

前 言 →

最近几年，HTML 5、Node.js 等新生技术给 JavaScript 带来了质的飞跃。很多对前端感兴趣或刚入门不久的朋友都在问笔者同一个问题，如何才能学好 JavaScript？笔者给出的答案是实践、实践、再实践。本书的几百段代码，全是笔者的原创，非常具有借鉴、学习、实践价值。学习编程就是学习一种思考方式，是学习看透事物本质的一种锻炼，只有不断学习代码才能知道怎样写出更好的代码。

JavaScript 中的那些槛儿

有些前端开发人员做了多年 JavaScript 开发，依然在面对问题时束手无策。这些问题很常见，你又能了解多少呢？

- 如何实现 JavaScript 的闭包？
- 为什么事件有冒泡捕获？
- 浏览器之间有何差异？
- 何为 DOM，何为 BOM，两者之间有何区别？
- JavaScript 里面的对象和 Java 的对象一样吗？
- XHTML 与 HTML 的区别是什么？
- JavaScript Function 的原理是什么？
- 浏览器都有哪些 JavaScript 引擎？
- 是否认识！、||、三元操作符、单链式写法、声明变量的缩略写法、惰性载入？
- 如何判断浏览器是否支持 HTML 5？
- 如何开发出跨设备、兼容性的代码？
- 如何在不同浏览器中调试代码？

以上所有内容在本书的代码中都有讲解，除这些常见的 JavaScript 门槛外，目前页面中常见的一些需求，在本书中都能够找到与之匹配的代码实例或解决方案。

如何学习 JavaScript

请记住以下 3 个词，只要您坚持下去，写 JavaScript 代码就是这么简单。

- **借鉴：**多阅读、多分析、多学习一些成功的 JavaScript 代码，多逛一些 JavaScript 类的技术社区、论坛、博客等，多与一些技术牛人交流学习，借鉴其经验、思路等。
- **思考：**思考的深度决定了代码质量，因此多思考、多问几个为什么会在很大程度上提高代码的质量。

第1章 学习 JavaScript 必须知道的事儿 →

从目前 JavaScript 的发展趋势来看，它确实是一门令人怦然心动的语言，但是要想学会并熟练运用这门语言，可不是一日之功。用什么开发工具？怎样执行？如何调试？这都是读者必须知道的 JavaScript 基础，本章将讲解与此相关的内容。

本章主要涉及的知识点有：

- 调用 JavaScript 代码：JavaScript 标签特性、JavaScript 的引入方式、JavaScript 的引入位置。
- JavaScript 的开发工具：目前主流 JavaScript IDE 的优缺点。
- 调试 JavaScript 代码：使用 WebInspector 和 Firebug 调试 JavaScript 的方法。

1.1 如何在 HTML 中调用 JavaScript 代码

JavaScript 刚推出时，Netscape 就面临一个亟待解决的问题：怎样在 Web 核心语言 HTML 中加入 JavaScript，可以呈现其效果而又不影响浏览器页面本身。为此，一个新的成员出现了——`<script>`，该标签用于定义客户端脚本语言。

1.1.1 `<script>`标签的定义

HTML 4.01 为`<script>`定义了 5 个属性。

- `type`：必需，表示脚本语言的 MIME 类型。可以理解为 `language` 替代属性。MIME 类型由两部分组成：媒介类型/子类型，在 JavaScript 中使用“`text/JavaScript`”，在非 IE 中，还可以使用“`application/JavaScript`”。为了保证最大兼容性，建议使用“`text/JavaScript`”。
- `language`：不赞成使用，用来规定脚本语言，目前受到非议。
- `charset`：可选，规定了引用外部文件的字符编码格式。如果外部文件与主文件中的编码不同，就会用到这个属性，默认字符编码是 ISO-8859-1。一般的浏览器会忽略这个属性，所以大多数开发人员不使用这个属性。
- `src`：可选，规定被包含的外部 URL 文件。
- `defer`：可选，规定脚本是否延迟到文档被完全载入或显示后再执行。

