把这些代码追加到前面例子的下面）:

```
    var openRequest = window.indexedDB.open('infoes_db');
    var db;
    openRequest.addEventListener('success', (event) => {
         ret.innerText = "打开成功！ ";
         db = openRequest.result;
         console.log("sucess");
    });
    openRequest.addEventListener("upgradeneeded", (event) => {
         ret.innerText = "新建或者升级成功！ ";
         db = openRequest.result;
```

```
        console.log("upgradeneeded");
    })
    openRequest.addEventListener("blocked", () => {
        ret.innerText = "被阻塞！";
        console.log("block");
    });
    openRequest.addEventListener("error", (event) => {
        ret.innerText = "打开失败！";
        console.log("error");
    });
```

如果打开成功，openRequest.result 中保存了 IDBDatabase 对象的一个实例化，以后就使用这个实例化对象来操作数据库，所以一般采用类似“db = openRequest.result;”语句保存到一个变量中。在 success 和 upgradeneeded 事件处理函数中，openRequest.result 等于 this.result，也等于 event.target.result。打开开发者工具，并刷新页面，看到的效果如下所示：

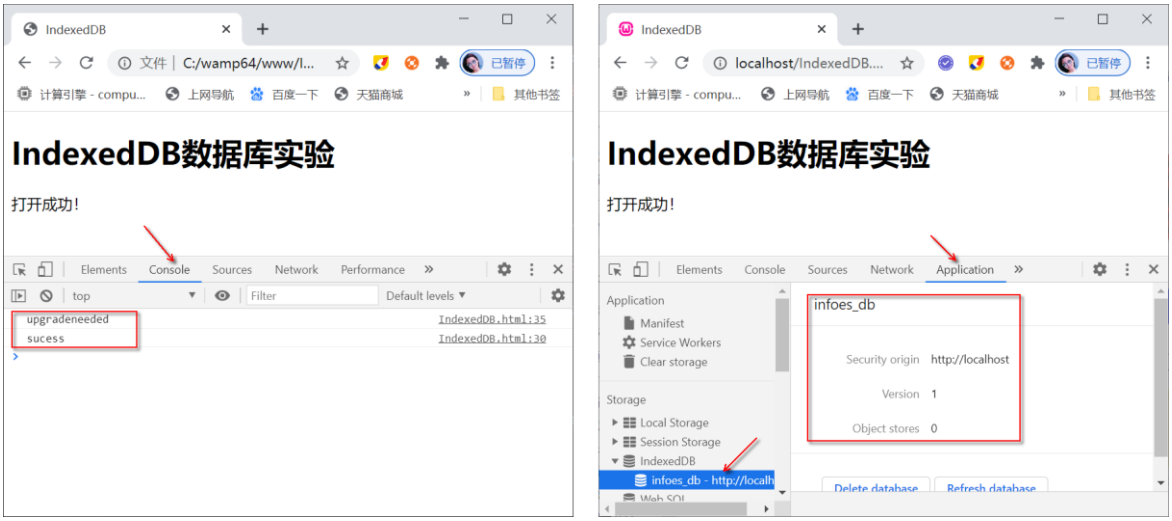


图 6.24 在开发者工具中查看 indexDB

调用 window.indexedDB.deleteDatabase(name: string)方法删除数据库，成功就触发 success 事件，其他已经打开的连接都会收到 versionchange 事件，删库失败会触发 error 事件。注意：删除不存在的数据库不会报错。

6.8.2.2 新建对象库

必须在 upgradeneeded 事件处理函数中新建对象库，调用 IDBDatabase 对象的方法 createObjectStore()创建对象库。它的声明如下：

```
IDBDatabase.createObjectStore(name: string, optionalParameters?: IDBObjectStore
Parameters): IDBObjectStore
```

形参 name 指明对象库的名字，但允许名字为空白。形参 optionalParameters 是可选的，它是一个对象，可能的属性有以下两个：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

keyPath – 指明对象中哪个属性作为键，可能的值有：①""或空，②用“.”分隔的若干标识符，③存储标识符的数组。如为①，则由“键/值”对中的“值”组合成单独存储的“键”；如为②且只有一个标识符，则这个唯一的标识符就是对象中的属性名称，如有有多个标识符，类似"name1.name2.name3.…"，则 name?对应各个子对象中的属性名称：

```
    {
        name1:{
            name2:{
                name3:{
                    …
                }
            }
        }
    }
```

如： { keyPath: "idCard" }、{ keyPath: "info.telephone" }

autoIncrement – 布尔类型，默认为 false，true 表示采用自增长键。如：{ autoIncrement: true }。

按照调用方法时是否提供了 keyPath 非空值，结合 autoIncrement 的取值，共有四种组合，见表 6.4 所示。

表 6.4 keyPath 和 autoIncrement 的组合

keyPath	autoIncrement	描述
×	false	对象库能保存任何值，比如整数、字符串、对象等。但插入数据时必须提供键。
×	true	对象库能储存任何值，插入数据时采用自增长键或额外提供键。
√	false	只能保存 JavaScript 对象，但对象中必须存在与键同名的属性。
√	true	只能保存 JavaScript 对象，如果对象中不存在与键同名的属性，则增加属性，属性的值就是自增长的键值，否则就使用对象属性作为键。

IDBDatabase.createObjectStore()方法返回 IDBObjectStore 对象的实例，它代表一个对象库，对象库中的数据按照索引排序存储，后面就使用这个对象实例来完成“增/删/改/查”操作了。方法失败的原因有这么个几个：⑴InvalidStateError（在版本变更事务之外调用本方法）；⑵TransactionInactiveError(数据库不存在)；⑶ConstraintError(对象库名重复)；⑷InvalidAccessError（实参错误：autoIncrement 等于 true，但是 keyPath 为空值）。

看例子：

针对下面的对象

```
const customerData = [
  { ssn: "444-44-4444", name: "张三", age: 35, email: "zsan@abc.com" },
  { ssn: "555-55-5555", name: "李四", age: 20, email: "lsi@home.org" },
  { ssn: "666-66-6666", name: "王二" },
  { ssn: "888-88-8888", name: "李四", age: 26, email: "alice@gmail.com" }
];
```

创建一个 customers 对象库，在前面例子的 upgradeneeded 事件处理函数中增加代码：

```
openRequest.addEventListener("upgradeneeded", (event) => {
    ret.innerText = "新建或者升级成功! ";
    db = openRequest.result;
    try {
        if (!db.objectStoreNames.contains('customers')) {
            var customerStore = db.createObjectStore("customers", { keyPath: "s
sn" });
        }
     } catch (error) {
        alert("新建对象库失败: ", error);
throw error;
     }
});
```

IDBDatabase.deleteObjectStore(name:string)方法删除对象库。

6.8.2.3 创建索引

由于只能通过索引查询，键同时是唯一索引，如果还需要通过其他方式查询（比如上面例子中的姓名和邮件），这时可采用 IDBObjectStore.createIndex()方法来创建更多的索引。此方法的声明如下：

```
IDBObjectStore.createIndex(indexName: string, keyPath: string | Iterable<string
>, objectParameters?: IDBIndexParameters): IDBIndex
```

形参 indexName 为索引名称，允许为空，索引名称在数据库中不能重复。keyPath 为建立索引的那个对象的属性名称，可为空、一个属性名或者序列（数组）。objectParameters 形参允许省略，类型为对象，包含的属性有：

unique – 是否唯一索引，默认为 false。

multiEntry – 只在 keyPath 的值为数组时才发挥作用。为布尔类型，当为 true 时，keyPath 数组中的每个元素都会建立一个索引，否则只为整个数组建立一个索引。

调用 IDBObjectStore.createIndex()方法成功时会返回一个 IDBIndex 对象，如果失败，可能的原因有：①ConstraintError（数据库中索引名称已经存在）；②InvalidAccessError（提供的 keyPath 的值是序列，并且在 objectParameters 对象中的 multiEntry 为 true）；③InvalidStateError（在版本变更事务之外新建索引，或者对象库被删除了）；④SyntaxError（keyPath 的值非法）；⑤TransactionInactiveError（此 IDBObjectStore 所属的事务未激活（可能被删除））。

例子：在 customers 对象库上对 name 建立非唯一索引，对 email 建立唯一索引。

```
if (!db.objectStoreNames.contains('customers')) {
     var customerStore = db.createObjectStore( "customers", { keyPath: "ssn" } );
     customerStore.createIndex( "nameIndex", "name" );
     customerStore.createIndex( "emailIndex", "email", { unique: true } );
}
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

方法 IDBObjectStore.deleteIndex(indexName: string)用于删除索引。

6.8.2.4 开启事务

IndexedDB 要求“增/删/改/查”操作必须安排在事务中，事务要跟对象库关联在一起。开启一个事务采用方法 IDBDatabase.transaction()，此方法的声明如下：

```
IDBDatabase.transaction(storeNames: string | string[], mode?: string, options?:
IDBObjectStoreParameters): IDBTransaction
```

形参 storeNames 指明若干个与本事务关联的对象库名称，可为字符串或字符串数组，如"customers"、["products","sales"]，如果要关联本数据库中全部的对象库，直接采用 IDBDatabase.objectStoreNames 即可。形参 mode 可选，定义访问类型，可取值"readonly"（默认值）、"readwrite"和"versionchange"，“增/删/改”操作一定要采用"readwrite"，“查询”操作虽然也可以采用"readwrite"，但是强烈建议采用"readonly"，创建、删除对象库和索引只能在"versionchange"类型的事务中完成，"versionchange"类型的事务中也可以做“增\删\改\查”。但是"versionchange"类型的事务是在触发 upgradeneeded 事件时自动创建的，因此事件处理函数处于"versionchange"类型的事务中，不能手工创建"versionchange"类型的事务。options 形参是可选的，给网页客户端提供一些改善性能方面的建议（在提交事务时是优先考虑性能还是数据完整性），它是一个对象，目前属性只有 durability 一个，此属性可取值"default"（默认值）（由客户端自行决定）、"relaxed"（数据被写到操作系统后表示事务提交完成，数据可能还在操作系统的内存高速缓存中）和"strict"（数据被写到磁盘后表示事务提交完成）。如果数据至关重要，即应用偏重于数据完整性而不是机器性能，那么最好采用"strict"。

IDBDatabase.transaction()方法调用成功会返回一个 IDBTransaction 对象，如果调用失败，则可能的原因有：①InvalidStateError（数据库已经关闭，调用了 IDBDatabase.close()方法）；②NotFoundError（指定的对象库不存在）；③TypeError（mode 参数非法）；④InvalidAccessError（storeNames 参数为空）。

IDBTransaction 对象有三个事件：①abort（事务被回滚）；②complete（事务被提交）；③error（出错）。一般给 complete 事件安排处理函数，用来提醒用户数据被可靠保存。

下面的例子开启一个“读/写”事务，稍后我们在此事务中向对象库中插入数据：

```
    openRequest.addEventListener('success', (event) => {
        ret.innerText = "打开成功！";
        db = openRequest.result;
        var myTrans = db.transaction(["customers"], "readwrite", {durability:"s
trict"});
        myTrans.addEventListener("complete", () => {
            alert("提交完成！");
        });
        console.log("sucess");
    });
```

由于本例子的事务只关联一个对象库 customers，所以第一个实参也可为"customers"字符串，如果关联多个，就只能采用字符串数组了，如果关联数据库中全部的对象库，直接采用 db.objectStoreNames，语句变为：

```
···db.transaction(db.objectStoreNames, "readw···
```

6.8.2.5 “增/删/改/查”操作

采用 IDBObjectStore 对象的方法完成“增/删/改/查”操作，此对象由 IDBTransaction.objectStore(name)方法返回，其中参数 name 为事务关联的一个对象库名称。IDBObjectStore 对象有如下一些方法：

1、add(value: any, key?: IDBValidKey): IDBRequest

插入数据。参数 value 为要插入的任何类型的数据，参数 key 指定键，可以省略此参数。方法返回 IDBRequest 对象，可根据对象的事件 error 和 success 来判断插入是否成功。

2、delete(key: string | IDBKeyRange): IDBRequest

删除数据。参数 key 指定被删除记录的键，格式可为字符串（如"444-44-4444"）、键范围（如 IDBKeyRange.lowerBound(10)）。表 6.5 是各种键范围表示法。

图 6.5 键范围表示法

代码	解释
IDBKeyRange.lowerBound(x)	keys≥**x**
IDBKeyRange.lowerBound(x, true)	keys>**x**
IDBKeyRange.upperBound(x)	keys≤**y**
IDBKeyRange.upperBound(x, true)	keys<**y**
IDBKeyRange.bound(x, y)	x≤keys≤**y**
IDBKeyRange.bound(x, y, true, true)	x<keys<**y**
IDBKeyRange.bound(x, y, true, false)	x<keys≤**y**
IDBKeyRange.bound(x, y, false, true)	x≤keys<**y**
IDBKeyRange.only(x)	key=**x**

3、clear(): IDBRequest

清除对象库中的全部记录。

4、put(value: any, key?: IDBValidKey): IDBRequest

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

修改或插入记录。参数 value 为任何类型的数据。可选参数 key 只有在对象库为自增长键（即 autoIncrement）时允许存在，其他情况必须省略参数 key，因为键包含在 value 参数中。当为自增长键且省略参数 key 时，就是插入记录，当为非自增长键并且 value 中的键不在对象库中时，也是插入一条新记录。

5、get(key: string | IDBKeyRange): IDBRequest

查询记录。参数 key 与 delete()方法的同名参数含义相同。

6、getAll(key?: string | IDBKeyRange, count?: number): IDBRequest

查询记录。如果两个参数都省略，则返回对象库中的全部记录，参数 count 规定返回记录数的上限，要求 0≤count≤2^{32}-1。

7、getKey(key: string | IDBKeyRange): IDBRequest

查询键。

8、getAllKeys(key?: string | IDBKeyRange, count?: number): IDBRequest

查询键。参数介绍参考 getAll()方法。

9、count(key?: string | IDBKeyRange): IDBRequest

查询记录数。如果省略参数 Key，则返回对象库中全部的记录数。

10、index(indexName: string): IDBIndex

打开一个索引，然后可以通过游标读取按该索引排序的所有记录。

11、openCursor(key?: string | IDBKeyRange, direction?: IDBCursorDirection): IDBRequest

通过游标获取满足条件的记录。如果省略 key 参数，则返回全部记录。参数 driection 定义游标遍历记录的方向，可能的实参有"next"（默认值）、"nextunique"、"prev"和"prevunique"，"next"定义游标的方向从结果集的首记录到尾记录，"prev"从结果集的尾记录到首记录，而"*unique"跳过那些与已经遍历过的记录有相同值的记录。

12、openKeyCursor(key?: string | IDBKeyRange, direction?: IDBCursorDirection): IDBRequest

通过游标获取满足条件的键。参数介绍参见 openCursor()方法。

插入记录

首先我们把“§6.2.8.2 新建对象库”小节中提到的 customerData 对象数组插入到 customers 对象库中。插入记录可采用 IDBObjectStore.add()或 IDBObjectStore.put()方法。看下面的代码：

```
myTrans.addEventListener("complete", () => {
        alert("提交完成！");
});
const customerData = [
    { ssn: "444-44-4444", name: "张三", age: 35, email: "zsan@abc.com" },
    { ssn: "555-55-5555", name: "李四", age: 20, email: "lsi@home.org" },
    { ssn: "666-66-6666", name: "王二" },
    { ssn: "888-88-8888", name: "李四", age: 26, email: "alice@gmail.com" }
];
```

```
    var customersStore = myTrans.objectStore("customers");
    customerData.forEach((customer) => {
            customersStore.add(customer);
    });
    console.log("sucess");
```

把代码“customersStore.add(customer);”更换为“customersStore.put(customer);”，效果一样。刷新页面，进入开发者工具，可以看到插入的数据。

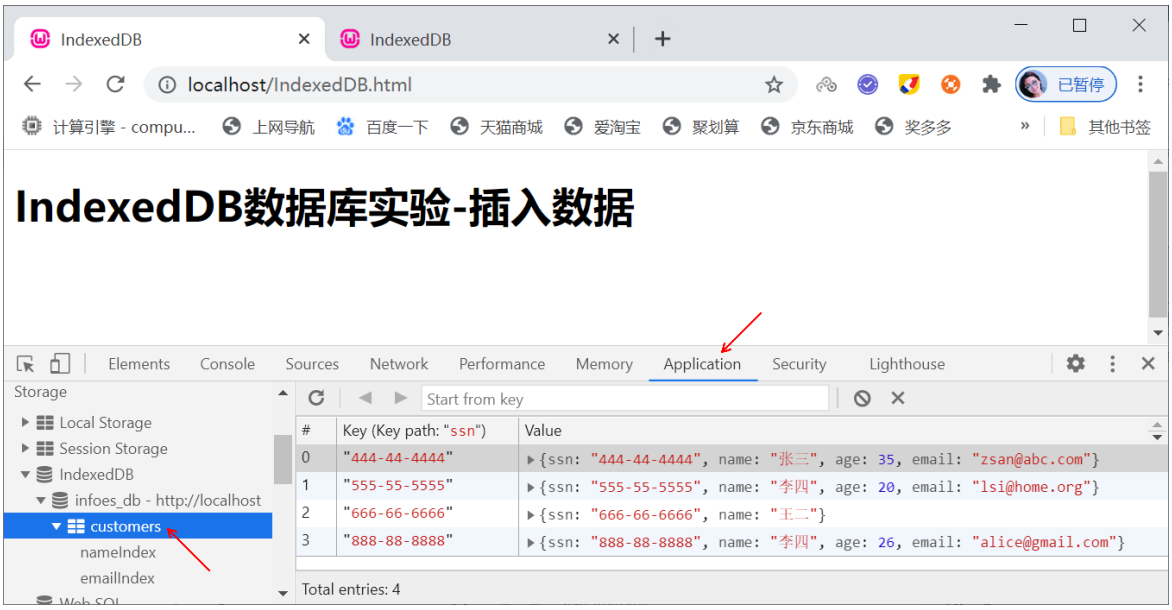


图 6.25 IndexDB 举例-插入数据

删除记录

删除记录可采用 IDBObjectStore.delete()或 IDBObjectStore.clear()方法。看下面的代码：

```
    openRequest.addEventListener('success', (event) => {
        ret.innerText = "打开成功！";
        db = openRequest.result;
        var deleteReq = db.transaction(["customers"], "readwrite").objectStore
("customers").delete("444-44-4444");
        deleteReq.addEventListener("success", (event) => {
            alert("删除成功！");
        });
        console.log("sucess");
    });
```

下面的代码删除满足条件（"444-44-4444"≤key≤"777-77-7777"）的全部记录。

```
var deleteReq = db.transaction(["customers"], "readwrite").objectStore("custome
rs").delete(IDBKeyRange.bound("444-44-4444", "777-77-7777"));
```

下面的代码删除对象库中所有记录：

```
db.transaction(["customers"], "readwrite").objectStore("customers").clear();
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

修改记录

修改记录采用 IDBObjectStore.put(value, key?)方法，假如现在要把 ssn="888-88-8888"的用户名由“李四”改为“王麻子”，这时形参 value 对应的实参为{ ssn: "888-88-8888", name: "王麻子", age: 26, email: "alice@gmail.com" }，代码如下：

```
openRequest.addEventListener('success', (event) => {
    ret.innerText = "打开成功！";
    db = openRequest.result;
    var myTrans = db.transaction(["customers"], "readwrite", { durability: "str
ict" });
    myTrans.addEventListener("complete", () => {
          alert("提交完成！");
    });
    var customersStore = myTrans.objectStore("customers");
    var putRequest = customersStore.put({ ssn: "888-88-8888", name: "王麻子
", age: 26, email: "alice@gmail.com" });
    putRequest.addEventListener("success", () => {
         console.log("修改成功！");
    });
    console.log("sucess");
});
```

如果只用实参 { ssn: "888-88-8888", name: "王麻子" } 行不行？语法没错，但是最终数据不但改了名字，而且 age 和 email 都被删除了，没有出现在实参中的对象属性意味着要删除。所以在实战当中，一般是先查询相应的记录，然后再修改相关属性的值。如果键不在对象库中，就插入一条新记录。

查询

查询可采用 IDBObjectStore 对象的方法：get(key)、getAll(key,count)、getKey(key)、getAllKey(key,count)、count(key)、index(indexName)、openCursor(key,direction)和 openKeyCursor(key,direction)，除了 index()方法外，其他方法都返回 IDBRequest 对象。

IDBRequest 对象的一些主要属性有：result（查询结果，可能是单值，也可能是数组）、error（查询失败时的错误信息）。IDBRequest 对象的事件有：success（查询成功时触发）、error（查询失败时触发）。一般我们在 success 的事件处理函数中处理查询结果。先看一个简单的例子：

```
    var openRequest = window.indexedDB.open('infoes_db');
    var db;
    openRequest.addEventListener('success', (event) => {
        db = event.target.result;
        var customersStore = db.transaction(["customers"], "readonly").objectSt
ore("customers");
        var getRequest = customersStore.get("444-44-4444");
        getRequest.addEventListener("success", (event) => {
            let getData = event.target.result;
            ret.innerText = "SSN=" + getData.ssn + " / 姓名
=" + getData.name + " / 年龄=" + getData.age + " / 邮件=" + getData.email;
        });
    });
```

随书配套资源的例子 chapter-6\indexdb.html 采用 getAll()获取全部的记录，并以表格形式显示在页面上。主要的代码如下。

```
var openRequest = window.indexedDB.open('infoes_db');
var db;
openRequest.addEventListener('success', (event) => {
    db = event.target.result;
    var customersStore = db.transaction(["customers"], "readonly").objectStore(
"customers");
    var getAllRequest = customersStore.getAll();
    getAllRequest.addEventListener("success", (event) => {
        displayData(event.target.result);
    });
});
```

event.target.result 中保存了查询出来的记录数组，比如 event.target.result[0].name 就是第一条记录的 name 字段，在 displayData(data)函数中显示在一个表中。接下来我们使用游标获取数据，采用方法 IDBObjectStore. openCursor(key,direction)，此方法虽然也返回 IDBRequest 对象，但是 IDBRequest.result 又是游标对象 IDBCursorWithValue，IDBCursorWithValue.value 属性保存了游标当前的数据记录，IDBCursorWithValue.continue() 方法移动游标指针，并再次触发 IDBRequest.success 事件。看例子：

```
openRequest.addEventListener('success', (event) => {
    db = event.target.result;
    var customersStore = db.transaction(["customers"], "readonly").objectStore(
"customers");
    var getAllRequest = customersStore.openCursor(IDBKeyRange.bound("444-44-
4444", "999-99-9999", true, false));
    getAllRequest.addEventListener("success", event => {
         var cursor = event.target.result;
         if (cursor) {
              let li = document.createElement("li");
              li.innerText = cursor.value.ssn + " , " + cursor.value.name + "
, " + cursor.value.age + " , " + cursor.value.email;
              document.body.appendChild(li);
              cursor.continue();
          } else {
              console.log("没有数据了");
          }
    });
});
```

本例子返回全部符合条件的记录，条件是"444-44-4444"<key≤"999-99-9999"。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.9 媒体流

媒体流由若干视频轨道和/或若干音频轨道组成，从多媒体设备（如话筒、耳机、摄像头等）获取的视频和音频内容会源源不断地进入浏览器，在浏览器里称为媒体源，像不断冒出水的泉眼一样。采用 MediaStream 对象可以从媒体流中随意删除和添加音/视频轨道，也可以对轨道重新组装成各种媒体流。把视频媒体流赋给<video>元素对象的 srcObject 属性，就可以播放视频，同样地把音频媒体流赋给<audio>元素对象的 srcObject 属性，即可播放音频。不过一些旧版浏览器的媒体对象没有 srcObject 属性，这时可用下面的代码来增加页面的普遍适应性：

```
if ('srcObject' in videoElem) {
        videoElem.srcObject = mediaStream;
} else {
     videoElem.src = URL.createObjectURL(mediaStream);
}
```

参见图 6.26 进一步从全局理解媒体流。

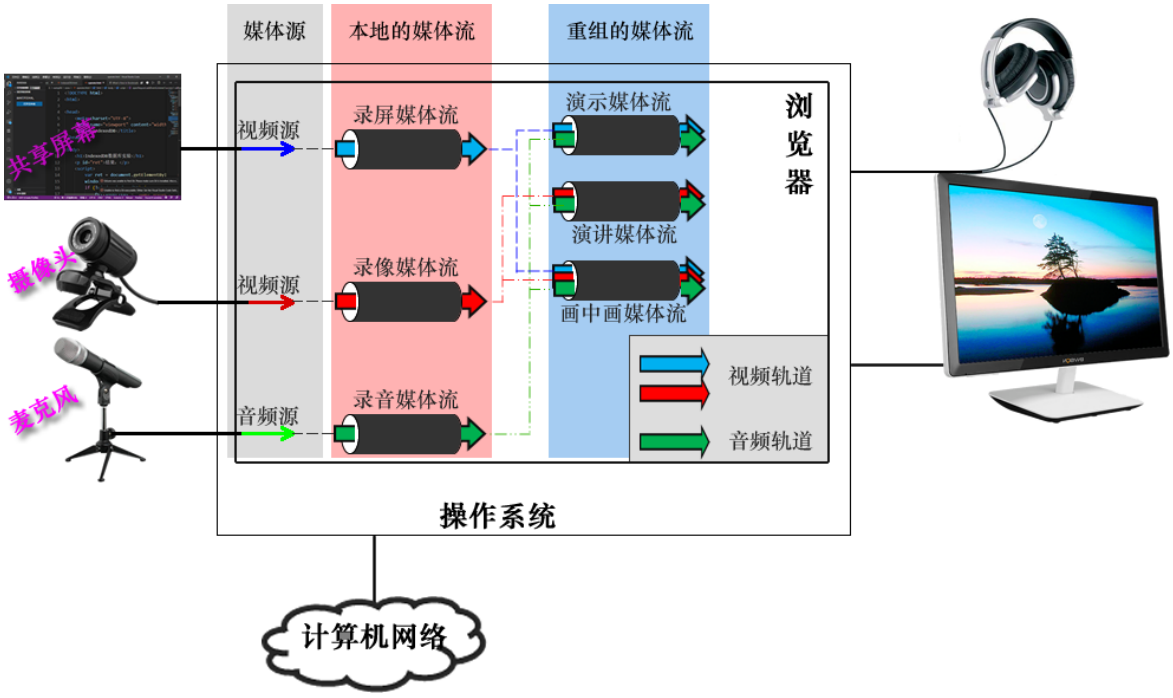


图 6.26 媒体流及媒体设备

在页面文档中操作媒体流，需要用到 MediaDevices、MediaStream、MediaStreamTrack、MediaStreamConstraints、MediaRecorder (可选)等几个接口对象。

如果电脑没有摄像头和麦克风，请安装虚拟摄像头 e2eSoft Vcam，以便于后续章节的学习。

6.9.1 MediaDevices

MediaDevices 为 navigator 的子对象，而 navigator 又是 window 的子对象。MediaDevices 接口从媒体输入设备（如麦克风、摄像头、共享屏幕等）获取媒体流，但是浏览器会弹出对话框，询问用户是否允许浏览器操作多媒体设备。MediaDevices 有一个 devicechange 事件，当连接的媒体设备发生变法时触发（比如插入或拔掉 USB 麦克风）。MediaDevices 对象常用的几个方法如下：

1、getUserMedia()

打开媒体输入设备获取媒体源，返回 MediaStream 类型的媒体流，调用方法的参数不同，返回的媒体流中包含的音/视频轨道也不同。此方法声明如下：

```
navigator.mediaDevices.getUserMedia(constraints : MediaStreamConstraints): Prom
ise
```

形参 constrains 用于获取满足条件的媒体轨道，它的类型 MediaStreamConstraints 是一个字典，这个字典只有两个属性 audio 和 video，它们的取值要么是布尔型要么是字典。我们看一些具体的例子，参考图 6.28。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

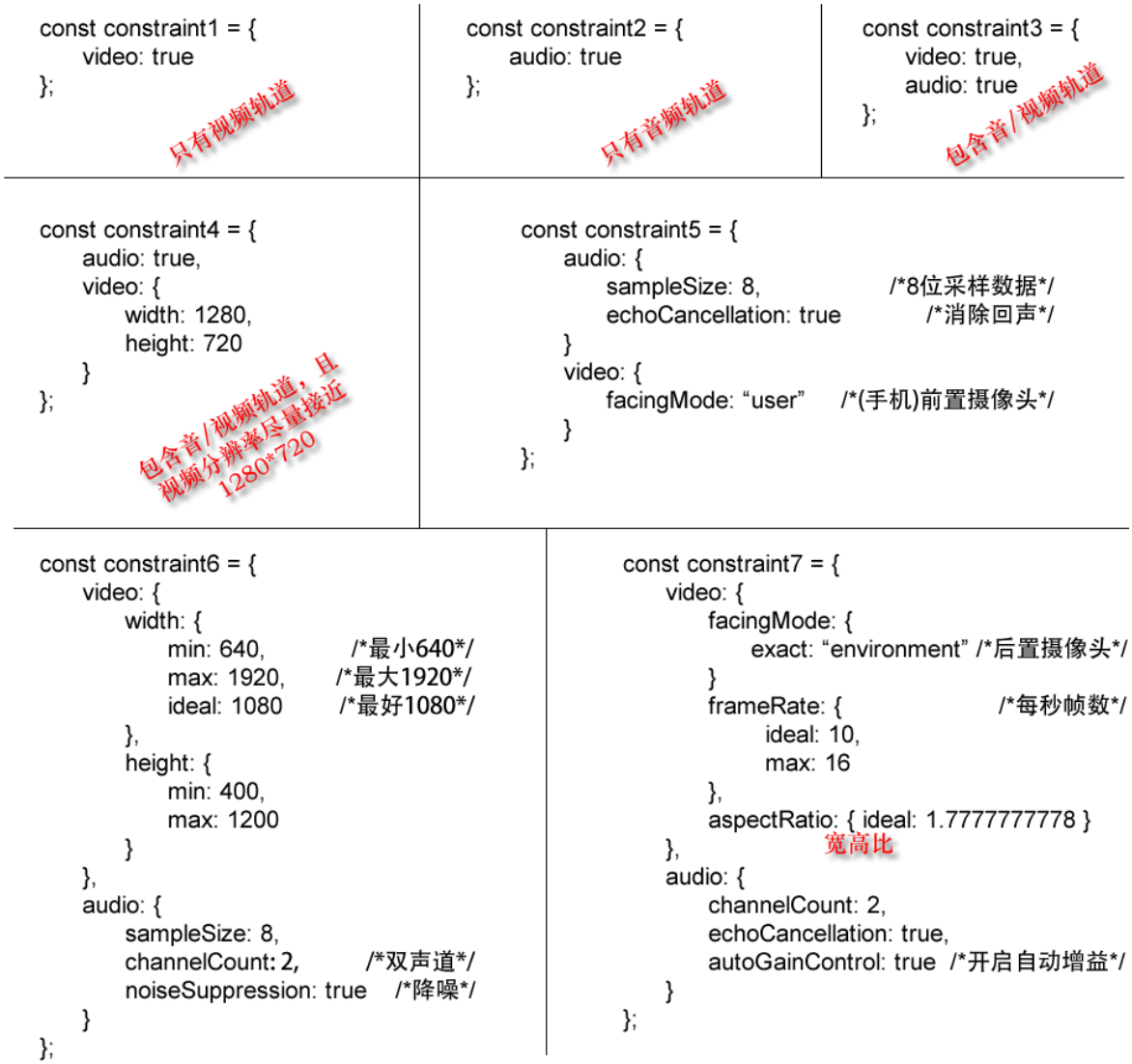


图 6.28 获取不同媒体源

对于音频轨道，还有这些限制属性：latency（双精度浮点数），sampleRate（采样频率）、volume（音量，取值范围[0.0,1.0]）。对于视频轨道，还有这些限制属性：facingMode（选择摄像头，还可取值"left"-左侧的摄像头、"right"-右侧的摄像头）、

另外 width:1080 等价于 width: { ideal: 1080 }，但与 width: { exact: 1080 }不同，因为“exact”表示“=”，而前者表示“≈”。条件是用来选择媒体源的，而不是设置媒体源，条件如果过于严格，有可能没有符合要求的媒体源，尽管电脑上可能连接了多个多媒体设备（如话筒、摄像头等）。

getUserMedia()方法返回一个 Promise，当成功获取请求的媒体时，例行处理程序将收到 MediaStream 对象。如果用户拒绝许可，或者匹配的媒体不可用，错误处理程序就会收到 DOMException 错误对象。下面看一个实际的例子：

```
<body>
    <video id="vid" autoplay playsinline></video>
<script>
    if (!navigator.mediaDevices || !navigator.mediaDevices.getUserMedia) {
          alert("不支持 getUserMedia()方法! ");
```

```
            throw 'exit';
        }
        const constraint = {
            audio: {
                echoCancellation: true,
                noiseSuppression: true
            },
            video: true
        }

        function successHandler(stream) {
            var vide = document.getElementById("vid");
            vide.srcObject = stream;
        }

        navigator.mediaDevices.getUserMedia(constraint).then(successHandler).catch
(
            err => console.log(err.name + ": " + err.message)
        );
</script>
<body>
```

本例子获取本地的音/视频源并组合成媒体流，最后把媒体流定向到<video>元素播放出来。采集音频时开启了降噪和消除回声功能。这样就捕捉到了摄像头画面并展示在页面中。

2、getDisplayMedia()

录频——把屏幕录制成媒体流，成为媒体流内的视频轨道，从而实现共享屏幕。此方法声明如下：

```
navigator.mediaDevices.getDisplayMedia(constraints ?  : MediaStreamConstraints):
 Promise
```

可选的实参 constrains 指定生成的视频轨道的一些要求，值的格式与 getUserMedia()方法实参一样，但是 video 属性的值如果是字典，则字典内部的属性名有所不同，主要有：

cursor - 定义鼠标指针是否出现在视频轨道中。可取值： "always"（除非鼠标移出共享区域，否则鼠标指针一直可见。这是默认值）、"motion"（当移动鼠标时，指针才显示）、"never"（鼠标指针不出现）。

displaySurface - 它指定要捕获的显示表面的类型。可取值："application"（包含被选中的应用程序打开的全部窗口）、"browser"（只包含浏览器上被选中的标签页）、"monitor"（包含整个屏幕）、"window"（只包含被选中的应用程序窗口）。

logicalSurface - 是否允许用户选择不在显示区域的界面，比如被覆盖的窗口、超出范围需要滚动才能浏览的界面等。

getDisplayMedia()方法返回一个 Promise，当成功获取请求的媒体时，例行处理程序将收到 MediaStream 对象。举例：把上面例子中的方法 getUserMedia()改为 getDisplayMedia()，其他代码保持不变。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

3、enumerateDevices()

列举全部可用的多媒体设备，如麦克风、摄像头、头戴设备等等。此方法的声明如下：

```
navigator.mediaDevices.enumerateDevices(): Promise
```

enumerateDevices()方法返回一个 Promise，当成功获取请求的媒体时，例行处理程序将收到 MediaDeviceInfo 对象数组，数组中的每一个元素代表一个媒体设备。MediaDeviceInfo 对象的只读属性有：deviceId（媒体设备的唯一标识符）、groupId（组标识符，属于同一个物理设备的若干个媒体设备属于同一组，比如带有内置摄像头和麦克风的显示器）、kind（媒体设备类型，类型有"videoinput"、"audioinput"、"audiooutput"）、label（描述，如"External USB Webcam"）。

下面的例子以有序列表方式列出本机支持的全部媒体设备：

```
<body>
    <h2>本机的多媒体设备：</h2>
    <ol id="devices"></ol>
    <script>
        if(!navigator.mediaDevices || !navigator.mediaDevices.enumerateDevices)
{
            console.log("不支持 enumerateDevices()方法！");
            throw "exit";
        }

        navigator.mediaDevices.enumerateDevices()
            .then(devices => {
                var oll = document.getElementById("devices");
                devices.forEach(function (device) {
                    let li = document.createElement("li");
                    li.innerText = "deviceId: " + device.deviceId + ", kind: "
 + device.kind + ", label: " + device.label;
                    oll.appendChild(li);
                })
            })
            .catch(function (err) {
                alert(err.name + ": " + err.message);
            });
    </script>
</body>
```

6.9.2 MediaStream

MediaStream 是 HTML5 中最重要的用来操纵媒体流的接口对象，比如从媒体流中删除轨道、添加轨道、创建新的媒体流等。MediaStream 对象的属性、事件和方法介绍如下：

1、属性

active – 布尔类型，true 表示媒体是活动的。活动的媒体流表明存在没有结束的轨道。

ended – 布尔类型，true 表示全部轨道播放结束。

id – 全球唯一 id 号（uuid）。

2、事件

addtrack – 增加轨道时触发。

removetrack – 删除轨道时触发。

active – 媒体流被激活时触发。

inactive – 停用时触发。

3、方法

addTrack(track) – 往媒体流中添加轨道，无返回值。参数 track 为 MediaStreamTrack 类型，表示被添加的媒体轨道。

clone() – 克隆媒体流。克隆出来的媒体流具有唯一的 id 号。

getAudioTracks() – 获取媒体流中全部的音频轨道，返回 MediaStreamTrack 对象数组。音频轨道就是属性 kind 的值为"kind"的轨道。

getVideoTracks() – 获取媒体流中全部的视频轨道，返回 MediaStreamTrack 对象数组。视频轨道就是属性 kind 的值为"video"的轨道。

getTrackById(id) – 返回指定 id 号的轨道。

getTracks() – 获取媒体流中的全部轨道，返回 MediaStreamTrack 对象数组。

removeTrack(track) – 从媒体流中删除轨道。参数 track 为 MediaStreamTrack 类型，指定被删除的轨道。

6.9.3 MediaStreamTrack

MediaStreamTrack 对象代表媒体流中的单个媒体轨道，主要是音频轨道和视频轨道，其他的轨道还有文本轨道、图片轨道等。此对象的属性、事件和方法介绍如下。

1、属性

contentHint – 轨道的内容类型的简短描述。

enabled – 布尔类型，如果为 true 表明能呈现播放效果，否则黑屏或者静音。如果轨道已断开连接，则可以更改该值，但不再起作用。比如通过设置音频轨道的 enabled 属性来实现开/关“静音”的按钮。

id –轨道的 uuid，由浏览器产生。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

isolated – 布尔类型，如果为 true 表明本轨道被隔离。轨道一旦被隔离，则页面文档就无法访问该轨道。当设置 RTCPeerConnection.peerIdentity 属性或轨道来自跨源时，轨道就会隔离。

kind – 轨道类型，"audio"表示音频轨道，"video"表示视频轨道。

label - 返回一个网页客户端（如浏览器）分配给轨道的标签，用于标识轨道源，如“内部麦克风”。 该属性的值允许为空，并且只要未连接任何源，它就一直为空。 如果取消轨道与其源的关联，那么 label 属性的值就不再改变了。

muted – 布尔类型，true 表示静音，false 表示取消静音。

readonly – 为 true 表示轨道内容是只读的（比如视频文件源，或者不能更改设置的摄像头）。

readyState – 轨道状态，如果状态是"live"，表明实时媒体数据正从源流入，此时可设置 enabled 属性来关闭或开启媒体数据输出；如果状态为"ended"，表明源不再流入新的媒体数据。

remote – 为 true 表示轨道源为 RTCPeerConnection。

2、事件

ended – 播放结束时触发，即值 readyState 更改为 end。

mute – muted 属性更改为 true 时触发，此时轨道暂时不再提供数据。

isolationchange – 只要 isolated 属性的值发生改变就触发。

unmute - muted 属性更改为 false 时触发。

3、方法

applyConstraints([constraints]) – 给轨道施加若干限制（如每秒帧数、消除回声、分辨率等），以满足页面或 app 的媒体呈现偏好。可选参数 constraints 定义具体的限制项，请参考 getUserMedia() 方法的同名参数介绍。没有出现在参数 constraints 中的限制项被恢复称默认值，所以如果省略参数 constraints，那么全部限制项被恢复默认值。本方法返回一个 Promise，如果成功施加限制，则确值，否则拒绝。看例子：

```
const constraints = {
     width: { min: 640, ideal: 1280 },
     height: { min: 480, ideal: 720 },
     advanced: [
         { width: 1920, height: 1280 },
         { aspectRatio: 1.333 }
     ]
};

navigator.mediaDevices.getUserMedia({ video: true })
.then(mediaStream => {
     const track = mediaStream.getVideoTracks()[0];
     track.applyConstraints(constraints)
     .then(() => {
          console.log("success!");
     })
```

```
    .catch(e => {
        console.log("failed@");
    });
});
```

clone() – 克隆轨道，新轨道除了 id 不同外，其他都相同。

getCapabilities() – 获取可更改的限制项。返回一个 MediaTrackCapabilities 对象，指明每个可更改属性的取值范围。注意，在不同平台和不同的网页客户端上调用本方法，返回的值可能不同。一旦知道允许施加哪些限制相，你就可以调用 applyConstraints()方法施加具体的限制了。

getConstraints() – 返回一个 MediaTrackConstraints 对象，其中包含了轨道最近施加的限制项。下面的函数用于选择前/后置摄像头：

```
function switchCameras(track, camera) {
    const constraints = track.getConstraints();
    constraints.facingMode = camera;
    track.applyConstraints(constraints);
}
```

getSettings() - 返回一个 MediaTrackSettings 对象，其中包含了轨道当前起作用的所有限制项。



图 6.29 getConstraints()和 getSettings()方法的输出

stop() – 停止媒体轨道，表示本轨道不再需要媒体源，但并不代表相应的媒体源就停止了，因为可能其他轨道还在使用这个媒体源，当一个媒体源的全部轨道都停止后，媒体源才停止。一旦调用 stop()方法，readyState 属性的值被值为"ended"，同时触发 ended 事件。下面的函数停止播放视频（videoElem 参数为<video>元素对象）：

```
function stopStreamedVideo(videoElem) {
    const tracks = videoElem.srcObject.getTracks();
    tracks.forEach( track => track.stop() );
    videoElem.srcObject = null;
}
```

最后我们来看一个综合例子，实现本小节开始的那个图形中的媒体流，代码在 chapter-6\mediastream.html 中，浏览效果（点击“演讲”按钮）如图 6.30。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

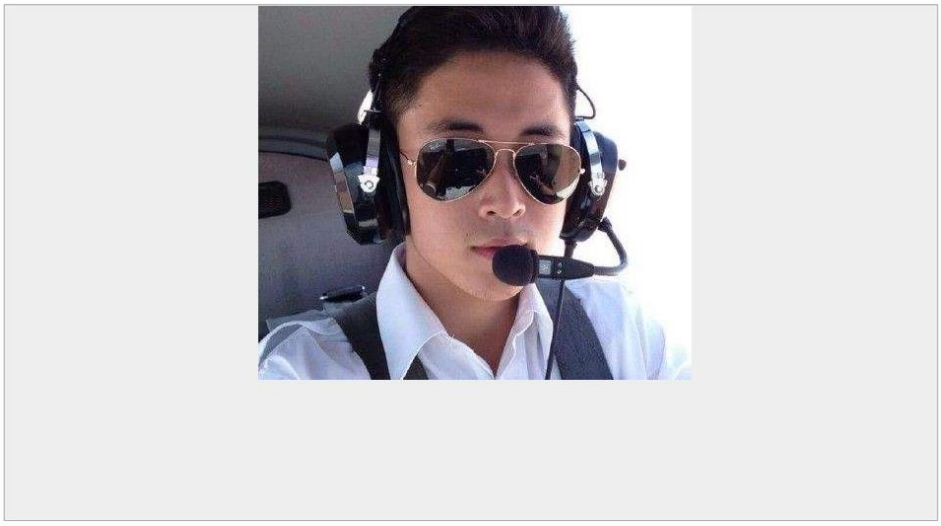


图 6.30 媒体流举例

不过目前 HTML5 的<video>元素不支持播放画中画视频（多视频轨道）。

6.9.4 MediaRecorder

MediaRecorder 接口对象用于录制媒体，录制完毕后可以通过<video>或<audio>元素播放出来，也可以存储到文件中。

首先采用如下所示的代码生产一个 MediaRecorder 对象。

```
var mediaRecorder = new MediaRecorder(stream[, options]);
```

构造器的参数 stream 指定被录制的媒体流，类型是 MediaStream，媒体流也可来自<audio>或<video>元素对象的 srcObject 属性、<canvas>元素对象的 captureStream()方法。可选参数 options 为包含如下属性的字典：

mimeType – 定义多媒体类型，常见的类型有"audio/mpeg"（MP3 音频）、"video/mp4"（ MPEG-4 视频）、"video/ogg"（Ogg 视频）、"video/quicktime"（苹果 QuickTime 视频）。

audioBitsPerSecond – 录制音频的比特率。

videoBitsPerSecond – 录制视频的比特率。

bitsPerSecond – 录制音视频的比特率。上面两个属性优先，如果定义了本属性且上面两个属性中的一个（比如 audioBitsPerSecond），那么本属性只影响上面两个属性中的另一个（如 videoBitsPerSecond）。

如果没有指定录制比特率，默认视频为 2.5Mbps，音频的比特率与采样率和声道数有关。看例子：

```
    var options = {
          audioBitsPerSecond: 128000,
```

```
        videoBitsPerSecond: 2500000,
        mimeType: "video/mp4"
    }
    var mediaRecorder = new MediaRecorder(stream, options);
```

MediaRecorder 接口对象的属性、事件和方法价绍如下：

1、属性

mineType（只读） - 返回 MIME type。

state(只读) - 返回 MediaRecorder 对象的当前状态，"inactive"(没开始录制或已停止)、"recording"（录制中）或"paused"（已开始录制但暂停了）。

stream（只读） - 返回被录制的媒体流。

ignoreMutedMedia – 布尔类型，为 false 表示当媒体轨道静音或黑屏时也录制。默认为 false。

videoBitsPerSecond（只读） - 返回当前的视频录制比特率。

audioBitsPerSecond（只读） - 返回当前的音频录制比特率。

2、事件

dataavailable – 以时间片为周期触发，如果调用 start()方法（开始录制）时没有指定时间片参数，则当整个媒体流录制完毕后触发。事件处理函数的形参为一 BlobEvent 对象，此对象的 data 属性保存了录制的媒体数据，因此我们可以不断收集录制的媒体数据并做进一步处理。

error – 出错时触发。事件梳理函数的参数对象的 error.name 属性定义了错误名字。

pause – 暂停录制时触发。

resume –从暂停状态恢复录制时触发。

start – 开始录制时触发。

stop – 录制结束时触发。有两种情况会导致录制结束：一是调用 stop()方法，二是媒体流终止。

3、方法

pause() – 暂停录制，同时触发 pause 事件，状态（MediaRecorder.state）置为"paused"。

requestData() – 处于录制状态时，调用本方法会触发 dataavailable 事件，并且创建一个新的 Blob 对象以存储后续录制的媒体数据。在非录制状态时调用本方法会引发 DOM InvalidState 错误。

resume() – 从暂停状态恢复录制，同时触发 resume 事件。。

start([timeslice]) – 开始录制，触发 start 事件，状态置为"recording"。录制的媒体数据保存在一个 Blob 对象中，当遇到下面四种情况之一时触发 dataavailable 事件：(1)由 timeslice 参数定义的毫秒

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

数到，(2)调用了 requestData()方法，(3)媒体流结束，(4)调用了 stop()方法。前两种情况会新建一个 Blob 对象以存储后续录制的媒体数据。

stop() – 结束录制，会先后触发 dataavailable 和 stop 事件，状态置为"inactive"。

接下来举一个录制音频的例子，代码参见 chapter-6\mediarecorder\目录中的三个文件，其中 index.html 为页面文件，audio-record.css 为样式文件，audio-record.js 为 js 代码程序，其中音频可视化介绍请参考后续章节“§6.10.6 音频可视化”。浏览的效果见图 6.31。

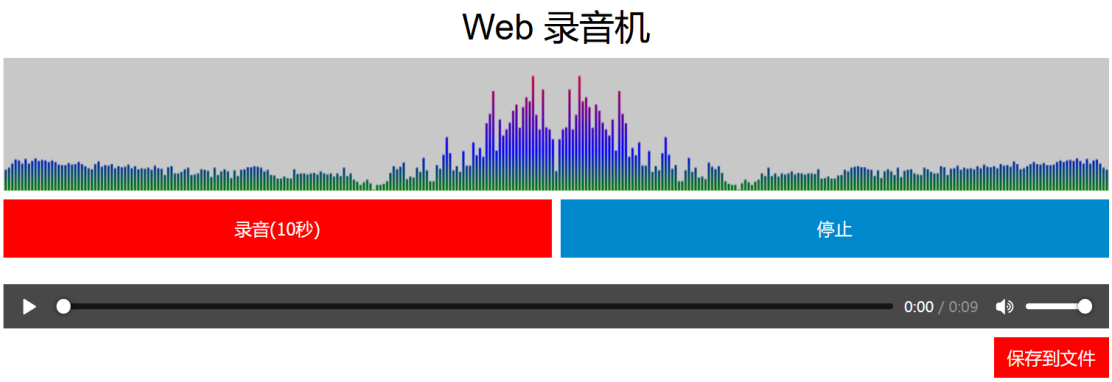


图 6.31 web 录音机

6.10 Web 音频 API

直接采用 HTML5 的元素<audio>和<video>播放音/视频，我们只能做简单的控制，比如播放、停止、调节音量大小，除此之外的诸如变调、混响、滤波、延时、压缩、均衡、上混和下混等高级控制，我们只有借助 AudioContext 接口对象了。当你熟悉了原生的 Web Audio Api 之后，在实际的项目开发当中，建议采用诸如 tone.js 等第三方的框架，比如类似 https://learningmusic.ableton.com/zh-Hans 这样的音乐教学网站就采用了 tone.js。

6.10.1 乐理知识

由于接下来要讲解处理音频的各种方法，例如变调、混响、滤波、均衡、延时等，所以有必要先了解一些乐理知识。

声音——由物体震动产生，并通过挤压空气等媒介传播，最终到达耳朵带动鼓膜震动，于是我们听到了声音。震动越强导致挤压空气的幅度越大，声音也就越大，如果用横坐标表示时间，纵坐标表示震动幅度，那么可用如图 6.32 所示的坐标系表示一段音频。

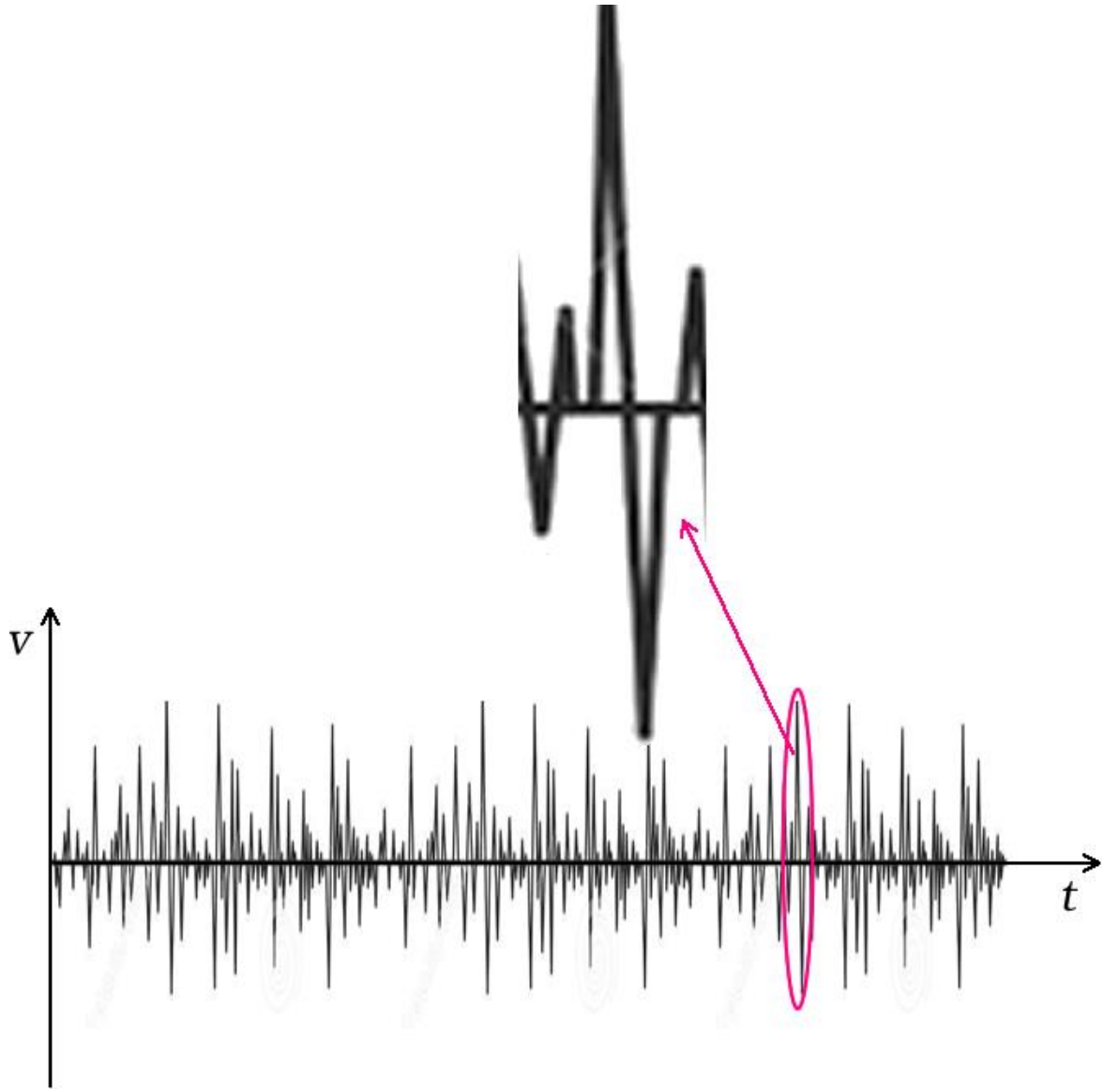


图 6.32 一段音频

音色——与震动的“物体”紧密相关，比如人的喉咙、鸟嘴、钢琴、小提琴、古筝等，小提琴与钢琴的音色各不相同，音色不同，则对应的声波波形也不同。参见图 6.33 所示。

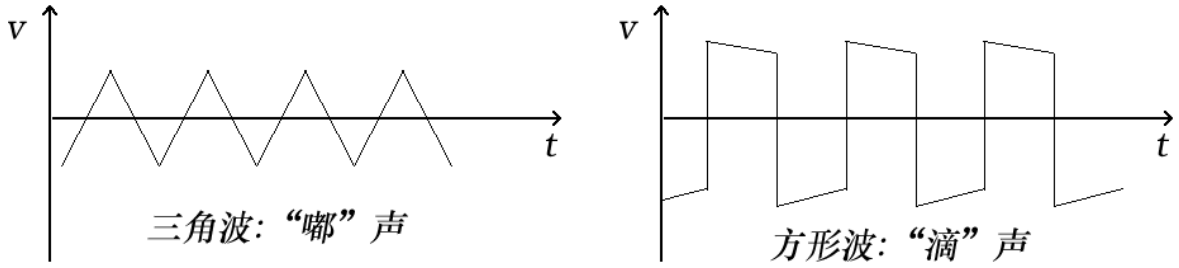


图 6.33 音频波形决定音色

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

响度——也称为音量，由声波的震幅决定，即声波坐标系中纵坐标的绝对值越大，震幅越强，声音也就越大。响度的单位是分贝，正常说话的响度为 50 分贝左右，大喊大叫则接近 70 分贝了。参考图 6.34。

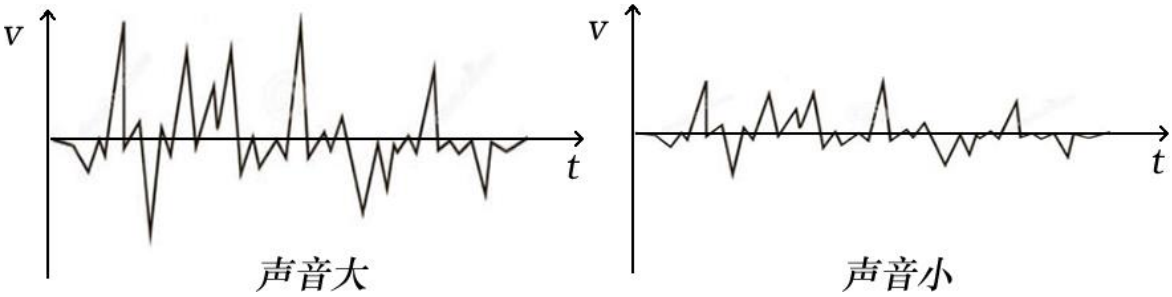


图 6.34 音频振幅决定音量大小

音高——由声波的频率决定，频率就是一秒钟震动多少次，单位是赫兹（Hz），比如 500Hz 就是每秒震动 500 次。人耳可听见的声音频率范围是 20Hz 至 20000Hz 之间。一个八度简谱音阶 1234567$\dot{1}$，对应的频率越来越高，且 $\dot{1}$ 的频率刚好是 1 的两倍，而且人们发现存在倍频关系的音听起来有相似性，比如 220Hz、440Hz、880Hz 三个音，尽管频率相差很大，但是入耳却类似。在通行的十二平均律中，互为倍频关系的两个音（含）之间均分为 12 个半音，共 1200 音分，即一个半音等于 100 音分。假如一个八度音阶的 1 对应的频率为 260Hz，那么各个简谱、频率和音分的关系参见图 6.35 下部分所示，显然此时低八度音阶的频率范围是 130Hz 到 260Hz，而高八度的频率范围为 520Hz 至 1040Hz 之间。

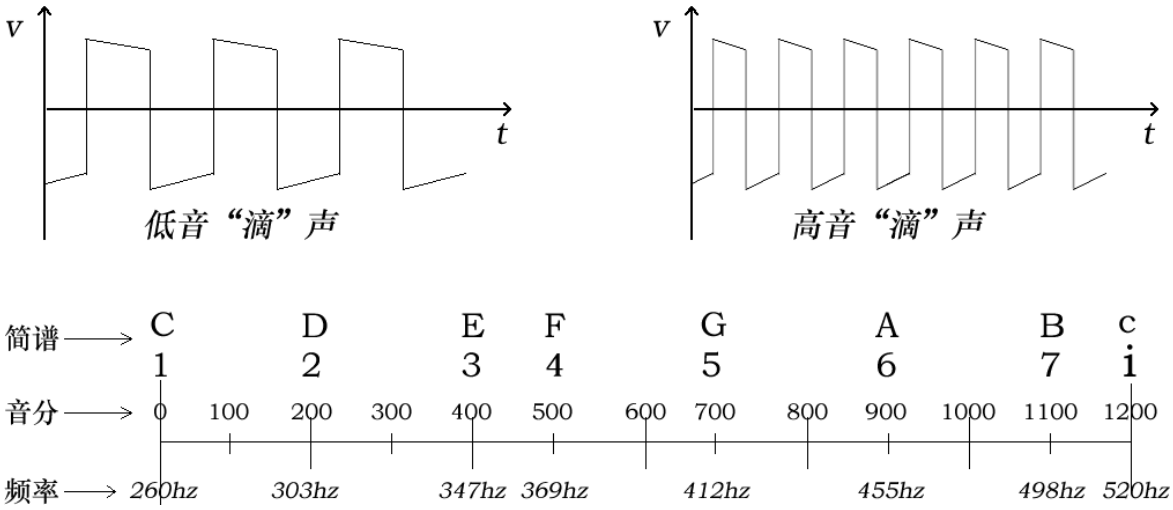


图 6.35 音频频率决定音高

一些英语国家用 CDEFGABc 代替数字简谱 1234567$\dot{1}$，对应关系参见上图。我们经常会发现一首数字简谱曲左上角标注类似“1=E”、“1=G”这样的字样，或者在口语或文字表述中提到“E 调”、“G 调”。“1=E”和“E 调”是同一个意思，即简谱 1 要发 3 的声音（因为字母简谱 E 等价于数字简谱 3），相当于告诉你 1 要提高两个全音，2 也要顺势提高两个全音，依此类推。图 6.36 是钢琴的标准键盘中音区，左侧的低音区和右侧的高音区没有绘制出来，白键是全音，黑键与相邻白键相差半个音。

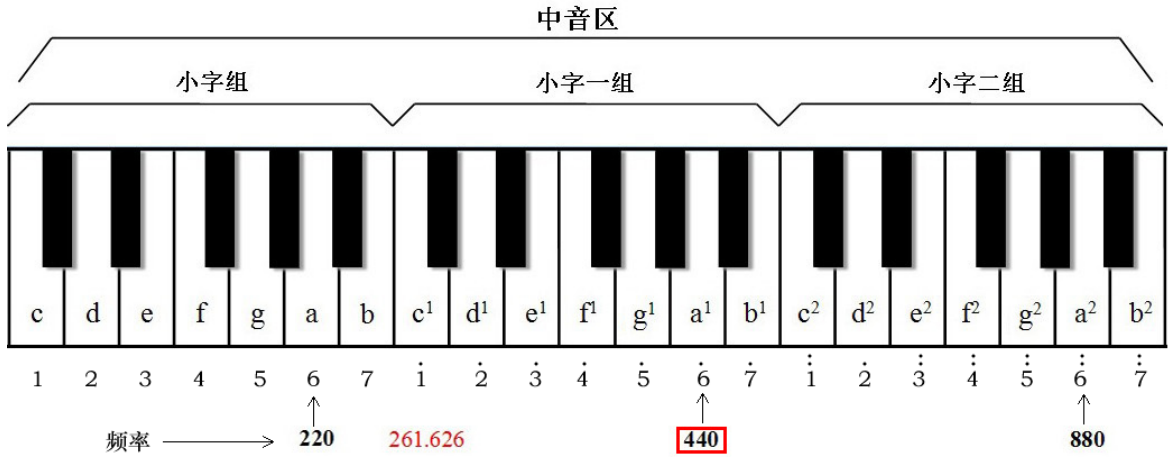
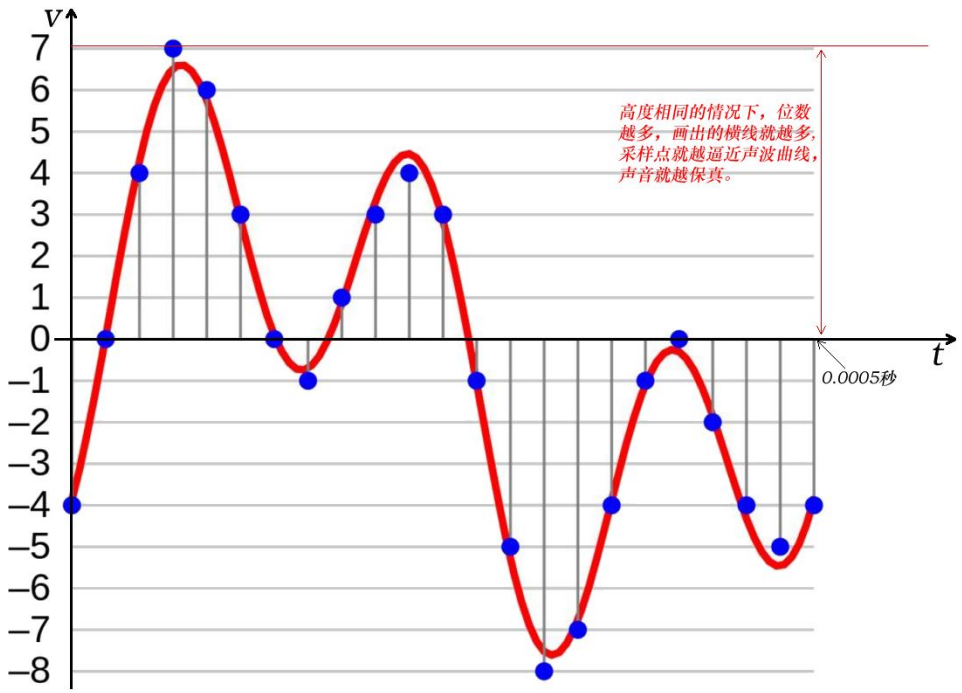


图 6.36 钢琴键盘的中音区

乐音——能轻易分辨出音色、音高的声音，例如钢琴、小提琴、古筝等发出的声音。

噪声——难以分辨出音高且比较混乱的声音，例如锣、鼓、爆炸声、瀑布声、雷鸣等等。

采样——物体震动的幅度在时间轴上是连续变化的，在坐标系中表现为连续曲线，见图 6.37 所示。为了把声音保存在电脑中，需要采样——即记录在时点 t_i 时的震动幅度 v_i，比如 0 秒幅度 0、0.0001 秒幅度 0.001、0.0002 秒幅度 0.004、……。每秒钟记录的数据个数称为采样率，采样率一般要远远大于声音的频率，否则会失真，采样率越高就越逼近原声，但数据量也越大，例如 mp3 音频的采样率为 44100Hz，即每秒记录 44100 个震幅数据。图 6.37 表示在 0.0005 秒内采样了 23 个数据，用 4 位二进制保存，可推算出采样率为 46000Hz。如果采样下来的每个数据采用 8 位二进制（即 1 个字节）保存，称为 8 位音频，如果用 16 位二进制（即 2 个字节）保存，则称为 16 位音频，目前 16 位音频是主流。位数越多，采样点越逼近声波曲线，因此声音越保真，比如 16 位二进制能储存-32768~+32767 之间的任何一个数字。



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

图 6.37 音频采样

对于上图，最终采样下来的数据是-4、0、4、7、6、3、0、-1、1、3、4、3、-1、-5、-8、-7、-4、-1、0、-2、-4、-5、-4。

音频 FFT——即音频快速傅里叶变换（Fast Fourier Transmit），根据傅氏理论可知：任意声波都可拆解成无数个不同频率（0 至∞）、不同相位和不同幅度的正弦波，参见图 6.38 所示。

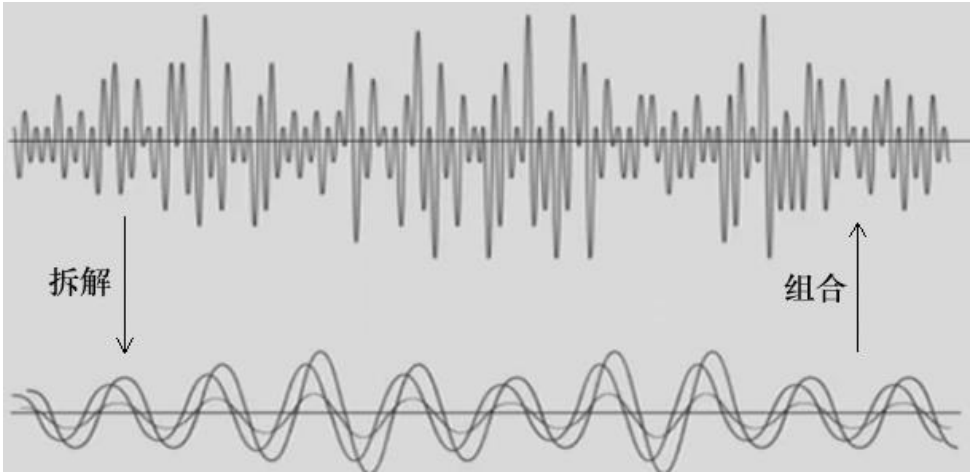


图 6.38 音频快速傅里叶变换

拆解之后可以去掉某些频率的正弦波（称为滤波）、或者调正某些正弦波的振幅（称为均衡），但没必要增加正弦波，因为声波本身就包含了任意频率的正弦波了。由于电脑只能处理离散数据，不能做无穷傅里叶变换，所以在电脑上只能拆解出有限个频率的正弦波，具体拆解出来的正弦波个数为每次傅里叶变换使用的采样数据个数的一半。例如采样频率为 44100Hz，而每次 FFT 使用 2048 个采样数据，那么只能拆解出 1024 个不同频率的正弦波了，而且 FFT 处理的频率为$\lfloor {}^{44100}/_{2048} \rfloor = 21Hz$，即做一次 FFT 的时间为 1/21≈46 毫秒，拆解出来的第 i 个正弦波的频率为$\mathrm{i} \times 21$赫兹。参考图 6.39。

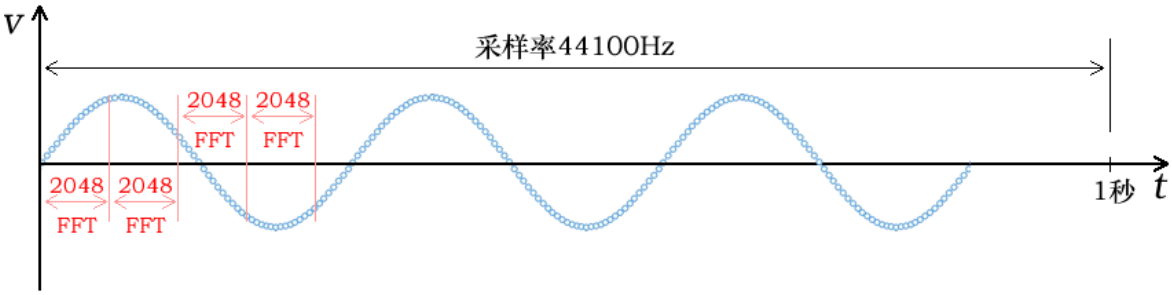


图 6.39 FFT 示意图

干声——直接录制的没有经过后期处理的原始声音。

湿声——经过后期处理（混响、调制、压限、变速等操作）的声音。

6.10.2 AudioContext

AudioContext 接口对象类似于录音棚，它创建各种输入节点来接收相应的音源，这些音源可来自麦克风、媒体流、音频文件和振荡器（由 JavaScript 编程产生的给定频率的声波）；它创建若干个处理节点来产生各种诸如延时、滤波、混音等音效；它创建若干输出节点把处理后的音频发到不同的目的地，比如喇叭、媒体流（后续可以把媒体流添加到 RTCPeerConnection 或保存到文件中）等。AudoContext 创建的每个节点有若干个输入口、若干个输出口，一个节点的输出口连接到另一个节点的输入口（采用节点对象的 connect()方法），从而形成一个音频处理有向图，见图 6.40 所示的一些例子。

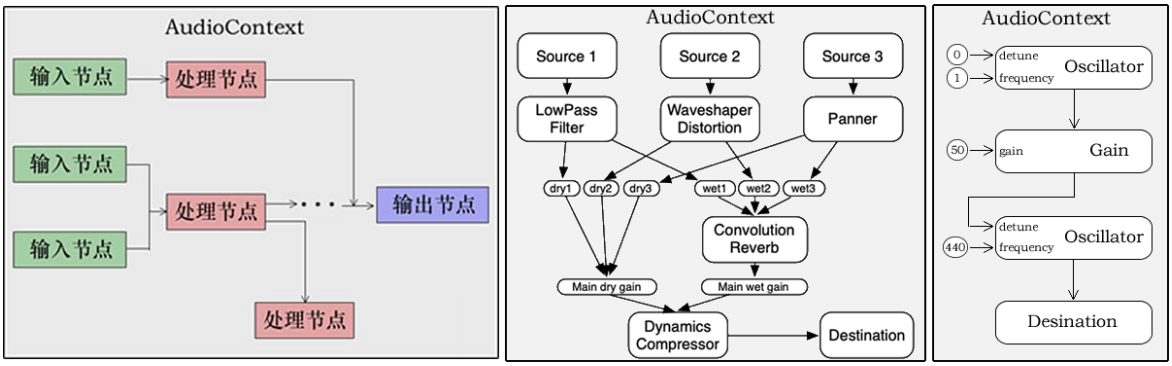


图 6.40 音频处理有向图

上图中间部分的 dry1、dry2、dry3、wet1、wet2、wet3、Main dry gain 都是 GainNode 节点（音量处理节点），其他节点参考下面的表格。