1.1.2 两种嵌入 JavaScript 代码的方式

使用<script>在网页中嵌入 JavaScript 代码有两种方式。

(1) 嵌入式，即直接在页面中包含 JavaScript 代码。

```
<script type='text/JavaScript'>
    alert("我是 JavaScript 代码");
</script>
```

(2) 外链式，即包含外部文件，通过定义 src 属性的 URL 引入文件。值得注意的是，src 还可以引用外部域的文件。这个强大的特性，让人又爱又恨，备受争议。从架构的角度看，外部引用比较受欢迎——易维护、能缓存、适应未来的发展，如下：

```
<script type='text/JavaScript' src="demo.JavaScript"></script>
```

按照惯例，所有的<script>代码，都应该放在<head>中。但是这样就带来了另一个问题，只有等所有的脚本加载完毕，才能够呈现页面的内容（浏览器遇到<body>才会呈现内容）。从用户体验的角度来看，如果页面有多个脚本需要加载，就会出现让用户以为页面没内容或等待一会儿才展示内容的问题，如下：

```
<html>
    <head>
        <title>html 例子</title>
        <script type='text/JavaScript' src='demo1.JavaScript'></script>
        <script type='text/JavaScript' src='demo2.JavaScript'></script>
    </head>
    <body>
        <!--待渲染的内容-->
    </body>
</html>
```

为了解决上述问题，在现代的浏览器中，我们一般将外部脚本引用放在<body>元素最后面，或者增加 defer 属性（脚本延迟加载），但是 defer 属性在一些浏览器中并不支持。

1.1.3 XHTML 与 HTML 对 JavaScript 解析的不同之处

XHTML，即可扩展超文本标记语言，编写规范要比 HTML 严格得多。例如以下代码就不能被 XHTML 解析：

```
<script type='text/JavaScript'>
    function demoJavaScript(x,y){
        if(x<y){
            alert("x 小于 y");
        }
    }
</script>
```

在 HTML 中，JavaScript 的一些特殊规则可以正确解析，但 XHTML 并不识别，按照 XHTML 的解析规则，“<”号被认为是开始标记，后边不能跟空格。

解决方案之一就是将“<”替换为“<”，但这只是解决燃眉之急的方法。另一种方

案就是利用特殊注解的方案，“//<![CDATA[”与“//]]>”的组合，如下：

```
<script type='text/JavaScript'>
//<![CDATA[
    function demoJavaScript(x,y){
        if(x<y){
            alert("x 小于 y");
        }
    }
//]]>
</script>
```

1.1.4 <noscript>如何用

谈到<script>，就不得不提及它的同胞兄弟<noscript>。<noscript>在浏览器不支持或禁用客户端脚本时很有用，如下：

```
<html>
    <head>
        <title>html 例子</title>
    </head>
    <body>
        <!--待渲染的内容-->
        <script type='text/JavaScript' src='demo1.JavaScript'></script>
        <script type='text/JavaScript' src='demo2.JavaScript'></script>
        <noscript>
            <p>此页面不支持（禁用）JavaScript，请更换浏览器或启用对脚本的支持。</p>
        </noscript>
    </body>
</html>
```

1.2 使用什么工具开发 JavaScript

目前流行的 JavaScript 开发工具有：Adobe Dreamweaver、SublimeText、AptanaStudio、WebStorm 等。下面分析一下这几款工具的优劣，具体选择哪种开发工具，请读者自行决定。

1.2.1 Adobe Dreamweaver 软件，推荐指数：3

Adobe Dreamweaver 由美国的 Macromedia 公司开发，是一套针对网页设计师打造的视觉集成开发工具，可用于构建网页和移动应用程序，目前版本是 Adobe Dreamweaver CS6。

【优点】

- 最快的开发效率。
- 利用其多屏幕预览面板的实时特性，快速检查工程在不同环境下呈现的效果。

- 强大的集成编码功能及代码导航器功能可以快速构建代码、部署项目。
- 项目的整体管理及控制。
- 实现了对 HTML、CSS、JavaScript 的智能提示，全方位地呈现设计与开发的完美协作。