一个音频处理有向图上不一定有处理节点，如简单播放器就是输入节点直接连接输出节点，或者不一定存在输出节点，如以图形方式显示实时音量大小的频谱可视化。表 6.6 列举了常用的节点。

表 6.6 各种音频处理节点

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

类型	名称	对象类型	实例化 (默认是AudioContext的方法)	特性					说明
				输入口数	输出口数	声道数模式	声道数	上/下混策略	
输入节点	振荡器	OscillatorNode	.createOscillator()	0	1	max	2	speakers	用JavaScript编程产生的给定频率的声波
	内存音频	AudioBufferSourceNode	.createBufferSource()	0	1		由源确定		存放在内存中的一段解码的音频
	<audio>音频	MediaElementAudioSourceNode	.createMediaElementSource()	0	1				来自<audio>或<video>元素的音频
	媒体流	MediaStreamAudioSourceNode	.createMediaStreamSource()	0	1				来自WebRTC或者话筒的媒体流
	媒体流轨道	MediaStreamTrackAudioSourceNode	.createMediaStreamTrackSource()	0	1				来自WebRTC或者话筒的媒体流轨道
	常量源	ConstantSourceNode	.createConstantSource()	0	1				输出固定值
处理节点	可视化	AnalyserNode	.createAnalyser()	1	1	max	2	speakers	实时时域或频域可视化分析
	自定义基础类	AudioWorkletNode	new AudioWorkletNode()						在AudioWorkletProcessor中处理的节点
	滤波器	BiquadFilterNode	.createBiquadFilter()	1	1	max	2	speakers	常用于过滤噪声
	声道合并器	ChannelMergerNode	.createChannelMerger()	可变，默认6	1	max	2	speakers	把多个声道合并到一个
	声道拆分器	ChannelSplitterNode	.createChannelSplitter()	1	可变，默认6	explicit	等于输出口数	discrete	把一个声道拆分成多个
	混响器	ConvolverNode	.createConvolver()	1	1	clamped-max	1、2或4	speakers	环境混响音效
	延时器	DelayNode	.createDelay()	1	1	max	2	speakers	对输入做延时后再输出
	动态混音器	DynamicsCompressorNode	.createDynamicsCompressor()	1	1	clamped-max	2	speakers	提供压缩效果，以降低信号最响部分的音量
	音量控制	GainNode	.createGain()	1	1	max	2	speakers	调整音量大小
	均衡器	IIRFilterNode	.createIIRFilter()	1	1	max	同输入	speakers	实现音调控制和图形均衡器
	3D环绕音效	PannerNode	.createPanner()	1	1	clamped-max	2	speakers	音频源信号在空间中的位置和行为
	立体声声相器	StereoPannerNode	.createStereoPanner()	1	1	clamped-max	2	speakers	向左或向右改变声源位置
	非线性失真器	WaveShaperNode	.createWaveShaper()	1	1	max	2	speakers	使用曲线向信号施加波形失真
输出节点	声响器	AudioDestinationNode	.destination	1	0	explicit	2	speakers	通过喇叭播放声音
	媒体流	MediaStreamAudioDestinationNode	.createMediaStreamDestination()	1	0	explicit	2	speakers	只有一个轨道的MediaStream，后续可把它加到WebRTC链接上去。

6.10.3 AudioNode

所有的音频节点对象都继承 AudioNode 对象，因此这些节点都拥有 AudioNode 对象的属性和方法。我们首先介绍 AudioNode 对象，它一些主要属性和方法有：

numberOfInputs - 输入口个数，输入节点的输入口个数为 0。

numberOfOutputs - 输出口个数，输出节点的输出口个数为 0。

channelCountMode – 确定在对该节点的输入进行上混和下混时如何计算通道数，该属性在没有输入的节点上没有意义。可取值："max" - 等于连接到同一个输入口的所有链接中的最大声道数；"clamped-max" - 即 clamp(0, channelCount, max)，当 channelCount 大于连接到同一个输入口的所有链接中的最大声道数时，最终声道数就等于所有链接中的最大声道数，否则就等于 channelCount；"explicit"- 最终声道数就等于 channelCount。例如在图 6.41 中：如果 channelCountMode="max"，第 1 个 input 的最终声道数=4，如果为"clamped-max"或"explicit"，最终声道数=2。

channelCount - 进行上混和下混时使用的声道数量，默认是 2。最终输出的通道数还与 channelCountMode 和 channelInterpretation 属性有关。

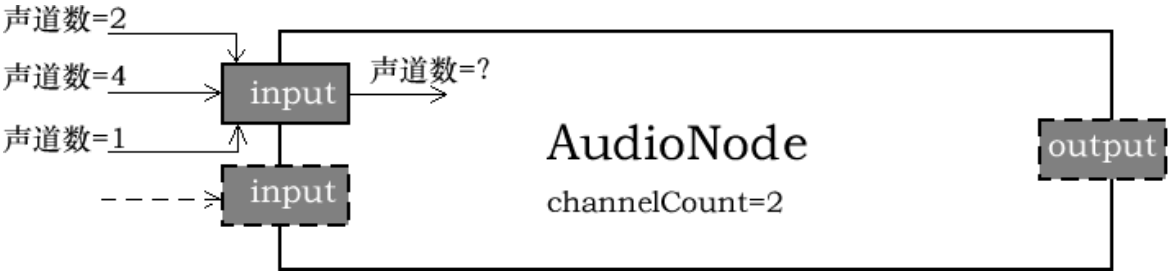


图 6.41 AudioNode 对象的属性

channelInterpretation – 定义上混和下混策略，该属性在没有输入的节点上没有意义。当输入声道数小于输出声道数时发生上混，否则发生下混。本属性可取值"discrete"（输入/输出按声道序号一一对应，多余的输出声道被静音或者多余的输入声道被忽略）、"speakers"策略定义的上混和下混操作见表 6.7。

表 6.7 上混/下混策略

输入声道	输出声道	上混/下混策略
单声道	双声道	上混——单声道被复制到左右声道，即： output.L = input.M, output.R = input.M
	4 声道	上混——单声道被复制到左右声道，左右环绕声道静音，即： output.L = input.M, output.R = input.M, output.SL = 0, output.SR = 0

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

	6 声道 (5.1)	上混——单声道被复制到中置声道，其他输出声道静音，即：output.C = input.M, output.RL= output.R = ioutput.SL = output.SR = output.LFE = 0
双声道	单声道	下混——输出声道为输入左右声道的平均值，即：output.M = 0.5 * (input.L + input.R)
	4 声道	上混——左右声道一一对应，左右环绕声道静音
	6 声道 (5.1)	上混——左右声道一一对应，其他声道静音
4 声道	单声道	Down-mix from quad to mono.
		下混——输出声道为输入四声道的平均值，即：output.M = 0.25 * (input.L + input.R + input.SL + input.SR)
	双声道	下混——输出左声道为输入左声道和左环绕声道的平均值，输出右声道为输入右声道和右环绕声道的平均值，即：output.L = 0.5 * (input.L + input.SL)，output.R = 0.5 * (input.R + input.SR)
	6 声道 (5.1)	上混——左右声道和环绕声道一一对应，输出中置和低音炮静音
6 声道 (5.1)	单声道	下混——计算公式为：output.M = 0.7071 * (input.L + input.R) + input.C + 0.5 * (input.SL + input.SR)
	双声道	下混——计算公司如下： output.L = input.L + 0.7071 * (input.C + input.SL) output.R = input.R + 0.7071 * (input.C + input.SR)
	4 声道	下混——计算公式如下： output.L = input.L + 0.7071 * input.C output.R = input.R + 0.7071 * input.C output.SL = input.SL output.SR = input.SR
其他非标布局		输入/输出按声道序号一一对应，多余的输出声道被静音或者多余的输入声道被忽略

connect(destination[, outputIndex, inputIndex]) – 本节点的输出连接到目标（由参数 destination 指定），目标如果是另一个节点，则音频流入那个节点的输入口，而如果目标是 AudioParam 接

口类型的实例，那么本节点的输出用于实时改变那个实例的值（比如 GainNode.gain 就是实时改变音量）。可选参数 outputIndex 和 inputIndex 指定本节点的输出口序号和目标节点的输入口序号，因为一个节点可能存在多个输出口和输入口。如果目标是节点，那么本方法返回目标节点，这样就方便我们写出这样的代码一次性串联多个节点：x.connect(y).connect(z).……。如果目标是 AudioParam，则本方法没有返回。

disconnect([destination, outputIndex, inputIndex]) – 断开连接，三个参数的含义与 connect()方法的参数一样。如果省略全部参数，则断开本节点的所有输出口与目标建立的连接。参数允许的组合有(destination)、(outputIndex)、(destination, outputIndex)、(destination, outputIndex, inputIndex)四种。

下面我们从最简单的音频处理有向图开始介绍各种流行的音效节点。

6.10.4 播放音乐文件

只有输入节点和输出节点，因为直接播放本地音乐文件，所以采用 AudioContext.createBufferSource()方法创建一个 AudioBufferSourceNode 类型的输入节点，输出节点为喇叭，即 AudioContext.destination 节点。参考图 6.42。

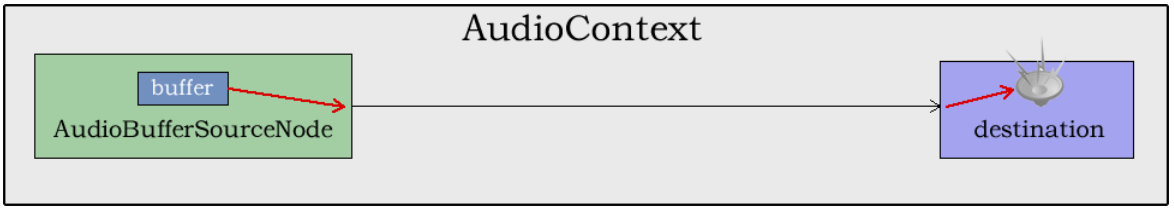
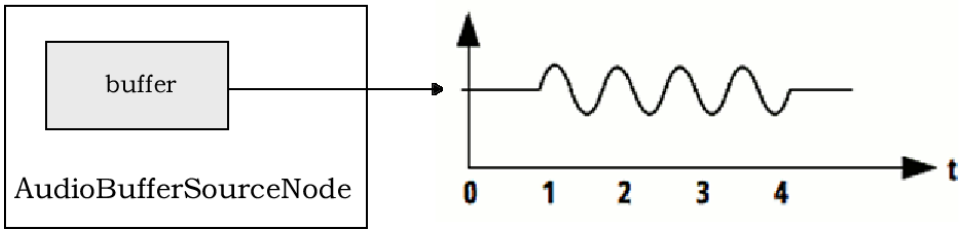


图 6.42 AudioBufferSourceNode 节点

AudioBufferSourceNode 类型的输入节点没有输入口，只有一个输出口，它直接把属性 buffer 中保存的解码音频按时序流到输出口，输出口的声道数由 buffer 的解码音频决定。一旦 buffer 中的解码音频数据流尽，AudioBufferSourceNode 就会自动销毁，所以如果要播放多个音频，就要多次实例化 AudioBufferSourceNode 对象，见图 6.43。



6.43 AudioBufferSourceNode 节点说明

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

由于保存在音乐文件中的音频数据是非解码的，所有我们需要采用AudioContext.decodeAudioData()方法把从文件读取的非解码音频进行解码，然后把解码音频数据赋给 AudioBufferSourceNode.buffer，最后调用 AudioBufferSourceNode.start()方法播放。AudioBufferSourceNode 对象的属性和方法介绍如下：

buffer – 类型为 AudioBuffer，保存待播放的解码音频数据，如果值置为 null，则表示静音单声道。

detune – 为 AudioParam 接口类型，用于调节音高，单位为音分，默认为 0。比如值比如 detune.value=100 就是提高一个半音，-100 降低一个半音，1200 提高一个八度。

loop – 为 true 则循环播放，默认为 false。

loopStart – 浮点数，单位为秒，定义每一轮循环的起点，默认为 0。

loopEnd – 浮点数，单位为秒，定义每一轮循环的终点，默认为 0（到歌曲结尾只有 0 秒）。

playbackRate – 为 AudioParam 接口类型，定义播放速率，默认为 1.0（正常速度）。比如 playblackRate.value=0.5 就是半速，2.0 就是倍速。

start() – 调用本方法开始播放音频。

stop() –停止播放。

AudioContext.decodeAudioData(ArrayBuffer, successCallback, errorCallback)方法就是对保存在 ArrayBuffer 中的非解码音频进行解码，如果解码成功再调用 sucessCallback(buffer)函数，其中实参 buffer 中保存了解码后的音频数据，解码失败则调用 errorCallback(error)，其中 error.err 保存了失败原因。也可采用 Promise 的写法：decodeAudioData(ArrayBuffer).then().catch()，具体参考下面的例子。

下面是完整的例子：

```
<input type="file" name="music" id="filePicker" accept="audio/*">
<script>
    var audioCtx = new AudioContext();
    var bufferSource = audioCtx.createBufferSource();
    bufferSource.connect(audioCtx.destination); //bufferSource-->destination

    function playMusic(evt) {
        const musicBuffer = evt.target.result; //这是保存了读取的未解码音频数据
        audioCtx.decodeAudioData(musicBuffer).then((decodeAudio) => {
            bufferSource.buffer = decodeAudio; //解码后的音频数据
            bufferSource.loop = true;
            bufferSource.start(); //开始播放音频
        }).catch((error) => console.log(error));
    }

    function readFile(evt) {
        var reader = new FileReader();
        reader.addEventListener("load", playMusic); //当读文件完成时调用 playMusic()
函数
         reader.readAsArrayBuffer(evt.target.files[0]);
```

```
    }
    window.addEventListener("load", () => {
        document.getElementById("filePicker").addEventListener("change", readFi
le);
    });
</script>
```

当我们通过输入框选择一个音乐文件后，调用 readFile(evt)函数，此函数成功读取音乐文件后再调用 playMusic(evt)函数，然后调用 decodeAudioData()对保存在 musicBuffer 中的未解码音频进行解码，解码成功后调用 start()方法播放。

下面我们在本例的基础上增加音量控制节点。

6.10.5 控制音量

控制音量采用 GainNode 节点，由 AudioContext.createGain()方法创建。GainNode 只有一个输入口和一个输出口，声道数 2、上混/下混策略为"speaker"，通道计数模式"max"。它除了继承了 AudioNode 对象的属性和方法外，增加了一个 AudioParam 接口类型的 gain 属性用来调节音量——设置 gain.value 或调用 AudioParam 接口的方法。参考图 6.44。

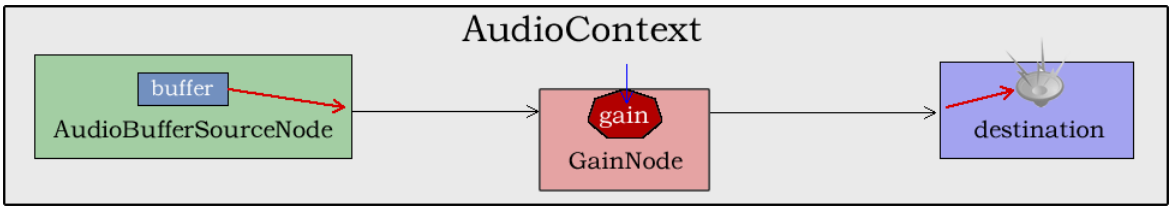


图 6.44 GainNode 节点用法举例

AudioParam 接口的属性和方法介绍如下。

defaultValue – 只读属性，表示刚开始时的默认音量。

value – 可写属性，代表当前的音量。初始值等于 defaultValue。

maxValue、minValue – 只读属性，定义了 value 属性的取值范围。

setValueAtTime(newValue, startTime) – 在指定的时间 startTime 更改属性 value 的值为 newValue。参数 startTime 为双精度类型，定义自从 AudioContext 创建以来的秒数，常常采用 AudioContext.currentTime+n 的形式，其中 AudioContext.currentTime 代表自从 AudioContext 创建以来至目前的秒数。例如 setValueAtTime(0.5, AudioContext.currentTime+5)表示 5 秒钟后音量调到最大音量的一半。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

linearRampToValueAtTime(newValue, endTime) – 从现在开始到 endTime 时点，把属性 vlue 的值线性调到 newValue，这样对于 GainNode 节点可实现淡入/淡出的音效。比如 linearRampToValueAtTime(0, AudioContext.currentTime+2)实现 2 秒钟内声音淡出。

exponentialRampToValueAtTime(newValue, endTime) – 按指数曲线改变属性 value 的值。通常用于调节音频频率或播放速率。

setTargetAtTime(newValue, startTime, timeConstant) – 从 startTime 开始经过 timeConstant 秒把属性 value 的值按指数曲线调到 newValue。如 gainNode.gain.setTargetAtTime(0, audioCtx.currentTime + 1, 1.5)。

setValueCurveAtTime(values, startTime, duration) – 从 startTime 时点开始在 duration 秒内按照由浮点数组 values 定义的顶点坐标$\left(values[i], startTime + \left(\frac{duration}{values.length-1}\right) \times i\right)$所构成的折线改变属性 value 的值。例如：

```
setValueCurveAtTime([0.5, 1, 0.5, 0, 0.5, 1, 0.5, 0, 0.5], audioCtx.currentTime
 + 1, 4);
```

绘成坐标如图 6.45 所示，属性 value 的值按照坐标中那条折线改变。

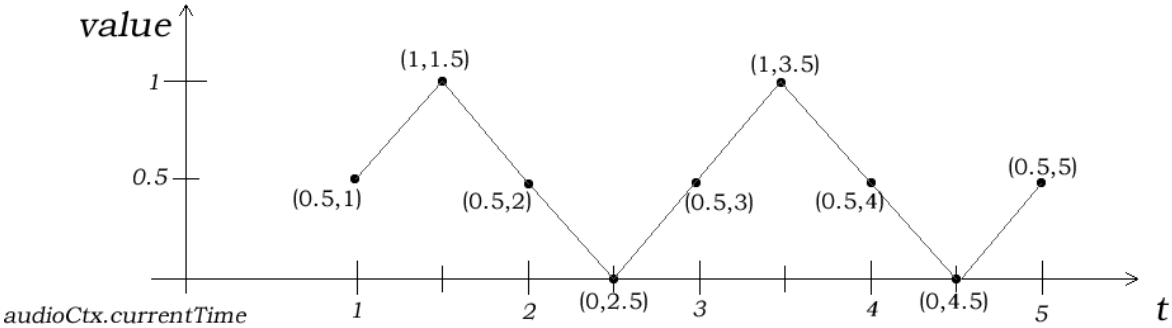


图 6.45 按照折线改变音量大小

cancelScheduledValues(startTime) – 取消 startTime 时点正在生效或之后将要生效的所有任务，但是调用本方法时生在生效的任务除外，调用本方法后安排的任务继续有效。例如 cancelScheduledValues(audioCtx.currentTime)取消所有任务，但是当前正在生效的任务不受影响。

cancelAndHoldAtTime(startTime) - 取消 startTime 时点之后将要生效的所有任务，而 startTime 时点正在生效的任务立即终止，但是属性 value 的值保留。看例子：

```
    gain.value = 1;
    gain.linearRampToValueAtTime(3, audioContext.currentTime + 2);
    gain.cancelAndHoldAtTime(audioContext.currentTime + 1);
    //gain.cancelScheduledValues(audioContext.currentTime + 1);
```

value 的当前值是 1（第 1 条语句），从现在开始的 2 秒中内 value 的值线性变为 3（第 2 条语句），1 秒钟时取消正在生效的任务（即第 2 条语句定义的任务），但是此时 value 的值已经变化到 2 了，所以它的值继续保持 2。但是如果把第 3 条语句改为第 4 条被注释的语句，那么第 2 条语句安排的任务根本不会启动，所以属性 value 的值一值没变，还是 1。

我们在上面“1、播放音乐文件”的例子上添加两个按钮，点击时声音逐渐淡出和淡入，再添加一个 range 类型的<input>元素来控制音量大小。

```
<input type="file" name="music" id="filePicker" accept="audio/*">
```

```
<input id="volume" type="range" min="0" max="100" value="100">
<span>音量控制</span>
<button id="fakeout">声音淡出</button> <button id="fakein">声音淡入</button>
<script>
    var audioCtx = new AudioContext();
    var bufferSource = audioCtx.createBufferSource();
    var gainNode = audioCtx.createGain();
    bufferSource.connect(gainNode).connect(audioCtx.destination); //bufferSourc
e-->gainNode-->destination

    function gainVolume(evt) {
        gainNode.gain.value = parseFloat(evt.target.value / 100.0);
    }
    function fakeoutVolume(evt) {
        gainNode.gain.linearRampToValueAtTime(0.0, audioCtx.currentTime + 2);
    }
    function fakeinVolume(evt) {
        gainNode.gain.setTargetAtTime(1.0, audioCtx.currentTime, 3);
    }
    document.getElementById("volume").addEventListener("change", gainVolume);
    document.getElementById("fakeout").addEventListener("click", fakeoutVolume);
    document.getElementById("fakein").addEventListener("click", fakeinVolume);

    function playMusic(evt) {
```

灰色以外的代码是新加的，拖动滑块调节音量大小，点击一下“声音淡出”按钮，声音就会慢慢变小直至无声，相反点击“声音淡入”按钮，声音由无声逐渐变大。浏览的效果如图 6.46 所示。



图 6.46 增加音量控制

6.10.6 拆分声道

我们在“§6.10.4 控制音量”的基础上拆分声道，并对其中一个声道做延时处理，最后合并声道再播放，音频处理有向图见图 6.47 所示。

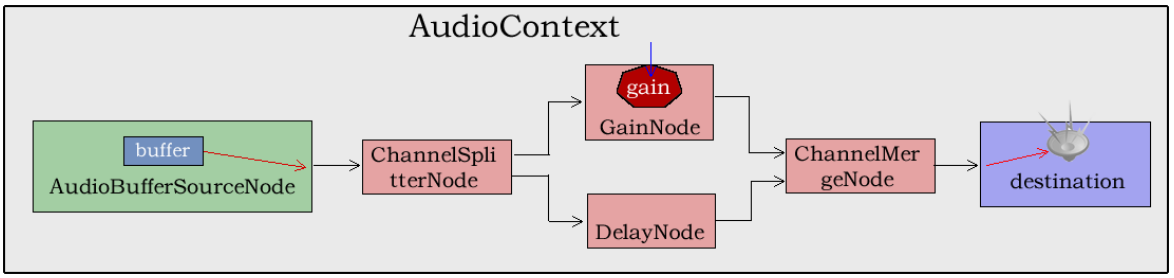


图 6.47 增加声道拆分节点

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

ChannelSplitterNode 节点比较简单，直接用 AudioContext.createChannelSplitter(numberOfOutputs)实例化即可，其中 numberOfOutput 指定拆分后的声道数，一个声道占用一个输出口。类似地，声道合并节点 ChannelMergerNode 采用 AudioContext.createChannelMerger(numberOfInputs)实例化。延时节点 DelayNode 采用 AudioContext.createDelay(maxDelayTime)实例化，其中参数 maxDelayTime 定义最大的延时秒数，取值范围为[1,180]，默认为 1 秒，具体的延时秒数要设置 AudioParam 接口类型的属性 delayTime（方法参见 gainNode 的 gain 属性）。在上一个例子的基础上添加下面的除灰色之外的代码。

```
    <script>
        var audioCtx = new AudioContext();
        var bufferSource = audioCtx.createBufferSource();
        var gainNode = audioCtx.createGain();
        var spliterNode = audioCtx.createChannelSplitter(2);
        var mergerNode = audioCtx.createChannelMerger(2);
        var delayNode = audioCtx.createDelay(10.0);
        delayNode.delayTime.setValueAtTime(2.0, audioCtx.currentTime);

        bufferSource.connect(spliterNode);//bufferSource-->spliterNode
        spliterNode.connect(gainNode,1);
        spliterNode.connect(delayNode,0);
        gainNode.connect(mergerNode,0,0);
        delayNode.connect(mergerNode,0,1);
        mergerNode.connect(audioCtx.destination);

        function gainVolume(evt){
            gainNode.gain.value = parseFloat(evt.target.value/100.0);
        }
```

6.10.7 音频可视化

以图形方式显示音频的音量大小或者各频率的振幅，此时要采用 AnalyserNode 节点，该节点使用快速傅里叶变换（FFT）把音频拆分出若干频率的正弦波，再把这些正弦波的振幅可视化即可，或者直接把 FFT 使用的采样数据可视化。绘制实时动画图形最常用的方法是采用<canvas>画布和 requestAnimationFrame()方法。我们把 AnalyserNode 加在 destination 节点前的旁路上，见图 6.48 所示。

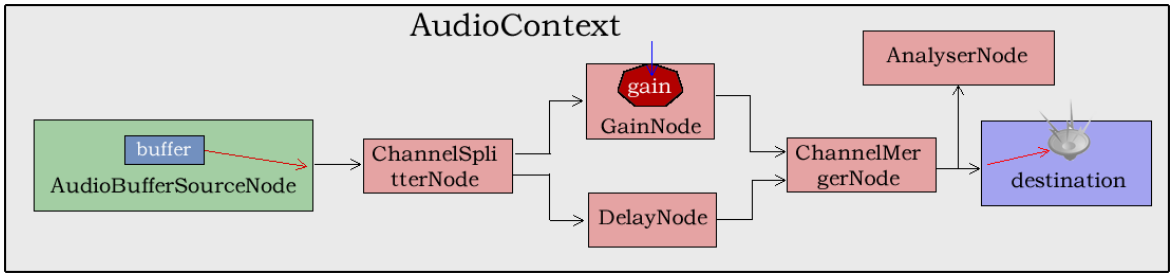


图 6.48 增加音频可视化节点

AnalyserNode 节点的属性、方法介绍如下：

fftsize – 每次执行快速傅里叶变换使用的采样数据个数，值一般为 2 的幂次方，默认为 2048。

frequencyBinCount – 只读属性，保存了拆分出的不同频率的正弦波的个数，它等于 fftsize 的二分之一。

minDecibels、maxDecibels – 分别定义最小和最大分贝。音频被拆分为不同频率的正弦波后，震幅等于或小于 minDecibels 的正弦波的震幅置为 0.0（getFloatFrequencyData()获取频域数据时）或 0（getByteFrequencyData()获取频域数据时），等于或大于 maxDecibels 的正弦波的震幅被置为 1.0 或 255。由于 FFT 计算出来的不同频率正弦波的震幅都为负数，所以 minDecibels 和 maxDecibels 取值为 0 或者负数，0 表示最大的声音。假如我们采用 getByteFrequencyData()方法获取频域数据，那么区间[minDecibels,maxDecibels]被映射到区间[0,255]，因此频率为 k 的正弦波的震幅 Y_k 被转换为[0,255]区间上的$\left\lfloor \frac{255}{maxDecibels-minDecibels} \times (Y_k - minDecibels) \right\rfloor$。minDecibels 的默认值为-100，maxDecibels 的默认值为-30。

smoothingTimeConstant – 时序数据的平滑常数，定义时间上存在先后关系的两个值之间的关联度，0 表示没有关联，1 表示两个值重度关联（几乎相等），默认是 0.8。例如频率为 k 的正弦波的震幅上一次计算的值为 Y_1，本次计算的值为 Y_2，如果平滑常数设置为 0，那么本次直接输出 Y_2，否则计算并输出它们的关联值（介于 Y_1 和 Y_2），因此绘制的曲线更平滑。

getFloatFrequencyData(dataArray) – 把 FFT 计算的频谱数据拷贝到 Float32Array 类型的 dataArray 数组中，dataArray 数组的元素个数一般等于 AnalyserNode.frequencyBinCount，如果不相等，那么多出来的被忽略。静音对应的频谱数值为-Infinity。datArray[i]就是频率为 i 的正弦波的震幅（分贝），一般频率越小，震幅越大。

getByteFrequencyData(dataArray) - 把 FFT 计算的频谱数据拷贝到 Uint8Array 类型的 dataArray 数组中，分贝值被映射到[0,255]区间。如果要求精度很高，请使用 getFloatFrequencyData()方法。

getFloatTimeDomainData(dataArray) – 直接把参与 FFT 计算的采集点数据（时域数据）拷贝到 Float32Array 类型的 dataArray 数组中，dataArray 数组的元素个数一般等于 AnalyserNode.fftsize。datArray[i]就是第 i+1 个采集点的数据。时域数据的个数是频域数据的 2 倍。

getByteTimeDomainData(dataArray) - 直接把参与 FFT 计算的采集点数据（时域数据）拷贝到 Uint8Array 类型的 dataArray 数组中。

本小节的代码如下：

```
<body>
<canvas id="visualizer" height="120px" width="800px"></canvas><br>
    //这里的语句参见上一小节的代码
    mergerNode.connect(audioCtx.destination);

    const canvas = document.getElementById("visualizer");
    const analyser = audioCtx.createAnalyser();
    mergerNode.connect(analyser);//mergerNode 同时连接 destination。
    visualize();
    function visualize() {
        const canvasCtx = canvas.getContext("2d");
        //analyser.fftsize 默认为 2048，所以每次做 FFT 计算使用 2048 给采样数据，产生 1024
个不同频率正弦波
        const bufferLength = analyser.frequencyBinCount; //1024
        const dataArray = new Uint8Array(bufferLength);
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
        var linear = canvasCtx.createLinearGradient(0, 0, 0, canvas.height);
        linear.addColorStop(0, "red");
        linear.addColorStop(0.7, "blue");
        linear.addColorStop(1, "green");

        draw()

        function draw() {
            const hWIDTH = canvas.width / 2;
            const HEIGHT = canvas.height;

            analyser.getByteFrequencyData(dataArray);//获取每个频率的震幅
            canvasCtx.fillStyle = 'rgb(200, 200, 200)';
            canvasCtx.lineWidth = 2;
            canvasCtx.strokeStyle = linear;
            canvasCtx.fillRect(0, 0, canvas.width, HEIGHT);

            canvasCtx.beginPath();

            let step = Math.floor(bufferLength * 3.0 / hWIDTH);
            let x = 0;

            for (let i = 0; i < bufferLength; i = i + step) {
                let y = dataArray[i] * HEIGHT / 256;
                canvasCtx.moveTo(hWIDTH + x, HEIGHT);
                canvasCtx.lineTo(hWIDTH + x, HEIGHT - y);
                canvasCtx.moveTo(hWIDTH - x, HEIGHT);
                canvasCtx.lineTo(hWIDTH - x, HEIGHT - y);
                x += 3;
            }
            canvasCtx.stroke();
            requestAnimationFrame(draw);
        }
    }

    function gainVolume(evt) {
        gainNode.gain.value = parseFloat(evt.target.value / 100.0);
    }
```

浏览的效果如图 6.49。

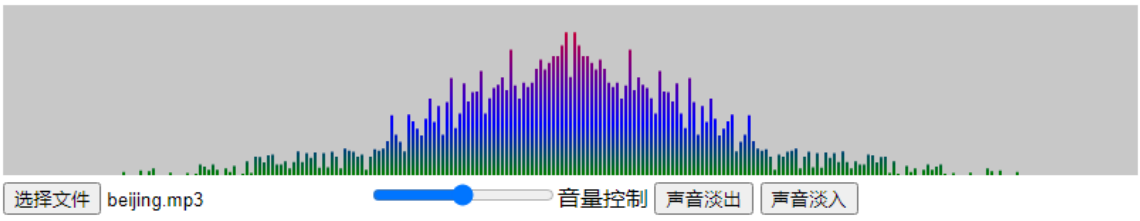


图 6.49 音频可视化举例

让我们尝试修改 analyser.minDecibels = -120、analyser.maxDecibels = -1、analyser.smoothingTimeConstant = 0.6，看看效果如何？

6.10.8 滤波

我们继续在上一节的基础上添加音频滤波器，过滤一些高频或者低频信号。IIRFilterNode 和 BiquadFilterNode 节点都能实现滤波，但前者是通用的滤波器，而后者是前者的特例，所以直接用后者能快速实现满足部分特定场合的滤波，但如果使用前者就能更灵活的定义滤波参数，从而针对各种场合定制最合适的滤波器。BiquadFilterNode 节点能实现若干种类的滤波器、音调控制器和均衡器。这次我们把 BiquadFilterNode 节点加在 ChannelMergerNode 节点之后，参见图 6.50。

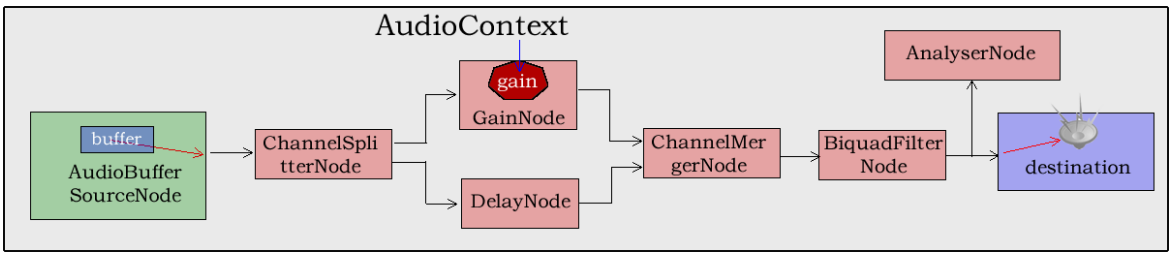


图 6.50 增加音频滤波节点

可采用 AudioContxt.createBiquadFilter()实例化一个 BiquadFilterNode 节点，它的属性和方法介绍如下：

frequency - 为 AudioParam 类型，定义当前算法中的参考频率（单位为 Hz），默认值为 350Hz。比如 frquency.value=5000，如果是高通滤波，则低于 5000Hz 的信号被过滤掉，如果是低通滤波，则高于 5000Hz 的信后被过滤掉。

detune – 变调——即改变频率，以音分为单位，100 音分为一个半音，为 AudioParam 类型，默认值为 0。比如 detune.value = 100 表示提高一个半音，detune.value=-200 表示降低一个全音。

Q – 为 AudioParam 类型的无量纲的品质因子，默认值为 1，取值范围一般为[0.0001,1000]。它定义为在一个振荡周期中存储在谐振器中的峰值能量与该周期每弧度损失的能量之比，Q 值越高，阻尼越弱，在不增加能量的情况下，震荡时间就越长。比如同一个摆锤在空气中振荡时具有较高的 Q 值，而浸入油中时则具有较低的 Q 值。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

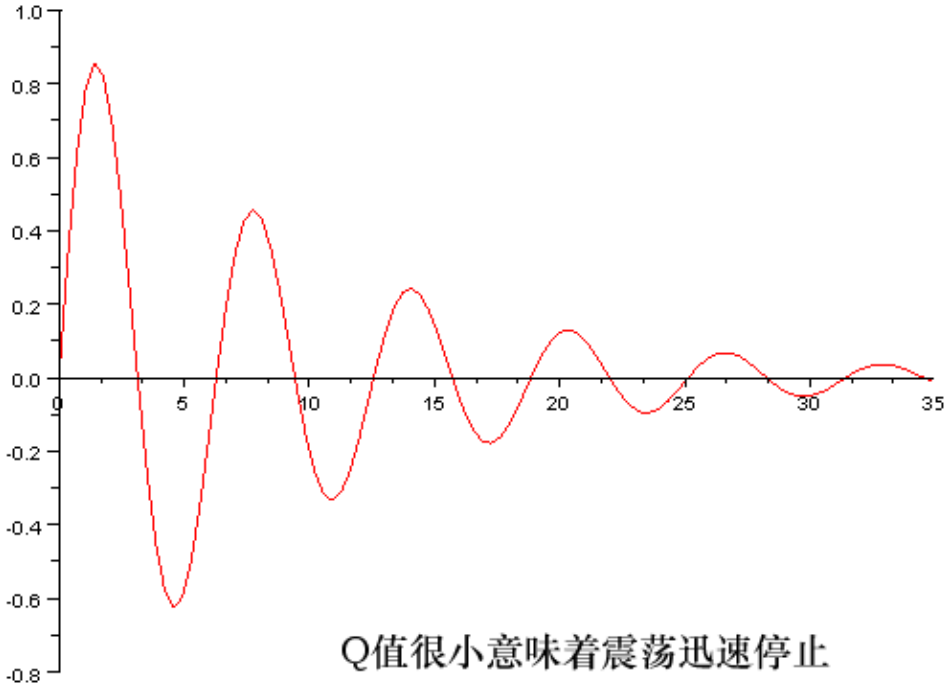


图 6.51 Q 值越大阻尼越小

gain - 表示当前过滤算法中使用的增益（放大或衰减音量），为 AudioParam 类型，单位为 dB，默认值为 0。当它取正值时，它是一个真实的增益，而当它取负值时，它是一个衰减。取值范围为-40 至 40。它于 gainNode 不同，gainNode 在静音至最大音量（假设为 maxVolume）之间调整，而本 gain 属性是在[maxVolume-40, maxVolume+40]之间调整。

type – 定义 BiquadFilterNode 节点执行的算法，算法不同，实现的功能也不同，默认为"lowpass"，即低通滤波器。type 属性的值为枚举类型，与其他属性的组合参见表 6.9。