【缺点】

- 难以把控代码。
- 效果不易统一。
- 由于代码的效果呈现是软件生成的，在一些严格要求精确效果的标准下，就不太符合要求。
- 成本高，不开源。
- 商业行为，决定了产品的生命周期，可能会衰败。

Adobe Dreamweaver 官网：<http://www.adobe.com/cn/products/dreamweaver.html>。

1.2.2 SublimeText，推荐指数：4

SublimeText 是一款非常不错的商业代码编辑器，是基于 Python 的跨平台文字编辑器，也是一款将类 VIM 编辑器经过扩充、增强、改良的多功能软件，相对 VIM 的学习成本要低。SublimeText 在软件工程师的美誉下被称为“神器”。

【优点】

- 强大的代码编辑支持。
- 简洁、性感的界面，令人眼前一亮。
- 小地图全文件预览特性，可以在代码文件中自由定位位置。
- 多种界面布局、全局免打扰模式，更具人性化的体验。
- 代码提醒、高亮、补全、折叠功能。
- 跨平台，可以在 Linux、Mac OS、Windows 平台上正常工作。
- 速度快，相对于其他拥有强大功能的编辑器，性能十分优越，一般不会造成假死、延迟现象。

【缺点】

- 中文支持不太好，GBK 支持得不太好，默认支持 UTF8 编码。
- 成本高，不开源。
- 商业行为，决定了产品的生命周期，可能会衰败。

SublimeText 官网：<http://www.sublimetext.com/>。

1.2.3 AptanaStudio，推荐指数：4

AptanaStudio 是基于 Eclipse 开发的集成式 Web 开发环境，也可以作为 Eclipse 的插件。它的最大特点就是对 JavaScript 编辑与调试的超强支持，这也是笔者推荐它的原因。

【优点】

- 代码编辑智能提示、自动补全功能、错误提示功能。
- 浏览器兼容性提示功能，方便开发人员跨浏览器开发。
- 类似 DOM 的文档树结构，帮助开发人员查看及分析文档结构。
- 融合了 JavaScript 调试器、PHP 开发环境。

【缺点】

- 对 HTML+CSS 支持有限。
- 默认不支持 GBK 编码。

AptanaStudio 官网：<http://www.aptana.com/>。

1.2.4 WebStorm，推荐指数：4.5

WebStorm 是 JetBrains 公司开发的一款 JavaScript 开发工具。在中国，被 JavaScript 开发者誉为“Web 前端开发神器”、“最强悍的 JavaScript IDE”等。它在 IntelliJIDEA 原有的超强 JavaScript 功能基础上，扩展、强化了很多功能。

【优点】

- 智慧型编辑，为代码开发人员提供极限装备，无论是代码补全、代码检测、还是批量代码分析、代码重构等功能，都游刃有余。
- 支持 ECMAScript、支持 CoffeeScript、支持节点。
- 代码跟踪、联想查询也是亮点。
- 最新版本加入了对 HTML 5 的支持，在未来竞争中，更具魅力。
- 代码质量分析，数百种特定语言代码检测工具，可以为代码提供质量检验并高亮提醒。
- 跨平台体验，在 Windows、Mac OS 或 Linux 平台上都可使用。

【缺点】

- 商业弱点。
- 成本高，有产品生命周期。
- 不支持可视化与代码之间的转换。
- 体系庞大、复杂。

WebStorm 官网：<http://www.jetbrains.com/webstorm/>。

1.3 如何在不同浏览器中调试 JavaScript 代码

估计大多数程序员在浏览器中调试 JavaScript 都是使用 `alert()`，这是最原始的方案，现在早就过时了。目前市面上流行的大多数 IDE 及编辑器都带有调试功能和元素查看器，在一些强大的 IDE 中，还有网络查看器、性能分析工具、资源工具等，你是否已经心动了？为了教会读者调试 JavaScript 代码，笔者特地选择两种最具代表性的调试工具 WebInspector

与 Firebug，只要这两种学会了，其他的 JavaScript 调试器都是大同小异。

1.3.1 WebInspector 调试工具

当今最流行的浏览器 Chrome 和 Safari，在调试时使用的都是 WebInspector，当然还有其他浏览器也使用 WebInspector。这些浏览器的调试界面外观稍有不同，本质还是一样的，如图 1-1 所示。

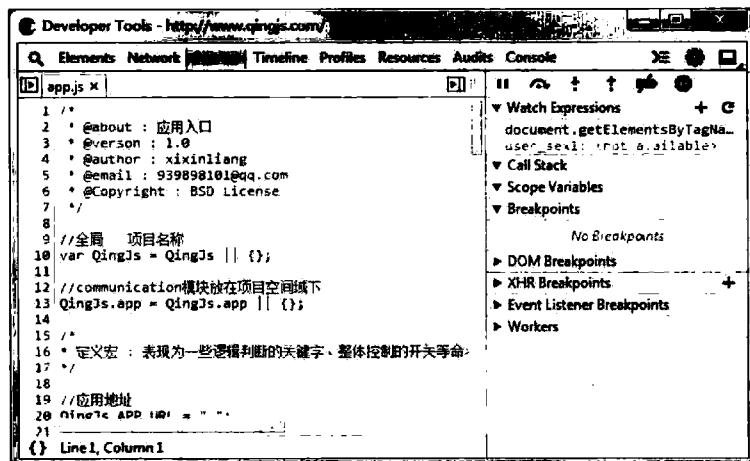


图 1-1 启用 Chrome 的 JavaScript 调试器界面

Chrome 开发工具可以按 F12 键打开，而 Safari 中的激活方式是在右上角设置里面选择“Preferences...”，然后在“Advanced”面板中，选择“ShowDevelopmenuinmenubar”启动调试器，如图 1-2 所示。



图 1-2 启用 Safari 调试工具

1.3.2 Firebug 调试工具

Firebug 是 Firefox 浏览器中的调试工具。只要我们在 Firefox 中安装了 Firebug 应用，就可以按 F12 键或右击鼠标开启调试，开启后的效果如图 1-3 所示。

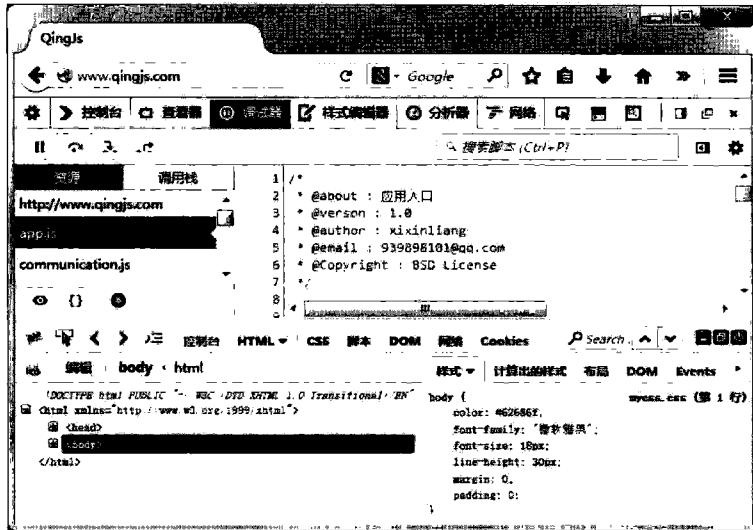


图 1-3 启用 Firefox 中的 Firebug 调试工具

无论是 WebInspector 还是 Firebug，大体功能都是类似的，下面简要概括一下：

- 元素查看、监控
- 源文件及所需资源
- HTTP 网络
- 脚本文件查看及调试
- 性能分析
- 代码和内存统计

另外，除了 WebInspector 和 Firebug 外，FirebugLite 也提供了强大的调试支持，尤其在 IE 中，调试非常方便。它不需要安装，只要引用一段 JavaScript 代码即可，如下：

```
<script type="text/JavaScript" src="https://getfirebug.com/firebug-lite.JavaScript"></script>
```