表 6.8 滤波算法

type	Description	frequency	Q	gain
lowpass	标准二阶谐振低通滤波器，具有 12dB / 倍频程衰减。 截止频率以下的频率通过，高于它的频率被衰减。	截止频率	定义截止频率附近的频率的衰减程度，Q 值越大，衰减越小。	无用
highpass	标准二阶谐振高通滤波器，具有 12dB / 倍频程衰减。 截止频率以下的频率被衰减； 高于它的频率通过。	截止频率	定义截止频率附近的频率的衰减程度，Q 值越大，衰减越小。	无用
bandpass	标准的二阶带通滤波器。 给定频率范围之外的频率会衰减，范围之内的频率通过。	给定范围内的中心频率	控制频带的宽度。 Q 值越大，频带越大。	无用

notch	标准陷波滤波器，也称为带阻或带阻滤波器。与带通滤波器相反：超出给定频率范围的频率通过，内部的频率被衰减。	给定范围内的中心频率	控制频带的宽度。 Q 值越大，频带越大。	无用
lowshelf	标准二阶 lowshell 滤波器，允许所有频率通过，但是低于参考频率的频率会被增强或衰减	参考频率	无用	正数则增益，如果为负数，则为衰减。
highshelf	标准二阶 hightshell 滤波器，允许所有频率通过，但是高于参考频率的频率会被增益或衰减	参考频率	无用	正数则增益，如果为负数，则为衰减。
peaking	峰值滤波器允许所有频率通过，但给定范围内的频率会被增益或衰减，范围之外的频率不变	给定范围内的中心频率	控制频带的宽度。 Q 值越大，频带越大。	正数则增益，如果为负数，则为衰减。
allpass	标准二阶全通滤波器。它允许所有频率通过，但会更改各个频率之间的相位关系。	发生相变的中心频率。	定义中心频率处的相位过渡的尖锐程度，Q 值越大表示越陡峭的过渡和更大的组延迟	无用

在上一节的例子中添加如下代码：

```
        delayNode.connect(mergerNode, 0, 1);
        var filterNode = audioCtx.createBiquadFilter();
        //filterNode.frequency.value = 250;
        mergerNode.connect(filterNode).connect(audioCtx.destination);

        const canvas = document.getElementById("visualizer");
        const analyser = audioCtx.createAnalyser();
        filterNode.connect(analyser);
        visualize();
```

滤波器的属性全取默认值，即参考频率 350Hz、Q 值 1（迅速衰减）、低通滤波器，从而实现的音效是高于 350Hz 频率的音频被过滤掉（被迅速衰减）。通过观察可视化图形，发现只有中间有条状。使上述代码被注释的语句生效，然后看看观察可视化图形有何变化。也可以采用下面的语句逐渐修改 frequency 的值：

```
filterNode.frequency.exponentialRampToValueAtTime(10, audioCtx.currentTime + 30
);
```