1.3.3 JavaScript 调试器

开发的过程中，调试器是必不可少的，也是最重要的。可以通过调试器来设置断点、监控变量、查看堆栈等。

设置断点有两种方法：

- (1) 单击 Sources 选项，在想要设置的行单击一下即可，如图 1-4 所示。
- (2) 通过在代码中添加 debugger 来设置断点，如下：

```
function debuggerTest(){
    //代码略
}
```

debugger
}

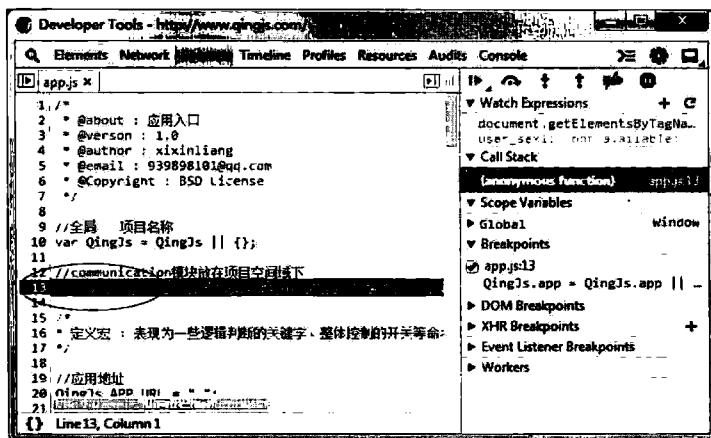


图 1-4 设置断点行

两种方法相比较，第一种方法更妥当，不会干扰代码环境。当代码执行到断点行时，就可以采取一定的操作，比如在调试面板的右上角执行播放、下一步、进入、跳出。

在调试器中，通过查看调用的堆栈 CallStack，可以看到函数的执行过程，如图 1-5 所示。

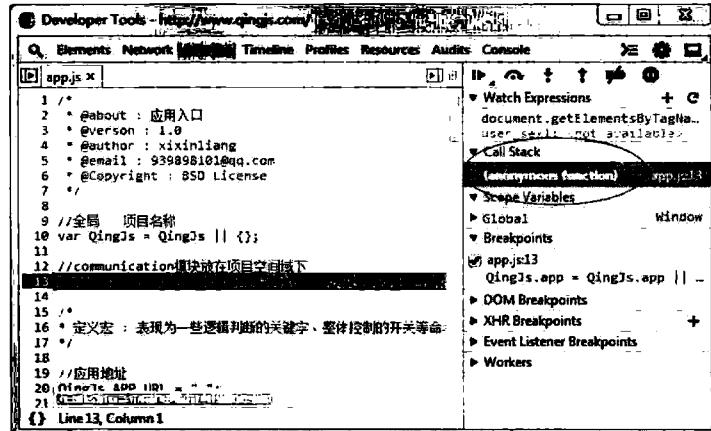


图 1-5 Chrome 右边的断点调试区域

1.3.4 控制台

控制台可以查看变量的值，也可以执行 JavaScript 代码。只要异步调用 `console.log()`，就可以输出想要的日志，如下：

```
console.log("我是日志");
console.log(3,4,{1:"one"});
```

`console.warn()` 和 `console.error()` 是警示级别，如下：

```
console.warn("一个警告");
console.error("一个错误");
```

```
//也可以这样写
try{
    //或许会报错的代码块
}catch(e){
    console.error("错误",e);
}
```

控制台还有如下一些其他比较有用的函数。

- 堆栈函数 `console.trace()`, 可以查看指定函数的调用关系。
 - `clear()`函数, 用来清除控制台中的 `log`。
 - `dir()`函数, 输出对象中的所有属性, 如:
- ```
dir({test:1,test2:2});
```
- `values()`函数, 以数组的形式打印出对象中的所有属性值。
  - `keys()`与 `values()`是一对, 会以数组方式打印对象中所有键(名字), 如图 1-6 所示。

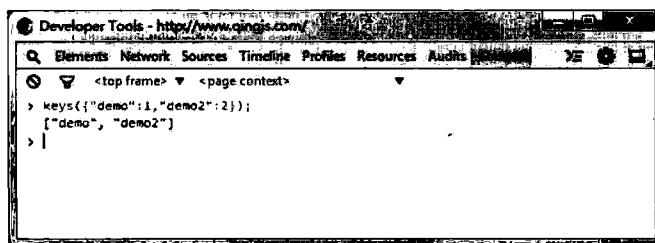


图 1-6 用 `keys()` 打印一个对象

注意: `console.log()`不是任何浏览器都支持, 如 IE 的低版本就不支持。

### 1.3.5 HTTP 分析

如果想知道 Web 在执行什么网络请求, 可以通过调试工具来查看, 包括网络请求的时间、请求的方式、地址等, 蓝色代表 `DOMContentLoaded` 触发的时间, 即 DOM 加载完成的时间。橙色(红色)代表 `load` 事件触发的时间。另外还有一条绿线, 是页面首次渲染的线, 在 Firebug 与 WebInspector 中看不见, 可以使用其他更底层的工具进行捕获, 如图 1-7 所示。

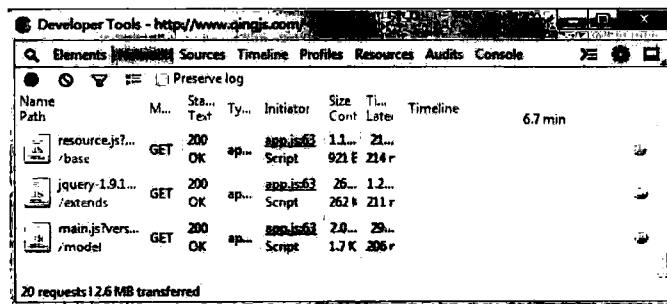


图 1-7 Chrome Network 面板

当单击某个请求时, 会看到请求的详细信息, 如图 1-8 所示。

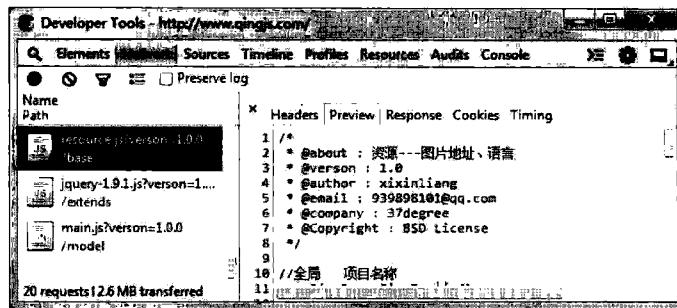


图 1-8 Chrome 中一条 Network 的详细信息

### 1.3.6 性能检测

大型的项目对性能的要求是很严格的，尤其是面对移动终端设备时。在调试工具中，Profile 可以精确地检测程序的性能。写法很简单，只要在想统计的代码外层添加 profile 代码即可，如下：

```
console.profile();
//要统计的代码
//.....
console.profileEnd();
```

当浏览器遇到 profileEnd() 时，就会将统计结果生成报表显示出来，或者在浏览器中使用 Profile 的 record 特性查看，如图 1-9 所示。

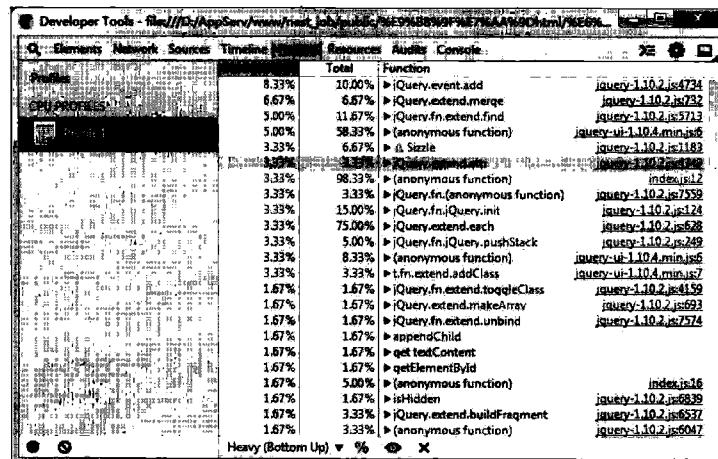


图 1-9 Chrome 的一条 Profile 信息

使用控制台函数 console.time() 与 console.timeEnd()，也可以实现同样的效果，当执行到 console.timeEnd() 时，后台把程序的执行时间（以毫秒为单位）发送到控制台，使用控制台的 API，将结果加入到测试代码中，就可以在整体上把控代码的性能，如下：

```
console.time("times");
//代码段落 ...
console.timeEnd("timesEnd");
```