下面的语句实现 lowshelf 滤波器，增益低于参考频率的音频：

```
        filterNode.type = "lowshelf";
        filterNode.gain.value = 20;
        mergerNode.connect(filterNode).connect(audioCtx.destination);
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.10.9 卷积混响

你应该听过《山路十八弯》这首歌吧，李琼站在半山腰的溪水旁面对山谷放声歌唱，歌声在山谷中久久回荡，使人心醉。但我要告诉你她是在安静的录音棚里录歌的，然后与山谷中录制的冲击响应音频（Impulse Rsponse，即 IR）卷积混响在一起，你可能不太相信。本小节介绍的 ConvolverNode 节点就是完成卷积混响功能的，以前一台硬件卷积混响设备是非常昂贵的，即使现在，价格也不菲。

卷积混响涉及两段音频：一段是干音——被卷积处理的音频，比如李琼在安静的录音棚里录制的《山路十八弯》，另一段是在声响空间中录制的非常短（一般小于 5 秒）的冲击响应音频，比如在山谷中燃放一个鞭炮或开一枪，把枪声和随后的回响声录下来，并以 wav 格式保存。录制冲击响应音频有几个要求：(1)存在混响反射的声响空间里录制，比如礼堂、山谷、隧道等，混响反射就是对原声的**响应**，(2)原声尽量瞬间完成（对周围环境形成一次**冲击**），且音域宽广，即包含尽可能多的频率，比如开一枪，(3)以无损格式存放，比如 wav 或 flac 格式，(4)原声的响应声消失后及时终止录音，或者做后期处理时掐头去尾，目的是保证冲击响应音频有效短暂（数秒内消失），(5)原声与冲击响应音频具备相同的采样率。网上有很多冲击响应音频素材供选择，这些素材遍布各种声响环境，你录制的同一段唱声如果与礼堂的冲击响应音频卷积混响，效果就与你站在礼堂唱歌一样，如果与山谷的冲击响应音频卷积混响，效果与你站在山谷唱歌一样。

卷积混响的执行过程大致是这样：首先把两段音频通过傅里叶变换拆分成各种频率的正弦波，然后针对被处理音频的每个正弦波找到相同频率的冲击响应音频拆分出来的正弦波（所以音域越宽广越好），两个正弦波做相乘累加计算：假设在时点 n，两个正弦波的采集数据分别设为 f(n)、g(n)，卷积混响后设为 v(n)，则存在公式$v(n)=\sum_{t=0}^{n}\big(f(t)\times g(n-t)\big)$，g(n-t)就是 g(t)左右翻**卷**的结果，然后与 f(t)乘**积**，最后累加。用通俗的语言描述就是：根据冲击响应音频的回响反射数据，对干声做同样的回响反射处理。

下面我们采用 ConvolverNode 节点完成卷积混响，首先自己清唱一段歌曲并录下来，解码后置入 AudioBufferSourceNode 的 buffer 中，作为被卷积混响的干音，然后从网上下载一个冲激响应音频，解码后置入 convolverNode 的 buffer 中，音频处理有向图如图 6.52。

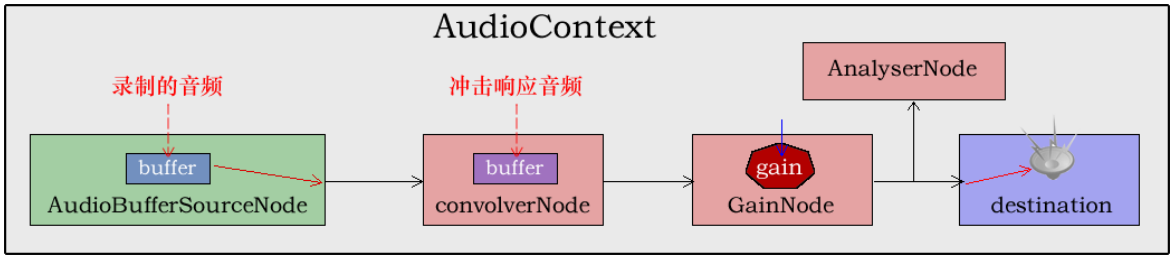


图 6.52 加入音频卷积节点的处理有向图

可使用 AudioContext.crteateConvolver()实例化 convolverNode 节点，它在自己的 buffer 属性所存储的解码音频上对原干执行线性卷积。convolverNode 对象有两个属性：

buffer - 为 AudioBuffer 接口类型，用于存储单声道或双声道或四声道的冲击响应解码音频。

normalize - 为布尔类型，它控制是否在设置 buffer 属性时通过等幂归一方法来缩放冲激响应音频数据。默认为 true，即等幂归一后卷积器的输出更为均匀。注意：如果修改了本属性的值，只有下次修改 buffer 属性时才生效。

接下来举一个完整的例子，效果图如下所示，首先在本地磁盘上准备两个音频文件，一个是普通的音乐文件，什么格式都可以，另一个是冲击响应音频。浏览效果见图 6.53。

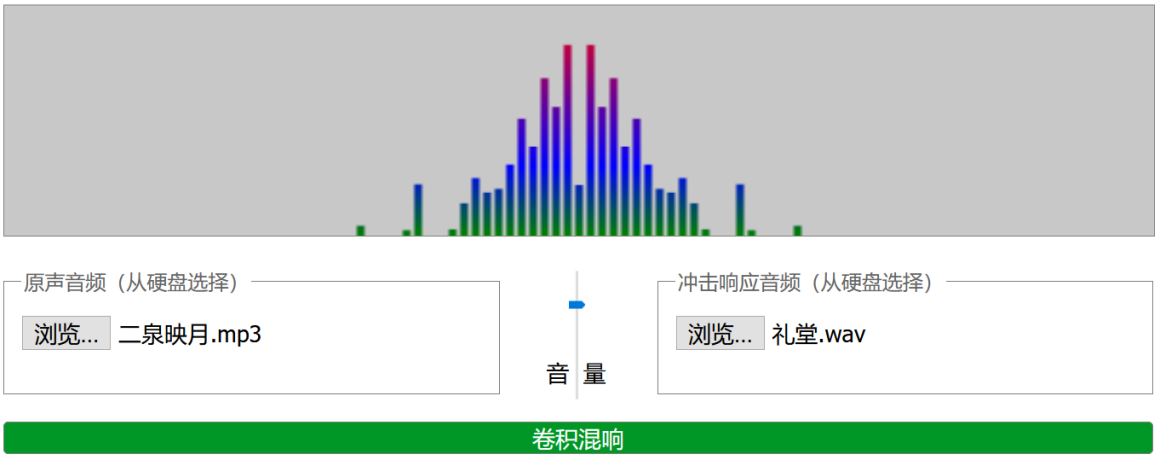
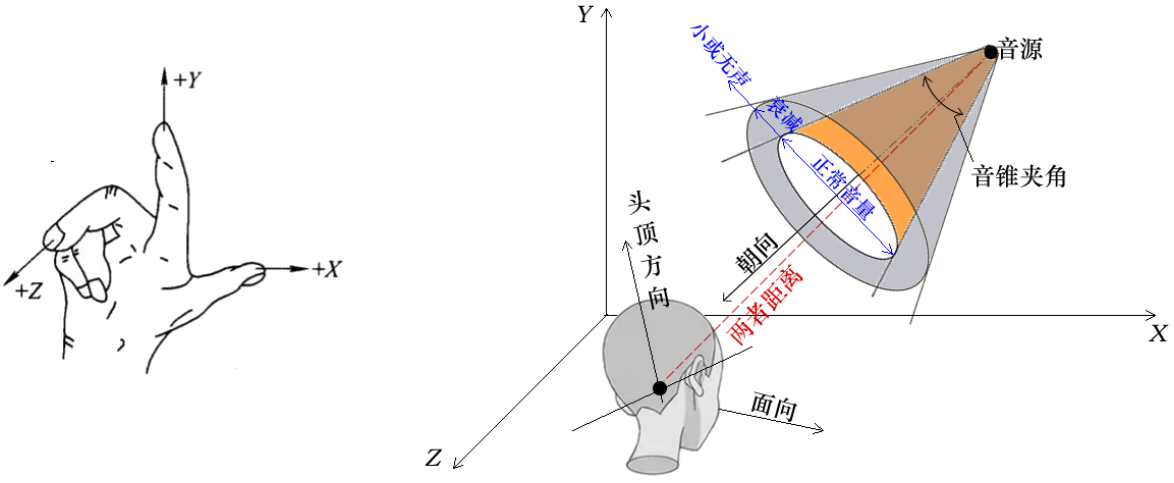


图 6.53 音频卷积处理举例

完整的代码请参考随书配套资源中的 chapter-6\convolver.html 文件。

6.10.10 3D 环绕

3D 环绕涉及听者的位置和声源的位置，而听者还涉及面向和双耳之间的连线与水平线的夹角两个参数，声源还涉及朝向和声锥角度两个参数。电脑的 3D 空间符合右手坐标系，具体参见示意图 6.54。



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

图 6.54 3D 环境中的听者和声源

听者的面向就是人的鼻尖方向，而头顶方向、面向和双耳连线三者之间互相垂直，所以我们通过确定头顶方向和面向来间接确定双耳方向。音锥的中心射线就是音源的朝向，音锥里面是正常的音量，音锥外面存在逐渐衰减的圆锥，再外面就是无声或者很小的声音了。Z 坐标是垂直于屏幕并指向人的眼睛。

使用 PannerNode 节点定义 3D 空间中的音源，包括位置、朝向和音锥夹角三个参数，使用 AudioContext 的属性 listener 定义 3D 空间中的听者，包括位置、面向和头顶方向三个参数。

PannerNode 节点有一个输入口和一个输出口，输入口允许单声道或双声道，而输出口只有双声道。采用 AudioContext.createPanner()实例化一个 PannerNode 对象，此对象的属性、方法介绍如下：

coneInnerAngle – 双精度类型，定义音锥夹角，单位为度，默认为 360 度（发出的声音没有方向），在音锥内声音不会衰变。如 panner.coneInnerAngle=30。

coneOuterAngle - 双精度类型，定义音量逐渐衰减圆锥的夹角，单位为度，默认为 0 度，在渐衰减圆锥的外面音量很小或无声（具体由 coneOuterGain 属性定义），从音锥到音量衰减圆锥之间，声音逐渐变小，即由正常音量过度到 coneOuterGain 属性定义的音量。如 panner.coneOuterAngle=46。

coneOuterGain – 双精度类型，定义衰减圆锥外面的音量大小，取值范围为[0,1]，默认为 0（即无声）。

maxDistance – 为双精度非负数类型，定义音源和听者之间的最大距离，当大于这个距离时，音量不再变化，默认值为 10000。

refDistance - 为双精度非负数类型，定义参考距离，当音源和听者之间大于这个距离时，音量开始衰减，默认值为 1。一般 maxDistance 要大于 refDistance。

distanceModel – 定义音源远离听者时音量衰减的算法，可取如下的值：

⑴"linear"：线性衰减，衰减公式为 1 - rolloffFactor * (distance - refDistance) / (maxDistance - refDistance)，其中 distance 为听者和音源的当前距离。

⑵ "inverse"（默认值）：音量衰减公式为 $refDistnce \div \left(refDistance + rolloffFctor \times \left(max(distance, refDistance) - refDistance\right)\right)$。

⑶"exponential"：音量指数衰减，衰减公式为 $\left(\frac{refDistance}{max(distnce, refDistance)}\right)^{rolloffFactor}$。

rolloffFactor – 当音源从参考距离（由 refDistance 属性定义）移动到最大距离（由 maxDistance 属性定义）时，定义衰减的音量，默认值为 1。取值范围与 distanceModel 属性有关：⑴当 distanceModel="linear"时，取值范围为[0,1]；⑵当 distanceModel="inverse"或"exponential"时，取值范围为[0,+∞]。

上面这个几个属性的关系参考图 6.55。

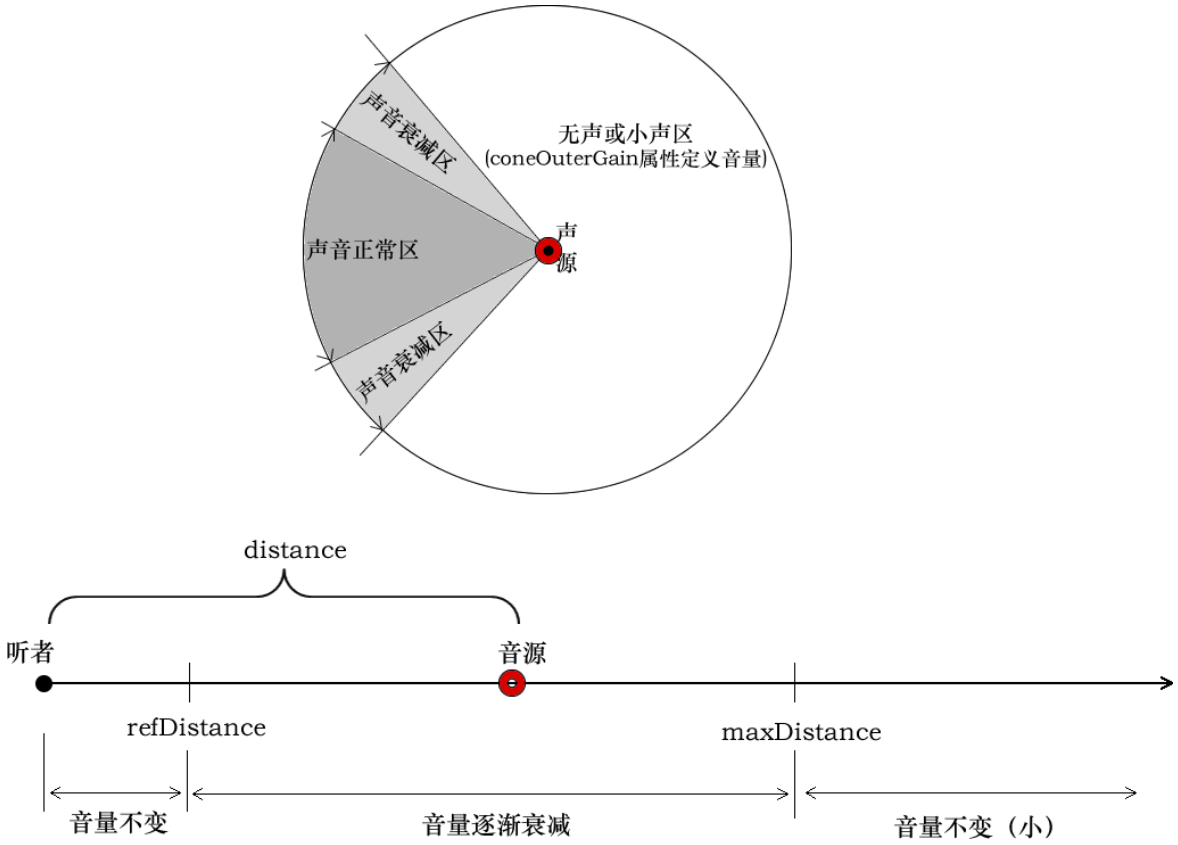


图 6.55 听者与音源之间的距离和角度

orientationX、orientationY、orientationZ – 均为 AudioParam 接口类型，定义音源朝向的向量，即从(0,0,0)指向(orientationX,orientationY,orientationZ)，默认值是(1,0,0)，即 X 轴方向。例如 orientationY.value=5 把 Y 坐标的值改为 5，此赋值语句等价于 orientationY.setValueAtTime(5, audioCtx.currentTime)。

positionX、positionY、positionZ – 均为 AudioParam 接口类型，定义音源的位置坐标，默认值为(0,0,0)，即就在坐标系的原点。

panningModel - 确定在 3D 空间中定位音频的算法，可取值"equalpower"（简单有效的等功率平移算法。这是默认值）、"HRTF"（卷积混响成立体声输出，空间感要优于 equalpower 算法，但是也更耗费计算资源）。原理图如图 6.56。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

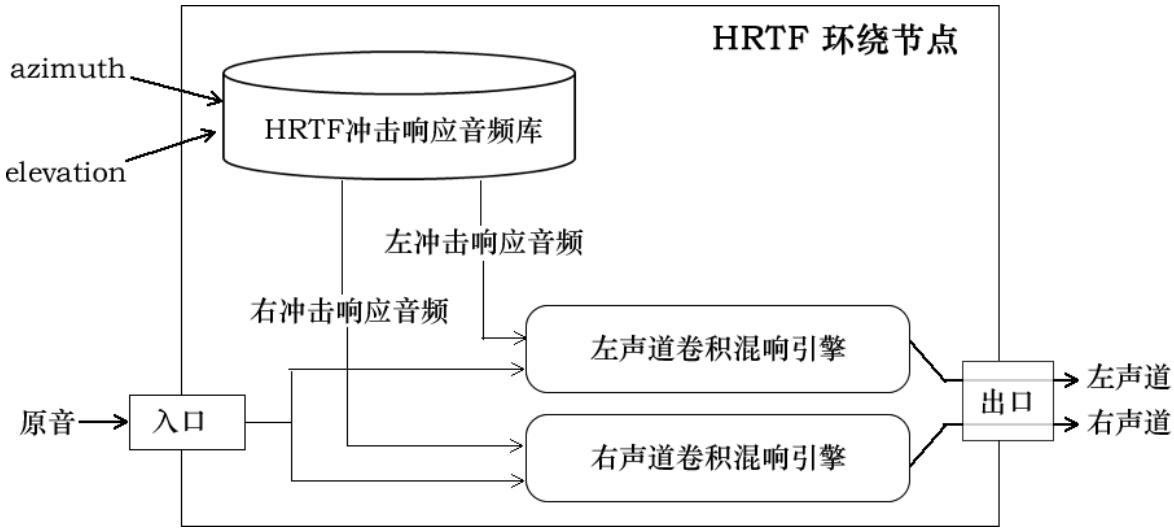


图 6.56 panningModel 属性说明

setPosition([x,y,z]) – 设置音源的位置坐标，即设置 positionX、positionY 和 positionZ 属性的值，如果省略参数，就是设置在坐标原点，即(0,0,0)。

setOrientation([x,y,z]) – 设置音源的朝向，即设置 orientationX、orientationY 和 orientationZ 属性的值，如果省略参数，就设置为(1,0,0)，即 X 轴的方向。

使用 AudioContext.listener 属性确定听者的有关信息，即位置、面向和头顶方向。这个属性为 AudioListener 接口对象，它有如下的属性：

positionX、positionY、positionZ – 均为 AudioParam 接口类型，定义听者的位置坐标，默认值为(0,0,0)，即就在坐标系的原点。

forwardX、forwardY、forwardZ – 均为 AudioParam 接口类型，定义听者面向的向量，即从(0,0,0)指向(forwardX,forwardY,forwardZ)，默认值是(0,0,-1)，即 Z 轴的反方向。

upX、upY、upZ – 均为 AudioParam 接口类型，定义听者头顶朝向，即从(0,0,0)指向(upX, upY, upZ)，默认值是(0,1,0)，即 Y 轴方向。

下面举一个例子，实现如图 6.57 所示的效果。



图 6.57 3D 环绕节点用法举例

网页代码参考 chapter-6 目录中的 panner.html、panner.css 和 panner.js 三个文件，如果有 5.1 系统，建议大家改改 Y 和 Z 轴坐标，体验下真正的 3D 环绕效果。

6.10.11 声音扭曲

采用 WaveShaperNode 节点可对音频震幅（音量）做非线性扭曲，设扭曲函数为 y=f(x)，即对于输入的一个采样点数据 x，输出为 y，只要 f 为非线性函数，那么就能实现非线性扭曲声音了。输入采样点数据一般都落在[-1,1]区间，且电脑没法表示连续函数，而只能采用函数曲线上的若干个坐标来近似模拟函数，如图 6.58 所示。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

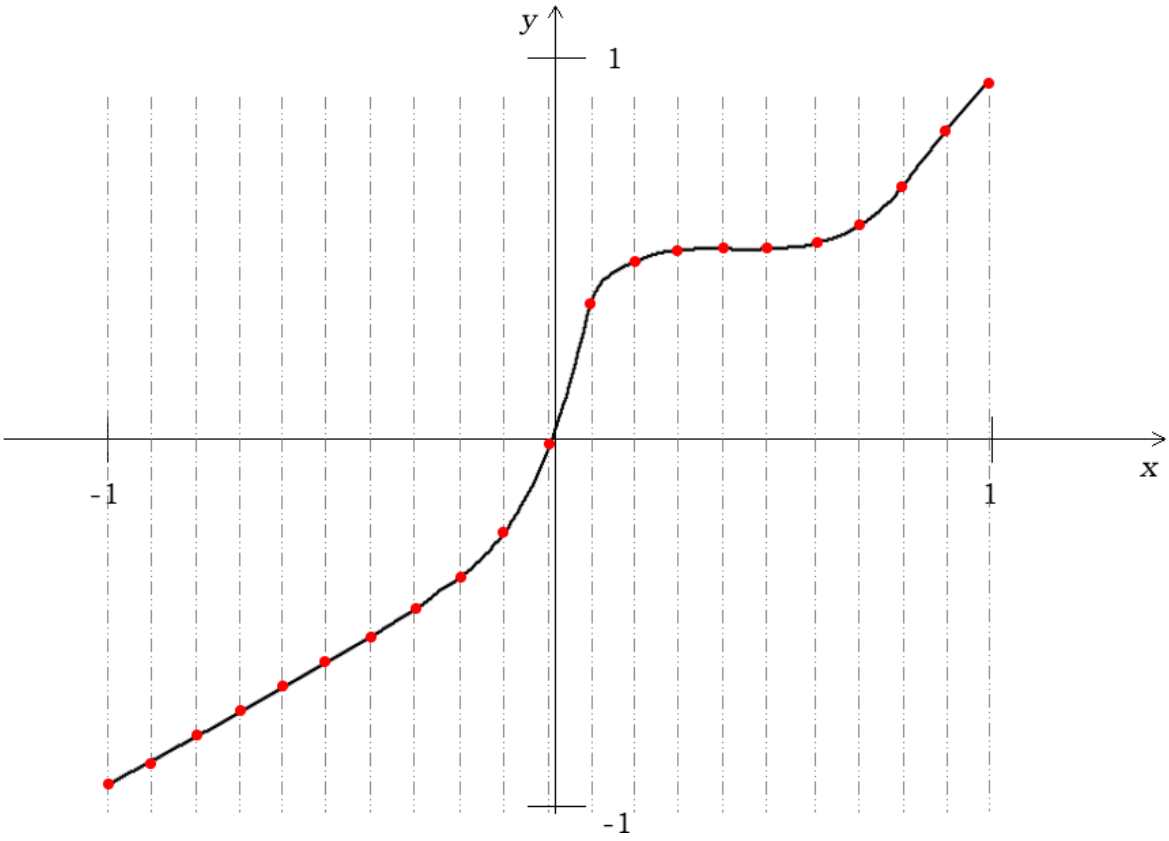


图 6.58 多点直线插值法模拟函数曲线

比如横坐标步进为 s，那么只要记录纵坐标 f(-1)、f(-1+s)、f(-1+2s)、f(-1+3s)、……、f(1)的值即可，当在电脑中恢复函数时，相邻两点之间采用直线补齐，即相邻两点坐标(-1+n*s, f(-1+n*s))、(-1+(n+1)*s, f(-1+(n+1)*s)之间画一根直线。显然步进 s 越小，记录的纵坐标数就越多，因而恢复的函数越逼近原始函数曲线。

采用 AudioContext.createWaveShaper()实例化一个声音扭曲处理节点 WaveShaperNode，此节点有两个属性：

curve – 为 Float32Array 类型的数组，存储了扭曲函数的若干个纵坐标值，假设数组元素个数为 N，那么横坐标的步进$s = \frac{1-(-1)}{N-1} = \frac{2}{N-1}$。对于输入 x，输出 y 的计算公式为：

$$y = \begin{cases} C[0] & ,v < 0 \\ C[N-1] & ,v \geq N-1 \\ (1-f) \times C[k] + f \times C[k+1] & ,\textit{其他情况} \end{cases}$$

其中$v = \frac{x+1}{\frac{2}{N-1}} = \frac{(N-1)}{2}(x+1)$，$k = \lfloor v \rfloor$，C 代表 curve 数组，$f = v - k$。由于 x 的取值范围为[-1,1]，所以 x+1 的取值范围就是[0,2]，k 就是输入 x 映射的数组下标。在实际编程时，一般根据一个具体的连续函数在 x∈[-1,1]区间内等分取 n 个点。

oversample – 枚举类型，定义超采样策略，超采样就是在输入采样的基础上增加更多的采样点，可取值'none'（不使用超采样，这是默认值）、'2x'（翻倍采样）、'4x'（4 倍采样）。倍数越大效果越好，但资源耗费也越大。

下面举一个实际的例子，音频处理有向图如图 6.59。

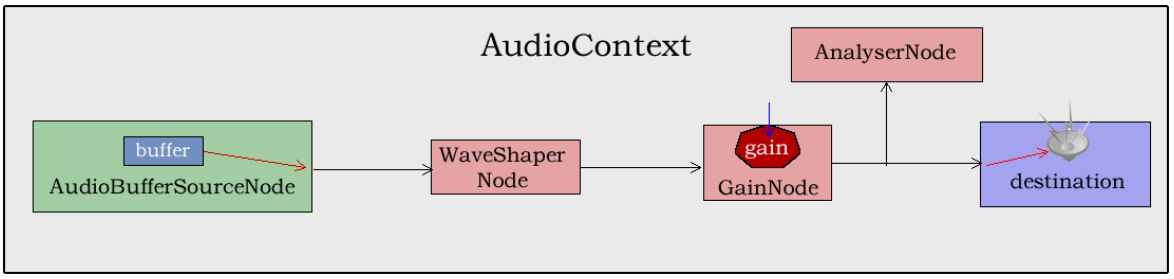


图 6.59 声音扭曲处理有向图

代码参见 chapter-6\waveshaper.html 文件，浏览的效果如图 6.60。

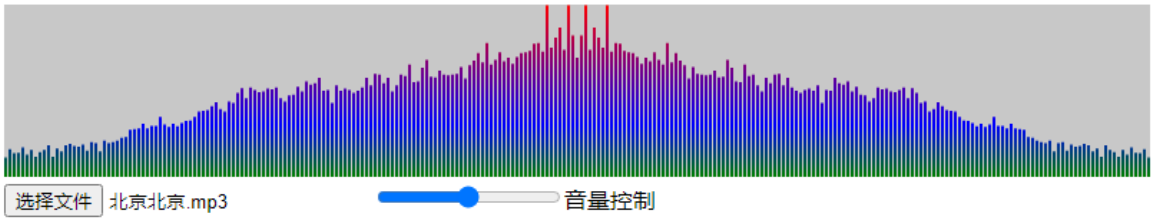


图 6.60 声音扭曲节点举例

本例子中我们采用反正切函数 atan(x)作扭曲函数，此函数在（-∞,+∞）区间中单调增，函数的值落在(-π/2，π/2)，代码中函数 generateWaveShapingCurve(amount)内的 x 取值范围为[-1,1]，所以 5x 的取值范围为[-5,5]，因此 atan(5*x)的曲线如图 6.61。

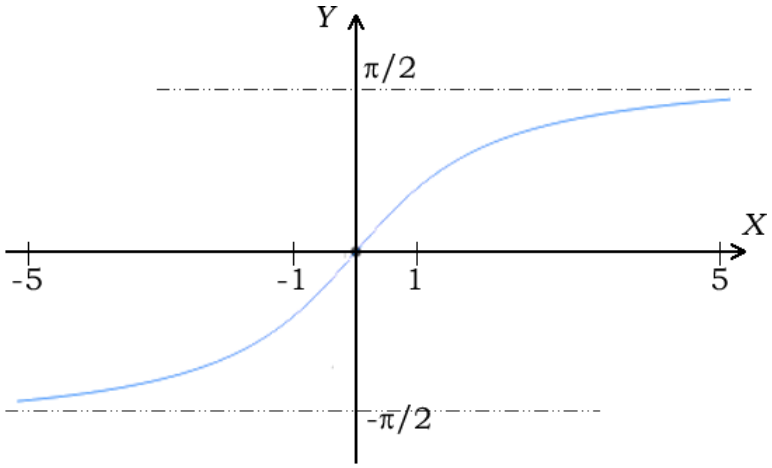


图 6.61 atan(5*x)曲线

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.11 WebRTC

WebRTC 为 Web Real-Time Communication 的缩写，意为 Web 实时通讯，是一种直接在浏览器之间建立点对点通讯的链接技术，从而采用 HTML5 页面也能实现如下应用场景：音频电话、视频会议、视频电话、远程演示、资料传输、实时广播等，瘟疫期间大家热衷于上网课、开在线视频会议，都是拜 WebRTC 技术所赐。通过引入一些中间设备，也能与传统语音终端（如 PSNT 电话机）建立连接。现代主流浏览器都原生支持 WebRTC，而不用安装额外的插件。要搞懂 WebRTC 技术，你必须先弄明白 NAT 网络和 NAT 穿透技术。

6.11.1 联网方式

6.11.1.1 NAT 网络

在实际生活当中，运行浏览器的那个设备，其接入互联网的方式从物理层面看可归纳为两类：(1)、直接连到互联网（设备拥有公网 ip 地址），(2)、通过 NAT 设备（如 wifi 路由器）接入互联网。参见示意图 6.63。



图 6.62 NAT 组网示意图

由于公网 ip 地址的稀缺性，很少有设备能直接配置公网 ip 接入互联网（很怀念在厦门大学读书的日子，那时实验室的每台电脑都是公网 ip），绝大多数的设备是经过 NAT 路由器接入互联网的，此时设备的网卡配置私有 ip，在路由器上做 NAT 转换，因此局域网内的众多设备能共享一个公网 ip 上网，这个共享的公网 ip 配置在 NAT 路由器的广域网口上（家里的 wifi 路由器的公网 ip 由电信部门随机分配，所以每次启动 wifi 设备，分配的公网 ip 可能是不同的）。不管是哪种接入方式，局域网内的设备一定能对外发起连接请求（比如浏览互联网上的网页），但是对于第二种方式，外面就不一定能访问局域网内的设备。

逻辑上 NAT 接入方式又划分为四种类型，即全锥型 NAT、地址受限锥型 NAT、地址端口受限锥型 NAT 和对称型 NAT，前三种统称为锥型 NAT 或非对称型 NAT。从局域网内部发送一个数据包到互联网上的某台设备，数据包中的源地址、源端口、目的地址、目的端口和协议组成一个五元组：（协议，源地址，源端口，目的地址，目的端口），路由器里 NAT 表（见表 6.9 所示）中的每条记录也对应一个五元组：（协议，内部地址，内部端口，目的地址，目的端口）。

当局域网内发出的数据包到达路由器时，这时 (1)如果是对称型 NAT，则比较包的五元组和 NAT 表中每条记录的五元组，如果找不到就增加一条 NAT 表记录，新纪录的“外部地址”字段填写路由器的互联网 ip，“外部端口”字段从端口池（1025~65535）中选择一个；(2)如果是锥型 NAT，则只比较五元组的前三个元素，即不管目的地址和目的端口，如果没找到就添加一条新记录。

表 6.9 NAT 表

协议	外部地址	外部端口	内部地址	内部端口	目的地址	目的端口
udp	102.232.16.188	4444	192.168.0.10	3333	222.16.98.20	5555

自动添加的 NAT 表记录是有生命期的，一般为 4 至 5 分钟，如果生命期内没有任何数据包通过，对应的表记录就会删除，相应的外部端口被放回端口池中。手工添加的表记录称为静态记录，静态记录没有生命期的限制，除非手动删除。至于对称 NAT，每一个不同的“目的地址:目的端口”都对应一条 NAT 表记录，也即“外部地址:外部端口”与“目的地址:目的端口”形成一一对应关系，构成对称性。而锥型 NAT 允许一个“外部地址:外部端口”对应多个“目的地址:目的端口”，构成圆锥型，见图 6.63 所示。

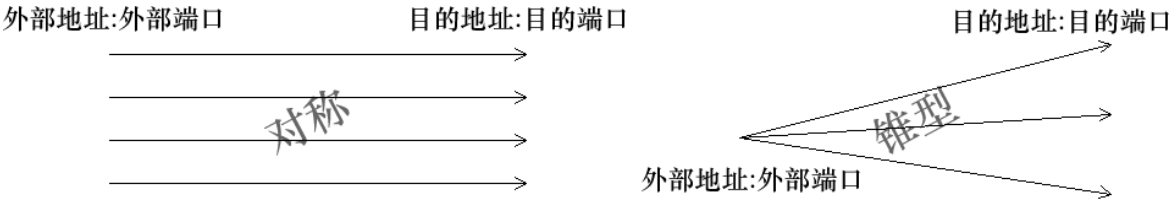


图 6.63 对称型和锥型 NAT

锥型 NAT 又分为三种类型，如果 (1)互联网上的任何设备都能访问某条 NAT 表记录中的“外部地址:外部端口”进而到达局域网内的某台设备，这就是全锥型 NAT；(2)只允许 NAT 表记录中的“目的地址:任意端口”访问本记录中的“外部地址:外部端口”，这就是地址受限锥型 NAT；(3)只允许 NAT 表记录中的“目的地址:目的端口”访问本记录中的“外部地址:外部端口”，这就是地址端口受限锥型 NAT。

从外面到达的数据包中抽出一个五元组：（协议，目的地址，目的端口，源地址，源端口），再从 NAT 表记录中抽出一个五元组：（协议，外部地址，外部端口，目的地址，目的端口），那么全锥型 NAT 只按次序比较两个五元组的前三个元素，地址受限锥型 NAT 按次序只比较前四个元素，而地址端口受限锥型 NAT 就要比较两个五元组的全部五个元素了。

另外只有由内到外的数据包经过路由器时才有可能添加动态 NAT 表记录，而从外至内的数据包到达路由器时只会按照相应的策略查询 NAT 表，查询命中就放行，否则就丢弃。动态 NAT 表中的记录是有生命期的，在生命期内没有被检索命中过的记录会被自动删除。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

由内到外的数据包经过路由器时会替换其中的源地址和源端口，从外面的来的数据要么被丢弃，要么被放行，被放行的数据包会替换其中的目的地址和目的端口，整个过程参考图 6.64。

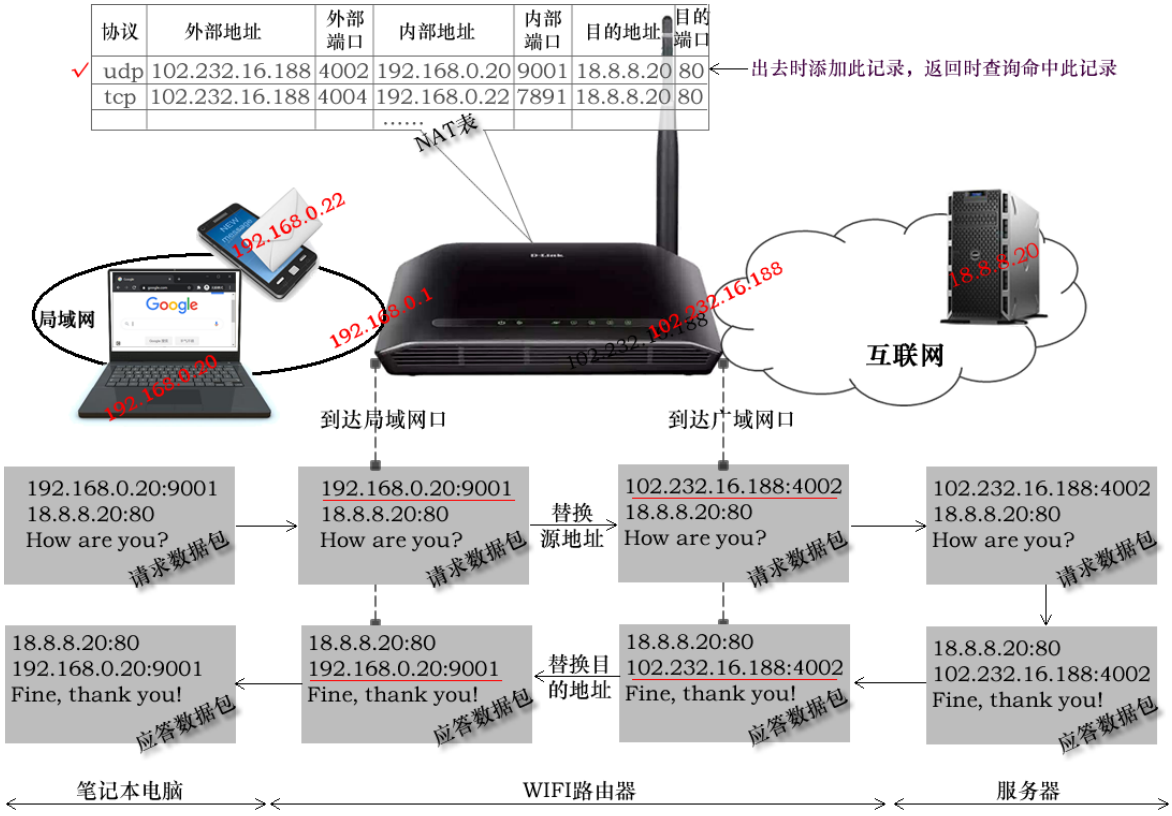


图 6.64 数据包经过 NAT 路由器的过程

局域网内的笔记本电脑首先向互联网上的服务器发起请求，请求数据包到达 wifi 路由器后，路由器查询 NAT 表（如果没有找到，就临时添加一条 NAT 记录），并替换数据包中的源 ip 地址和源端口为路由器的广域网口 ip 和端口，然后发到广域网上，服务器收到请求数据包后，构造应答数据包，目的 ip 地址填上 wifi 路由器的广域网口 ip，路由器收到应答数据包后，替换目的 ip 地址为局域网内的笔记本的 ip，然后发送到局域网上，这样最终笔记本就会收到应答数据包了。

6.11.1.2 NAT 穿透

点到点连接不需要中央转发服务器，使得流量分散，这在基于互联网的视频/语言通讯领域应用非常广泛。由于绝大多数设备都在 NAT 网关的后面，所以点到点连接必须穿透对方的 NAT 网关，即 NAT 穿透。那么位于不同局域网内的设备到底如何建立点到点连接呢？下图是最一般的网络拓扑图，即局域网内的设备都通过 NAT 网关（路由器）接入因特网，这些设备平时都能与因特网上的设备通讯，比如上网、看在线视频等，因为网站服务器和视频服务器都在因特网上。现在要求图 6.65 中的笔记本能与另一个局域网内的台式机直接通讯，即所谓的点到点通讯（或称 p2p 通讯），那怎么办呢？

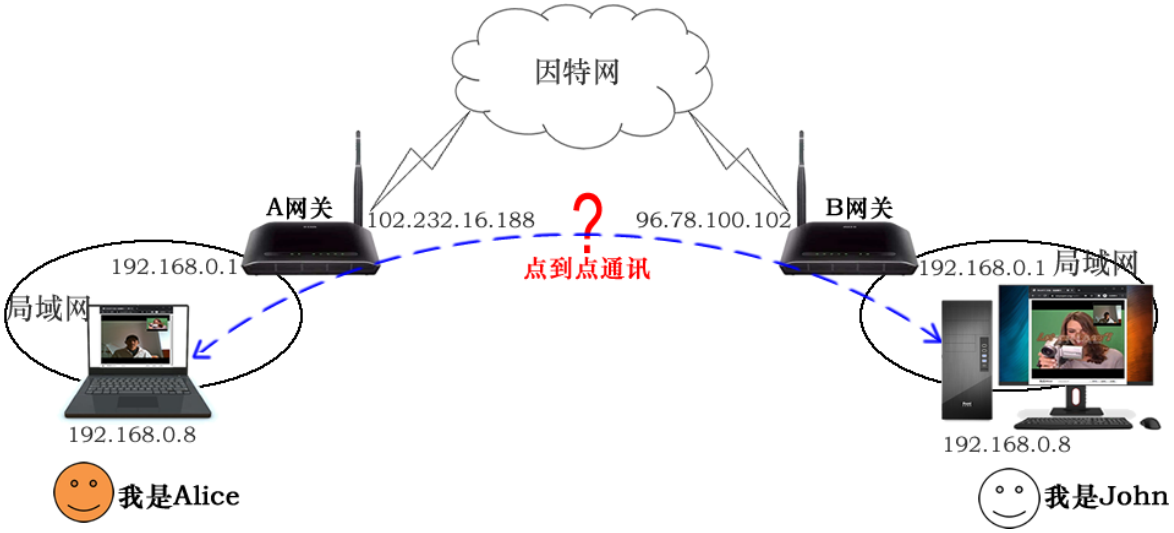


图 6.65 显示生活当中最普遍的网络拓扑

由于两台设备的 ip 地址都是 192.168.0.8，因此任何一方不能直接访问对方的 ip（操作系统会认为就是访问自己）。那笔记本访问对方网关（即 B 网关）的外部地址 96.78.100.102 的某个端口（比如 2000），行吗？参见图 6.66 所示。

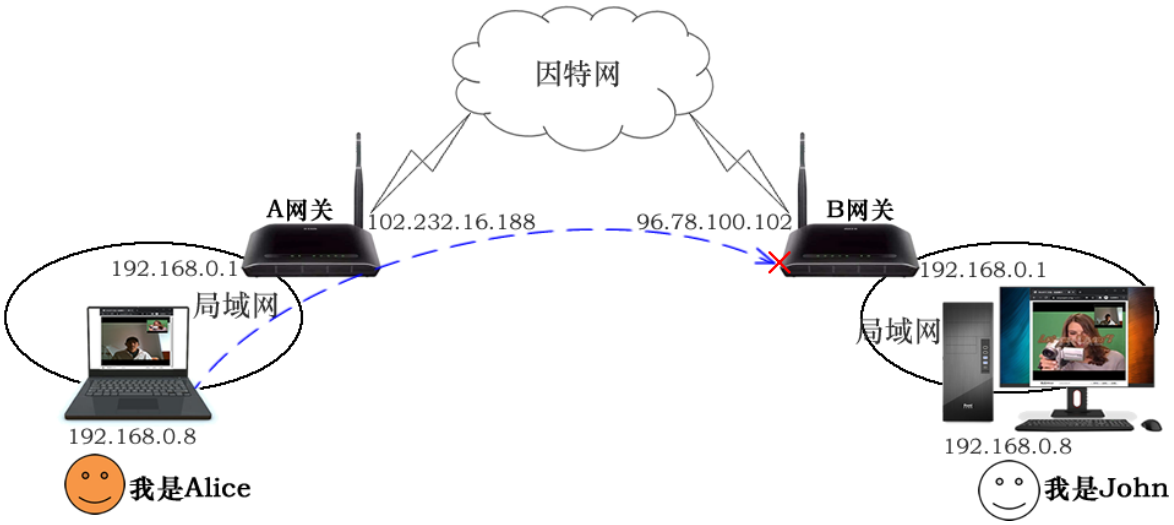


图 6.66 访问对方网关

当数据包到达 B 网关时，数据包中的源地址:源端口为 102.232.16.188:5001（不妨假设数据包离开 A 网关时源端口被替换为 5001），目的地址:目的端口为 96.78.100.102:2000，这时 B 网关会在 NAT 表中按如下条件查询记录：外部地址=96.78.100.102 and 外部端口=2000 and 协议=<数据包中的协议>。如果查询到一条记录且记录中的内部地址刚好就是 192.168.0.8，这时再满足如下条件之一就成功：(1)记录是静态的；(2)B 网关为全锥型。其他情况 B 网关都会丢弃数据包，从而导致笔记本连接台式机失败!在现实生活中,那笔记本是怎么知道 B 网关的外部地址和端口的？Alice 打电话问 John，John 也不知道。你知道你家 wifi 路由器的外部地址吗？不知道吧！

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

唯一的解决办法是借助因特网上的一台公共设备——信令服务器，首先笔记本和台式机都访问这台信令服务器，把自己的姓名、网关外部地址和外部端口等信息登记在信令服务器的 TRACE 表中，各自的网关会自动在 NAT 表中增加一条动态记录。等等，Alice 和 John 是怎么知道自己的网关外部地址和外部端口呢？这要询问 STUN 服务器了——STUN 的功能之一就是反馈客户端的网关 ip 地址和端口号。参见图 6.67 所示。

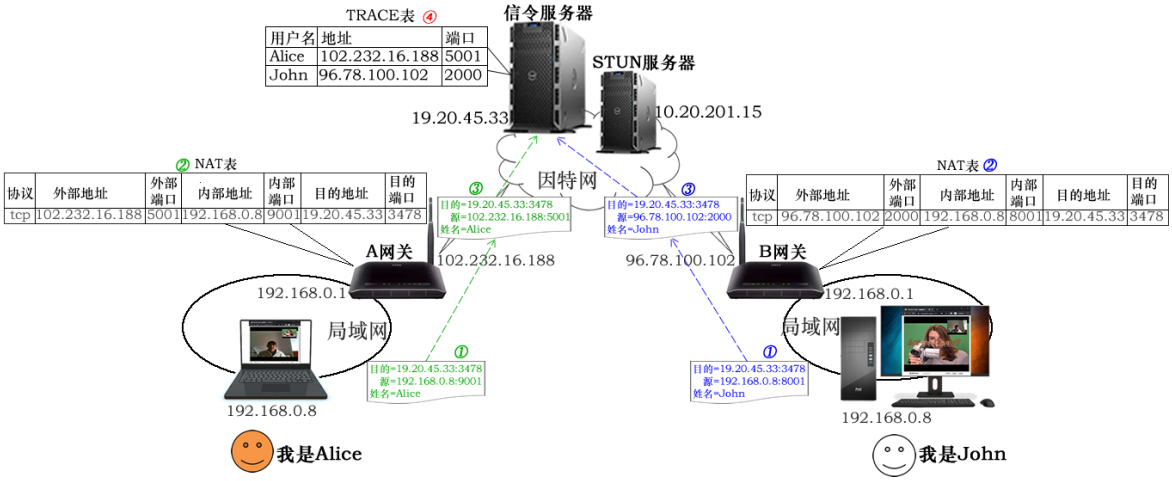


图 6.67 需要信令和 STUN 服务器参与

接下来 Alice 与 John 建立点到点连接的过程如下：

1、Alice（笔记本电脑）向信令服务器询问 John（台式机）的地址和端口。

2、信令服务器向 Alice 返回 John 的地址和端口，即 96.78.100.102:2000。

3、Alice 向 96.78.100.102:2000 发起连接请求。

5、B 网关收到连接请求后查询 NAT 表，成功命中一条记录，然后修改数据包中的目的地址和端口为 192.168.0.8:8001，最后把数据包发给台式机。

6、John 给 Alice 的应答包沿路返回。

穿透成功！！

如果 B 网关不是全锥型 NAT 路由器，穿透过程会更复杂一些，但是还是存在一些特殊情况是没法直接建立 p2p 连接的（见表 6.10），这时需要通过转发服务器才行。

表 6.10 双方网关类型决定是否能穿透

A 网关类型	B 网关类型	能否穿透
全锥型	全锥型	√
全锥型	地址受限锥型	√
全锥型	地址端口受限锥型	√
全锥型	对称型	√
地址受限锥型	地址受限锥型	√
地址受限锥型	地址端口受限锥型	√
地址受限锥型	对称型	√
地址端口受限锥型	地址端口受限锥型	√

地址端口受限锥型	对称型	╳
对称型	对称型	╳

从表 6.10 可知，只有当一方网关为对称型，另一方为对称型或地址端口受限型时，是不可能直接建立点到点连接的，不过这种情况在实际的生活当中占比不到 20%。针对不能建立点到点连接的问题，解决的唯一办法是引入因特网上的转发服务器——TURN 服务器，参见图 6.68 所示。

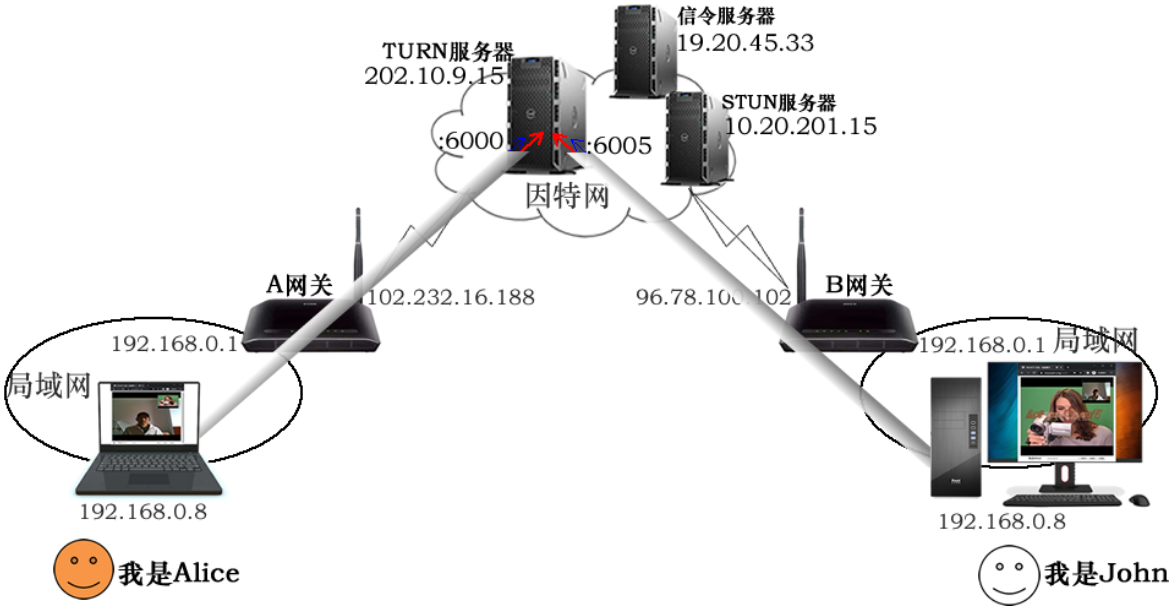


图 6.68 引入转发服务器 TURN

即 Alice 先与因特网上的 TURN 服务器建立长连接（一般为 tcp 连接），这相当于挖了一条通道，TURN 服务程序会为每一条“通道”额外监听一个端口（假设为 6000），以后只要 202.10.9.15:6000 收到的任何数据包都会转发给 Alice，Alice 的应答数据包沿路返回。然后 Alice 把 202.10.9.15:6000 登记到信令服务器中。如果 John 要联系 Alice，他直接访问 202.10.9.15:6000 即可，其中 202.10.9.15 为 TURN 服务器的 ip 地址。

实操实验

1）目的

从家里访问公司电脑的 windows10 桌面。条件：一台安装了 ubuntu 18.04 的阿里云主机（假设云主机的 ip 地址为 202.10.9.15）。

2）操作

云主机：打开/etc/ssh/sshd_config 文件中的 GatewayPorts = yes 配置项，然后执行命令 systemctl reload ssh 使得新配置生效。让防火墙放行 tcp 3389 端口。创建用户 moodisk，并设置密码 A1b2C3。

办公室 windows 10 电脑：打开 PowerShell 窗口，执行命令与云主机建立“通道”：ssh -p 22 -qngfNTR "[::]:3389:localhost:3389" moodisk@202.10.9.15。开启远程桌面连接（右击此电脑》属性》远程设置》远程》允许远程连接到此计算机）。让防火墙放行 tcp 3389 端口。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

3）验证

从其他任何一台 windows 电脑上，按 Win+r，输入 mstsc 并回车启动远程桌面客户端，见图 6.69。

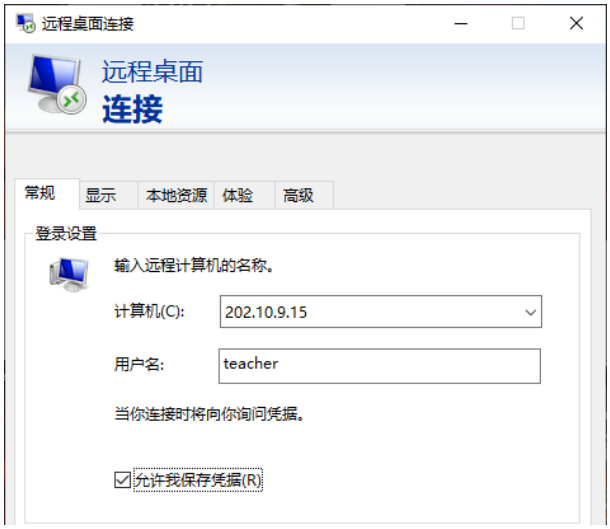


图 6.69 windows 远程桌面客户端

输入云主机的 ip 地址 202.10.9.15 和办公室 Windows 10 电脑上的用户 teacher，点击“连接”按钮后再输入 teacher 用户的密码即可登陆了。

6.11.1.3 ICE 候选者

WebRTC 技术引入了 ICE 候选者的概念，ICE 为 Interactive Connectivity Establishment 三个单词的首字母缩写，译为互联建立。ICE 候选者就是我方提供的若干个建立互联的方式，供对方选择我方最优的一个。主要的 ICE 候选者有三个：主机候选者、反射候选者和中继候选者，但并不表示我方一定同时拥有这三个候选者。

如果我方直接接入互联网，即设备上直接配有公网 ip 地址，那么对方可以直接访问我方，这样的互联方式就称为主机候选者，主机候选者是最优的候选者。但在实际生活当中，这种情况很少见，参考图 6.70 所示。

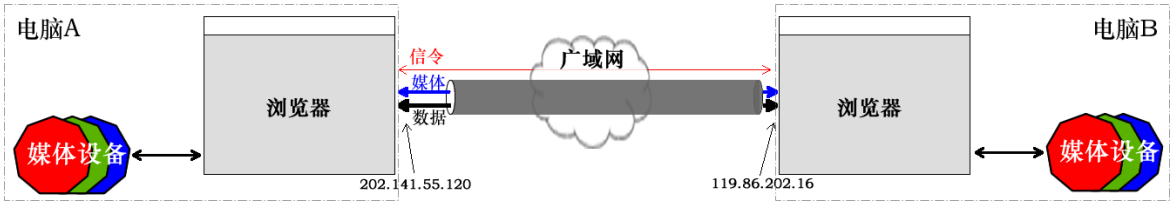


图 6.70 主机候选者

如果我方通过 NAT 网关接入因特网，这时需要引入因特网上的一台 STUN 服务器。STUN 服务器的功能之一是响应客户端的 stun 请求，返回客户端接入因特网的 ip 地址和端口号（比如最外层 NAT 路由器因特网口的 ip）。这种由 STUN 服务器“反射”回来的“ip 地址:端口”就称为反射候选者，对方需要访问我的反射候选者才可能建立 p2p 连接。参见图 6.71 所示。

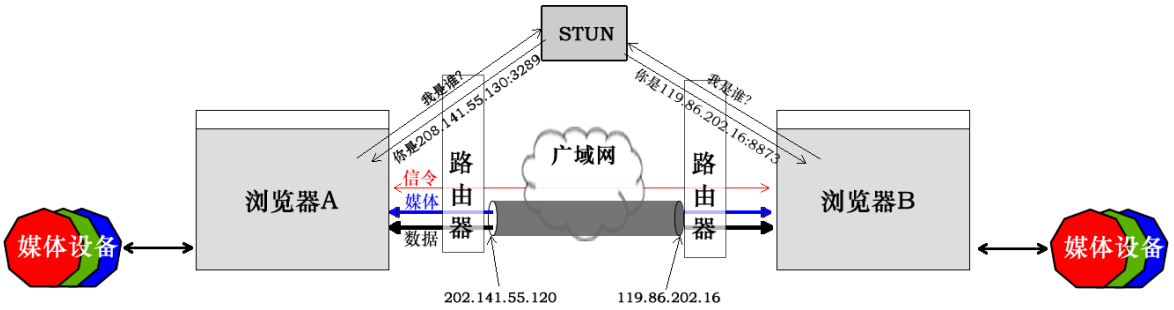


图 6.71 反射候选者

主机候选者和反射候选者都无效的情况下，就要使用我方的中继候选者了。中继候选者的“ip 地址:端口”暴露在 TURN 服务器上，我方和 TURN 服务器之间建立了长连接（具体介绍参见前面的小节），对方只能通过 TURN 服务器的 ip 地址和我方在 TURN 服务器上“登记”的端口间接和我建立连接，参见示意图 6.72。

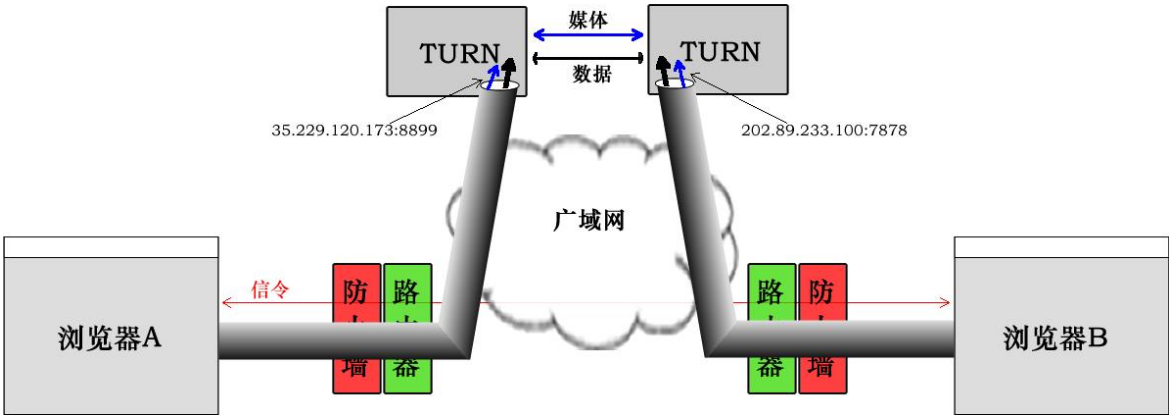


图 6.72 中继候选者

在实际生活当中，建立链接需要由双方（称为 ICE 代理，比如浏览器）协商确定，协商的过程大致是：双方各自提交自己支持的全部候选者，然后进行“打洞”匹配以确定双方最优的 ICE 候选者。交换候选者通过信令完成，信令只规定消息内容的格式，至于双方如何传递信令消息，WebRTC 并没有规定，在具体的实践当中可以采用邮件方式、XHR、SSE 或 WebSocket 等方式，更多人采用 socket.io 库开发信令系统，因为它包含（聊天）房间的概念。

我们只要使用 WebRTC 的几个接口对象，就能轻松实现点到点通讯，技术细节全部隐藏在接口对象中。如果要用到 STUN 和 TURN 服务器，可以自己搭建、或者使用免费的、或者使用商用的都行。下面是一些免费的 STUN/TURN 服务器：stun.l.google.com:19302、stun1.l.google.com:19302、stun2.l.google.com:19302、stun3.l.google.com:19302、stun4.l.google.com:19302、stun01.sipphone.com、stun.ekiga.net、stun.fwdnet.net、stun.ideasip.com、stun.iptel.org、stun.rixtelecom.se、stun.schlund.de、stunserver.org、stun.softjoys.com、stun.voiparound.com、stun.voipbuster.com、stun.voipstunt.com、stun.voxgratia.org 、 stun.xten.com ， 不 过 在 使 用 前 最 好 在 https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/网站上测试是否可用。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.11.1.4 信令过程

图 6.73 是 WebRTC 最一般的网络拓扑图。

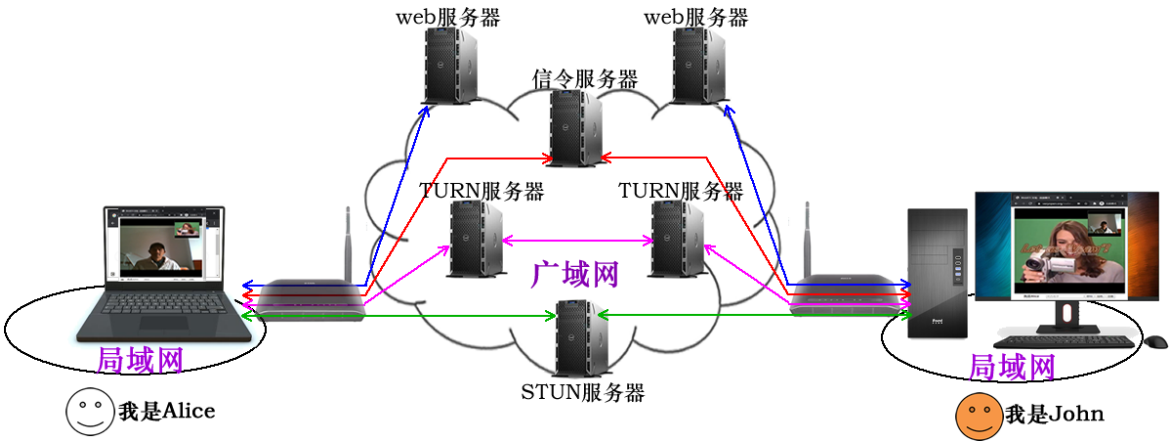


图 6.73 WebRTC 网络拓扑图

建立点到点通讯的大致过程如下：

1、Alice 和 John 各自浏览网页。浏览器自动下载页面前端文件（*.html/*.js/*.css），然后执行*.js。

2、Alice 和 John 各自在信令服务器上注册，信令服务器记录他们的名字。

3、Alice 和 John 各自实例化一个 RTCPeerConnection 对象。

4、Alice 创建一个呼叫请求（即 offer），然后把 offer 发给信令服务器。

5、信令服务器把 Alice 的 offer 发给 John。

6、John 收到 Alice 的 offer 后创建一个应答（即 answer），然后把 answer 发给信令服务器。

7、信令服务器把 John 的 answer 发给 Alice。

8、John 把浏览器收集到的新 ICE 候选者发给信令服务器，信令服务器然后转发给 Alice。Alice 把浏览器收集到的新 ICE 候选者发给信令服务器，信令服务器然后转发给 John。注：浏览器收集候选者需要用到 STUN 和 TURN 服务器。

下面两步不需要使用信令服务器。

9、双方浏览器对收集到的候选者进行打洞配对（即 P2P 穿透），从而建立最佳的点到点连接。具体使用什么候选者（主机候选者、反射候选者或者中继候选者）由浏览器自行决定，Alice 和 John 本人并不知道。

10、双方把媒体流（摄像头、话筒或者截屏等）附加到点到点链接上，于是视频聊天开始了。

11、John 向信令服务器发送“挂机”消息，信令服务器转发给 Alice。

12、双方关闭点到点连接。

图 6.74 是建立点到点连接的详细的握手过程。

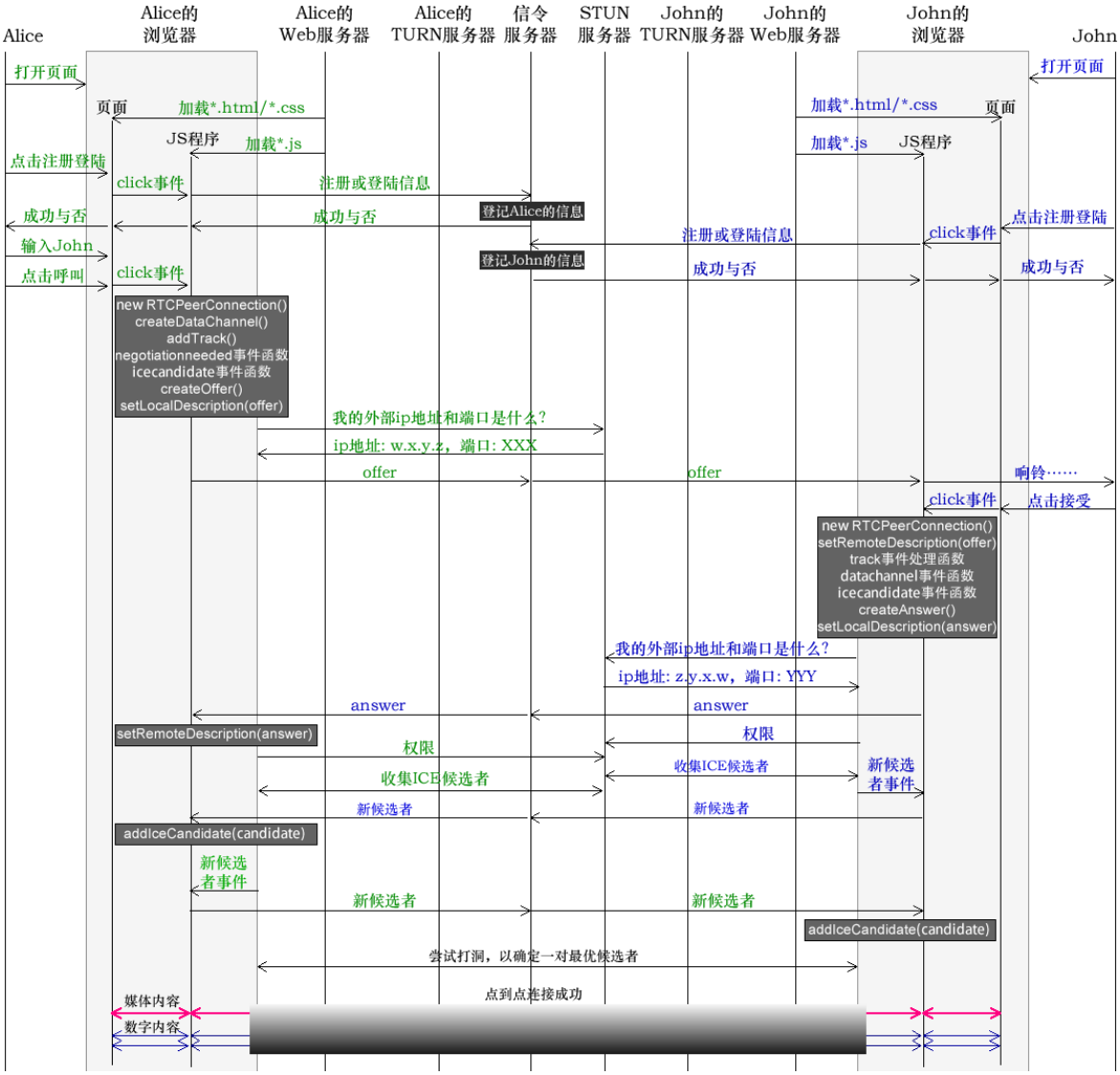


图 6.74 建立点到点连接的握手过程

编程时我们只要关注“页面”和“JS 程序”两部分，上图中，凡是箭头的起点和终点不是落在页面或 JS 程序的部分，都是由浏览器自动完成，不用我们编写代码。另外上图中并没有看到任何箭头的起/终点落在 TURN 转发服务器上，但是浏览器“打洞”匹配最优候选者对时，最终建立的 p2p 连接可能不得不采用了 TURN 转发服务器。在实际的应用场景中，如果没有 TURN 转发服务器，会有不到 20%的概率不能建立 p2p 连接。

信令服务器只是转发双方的信令消息而已，基于 web 的实现信令服务器的最佳技术就是 HTML5 的 WebSocket 了。随书配套资源 chapter-6\signalserver.js 是基于 WebSocket 的库 ws 实现的一个简单的信令服务器：能接受用户注册、转发 offer 和 answer、转发候选者等。运行该信令服务器，

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

需要安装 node.js，再采用命令 npm install -g ws 安装 ws 库，最后采用命令 node signalserver.js 启动信令服务器。信令服务器最好部署在云主机上，而且要让 9090 端口穿过防火墙。

注意：如果网站页面采用 https 协议，那么 WebSocket 也必须采用 wss 安全协议，这时需要用下面的代码代替 signalserver.js 中的开头两行：

```
const fs = require('fs');
const https = require('https');
const WebSocketServer = require('ws').Server;
const options = {
    key: fs.readFileSync(process.cwd() + '/server.key'),
    cert: fs.readFileSync(process.cwd() + '/server.crt')
};
const server = https.createServer(options, (req, res) => {
    res.writeHead(200);
    res.end("这是加密 WebSocket 服务！");
}).listen(9090, '0.0.0.0');

var wss = new WebSocketServer({ server: server });
```

可采用如下的命令产生合法的主机证书：

```
letsencrypt certonly --standalone -d stun.veryopen.org
```

其中 stun.veryopen.org 是作者申请的域名，大家在操作时要用自己的的域名替换它。

6.11.2 搭建实验环境

很多人没有公网 ip 或者云主机，为了不影响本章的学习，我们可以采用虚拟机软件 VMWare 搭建“图 6.73 WebRTC 网络拓扑图”所示的实验环境。我们采用两台物理机联网在一起，均安装 VMWare 虚拟机软件。不失一般性，我们只在一台物理机上创建一台虚拟机担任各种服务器角色，此虚拟机的网卡采用桥接模式，安装 ubuntu 18.04 Linux 操作系统，再在两台物理机上各创建一台虚拟机，网卡采用 NAT 模式，虚拟机里面安装 windows 10 操作系统。这样物理机所在的网络就相当于因特网了，因此那台安装了 ubuntu 18.04 操作系统的虚拟机就是在“因特网”上了。我们做如下设定：“因特网”的网段为 20.15.18.0/24，NAT 内部的局域网都是 192.168.10.0/24，网关为 192.168.10.2，两台物理机的 ip 分别为 20.15.18.8 和 20.15.18.9，ubuntu 18.04 虚拟机的 ip 地址为 20.15.18.10，两台 windows 10 虚拟机的 ip 地址均为 192.168.10.20，具体参考示意图 6.75。

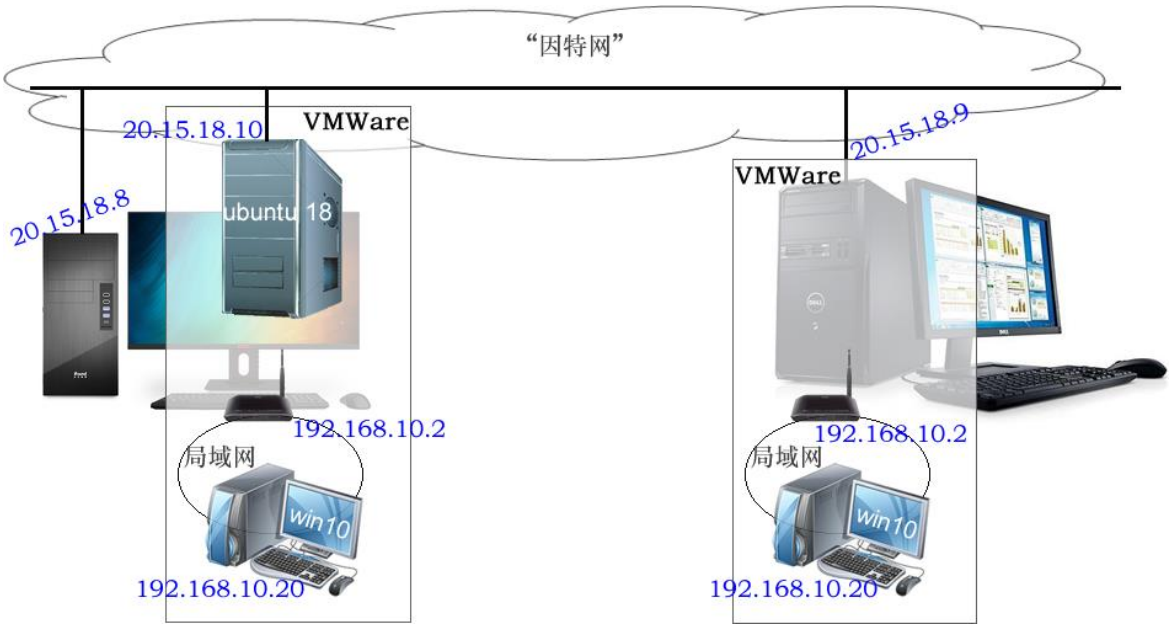


图 6.75 搭建实验环境

配置 NAT 局域网：启动 VMWare，然后点击菜单“编辑》虚拟网络编辑器”，再点击“更改设置”按钮，然后参考图 6.76 操作：

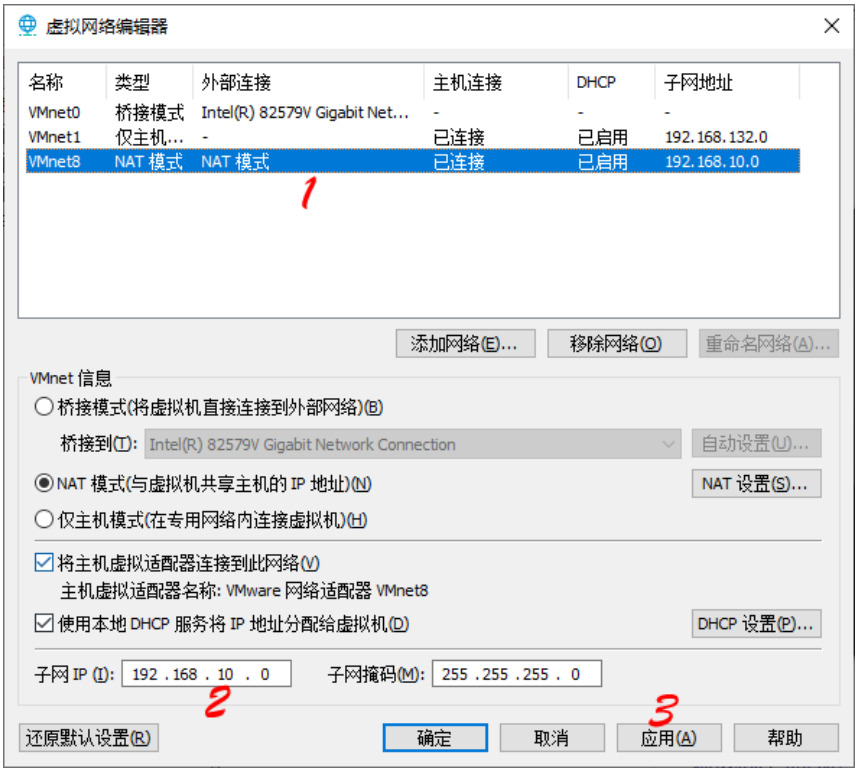


图 6.76 配置 NAT 局域网

如果要查看网关，请点击“NAT 设置”按钮。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.11.3 搭建 STUN/TURN

采用免费的 STUN/TURN 服务器平时学习测试没问题，在生产环境中由于 TURN 服务器要承载大量的转发流量，免费的显然力不从心，谁愿意免费为你提供大流量带宽呢。所以需要自行搭建 STUN、TURN 以及 WEB 服务器，同时还需要开发信令服务程序（一般采用基于 WebSocket 的 socket.io 库）。下面采用目前流行的开源 coturn 软件自行搭建 STUN 和 TURN 服务器。

1、前提条件

（1）一台配有公网 ip 的机器，里面安装 Linux 操作系统。比如购买一台硬件服务器，租用公网 ip，就放在公司内部，或者租用云主机，国内阿里云主机、国际亚马逊、谷歌云或者微软的 Azure 云主机都是不错的选择。笔者租用一台谷歌云主机，里面安装 ubuntu server 18.04。

注：或者用我们上一小节安装了 ubuntu 18.04 的虚拟机充当。

（2）申请一个域名，域名 DNS 解析到第（1）步准备的那台机器的公网 ip 地址。笔者申请的域名为 veryopen.org，二级域名 www.veryopen.org 访问我的个人网站，stun.veryopen.org 访问我的 STUN/TURN 服务，signal.veryopen.org 访问信令服务。

注：或者自己随便用一个域名，比如 abc.com，并把“20.15.18.10 abc.com”加到两台 windows 10 的 c:\windows\system32\drivers\etc\hosts 和 ubuntu 的/etc/hosts 文件中。

2、搭建过程

登陆到 linux，然后采用命令 sudo -s 切换到超级用户命令行。

（1）安装 coturn:

```
root@instance-1:~# apt  install  -y  coturn
```

（2）激活 coturn

```
root@instance-1:~# echo  "TURNSERVER_ENABLED=1">>/etc/default/coturn
```

（3）配置 coturn

```
root@instance-1:~# vim  /etc/turnserver.conf
```

确保/etc/turnserver.conf 文件中存在表 6.11 第一列的配置项

表 6.11 turnserver.conf 的配置项

配置项	说明
listening-ip=10.128.0.2	监听一块网卡的 ip 地址，ip 要根据实际情况填写

external-ip=35.238.210.160/10.128.0.2	35.238.210.160 是云主机的公网 ip（即 IaaS 的浮动 ip），要根据具体情况填写。如果是实验环境，省略 external-ip
min-port=10000	定义转发端口池中的最小 udp 端口号。默认是 49152。
max-port=20000	定义转发端口池中的最大 udp 端口号，中继候选者的端口从[10000,20000]中随机挑选。本参数默认值为 65535。
lt-cred-mech	启用用户身份认证。用户采用命令 turnadmin 创建，见下文
realm=veryopen.org	定义域名，并且要为本域名创建证书
cert=/etc/letsencrypt/live/veryopen.org/fullchain.pem	域名的证书文件。证书采用 letsencrypt 命令产生，见下文
pkey=/etc/letsencrypt/live/veryopen.org/privkey.pem	域名证书的私钥文件

（4）给域名签发证书

letsencrypt 是开源证书管理工具，用它签发的域名证书，目前主流浏览器都支持。先安装 letsencrypt，然后再签发：

```
root@instance-1:~# apt install -y letsencrypt
root@instance-1:~# letsencrypt certonly --standalone -d veryopen.org
```

签发的证书文件是 /etc/letsencrypt/live/veryopen.org/fullchain.pem，证书私钥文件是 /etc/letsencrypt/live/veryopen.org/privkey.pem

（5）创建转发用户

```
root@instance-1:~# turnadmin -a -u zhanhsan -r veryopen.org -p A1b2C3
```

创建用户 zhangsan，密码是 A1b2C3。

（6）重启 coturn 服务

```
root@instance-1:~# systemctl restart coturn.service
```

（7）开启防火墙

```
root@instance-1:~# netstat -tunlp
```

如看到如下的内容，说明 coturn 服务启动正常：

```
tcp        0      0 10.128.0.2:5349         0.0.0.0:*               LISTEN      24828/turnserver
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
tcp        0      0 10.128.0.2:5349         0.0.0.0:*               LISTEN      24828/turnserver
tcp        0      0 10.128.0.2:3478         0.0.0.0:*               LISTEN      24828/turnserver
tcp        0      0 10.128.0.2:3478         0.0.0.0:*               LISTEN      24828/turnserver
udp        0      0 10.128.0.2:5349         0.0.0.0:*                           24828/turnserver
udp        0      0 10.128.0.2:5349         0.0.0.0:*                           24828/turnserver
udp        0      0 10.128.0.2:3478         0.0.0.0:*                           24828/turnserver
udp        0      0 10.128.0.2:3478         0.0.0.0:*                           24828/turnserver
```

发现 coturn 服务监听的 tcp 和 udp 端口都是 3478 和 5349，其中 5349 是加密端口，类似 WEB 服务的 443（https），3478 是非加密端口，类似 WEB 服务的 80（http）。当以后产生一个中继候选者时，coturn 服务从[10000, 20000]的 udp 端口池中任选一个并监听它，udp 端口池由参数 min-port 和 max-port 定义（参见表 6.11 中的配置）。另外 coturn 服务还有一个备用端口 3479，如果 3468 被占用，coturn 服务就监听备用端口 3479。所以防火墙要允许这些端口进入：tcp:3478,3479,5349；udp:3478,3479,5349,10000-20000。谷歌云主机防火墙设置：导航菜单》VPC 网络》防火墙，然后参考图 6.77。

	名称	类型	目标	过滤条件	协议/端口	操作	优先级	网络 ↑	日志
☐	webrtc	入站	应用到所有实例	IP 地址范围: 0.0.0.0/0	tcp:3478,3479,5349,9090 udp:3478,3479,5349,10000-20000	允许	1000	default	关闭

图 6.77 谷歌云主机防火墙配置

（8）验证 coturn 服务

用 firefox 浏览器打开网站 https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/，先删除里面的 stun:stun.l.google.com:19302，然后参照图 6.78 所示填写，再点击“Add Server”。

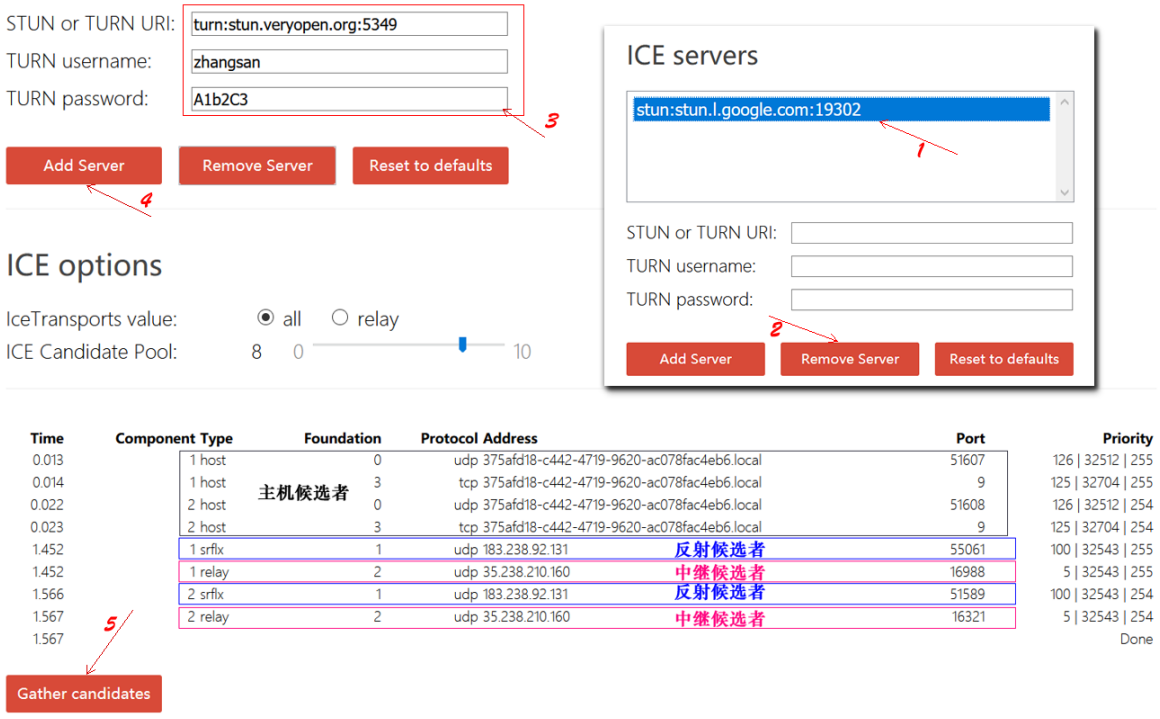


图 6.78 验证 TURN/STUN 服务器

反射候选者的“ip 地址:端口”就是运行浏览器的电脑所在局域网对应的最外层 NAT 网关的外部地址和外部端口。本例中，对方会尝试打洞访问“183.238.92.131:55601”或者“183.238.92.131:51589”。中继候选者的 ip 地址就是 TURN 服务器的浮动 ip 地址（因为我租用谷歌的云主机做 TURN 服务器，谷歌云主机在云端的租户网络上，租户网属于私有网，必须给云主机分配一个浮动 ip 地址才允许外面访问进来，这个浮动 ip 就是公网 ip 地址），这时在云主机里可以看到中继候选者的端口 16988 和 16321 处于监听状态，这两个 udp 端口都落在端口池[10000,20000]内。中继候选者的“ip:端口”中的 ip 就是 TURN 转发服务器的公网 ip，端口是 TURN 服务器上被监听的 UDP 端口。

（9）使用 coturn

在 js 代码中，实例化一个 RTCPeerConnection 对象时，针对参数对象 RTCConfiguration 的 iceServers 属性设置成 STUN/TURN 服务器的 URL，我们搭建的 STUN/TURN 服务器的合法 URL 有：

"stun:stun.veryopen.org" 、 "stun:stun.veryopen.org:5349" 、 "turn:stun.veryopen.org" 、 "turn:stun.veryopen.org:5349"。例如：

```
var configuration = {
    iceServers: [
      { urls: ["stun:stun.xten.com", "stun:stun.veryopen.org"] },
      { urls: "turn:stun.veryopen.org", username: "zhangsan",  credential: "A1b
2C3" }
    ]
};
myPeerConnection = new RTCPeerConnection(configuration);
```

详细介绍参见后续章节。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.11.4 实现步骤

WebRTC 编程主要考虑三个问题：一是如何与对端建立连接；二是如何获取本地媒体源（如话筒、耳机、摄像头等）并把媒体流定向到第一步中建立的连接上，或者从连接上分离媒体流并定向到本地媒体播放设备；三是如何传递任意类型的数据。

除了“§6.9 媒体流”小节讲过的多媒体对象外，WebRTC 编程还涉及这些接口对象：RTCPeerConnection、RTCSessionDescription、RTCIceCandidate、RTCDataChannel。其中 RTCPeerConnection 和 RTCDataChannel 是最核心的两大接口对象。

6.11.4.1 媒体点对点

想想你现在要打电话推销适合酒店客房使用的云终端产品，你不知道客户的名称和电话号码。你会首先从黄页中或打 114 查询 XX 城市的酒店，包括名称和联系电话，然后一一拨打电话推销产品。其中打 114 查询酒店名称和联系电话的过程就是信令过程，固定电话机就相当于 RTCPeerConnection 接口对象，与酒店通话就是点对点通讯。当然酒店在 114 登记自己的名称和联系电话也算是信令过程。酒店名称、联系电话和通信地址就是信令，至于你如何获取这些信令，除了问 114 外还有很多其他方式，比如上网搜索、查找黄页、购买等。在 WebRTC 应用中，视频分辨率、媒体编码/解码器、源地址和时序等构成了信令的主要内容。

下面我们先来看看 RTCPeerConnection 这部“电话机”。

可采用 pc = new RTCPeerConnection([configuration])语句实例化一个 RTCPeerConnection 对象，可选实参 configuration 为 RTCConfiguration 类型字典，专门提供实例的配置参数，如果实例化 RTCPeerConnection 时省略 configuration 实参，那么新实例的参数全部取默认值。

1、RTCConfiguration 字典

其中的属性都是可选的，这些属性有：

bundlePolicy - 指定捆绑策略，值为枚举类型："balanced"（默认策略。浏览器首先为添加的每种类型的内容(音频、视频和数据通道)各创建一个 RTCDtlsTransport。但如果对方不支持 BUNDLE 协议，那么只为一个音频轨道和视频轨道各创建一个 RTCDtlsTransport，而忽略其他内容轨道）、"max-compat"（浏览器最初为所有媒体轨道创建一个 RTCDtlsTransport，为数据通道创建一个单独的 RTCDtlsTransport）、"max-bundle"（开始只创建一个 RTCDtlsTransport，如果对方不支持 BUNDLE，那么只协商建立一个媒体轨道），当代的主流浏览器斗志 BUNDLE 协议。

certificates - 为 RTCCertificate 对象类型的数组，存放建立连接时需用的证书。如果不想使用默认的证书，你可以调用 RTCPeerConnection.generateCertificate()方法定制证书，例子参考下面讲解 generateCertificate()方法的部分内容。

iceCandidatePoolSize - 16 位无符号整数，用来定义预取的 ICE 候选者池的大小，默认值为 0（表示不预取候选者）。

iceServers - 为 RTCIceServer 类型的数组，数组的每一个元素定义了一台服务器，浏览器在建立点到点连接时会使用这些服务器，典型的是 TURN/STUN 服务器。如果省略这些服务器，那么成功建立连接的概率会大大降低。RTCIceServer 对象的属性有：urls（必须的属性，为一个字符串或字符串数组，指定服务器的 URL，一个 URL 的格式为"<stun|turn>:<域名或 ip>[:<端口号>]"，其中 turn 协议指明是 TURN 转发服务器，stun 协议就是 STUN 服务器。例如 turn:stun.veryopen.org）、username 和 credential（可选属性，定义用户名和身份验证信息，如密码。只有 turn 协议才可能需要验证身份）、credentialType（定义 credential 属性的类型，可取值"password"或"oauth"，前者为默认值，表示 credential 属性的值为密码）。看例子：

```
var configuration = {
    iceServers: [
        {
            urls: "turn:stun.veryopen.org:5349",
            username: "zhangsan",
            credential: "A1b2C3"
         }, {
            urls: [
                  "stun:stun.veryopen.org",
                  "stun:stun.xten.com",
                  "stun:stun.schlunde.de",
                  "stun:stun.l.google.com:19302"
            ]
        }
     ]
};
var pc = new RTCPeerConnection(configuration);
```

iceTransportPolicy - 指定 ICE 代理在连接协商期间应使用的传输选择策略，可取值"all"（默认值）和"relay"，前者表明 ICE 代理能使用任何候选者，后者表明只使用需要转发的候选者。

peerIdentity - 指明对端的 id，本属性默认为 null。每个 RTCPeerConnection 实例都有一个唯一的 id。如果指明了对端的 id，除非通过身份验证，否则没法建立连接。

rtcpMuxPolicy - 为了支持非多路复用的 RTCP，本属性定义 RTCP 多路复用策略，浏览器在收集候选者时使用此策略。可取值"require"（默认值。指示 ICE 代理只收集 RTP 候选者，并在其上作多路复用 RTCP，如果对端不支持 RTCP 多路复用，则没法建立连接）、"negotiate"（指示 ICE 代理同时收集 RTP 和 RTCP 候选对象。如果对端可以复用 RTCP，则将 RTCP 候选对象复用到相应的 RTP 候选对象之上，否则，将分别返回 RTP 和 RTCP 候选者）。

2、RTCPeerConnection 对象

此对象的主要属性有如下一些：

canTrickleIceCandidates – 布尔类型，反映对端是否接受 trickled ICE 候选者。trickled ICE 协议是 ICE 协议的扩展，该协议允许 ICE 代理以增量方式发送和接收候选者，而不是一次性交换完整候选者列表。通过这种增量配置，ICE 代理可以在收集候选者的同时开始连接检查，这样能大大缩短 ICE 处理完成所需的时间。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

connectionState – 当前链接状态，可能的值有"connecting"（正在建立连接）、"connected"（连接成功）、"failed"（失败）、"disconnected"（已拆线）。

currentLocalDescription – 为 RTCSessionDescription 对象类型，描述了最近成功协商建立连接后的本地端。RTCSessionDescription 对象包含两个属性：type（定义当前状态，可能的值"answer"、"offer"）、sdp（会话描述协议）。

currentRemoteDescription -描述了最近成功协商建立连接后的对端。

iceConnectionState – 反应与 RTCPeerConnection 关联的 ICE 代理（如浏览器）的状态，可能的值有"new"（正在收集候选者）、"checking"（正在匹配两端的候选者）、"connected"（成功匹配出一对候选者并且建立了连接，可能还在继续收集更多的候选者）、"completed"（完成了所有候选者的收集和匹配工做，并且建立了连接）、"failed"（匹配不成功，建立连接失败）、"disconnected"（临时掉线，可能一会儿恢复成 connected 状态）、"closed"（关闭，不再处理新的请求）。

iceGatheringState – 反应 ICE 代理收集后选者的情况，可能的值有"new"（还没有开始建立连接）、"gathering"（正在收集候选者）、"complete"（收集完毕）。

localDescription – 描述本地端的一个 RTCSessionDescription 对象。

remoteDescription - 描述对端的一个 RTCSessionDescription 对象。

signalingState – 反映信令过程处于的阶段。

3、RTCPeerConnection 对象的主要方法

addIceCandidate(candidate) – 把对方发过来的候选者加入本地 ICE 代理（如浏览器）。返回一个 Promise 对象。

addTrack(track[, stream...]) – 添加媒体轨道，触发对端 ontrack 事件。参数 track 为 MediaStreamTrack 对象，可选参数 stream...为若干个 MediaStream 对象，轨道被添加到这些媒体流中。一个 RTCPeerConnection 链接包含若干媒体流，一个媒体流包含若干媒体轨道，媒体流其实就是对媒体轨道进行分组。本方法返回一个 RTCRtpSender 对象。一个 RTCRtpSender 对象配对一个 RTCRtpReceiver 对象，它们组成一个 RTCRtpTransceiver 对象。如果一个 RTCRtpReceiver 实例的配对端没有添加任何媒体流，那么它是“静音”的（不会传递数据）。如果省略 stream 参数，那么媒体轨道是“裸露”的，由对方决定最终把轨道加入什么媒体流。请看图 6.79 所示的例子。

```
async openCall(pc) {
    const gumStream = await navigator.mediaDevices.getUserMedia({
        video: true,
        audio: true
    });
    for (const track of gumStream.getTracks()) {
        pc.addTrack(track);
    }
}
```

我方

```
let inboundStream = null;
pc.ontrack = ev => {
    if (ev.streams && ev.streams[0]) {
        videoElem.srcObject = ev.streams[0];
    } else {
        if (!inboundStream) {
            inboundStream = new MediaStream();
            videoElem.srcObject = inboundStream;
        }
        inboundStream.addTrack(ev.track);
    }
}
```

对方

图 6.79 把轨道加入 p2p 链接

close() – 关闭连接。

createAnswer([options]) – 创建一个应答 SDP 对象，作为对方发来的 offer 的回应。可选参数 options 提供额外的配置项 voiceActivityDetection（默认值为 true，即启用语音活动检测，当静音时就不发送，从而减少了用于音频的网络带宽；如果值为 false，静音音频照样发送。“静音”就是声音小到一定程度），如 pc.createAnswer({ voiceActivityDetection: false })。本方法返回一个 Promise 对象。

createDataChannel(label[, options]) - 创建一个可传递任何数据的通道，对于文字交流、文件分享特别有用。参数 label 为本通道命名，如"textChat"，可选参数 options 为字典，为通道指定一些配置内容，字典的属性可是可选的，具体有这些：ordered（数据从发送端和接收端是否保持相同的顺序，默认为 true，false 允许乱序到达）、maxPacketLifeTime（传递一个数据包的最大毫秒数，默认为 null，即没有限制）、maxRetransmits（指定一个消息的最大重传次数，在不稳定的线路上超时没有到达的消息将重传。默认值为 null）、protocol（定义 RTCDataChannel 上使用的子协议的名称，默认为空字符串""）、negotiated（是否带外协商建立数据通道。默认为 false，即带内协商，一方调用 createDataChannel()方法，对方监听 ondatachannel 事件；值为 true 时表示带外协商，两方都调用 createDataChannel()方法）、id（为本通道指明一个 ID，取值范围[0, 65534]。如果省略，则有浏览器自动产生 ID）。本方法返回一个 RTCDataChannel 对象。看下面带外协商的例子：

```
//双方都执行相同的代码
var pc = new RTCPeerConnection(options);
var channel = pc.createDataChannel("wechat", {
      negotiated: true,
      id: 668
});
channel.onopen = function (event) {
      channel.send('您好!');
}
channel.onmessage = function (event) {
      console.log(event.data);
}
```

图 6.80 是带内协商的例子：

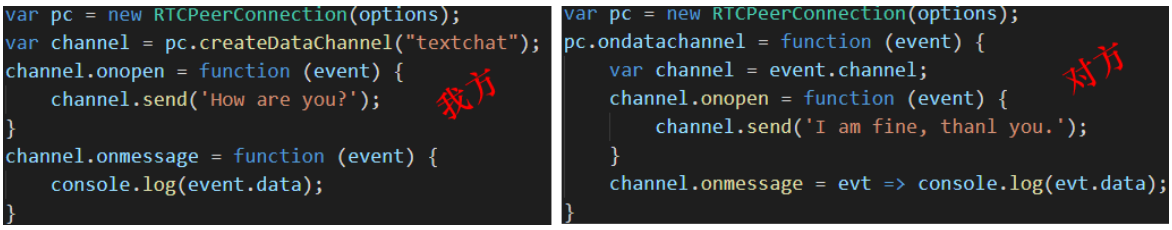


图 6.80 带内协商

createOffer([options]) – 创建一个类型为 SDP 对象的 offer，接下来会把此 offer 作为信令过程发给对方。SDP 就是会话描述协议。可选参数 options 为 RTCOfferOptions 类型的字典，定义一些额外配置项：iceRestart（为 true 就指示 RTCPeerConnection 对象重启 ICE 协商，默认为 false。重启 ICE 协商最好采用 RTCPeerConnection.restartIce()方法，如果 ICE 代理存在此方法的话）、voiceActivityDetection（参见 createOffer 方法）。本方法返回一个 Promise 对象。看例子：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
pc.oniceconnectionstatechange = function (evt) {
    if (pc.iceConnectionState === "failed") {
            if (pc.restartIce) {
                 pc.restartIce();
            } else {
                 pc.createOffer({ iceRestart: true }).then(pc.setLocalDescripti
on).then(sendOfferToServer);
            }
     }
}
```

generateCertificate(keygenAlgorithm) - 这是一个静态方法，用于创建 X.509 证书和相应的私钥。参数 keygenAlgorithm 指定证书生成算法，可为字符串或对象，常见的算法有 RSASSA-PKCS1-v1_5、ECDSA、RSA-PSS 和 HMA 等，其中前两者支持的浏览器最多。本方法返回一个 Promise 对象，成功时确值为包含新密钥的 RTCCertificate 对象，此对象包含的证书可用于 RTCPeerConnection 建立 DTLS 安全会话。证书的有效期默认为 30 天，可通过 expires 属性定义有效期，单位为毫秒。看例子：

```
    RTCPeerConnection.generateCertificate({
            name: 'RSASSA-PKCS1-v1_5',
            hash: 'SHA-256',
            modulusLength: 2048,
            expires: 15552000000,//有效期 180 天
            publicExponent: new Uint8Array([1, 0, 1])
    }).then(function (cert) {
          var pc = new RTCPeerConnection({ certificates: [cert] });
    });
```

由于 generateCertificate()是静态方法，所以总是这样调用：

```
RTCPeerConnection.generateCertificate(…)
```

再看一个例子：

```
    RTCPeerConnection.generateCertificate("ECDSA").then( cert => {
          var pc = new RTCPeerConnection({ certificates: [cert]});
    });
```

getConfiguration() - 返回一个 RTCConfiguration 对象，如果之前调用过 setConfiguration()方法，则返回的 RTCConfiguration 对象就是最近调用 setConfiguration()方法的实参，否则就是 RTCPeerConnection 实例化时的参数对象。

getReceivers() - 返回一个 RTCRtpReceiver 对象的数组，数组中的每个元素代表一个 RTP 接收器。RTP 接收器就是管理 RTCPeerConnection 上 MediaStreamTrack 的数据接收和解码。

getSenders() - 返回一个 RTCRtpSender 对象的数组，数组中的每个元素代表一个 RTP 发送者。RTP 发送者对象中的方法和属性可用于检查、控制轨道数据的编码和传输。看例子：

```
    function setMuting(pc, muting) {
          let senderList = pc.getSenders();
          senderList.forEach(sender => sender.track.enabled = !muting);
    }
```

这个自定义函数用于设置“静音”或“非静音”，音频静音就是无声，视频静音就是黑屏。比如 setMuting(pc, true)调用就是置为静音。

getStats(selector) - 返回 Promise 对象，调用成功则确值为 RTCStatsReport 对象数组——保存了连接的状态信息，具体内容与 selector 参数有关。selector 指明具体的媒体轨道（MediaStreamTrack）或为 null。默认为 null，此时就是整个 RTCPeerConnection 的状态信息。每个 RTCStatsReport 对象实例至少包含如下三个属性：id（唯一标识被检查的对象）、type（被检查对象的类型）、timestamp（检查的时间，单位为毫秒），其他属性与被检查的对象有关。下面的例子输出全部的状态信息：

```
function printStatus() {
    myPeerConnection.getStats(null).then(stats => {
        stats.forEach(report => {
            Object.keys(report).forEach(statName => console.log(statName+":"+re
port[statName]));
        });
        console.log("-------------------------------------------------------");
    });
}
```

getTransceivers() - 返回 RTCRtpTransceiver 对象列表，这些对象在链接会话中发送和接收数据。下面的例子终止全部的 transceivers：

```
myPeerConnection.getTransceivers().forEach(transceiver => {
     transceiver.stop();
});
```

removeTrack(sender) - addTrack() 方法的逆向操作，停止发送媒体数据，不会删除相应的 RTCRtpSender 对象。看例子：

```
    //呼叫:
    pc = new RTCPeerConnection();
    var track = stream.getVideoTracks()[0];
    sender = pc.addTrack(track, stream);
    ……
    //挂机:
    pc.removeTrack(sender);
    pc.close();
```

restartIce() - 本方法调用之后，下一次使用 createOffer（）创建的 offer 一旦通过您的信令机制被发送到对方，便会导致 ICE 重新启动。 重新启动 ICE 实际上会重置 ICE，以便它使用新的凭据创建所有新的候选者。看例子：

```
    pc.addEventListener("iceconnectionstatechange", event => {
         if (pc.iceConnectionState === "failed") {
              pc.restartIce();
         }
    });
```

setConfiguration(configuration) - 修改 RTCPeerConnection 实例的当前配置，参数 configuration 为 RTCConfiguration 对象类型。看例子：

```
    const rnewConfig = {
          iceServers: [{
              urls: "turn:stun.veryopen.org",
              username: "allie@veryopen.org",
              credential: "123456"
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
        }]
    };
    pc.setConfiguration(newConfig);
    pc.restartIce();
```

setLocalDescription([sessionDescription]) - 修改连接的本端描述。可选参数 sessionDescription 为 TCSessionDescription 对象类型，用于指定配置项，如果省略此参数，则由 WebRTC 系统自动决定配置。返回 Promise 对象，确值的处理函数没有参数。看例子：

```
myPeerConnection.createOffer().then( offer => {
    myPeerConnection.setLocalDescription(offer).then(()=> send2Peer(offer));
});
```

setRemoteDescription([sessionDescription]) - 当收到 offer 或 answer 后，调用本方法设置连接的远端描述。参数和返回值与 setLocalDescription()方法相同。

4、RTCPeerConnection 对象的主要事件

connectionstatechange – 属性 connectionState 的值发生改变时触发。

datachannel – 当对端调用 createDataChannel()方法时触发，即对方添加了数据通道。看例子：

```
    pc.addEventListener("datachannel", ev => {
          receiveChannel = ev.channel;
          receiveChannel.onmessage = myHandleMessage;
          receiveChannel.onopen = myHandleOpen;
          receiveChannel.onclose = myHandleClose;
    }, false);
```

icecandidate – 发现新的候选者，或者候选者收集完毕时触发。如果是前一种情况，就要把新的候选者发送给对方。看例子：

```
    pc.addEventListener("icecandidate", ev => {
          if (ev.candidate) {//发现新的候选者
              sendCandidateToRemotePeer(ev.candidate)
          }
    });
```

iceconnectionstatechange – 当属性 iceConnectionState 的值发生改变时触发。

icegatheringstatechange – 当属性 iceGatheringState 的值发生改变时触发。

signalingstatechange – 当属性 signalingState 的值发生改变时触发。

track – 对方添加了媒体轨道后触发。看例子：

```
    pc.addEventListener("track", e => {
          videoElement.srcObject = e.streams[0];
    }, false);
```

下面我们利用前面介绍的信令服务器 signalserver.js，实现两个人的视频聊天页面，代码参考 chapter-6\webrtc.html 和 chapter-6\webrtcclient.js 两个文件。在生产环境中，为了增加代码对更多浏览器的适应性，建议引入 adapter.js。

如果要走安全协议 wss，请用 var conn = new WebSocket('wss://stun.veryopen.org:9090');替换 client.js 中的相应语句。最好结合 server.js 代码一起阅读，比如注册时向服务器发送信令消息{type: "login", name:"Alice" }，服务器就把"Alice"保存起来。client.js 用到的 STUN/TURN 服务器 stun.veryopen.org 是作者在谷歌云上自行搭建的，具体搭建的过程见后续章节，你也可以采用一些免费的服务器。

从 chrome 浏览器的两个标签页上浏览 webrtc.html 页面，输入自己的名字（比如 John），然后再输入对方的名字（比如 Alice），最终看到的效果如图 6.81 所示。在 chrome 浏览器的新标签页上打开 chrome://webrtc-internals/，可以查看 WebRTC 建立连接的详细过程。

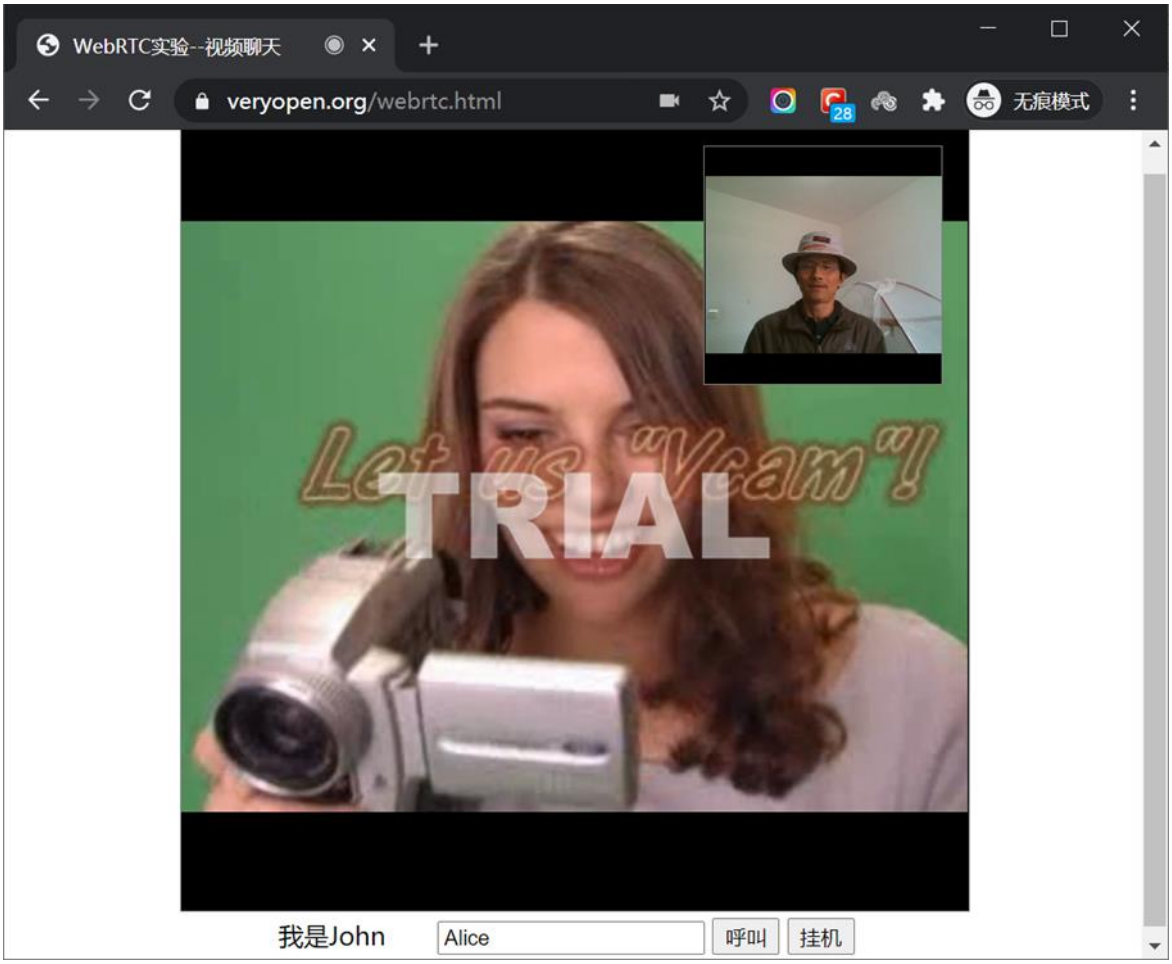


图 6.81 WebRTC 举例

6.11.4.2 数据点对点

采用 RTCDataChannel 接口对象把数据通道附加到 RTCPeerConnection 会话上，从而实现任意数据传输，比如文字聊天、文件共享等。RTCPeerConnection 会话上允许附加多个 RTCDataChannel 数据通道。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

调用 RTCPeerConnection 实例的方法 createDataChannel()返回一个 RTCDataChannel 对象，createDataChannel()方法的具体用法参见前面章节介绍 RTCPeerConnection 的内容。

RTCDataChannel 对象具有如下一些属性：

binaryType - 当传递二进制数据时，定义其对象类型，可取值"blob"或者"arraybuffer"（默认值），前者表示采用 Blob 对象，后者表示采用 ArrayBuffer 对象。

bufferedAmount - 只读属性，表示正在发送队列中的数据的字节数。当它小于 bufferedAmountLowThreshold 属性的值时触发事件 bufferedamountlow，bufferedAmountLowThreshold 属性的默认值为 0。

id - 只读属性，唯一标识一个数据通道的数字，介于 0 到 65534 之间，默认值为 null。

label - 只读属性，数据通道的名字，允许不唯一。

maxPacketLifeTime - 传送一个数据包的最大毫秒数。默认为 null，表示没有限制。

maxRetransmits - 最大重传次数，默认值为 null。

negotiated - 布尔类型，true 表示由 web app 协商建立连接，false 表示由 WebRTC 层协商，默认值为 false。

ordered - 布尔类型，默为 true，表示数据到达的顺序与发送的顺序相同，false 表示允许乱序到达。

protocol - 定义数据通道采用的子协议，默认为空字符串。协议名字具体由程序员定义，如"json"、"raw"等。

readyState - 数据通道的当前状态，可能的值有"connecting"（正在建立连接）、"open"（建立连接成功，可以传送数据了）、"closing"（关闭中）、"closed"（已关闭）。

RTCDataChannel 对象具有如下一些方法：

close() - 关闭数据通道。

send(data) - 发送数据，参数 data 可为 USVString、Blob、ArrayBuffer 或 ArrayBufferView 类型。看例子：

```
dataChannel.send(JSON.stringify({ "message": msg, "timestamp": new Date() }));
```

RTCDataChannel 对象具有如下一些事件：

bufferedamountlow - 发送队列中的字节数小于 bufferedAmountLowThreshold 属性的值时触发。

close - 关闭数据通道时触发。

message - 收到数据时触发，收到的数据在实参的 data 属性中。如：

```
dataChannel.addEventListener("message",({data}) => console.log(data));
```

open - 建立连接成功时触发，接下来就可以传送数据了。

下面我们结合前面介绍的信令服务器 signalserver.js，开发文字聊天和文件分享的 web 网站，实现如图 6.82 所示的浏览效果，在两个浏览器中打开网站 https://www.veryopen.org/datachannel.html，然后各自注册并登陆一个用户，最后任由一方呼叫另一方建立点到点连接。此后便可以输入文字聊天了，或者分享任意文件。



图 6.82 用数字点对点实现文字聊天

代码参见 chapter-6\datachannel\ 目录中的三个文件 datachannel.html、datachannel.css 和 datachannel.js，基本思路是：首先在信令服务器上注册，然后通过信令服务器把我的 offer 转发给对方，对方收到 offer 后沿路返回一个 answer，最后各自把浏览器发现的候选者通过信令服务器转发给对方。浏览器开始局域网打洞，从而选择最佳的候选者对，这样就建立了点对点链接。接下来任何一方可把数据通道附加到点对点链接，另一方响应事件即可。此后可以开始文字聊天和分享文件了。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.12 Web 组件

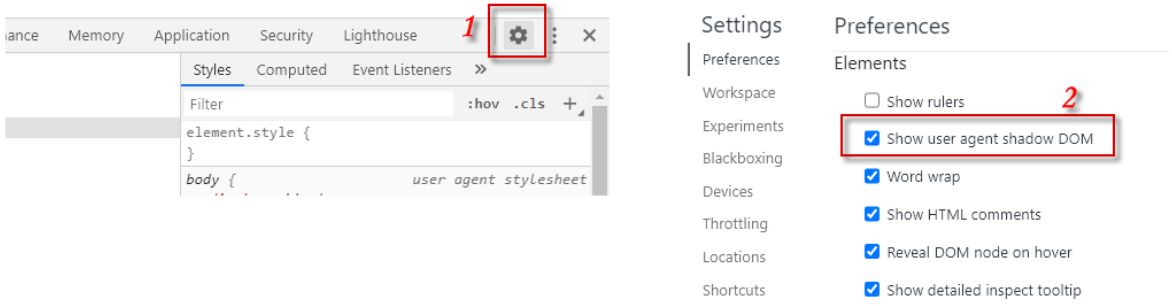
只要你做过程序开发人员，你应该就明白函数或组件或模块的作用：封装和重用。比如通过顶点坐标计算三角形的面积，如果项目中只计算一次三角形面积，那么真的没必要为此单独开发一个函数或组件，因为没有重用性（多次使用），这时直接把计算面积的代码写在程序中本来的位置就可以了。但是如果项目中经常要用到通过顶点坐标计算三角形面积的话，聪明的程序员就会单独开发一个计算三角形面积的组件（或至少写一个函数），在需要计算面积的地方引用这个组件（或调用函数）即可，这样一方面计算面积的具体代码被封装在组件或函数中，引用组件或调用函数的人根本不需要了解三角形面积是具体怎么计算出来的，另一方面每次引用组件或调用函数本质上就是重复执行那些被封装的代码。

现在的问题是：这种妙极了的“封装和重用”技巧能否用于网站前端开发？因为一个稍具规模的网站，其前端代码（Html5+CSS3+JavaScript）动辄几万行多则十几万行是很常见的事，而且可以肯定这种“封装和重用”的需求也很旺盛。

为此 web 组件诞生了，web 组件这是个统称，具体的技术包括以下三种：定制元素、影子 DOM 和 HTML 模板，这些技术可以单独使用，也可以组合使用。什么？React、Vue 就是前端组件？这我知道，但我在这里讲的是由浏览器厂家推动的原生 web 组件。相比第三方组件框架，原生组件具有明显的性能优势！

6.12.1 定制元素

迄今为止，你用<video>元素应该很 happy 了吧，在页面上播放视频太简单了，就一个元素搞定！但我要告诉你：你的 happy 是建立在巨人肩膀上的，因为类似<audio>、<details>、<input>、<meter>、<progress>、<textarea>、<video>等这样的元素是由浏览器厂家在基础元素上开发的原生定制元素，不同于第三方组件（如 React、Vue 等），原生定制元素简单直接，不用加载任何外部模块，代码量小。我们通过浏览器的开发者工具就能“窥探”这种定制元素的内部结构。对于 chrome 浏览器，按 Ctrl+Shift+i 或 F12 进入开发者工具，然后点击齿轮图标打开设置，勾选 Elements 下的“Show user agent shadow DOM"，具体参考下图操作。



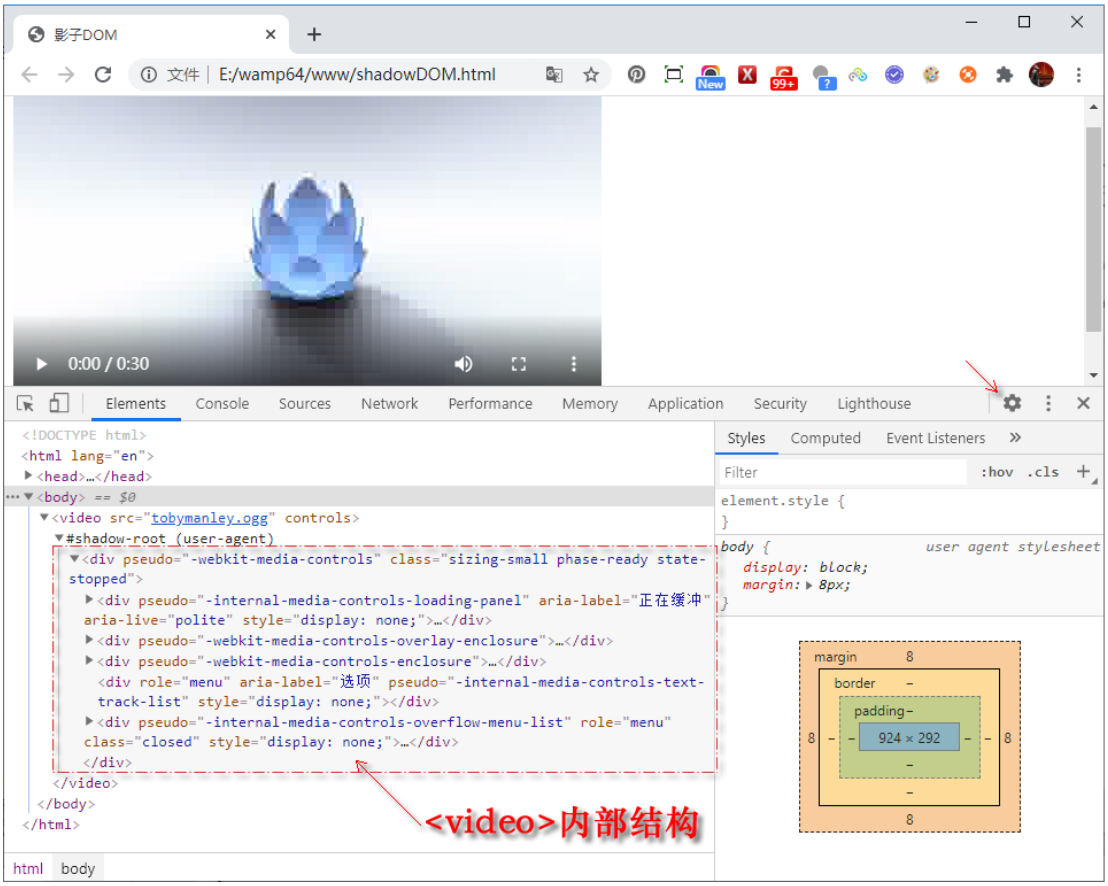


图 6.65 查看定制元素的内部结构

继续深入“窥探”，如图 6.66，可以发现<video>内部的很多元素都带有 pseudo 属性：

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk



图 6.66 <video>内部的很多元素都带有 pseudo 属性

这种带有 pseudo 属性的元素称为伪元素，表明我们可以采用伪元素选择器来修改<video>元素的样式，比如当鼠标停靠播放按钮时，按钮的背景色改为黄色，参考下面的代码。

```
<style>
      video::-webkit-media-controls-play-button:hover {
          background-color: yellow;
          border-radius: 50%;
      }
</style>
```

代码中的-webkit-media-controls-play-button 就是上图播放按钮的 pseudo 属性的值。我们还可以窥探<input type="range">的影子 DOM，参考图 6.67。

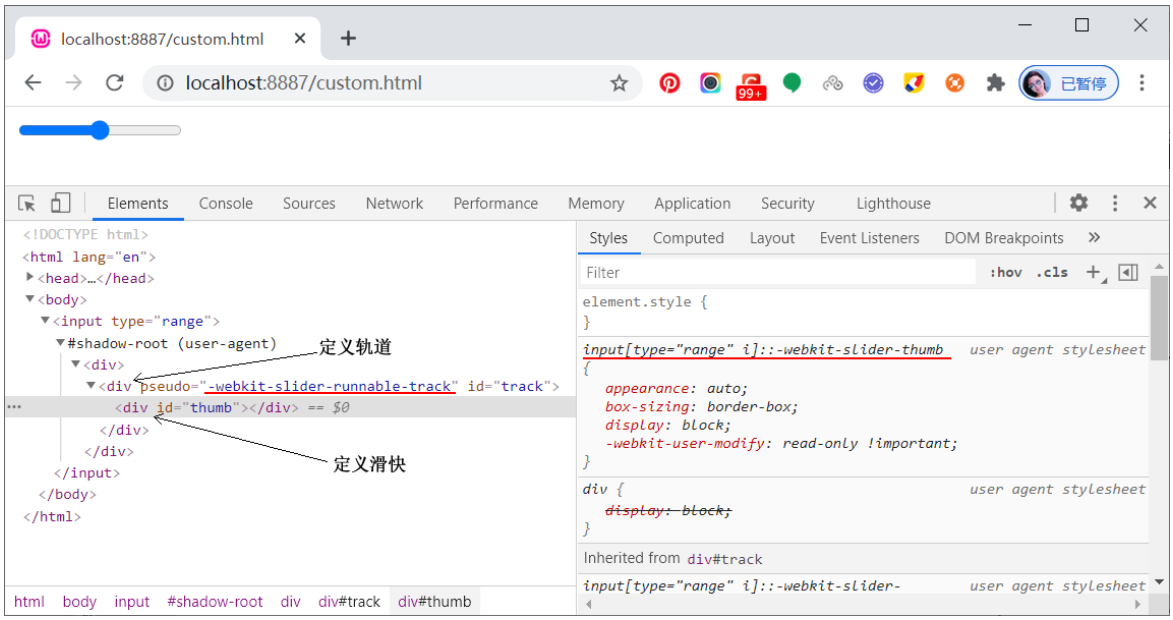


图 6.67 <input type="range">的影子 DOM

这样在样式选择器 input[type=range]::-webkit-slider-runnable-track 中给 background-image 指定图片就可以定制轨道，在样式选择器 input[type=range]::-webkit-slider-thumb 中给 background-image 指定图片就可以修改默认的滑块，具体的样式代码请参考“§4.9.3 <input>元素”小节。

令人兴奋的是，我们自己也能定制全功能的元素！为了避免与浏览器厂家定制的元素同名，厂家承诺他们定制的元素名称当中不含“-”字符，所以我们定制的元素名称当中至少要包含一个“-”字符了，以免元素的名字冲突，比如<my-product>、<contact-info>、<my-basic-resume>等都是些好的元素名字。

firefox 浏览器需要把 devtools.inspector.showAllAnonymousContent 设置为 true(低于 76 版的 firefox，把 devtools.inspector.showUserAgentShadowRoots 参数设为 true)，然后重启浏览器才能“窥探”定制元素的内部结构，具体操作是在浏览器中输入 about:config 并回车，然后搜索 devtools.inspector.showAllAnonymousContent(或者 devtools.inspector.showUserAgentShadowRoots)，见图 6.68。

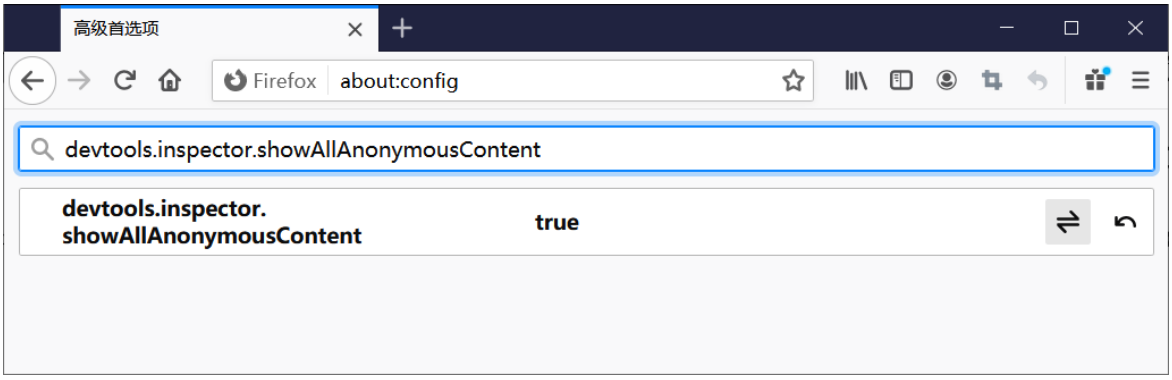


图 6.68 firefox 开启“窥探”影子 DOM 的功能

好了，下面看看具体如何定制我们自己的元素，这时 JavaScript 又要派上用场了，是的，只能用 JS 代码来定制元素。我们需要采用 customElements.define()函数，其基本语法如下：

```
void customElements.define(name, constructor, options);
```

参数 name 就是元素名，最少包含一个“-”符号，constructor 为新元素的构造器，一般为一个类对象，可选参数 options 为对象类型，目前只支持 extends 成员属性，指明一个继承的内置元素名字，看例子：

```
customElements.define('popup-info', PopUpInfo);
```

本例子新建元素<popup-info>，没有继承任何内置元素，这样的定制元素称为独立定制元素，独立定制元素的用法与原生内置元素一样，比如<popup-info>blabla</popup-info>。如果要继承内置元素，就称为定制内置元素，即修改原生的元素，如下面的代码：

```
customElements.define('word-count', WordCount, { extends: 'p' });
```

本例子不是创建了一个新元素，而是修改了原生 元素，采用 blabla 或者 document.createElement("p", { is: "word-count" })来使用修改后的 元素。

显然，调用 customElements.define()函数之前，我们必须要先准备好第二个参数：构造器，比如上面例子当中的 PopUpInfo 和 WordCount 类对象。类对象的基本框架如下。

对于独立定制元素，相应的类对象的框架如下：

```
    class 类对象名称 extends HTMLElement {
        constructor() {
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
            super();   //首先执行基类的构造器
            //接下来可以初始化状态、设置监听器或者创建影子 DOM
            ...
        }
       //各种生命周期函数
       ...
    }
```

对于定制内置元素，它的类对象的框架如下：

```
    class 类对象名称 extends 内置元素接口 {
        constructor() {
            super();   //首先执行基类的构造器
            //接下来可以初始化状态、设置监听器或者创建影子 DOM。
            ...
        }
       //各种生命周期函数
       ...
    }
```

内置元素接口=HTML+首字母大写的内置元素+Element，比如<video>对应的内置元素接口为HTMLVideoElement、<ul>对应 HTMLUlElement、对应 HTMLSpanElement，具体参考https://developer.mozilla.org/en-US/docs/Web/API。

定制元素的四个生命周期函数如下：

connectedCallback() - 当元素被插入到 DOM 时触发执行。我们常在这里做一些加载资源和渲染的工作。

disconnectedCallback() - 当元素从 DOM 中移除时触发执行。我们常在这里作一些清理工作。

attributeChangedCallback(attrName, oldVal, newVal) - 当被监视的元素属性被改变（添加、移除、更改、替换）时触发执行。注意：只有列在类成员属性 observedAttributes 中的元素属性才被监视。

adoptedCallback() - 当元素被移到新页面文档时触发执行。比如调用 document.adoptNode(el)移动元素。

这四个生命周期函数是可选的，如果需要就用，否则就不用。下面是一个包含各个生命函数的类对象框架：

```
class ClassName extends HTMLElement {
    constructor() {
       super();
       ...
    }
    connectedCallback() {
       ...
    }
    disconnectedCallback() {
       ...
    }
    attributeChangedCallback(attrName, oldVal, newVal) {
       ...
    }
    disconnectedCallback() {
```

```
        ...
    }

}
```

接下来我们来看一个完整的独立定制元素和使用独立定制元素的例子，本例子新建一个独立元素<flag-icon>，该元素会根据属性 country 的值显示相应的国旗图标，不过事先要在网页所在目录中放置各种国旗图片，图片文件名为国家代码.png，比如美国国旗的图片文件名为 us.png。

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>独立定制元素&lt;flag-icon&gt;</title>
    <script type="module">
        class FlagIcon extends HTMLElement {
            constructor() {
                super();
                this._countryCode = null;
            }

            static get observedAttributes() {
                //监视元素的 country 属性的变化:
                return ["country"];
            }

            //当元素的 country 属性的值发生改变时触发执行:
            attributeChangedCallback(name, oldValue, newValue) {
                this._countryCode = newValue;
                this._updateRendering();
            }

            connectedCallback() {
                this._updateRendering();
            }

            //引用 this.country 时调用此函数:
            get country() {
                return this._countryCode;
            }

            //当给 this.country 赋值时调用此函数:
            set country(v) {
                this.setAttribute("country", v);
            }

            _updateRendering() {
                if (this.ownerDocument.defaultView)
                    this.innerHTML = "<img src='" + this.country + ".png'>";
            }
        }
        customElements.define("flag-icon", FlagIcon);
    </script>
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
</head>

<body>
    <flag-icon country="cn"></flag-icon>
</body>

</html>
```

代码中 this 代表 DOM 上的<flag-icon>元素对象，this.ownerDocument 代表<flag-icon>元素对象所在页面的 document 对象，而 this.ownerDocument.defaultView 代表<flag-icon>元素对象所在页面的 document 对象所在的 window 对象（参考“§2.4 DOM”小节），如果页面还没有完成渲染，this.ownerDocument.defaultView 的值为 null，语句 this.getAttribute("country")获取<flag-icon>元素的属性 country 的值。换句话说，只有在页面本身显然完成之后，才显示国旗图标。如果缺少 set country(v) { … }语句，那么在外部通过 JS 代码直接修改<flag-icon>元素的属性 country 的值（餐参考下面的代码），就不会触发执行 attributeChangedCallback()方法。

```
    <flag-icon country="us"></flag-icon>

    <script>
        const element = document.querySelector('flag-icon');
        element.country = 'jp';
    </script>
```

独立定制元素的使用与诸如 的原生内置元素的使用没有任何区别，当然也可以添加事件监听器，比如：

```
var newContact = document.createElement('contact-info')//或 new ContactInfo();
document.body.appendChild(newContact);
document.querySelector('contact-info').addEventListener('click', function () {
      ...
});
```

定制元素的事件监听器也可以在类对象的 contructor()构造函数中指定。如果你需要的定制元素与原生的内置元素差异不大，最好采用定制内置元素，这样相同部分从内置元素继承，额外补上差异部分即可。相反，如果你需要的元素功能复杂（比如商品的缩略图、智能表格等），这时就最好采用独立定制元素了，不过结合后面讲到的影子 DOM 和 HTML 模板技术就更能发挥定制元素的威力。

实操练习

设计一个提示图标的独立定制元素<pop-tips>，当鼠标停靠图标上面时弹出提示信息，见图 6.69 所示。提醒：采用 this. getAttribute('img')获取<pop-tips>元素的 img 属性的值。

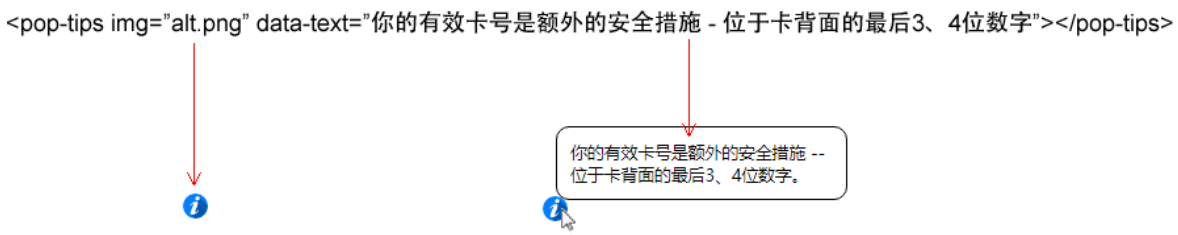


图 6.69 定制元素实验

参考资料

1、https://www.robinwieruch.de/web-components-tutorial

6.12.2 影子 DOM

在“§2.5 DOM 模型树”小节中，我们讲过一个页面内的元素对象组成了一颗 DOM 树，实际上这个 DOM 树上的某个枝条允许形成一个封闭的区域——封闭名字空间（内外的样式和行为互不影响），树枝作为整体可以附加到某个页面的 DOM 树上，或者移走。参考示意图 6.70。

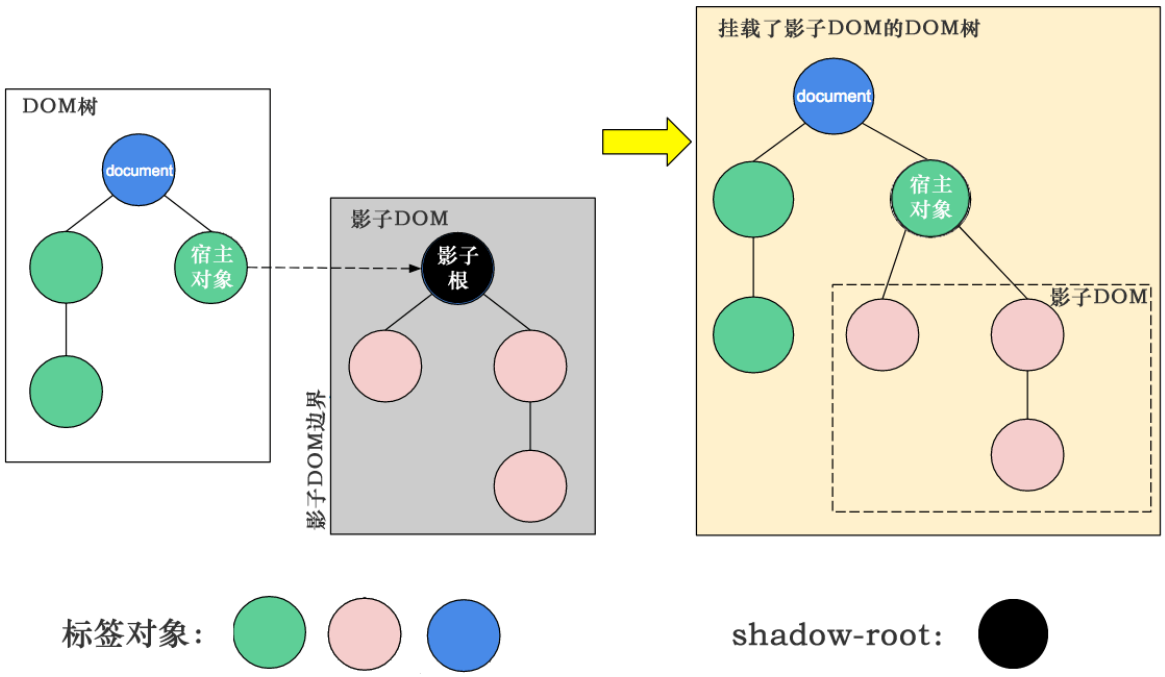


图 6.70 影子 DOM 说明

除了影子根不是元素对象外，其他的都是元素对象，影子 DOM 内的样式和行为不会影响外部 DOM，外部 DOM 中的样式和行为也不会影响影子 DOM。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

对于下面的代码，我们可以“窥探”附加在<audio>这个宿主对象的影子 DOM 的内部结构如图 6.71。

```
<audio src="beijing.mp3" controls></audio>
```

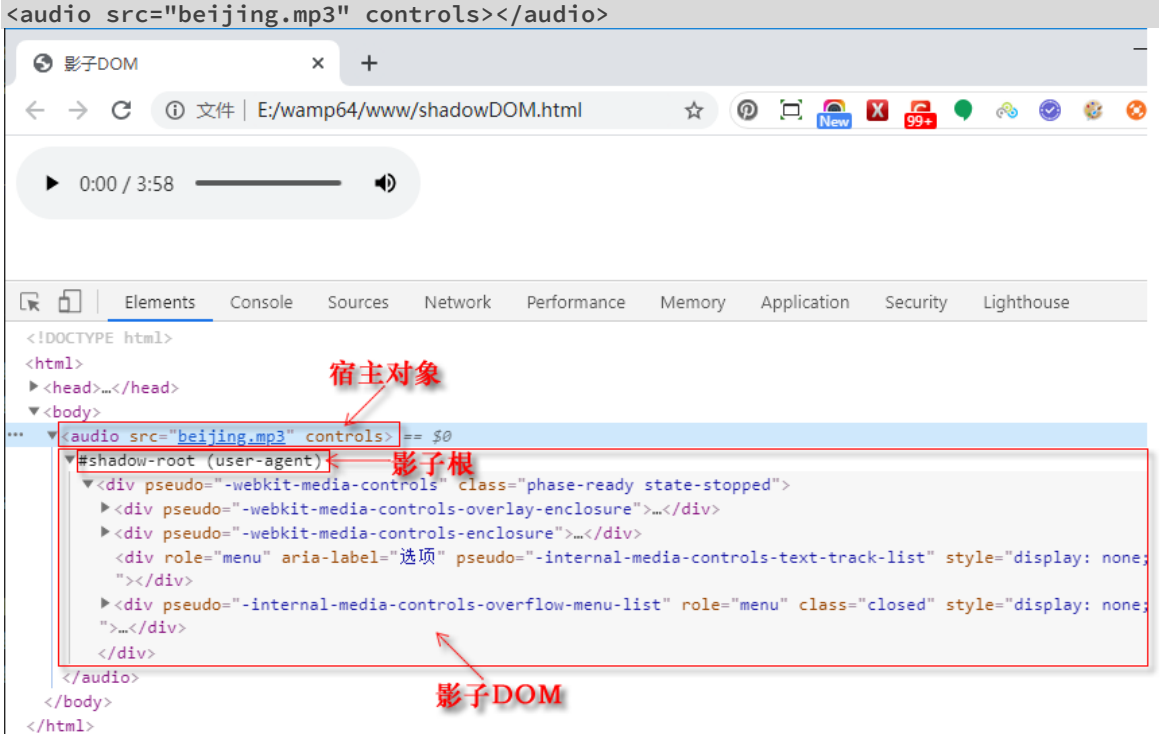


图 6.71 <audio>的影子 DOM

<audio>元素就是由浏览器厂家开发的定制元素，在这个定制元素的类对象中生成了一棵影子 DOM 树，此树上包含若干个带有 pseudo 属性的 元素，我们可以对诸如::-webkit-media-controls-enclosure 这样的伪元素设置样式。当然也可以在外面通过类似下面的语句修改影子内部的内容：

```
document.getElementById("item-info").shadowRoot.getElementById("img2").style.animation = "none";
```

影子 DOM 树可以附加在原生的元素对象上，但更多的情况是附加在定制元素对象上，采用宿主对象的方法 attachShadow()完成附加操作，这个方法可带一个参数，形式如下：

{mode: 'open'} - 允许在外部采用 JS 语言通过宿主对象的 shadowRoot 属性访问影子 DOM。

{mode: 'closed'} - 不允许从外部访问影子 DOM，因为此时宿主对象的 shadowRoot 属性等于 null。

比如下面的代码把一棵空的影子 DOM 树附加在原生 对象上：

```
<!DOCTYPE html>
<html>

<head>
    <meta charset="UTF-8">
    <title>影子 DOM</title>
    <style>
        span {
            background-color: blue;
        }
    </style>
```

```
</head>

<body>

    <div id="host"></div>
    <script>
        document.getElementById('host').attachShadow({mode: 'open'});
    </script>
</body>

</html>
```

由于是带参数{mode: 'open'}调用 attachShadow()方法，因此在影子外部可以通过 document.getElementById('host').shadowRoot 属性访问影子 DOM 树。“窥探”影子 DOM 的内部结构见图 6.72 左侧部分。

接下来在影子 DOM 树中增加元素对象，代码如下：

```
    let shadowR = document.getElementById('host').attachShadow({mode: 'open'});
    let sp = document.createElement('span');
    shadowR.appendChild(sp);
```

可以发现外部的样式没有影响到影子 DOM 中的元素，这是影子 DOM 的内部结构见图 6.72 中间部分。我们继续在影子 DOM 中添加样式表，在 shadowR.appendChild(sp);下添加如下代码：

```
    let sty = document.createElement('style');
    sty.textContent = `
    span {
        display: block;
        width:60px;
        height: 60px;
        border-radius: 50%;
        background-color: red;
    } `;
    shadowR.appendChild(sty);
```

现在浏览器中显示了一个红色圆，而影子 DOM 的内部结构见图 6.72 右侧部分。

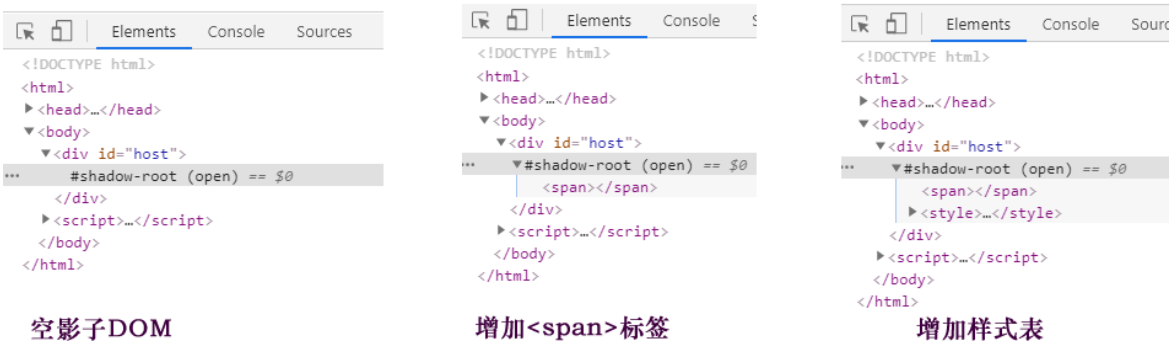


图 6.72 修改标准元素的影子 DOM

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

下面用实际的例子来新建一个独立定制元素<product-info>，并且附加影子 DOM，形成图 6.73 所示的效果。

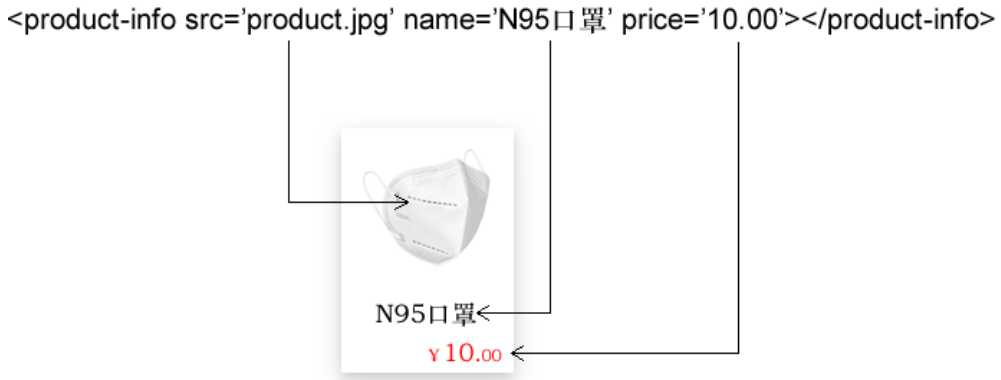


图 6.73 新建商品元素

要结合独立定制元素和影子 DOM 技术，在独立定制元素的类对象构造函数中完成影子 DOM、并增加<img>和两个<p>元素，而这三个元素又包含在 元素当中，其中<img>用来显示商品图片，<p>显示商品名称和价格，详细的代码如下：

```
<head>
    <script type="module">
        window.onload = function () {
            class ProductInfo extends HTMLElement {
                constructor() {
                    super();

                    //this 代表父 DOM 树中的<product-info>元素对象
                    var shadow = this.attachShadow({ mode: 'open' });

                    var container = document.createElement('div');
                    container.setAttribute('id', 'container');

                    var img = document.createElement('img');
                    img.setAttribute('src', this.getAttribute('src'));

                    var pName = document.createElement('p');
                    pName.setAttribute('id', 'name');
                    pName.innerText = this.getAttribute('name');

                    var pPrice = document.createElement('p');
                    pPrice.setAttribute('id', 'price');
                    var pri = this.getAttribute('price').split('.');
                    pPrice.innerHTML='<small>
￥</small>'+pri[0] + '.<small>' + pri[1] + '</small> ';

                    var styl = document.createElement('style');
                    styl.textContent = `
                    #container {
                        width: 120px;
                        height: 170px;
```

```
                        box-shadow: 0px 4px 8px rgba(102,102,102,0.2),
0px 6px 20px rgba(102,102,102,0.19);
                        text-align: center;
                        margin: 10px;
                    }
                    img {
                        max-width: 100px;
                        max-height: 100px;
                    }
                    #name {
                        margin-bottom: -10px;
                    }
                    #price {
                        text-align: right;
                        color: red;
                    }`;

                    container.appendChild(styl);
                    container.appendChild(img);
                    container.appendChild(pName);
                    container.appendChild(pPrice);
                    shadow.appendChild(container);
                }

            }
            customElements.define('product-info', ProductInfo);
        }
    </script>
</head>

<body>
    <product-info src='product.jpg' name='N95口罩' price='10.59'></product-info>
    <product-info src='book.jpg' name='通往奴役之路' price='33.10'></product-info>
</body>
```

构造函数中的 this 代表页面 DOM 树中的<product-info>元素对象，所以语句 this. getAttribute('src') 就是获取<product-info>元素的属性 src 的值，即图片文件名。另外在影子 DOM 中增加了样式表，由于影子 DOM 的隔离性，这些样式不会影响外部的元素，当然外部的样式也不会影响影子 DOM 内部的元素。最终在开发者工具中看到的效果如图 6.74。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

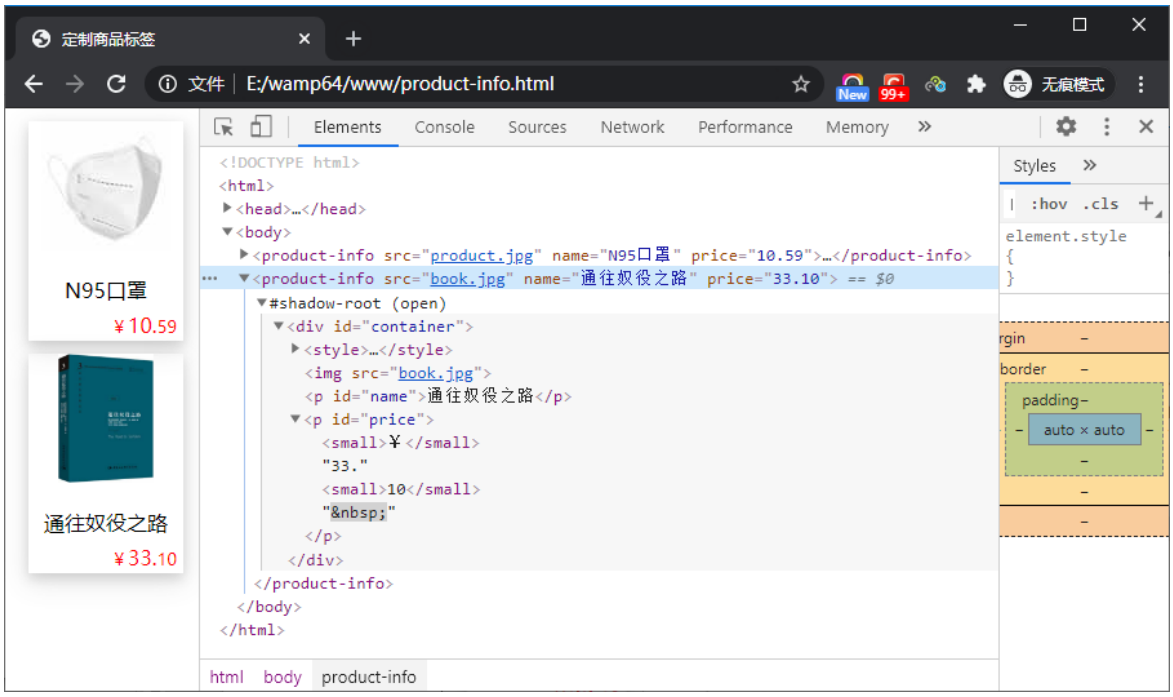


图 6.74 <product-info>元素的影子 DOM

实操练习

在上面例子的基础上，增加“购买”按钮，点击按钮就弹出窗口显示“已经放入购物车”，类似图 6.75 所示。



图 6.75 影子 DOM 实验

6.12.3 HTML 模板

影子 DOM 结合定制元素完美实现了封装，也实现了某种程度的复用，比如一旦实现了<product-info>元素（参考上一小节的例子），我们就可以在页面文件中随时使用此元素。而本小节要讲述的 HTML 模板纯粹是为了代码复用，把网站开发中经常用到的具备相同 HTML 结构、样式和行为的代码剥离出来，“打包”成一个 HTML 模板，然后在需要这些共性代码的地方插入模

板的一个实例，这样被“打包”到模板中的代码就实现了复用，而且实例化模板时还可以传递具体的内容。HTML 模板相当于 JavaScript 语言的函数，实例化模板相当于调用函数。

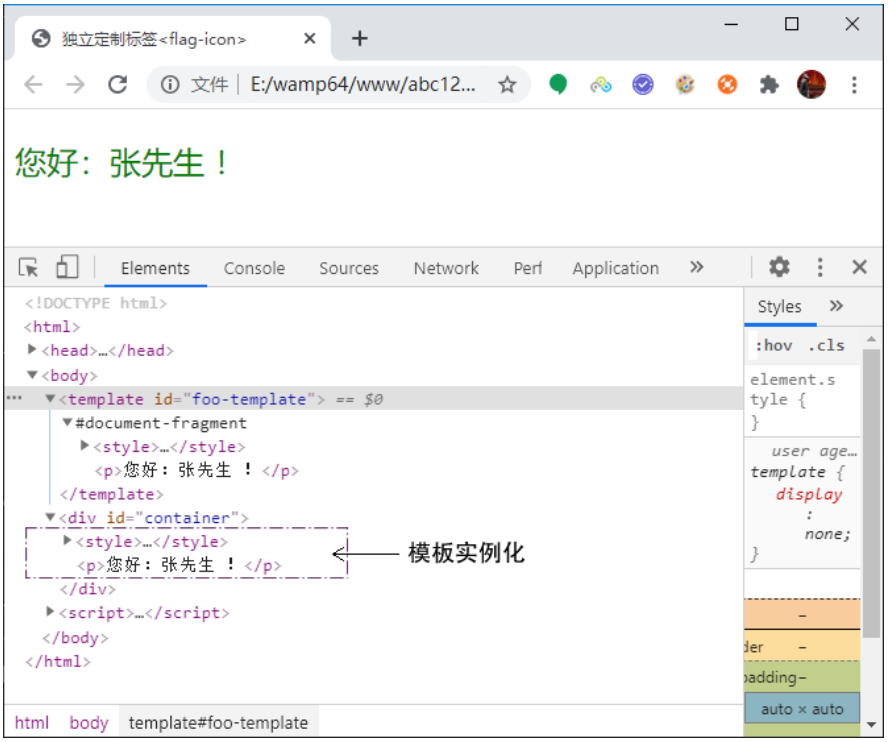
定义模板采用<template>元素，HTML 绝大多数元素都可以作为<template>的后代，其中<slot>元素用来传递参数。<template>元素直接写在网页文件中，浏览器渲染页面时会忽略<template>元素及其后代元素，只有模板的实例才会被渲染出来。先看一个简单的例子：

```
<body>
    <template id="foo-template">
        <style>
            p {
                color: green;
                font-size:x-large;
            }
        </style>
        <p>您好：张先生 ！ </p>
    </template>

    <div id="container"></div>

    <script>
        let ele = document.getElementById('foo-template');
        let box = document.getElementById('container');
        box.appendChild(ele.content.cloneNode(true));
    </script>
</body>
```

ele.content.cloneNode(true)就是复制<template>元素对象的后代，如果带 false 参数调用 cloneNode()方法，则只复制对象本身。对于上面这个例子，在 chrome 开发者工具中看到的效果如图 6.76。



本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

图 6.76 HTML 模板举例

如果每次实例化要求向不同的人打招呼，诸如“您好：王先生 ！”、“您好：李总统 ！”等，这时就要采用<slot>元素来传递参数，不过必须要结合独立定制元素和影子 DOM 才行。首先修改上面的模板，把“ 您好：张先生 ！ ”改为：

```
<p>您好：<slot name='ming'>张先生</slot> ！</p>
```

假设<hello-you>是独立定制元素，那么采用下面的代码就能用“王先生”来实例化模板：

```
   <hello-you>
       <span slot="ming">王先生</span>
   </hello-you>
```

slot 属性的值要等于模板中<slot>元素的属性 name 的值，实例化时浏览器会用“王先生”替换模板中的“<slot name='ming'>张先生</slot>”。如果浏览器不支持<slot>元素，就不能传递参数，这时继续显示“张先生”。也可以这样实例化：

```
   <hello-you>
        <h1 slot="ming">李总统</h1>
   </hello-you>
```

而<hello-you>独立定制元素的 JS 代码如下：

```
class HelloYou extends HTMLElement {
    constructor() {
        super();
        let template = document.getElementById('foo-template');
        const shadowRoot = this.attachShadow({ mode: 'open' });
        shadowRoot.appendChild(template.content.cloneNode(true));
    }
}
customElements.define('hello-you', HelloYou);
```

当然模板中允许出现多个<slot>，这样就能传递多个参数，但是每个<slot>元素的 name 属性的值不能相同，这样在实例化时就能通过元素的 slot 属性建立一一对应关系，参见图 6.77 所示。

```
<template id="contact-card">
  <div>
    <p>姓名：<slot name='name'>XXX</slot></p>
    <p>电话：<slot name='tel'>12345678</slot></p>
    <p>邮件：<slot name='mail'>wlm@veryopen.org</slot></p>
    <p>地址：<slot name='address'>XXYYZZ</slot></p>
  </div>
</template>

<contact-ids>
    <b slot='name'>刘几根</b>
    <b slot='tel'>13670065540</b>
    <b slot='mail'>liujg@gmail.com</b>
    <b slot='address'>秃顶县光明村13号</b>
</contact-ids>
```

图 6.77 传递多参数的模板

实操练习

采用 HTML 模板，结合影子 DOM 和独立定制元素<id-card>，在页面文档中如下的代码就能显示图 6.78。

```
<id-card>
    <img slot='photo' src='idcard.jpg' />
    <span slot='name'>张三</span>
    <small slot='id'>987654321097844</small>
    <i slot="expire">2022-10-01</i>
</id-card>
```



图 6.78 HTML 模板实验

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

6.13 Web Crypto API

针对对称和非对称密钥的管理以及应用，Web 也提供了自己的解决方案——Web Crypto API。目前主流浏览器都支持 SubtleCrypto 对象，它是 Web Crypto API 的具体实现，可以通过 window.crypto.subtle 属性获取 SubtleCrypto 的一个实例。SubtleCrypto 对象用于管理对称和非对称密钥、加/解密、签名以及签名验证。只能在 https 协议下使用此对象。

SubtleCrypto 对象没有属性，只有下面的方法。

1. dist()：创建固定长度并且抗碰撞的摘要。
2. generateKey()：产生一个新密钥（对称密钥）或者一队公/私钥（非对称密钥）。
3. deriveBits()：从现有的密钥导出一定长度的 bit 位。
4. deriveKey()：从既有的密钥派生新的对称密钥。
5. exportKey()：导出密钥。
6. importKey()：导入由 exportKey()导出的密钥。
7. wrapKey()：导出密钥并对之加密处理。
8. unwrapKey()：导入由 wrapKey()方法导出的密钥。
9. sign()：产生数字签名。
10. verify()：验证签名。
11. encrypt()：加密。
12. decrypt()：解密。

其中方法 1 产生摘要，它与密钥无关，方法 2~8 用于管理密钥，其他的方法是使用密码。

6.13.1 产生摘要

方法 digest()对数据产生固定长度的摘要，摘要一般用于签名，本质上就是做哈希运算，输入的数据不同，产生的摘要完全不同。

digest()方法的使用语法如下：

```
const digest = crypto.subtle.digest(algorithm, data);
```

参数 algorithm 定义摘要算法，为字符串类型，可能取值有：'SHA-1'、'SHA-256'、'SHA-384'或'SHA-512'，建议不要用'SHA-1'，不安全。比特币和以太币都采用'SHA-256'算法，产生的摘要固定为 32 个字节。参数 data 为 ArrayBuffer 或 ArrayBufferView 类型，包含了输入的数据。

crypto.subtle.digest()方法返回一个 Promise，成功后确值为一个 ArrayBuffer 对象，里面存储了产生的摘要数据。

例子 1、产生摘要：

```
const text = '世界的领导者也必须是和平的缔造者———泽连斯基';

async function digestMessage(message) {
    const encoder = new TextEncoder();
    const data = encoder.encode(message);
    const hash = await crypto.subtle.digest('SHA-256', data);
    return hash;
}

digestMessage(text).then(digestBuffer => console.log(digestBuffer.byteLength));
```

6.13.2 管理密钥

6.13.2.1 产生密钥

可以调用方法 generateKey()或 deriveKey()产生密钥，前者每次调用会产生新的密钥，后者在相同的“材料”上会派生出相同的对称密钥。

使用 **generateKey()**方法产生新的密钥，它的使用语法如下：

```
const result = crypto.subtle.generateKey(algorithm, extractable, keyUsages);
```

参数 algorithm 为一字典对象，指定密钥生成算法以及提供给算法的参数，算法不同，参数也不同：

- 当采用 ECDSA 或 ECDH 算法时，algorithm 为一 EcKeyGenParams 对象。EcKeyGenParams 对象的属性和取值为{name: 'ECDSA'或'ECDH', namedCurve: 'P-256'或'P-384'或'P-521'}。例如以太坊和比特币使用 ECDSA-Secp256k1 椭圆曲线算法得到私钥和公钥的，因此 algorithm 参数为{name:'ECDSA', namedCurve:'P-256'}。
- 当采用 RSASSA-PKCS1-v1_5、RSA-PSS 或 RSA-OAEP 算法时，algorithm 为一 RsaHashedKeyGenParams 对象。RsaHashedKeyGenParams 对象的属性和取值为{name:'RSASSA-PKCS1-v1_5'或'RSA-PSS'或'RSA-OAEP', modulusLength:<密钥长度，最少 2048>, publicExponent:<公共指数，一般为[0x01, 0x00, 0x01]，即 65537>, hash: 'SHA-256'或'SHA-384'或'SHA-512'}，例如{name:'RSA-PSS', modulusLength: 4096, publicExponent:[0x01, 0x00, 0x01], hash: 'SHA-256'}。
- 当采用 HMAC 算法时，algorithm 为一 HmacKeyGenParams 对象。HmacKeyGenParams 对象的属性和取值为{name:'HMAC', hash: 'SHA-1'或'SHA-256'或'SHA-384'或'SHA-512', length:<可选的，指明密钥长度>}。
- 当采用 AES-CTR、AES-CBC、AES-GCM 和 AES-KW 算法时，algorithm 为一 AesKeyGenParams 对象。AesKeyGenParams 对象的属性和取值为{name:'AES-CBC'或'AES-CTR'或'AES-GCM'或'AES-KW', length: 128 或 192 或 256}

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

其中 RSASSA-PKCS1-v1_5、RSA-PSS、RSA-OAEP、ECDSA 和 ECDH 都是非对称算法，其他的都是对称算法。

参数 extractable 为布尔类型，定义是否允许采用 exportKey()或者 wrapKey()导出密钥。

参数 keyUsages 为一数组，列出创建的密钥能用在何处，允许出现在数组的值有：encrypt（可用于加密）、decrypt（可用于解密）、sign（可用于签名）、verify（可用于验证签名）、deriveKey（可用于导出另一个密钥）、deriveBits（可用于导出比特位）、wrapKey（可用于加密另一个密钥）、unwrapKey（可用于解密另一个密码）。例如["encrypt","decrypt"]。

crypto.subtle.generateKey()方法返回一个 Promise，成功时确值为一个 CryptoKey 对象（对称算法）或者 CryptoKeyPair 对象（非对称算法）。

CryptoKey 对象有四个属性：

- type：为字符串，指明本对象的类型，可能的值有"secret"（可用于对称算法的密钥）、"private"（非对称算法产生的 CryptoKeyPair 对象的私钥部分）、"public"（非对称算法产生的 CryptoKeyPair 对象的公钥部分）。
- extractable：为布尔类型，定义是否允许采用 exportKey()或者 wrapKey()导出密钥。true 允许导出。
- algorithm：为一个描述算法的对象实例，可能的对象有 EcKeyGenParams、RsaHashedKeyGenParams、HmacKeyGenParams 和 AesKeyGenParams，这几个对象上面已经介绍过。
- usages：介绍参见 generateKey()方法的 KeyUsages 参数。

CryptoKeyPair 对象有两个属性：

- privateKey：一个代表私钥的 CryptoKey 对象实例。
- publicKey：一个代表公钥的 CryptoKey 对象实例。

举例 2、产生非对称公/私钥：

```
crypto.subtle.generateKey(
    {
        name: "ECDSA",
        namedCurve: "P-256"
    },
    true,
    ["sign", "verify"]
).then(rst => console.log(rst));
```

举例 3、产生对称密钥：

```
crypto.subtle.generateKey(
    {
      name: "AES-CTR",
```

```
    length: 256
  },
  true,
  ["encrypt","decrypt"]
).then(rst => console.log(rst));
```

调用 **deriveKey()** 方法派生对称密钥，它的语法如下：

```
const result = crypto.subtle.deriveKey(
  algorithm,
  baseKey,
  derivedKeyAlgorithm,
  extractable,
  keyUsages
);
```

参数 algorithm 定义派生算法相应的对象实例：

- 当使用 ECDH 导出算法（允许两个拥有公/私钥的人导出一个共享对称密钥）时，algorithm 参数为一 EcdhKeyDeriveParams 对象实例，EcdhKeyDeriveParams 有两个属性：name 属性的值为 "ECDH"，public 属性的值为对方的公钥（CryptoKey 对象）。也就是用我的私钥和对方的公司派生一个共享对称密钥。
- 当使用 HKDF 导出算法（从高熵输入[如 ECDH 秘密]派生）时，algorithm 参数为一 HkdfParams 对象实例，此对象有四个属性：name 属性的值为"HKDF"；hash 属性指定摘要算法，可能取值为"SHA-1"、"SHA-256"、"SHA-384"或"SHA-512"；salt 属性为 BufferSource 类型，添加“盐”，一般是与摘要算法输出的长度相同的随机数；info 属性为 BufferSource 类型，表示上下文信息，一般为空字符串。
- 当使用 PBKDF2 导出算法（从低熵输入[如密码]派生）时，algorithm 参数为一 Pbkdf2Params 对象实例，此对象有四个属性：name 属性的值为"PBKDF2"；hash 属性指定摘要算法，可能取值为"SHA-1"（尽量不要用）、"SHA-256"、"SHA-384"或"SHA-512"；salt 属性为 BufferSource 类型，添加“盐”，至少为 16 个字节的随机数；iterations 属性指定重复 hash 计算的次数，次数越多越难以暴力破解。

参数 baseKey 为 CryptoKey 对象类型，代表派生算法的输入，当使用 ECDH 派生算法时，baseKey 为 ECDH 私钥（即我的私钥）。

参数 derivedKeyAlgorithm 定义派生密钥的用途，它是：HmacKeyGenParams 对象（用于 HMAC）、AesKeyGenParams 对象（用于 AES-CTR、AES-CBC、AES-GCM 或 AES-KW）。

参数 extractable 和 keyUsages 意义与 generateKey()方法相同，前面已经做过介绍。

crypto.subtle.deriveKey()方法返回一个 Promise，成功时确值为一个 CryptoKey 对象。

例子 4、下面的例子摘自 https://github.com/mdn/dom-examples/blob/master/web-crypto/derive-key/ecdh.js

```
/*
派生 AES 密钥，参数：
- 我的 ECDH 私钥
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
   - 对方的 ECDH 公钥
   */
   function deriveSecretKey(privateKey, publicKey) {
      return window.crypto.subtle.deriveKey(
         {
            name: "ECDH",
            public: publicKey
         },
         privateKey,
         {
            name: "AES-GCM",
            length: 256
         },
         false,
         ["encrypt", "decrypt"]
      );
   }

   async function agreeSharedSecretKey() {
      // 新产生 Alice 的非对称密钥对:
      let alicesKeyPair = await window.crypto.subtle.generateKey(
         {
            name: "ECDH",
            namedCurve: "P-256"
         },
         false,
         ["deriveKey"]
      );

      // 新产生 Bob 的非对称密钥对:
      let bobsKeyPair = await window.crypto.subtle.generateKey(
         {
            name: "ECDH",
            namedCurve: "P-256"
         },
         false,
         ["deriveKey"]
      );

      // Alice 利用自己的私钥和 Bob 的公钥派生一个对称密钥:
      let alicesSecretKey = await deriveSecretKey(alicesKeyPair.privateKey, bobsKeyPair.publicKey);

      //  Bob 利用自己的私钥和 Alice 的公钥派生一个相同的对称密钥，由于各自能派生相同的对称密钥，所以杜绝了传递密钥过程中
泄密的可能性:
      let bobsSecretKey = await deriveSecretKey(bobsKeyPair.privateKey, alicesKeyPair.publicKey);

      // Alice 用派生的密钥加密数据并发给 Bob:
      let encryptButton = document.querySelector("#ali");
      encryptButton.addEventListener("click", () => {
         encrypt(alicesSecretKey);
      });

      // Bob 用派生的密钥解密收到的数据:
      let decryptButton = document.querySelector(".ecdh .decrypt-button");
      decryptButton.addEventListener("click", () => {
         decrypt(bobsSecretKey);
```

```
    });
  }
```

调用 **deriveBits()**方法从现有的密钥派生某种长度的比特位（作为共享对称密钥），它的语法如下：

```
const result = crypto.subtle.deriveBits(
   algorithm,
   baseKey,
   length
);
```

参数 algorithm 和 baseKey 与 deriveKey()方法的同名参数意思相同，参数 length 定义导出的比特位数，为了兼容更多的浏览器，最好是 8 的倍数（即多少个字节的二进制数）。

crypto.subtle.deriveBits()方法返回一个 Promise，成功时确值为一个 ArrayBuffer 对象，里面包含了导出的比特位。

6.13.2.2 导入/导出密钥

采用 **exportKey()**方法导出没有被加密的密钥，它的使用语法如下：

```
const result = crypto.subtle.exportKey(format, key);
```

参数 format 定义导出的格式，可能的取值为"raw"（原始格式）、"pkcs8"（PKCS #8 格式）、"spki"（SubjectPublicKeyInfo 格式）或"jwk"（JSON Web Key 格式）。

参数 key 为被导出的密钥（CryptoKey 对象实例）。

本方法返回一个 Promise，成功后确值为：(1)当 format 参数为"jwk"时，JSON 对象；或者(2)其他情况，为 ArrayBuffer。

例子 5、导出 JSON 格式：

```
   function ab2str(buf) {
      return String.fromCharCode.apply(null, new Uint8Array(buf));
   }

   crypto.subtle.generateKey(
      {
         name: "ECDSA",
         namedCurve: "P-256"
      },
      true,
      ["sign", "verify"]
   ).then(rst => exportCryptoKey(rst.privateKey));

   async function exportCryptoKey(key) {
      const exported = await window.crypto.subtle.exportKey(
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

```
        "jwk",
        key
    );
    console.log(JSON.stringify(exported, null, " "));
}
```

采用 **importKey()**方法导入密钥，它的使用语法如下：

```
const result = crypto.subtle.importKey(
    format,
    keyData,
    algorithm,
    extractable,
    keyUsages
);
```

参数 format 与 exportKey()方法的同名参数意义相同，参数 extractable 和 keyUsages 与 generateKey()的同名参数意义一样。参数 keyData 指定被导入的密钥，类型可为 ArrayBuffer、TypedArray、DataView 或 JSONWebKey 对象。参数 algorithm 为一字典对象，定义被导入密钥的算法和算法用到的参数，可能取值如下：

- 当使用 RSASSA-PKCS1-v1_5、RSA-PSS 或 RSA-OAEP 算法时，algorithm 参数为 RsaHashedImportParams 对象，此对象有两个属性：{name: "RSASSA-PKCS1-v1_5"或"RSA-PSS"或"RSA-OAEP"，hash: "SHA-1"或"SHA-256"或"SHA-384"或"SHA-512"}，不过尽量不要使用"SHA-1"。
- 当使用 ECDSA 或 ECDH 算法时，algorithm 参数为 EcKeyImportParams 对象，此对象有两个属性：{name: "ECDSA"或"ECDH"，hash: "P-256"或"P-384"或"P-521"}。
- 当使用 HMAC 算法时，algorithm 参数为 HmacImportParams 对象，此对象有两个属性：{name: "HMAC"，hash: "SHA-1"或"SHA-256"或"SHA-384"或"SHA-512"，length:<可选的，指明密钥长度>}。
- 当使用 AES-CTR、AES-CBC、AES-GCM 或 AES-KW 算法时，algorithm 参数为{name: "AES-CTR"或"AES-CBC"或"AES-GCM"或"AES-KW"}。
- 当使用 PBKDF2 算法时，algorithm 参数为"PBKDF2"。
- 当使用 HKDF 算法时，algorithm 参数为"HKDF"。

crypto.subtle.importKey()方法返回一个 Promise，成功时确值为一个 CryptoKey 对象。

例子 6、原始密钥导入：

```
const rawKey = window.crypto.getRandomValues(new Uint8Array(32));
function importSecretKey(rawKey) {
    return window.crypto.subtle.importKey(
        "raw",
        rawKey,
        "AES-GCM",
```

```
            true,
            ["encrypt", "decrypt"]
        );
    }
```

采用 **wrapKey()** 方法导出密钥并加密存储，出于安全起见。它的使用语法如下：

```
const result = crypto.subtle.wrapKey(
    format,
    key,
    wrappingKey,
    wrapAlgo
);
```

参数 format 和 key 参见 exportKey()方法的同名参数，参数 wrappingKey 指明用来加密的密钥（产生时必须声明是"wrapKey"用途，参见 generateKey()方法的 keyUsages 参数），类型为 CryptoKey。参数 wrapAlgo 定义加密算法和加密算法用到的参数，可能的取值如下：

- 当使用 RSA-OAEP 算法时，wrapAlgo 参数为 RsaOaepParams 对象，此对象为{name: " RSA-OAEP", label:<字节数组>}，其中 label 是可选的，是绑定到密文的字节数组。除非您的应用程序需要标签，否则您可以省略此参数，它不会影响加密操作的安全性。
- 当使用 AES-CTR 算法时，wrapAlgo 参数为 AesCtrParams 对象，此对象为{name: "AES-CTR", counter:<16 字节的数组>， length:<counter 右侧多少位作为计数器>}，counter 右侧 length 位作为计数器，剩下的作为 nonce，例如 length 为 64，那么 counter 剩下的 64 位二进制就作为 nonce。
- 当采用 AES-CBC 算法时，wrapAlgo 参数为 AesCbcParams 对象，此对象为{name: "AES-CBC", iv:<16 个字节的随机数数组>}。
- 当采用 AES-GCM 算法时，wrapAlgo 参数为 AesGcmParams 对象，此对象为{ name: "AES-GCM", iv:<16 个字节的随机数数组>[， additionalData:<字节数组，总的二进制位数要小于 2^64-1>, tagLength: 96 或 104 或 112 或 120 或 128]}。
- 当采用 AES-KW 算法时，wrapAlgo 参数为"AES-KW"或者{name:"AES-KW"}。

crypto.subtle.wrapKey()方法返回一个 Promise，成功时确值为一个 ArrayBuffer 对象（里面包含了加了密的导出密钥）。

采用 unwrapKey()方法导入由 wrapKey()导出的密钥，它的语法如下：

```
const result = crypto.subtle.unwrapKey(
    format,
    wrappedKey,
    unwrappingKey,
    unwrapAlgo,
    unwrappedKeyAlgo,
    extractable,
    keyUsages
);
```

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

参数 format、extractable 和 keyUsages 请参考 importKey()方法的同名参数，参数 unwrapAlgo 与 wrapKey()方法的 wrapAlgo 参数意义相同；wrappedKey 参数为存储了加密密钥的 ArrayBuffer 对象；参数 unwrappingKey 为 CryptoKey 对象，里面存储了用来解密的那个密钥（必须具备“unwrapKey”功能）；参数 unwrappedKeyAlgo 与 importKey()方法的 algorithm 参数大同小异，只不过没有 PBKDF2 和 HKDF 算法而已。

crypto.subtle.unwrapKey ()方法返回一个 Promise，成功时确值为一个 CryptoKey 对象。

6.13.3 使用密钥

6.13.3.1 签名/验证

使用 **sign()**方法对数据签名，它的使用语法如下：

```
const signature = crypto.subtle.sign(algorithm, key, data);
```

参数 algorithm 指名签名算法及其参数，签名算法不同，algorithm 的取值也不同：

- 当使用 RSASSA-PKCS1-v1_5 算法时,algorithm 参数为{ "name": "RSASSA-PKCS1-v1_5" }或者"RSASSA-PKCS1-v1_5"。
- 当使用 RSA-PSS 算法时，algorithm 参数为 RsaPssParams 对象。
- 当使用 ECDSA 算法时，algorithm 参数为 EcdsaParams 对象。
- 当使用 HMAC 算法时，algorithm 参数为"HMAC 或者{ "name": "HMAC" }。

参数 key 为一 CryptoKey 对象，定义了签名的密钥，对于非对称算法，key 为私钥；参数 data 为被签名的数据，类型为 ArrayBuffer 或 ArrayBufferView。

crypto.subtle.sign()方法返回一个 Promise，成功后确值为一个 ArrayBuffer 对象，里面存储了签名后的数据。

例子 7、签名：

```
   const messageBox = document.getElementById("inputMessage");
   let message = messageBox.value;
   let enc = new TextEncoder();
   let encoded = enc.encode(message);

   let signature = await window.crypto.subtle.sign(
      {
         name: "ECDSA",
         hash: { name: "SHA-256" },
      },
      privateKey,
      encoded
```

```
    );
```

使用 **verify()**方法对数据签名，它的使用语法如下：

```
const result = crypto.subtle.verify(algorithm, key, signature, data);
```

参数 algorithm、key 和 data 与 sign()方法的同名参数意义一样，但是对于非对称算法，key 为公钥；参数 signature 是签名后的数据，为 ArrayBuffer 类型。

crypto.subtle.verify()方法返回一个 Promise，成功后确值为一个布尔值：验证成功为 true，验证失败为 false。

例子 8、验证：

```
    function getMessageEncoding() {
        const messageBox = document.querySelector(".ecdsa #message");
        let message = messageBox.value;
        let enc = new TextEncoder();
        return enc.encode(message);
    }

    async function verifyMessage(publicKey) {
        const signatureValue = document.querySelector(".ecdsa .signature-value");
        signatureValue.classList.remove("valid", "invalid");

        let encoded = getMessageEncoding();
        let result = await window.crypto.subtle.verify(
            {
                name: "ECDSA",
                hash: { name: "SHA-384" },
            },
            publicKey,
            signature,
            encoded
        );

        signatureValue.classList.add(result ? "valid" : "invalid");
    }
```

6.13.3.2 加密/解密

使用 **encrypt()**方法对数据加密，它的使用语法如下：

```
const result = crypto.subtle.encrypt(algorithm, key, data);
```

参数 algorithm 参考 wrapKey()的参数 wrapAlgo，参数 key 为一 CryptoKey 对象，定义了用来加密的密钥，对于非对称算法，key 为公钥；参数 data 为被签名的数据，类型为 BufferSource。

crypto.subtle.encrypt()方法返回一个 Promise，成功后确值为一个 ArrayBuffer 对象，里面存储了加密后的数据。

本书作者拥有本书中包含的文字和图片以及随书配套资源的全部版权。如果需要配套资源请联系 183407563@qq.com 或微信 moodisk

使用 **decrypt()**方法对数据加密，它的使用语法如下：

```
const result = crypto.subtle.decrypt(algorithm, key, data);
```

参数与 ecnrypt()方法的同名参数意义相同，只不过对于非对称解密算法（如 RSA-OAEP），key 为私钥。

crypto.subtle.decrypt()方法返回一个 Promise，成功后确值为一个 ArrayBuffer 对象，里面存储了解密后的数据。