## 1.4 总结

学完本章，在某种程度上说，你已经掌握了 JavaScript 的一些实用技巧，下面来梳理一下本章的内容：

- 在 HTML 页面中引入 JavaScript 有两种方案——嵌入式与外链式，一般选择第二种方案引入 JavaScript 代码。为了增加用户体验，还可以通过设置 defer 属性或在标签尾部添加脚本的方式来实现延迟加载。由于浏览器的支持程度不一，选择外链式引入 JavaScript 会比较友好。
- 介绍了一些现在比较流行的 JavaScript IDE，都很强大，各有千秋。
- JavaScript 调试一直是一个比较重要、有趣的课题，现在各大主流浏览器对前端 JavaScript 调试支持得比较完美。本章介绍了 WebInspector 与 Firebug 的基本调试方法，其他的方法读者可以深入研究。

# 第 2 章 表单常用代码

表单是前端程序员经常接触的元素，也是用户在网页上看到的基本内容。Web 应用中的大部分数据都是通过表单的方式进行收集的，因此表单的处理非常重要。一般的 JavaScript 程序只是实现一些简单的表单操作技术——获取表单的值、隐藏表单、修改样式等。其实，表单的操作技术不仅仅是这些，还有很多其他有趣的地方。

本章主要涉及的知识点如下。

- 实现一些常见的表单文本区域操控技术：禁止输入、字符限制、去除空格等。
- 表单域和文件结合：文件上传、文件限制、清除上传文件等。
- 字符串操作：小写转大写、数字转字符串等。
- 表单验证：密码强度、常见验证规则等。
- 表单选取：单选框的选中、复选框全选等。

## 2.1 去除字符串左右两边的空格

在日常工作中，过滤表单中一些特殊的字符是很常见的功能。比如文本中要求输入单纯的数字，但用户有时会误输入一些多余的空格或其他字符混合的文本，这显然不符合输入要求。下面一起来学习怎样去除字符串左右两边的空格，图 2-1、图 2-2 是本例的效果图。

需要过滤空格      过滤

图 2-1 过滤前含有空格的 input 表单

需要过滤空格      过滤

图 2-2 过滤后没有空格的 input 表单

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 var _rstrsBtn = document.getElementById("rstrsBtn"), // 获取过滤按钮对象
04 _strs = document.getElementById("strs"); // 获取被过滤元素
05
06 _rstrsBtn.onclick = function(){ // 去除空格
07 _strs.value = _strs.value.replace(/^(\s|\u00A0)+|(\s|\u00A0)+$/g, "");// 正则替换
08 }
}
```

```
09 };
10 </script>
```

本例 HTML 代码如下：

```
01 <h2>测试去除字符串左右两边的空格</h2>
02 <input type='text' id='strs' value=' 需要过滤空格 '>
03 <input type='button' id='rstrsBtn' value=' 过滤 '>
```

#### 【代码说明】

- JavaScript 代码第 06~08 行是去除字符串左右空格的关键代码。
- JavaScript 代码第 07 行的 replace() 是 JavaScript 中的原生函数，此处使用 replace() 替换一个与正则表达式匹配的子串，性能相对比较优越。其中的正则“`^\s\u00A0)+|(\s|\u00A0)+$/g`”是关键部分，“`\s`”用来匹配任何空白字符。
- 在 index.html 中创建两个 input，供读者测试代码。

## 2.2 验证用户是否输入

表单中经常有些必填项，如果用户没有输入则不允许提交表单。例如在注册用户时，必须填写用户名才能提交注册信息。验证用户是否输入时，通常需要先过滤再验证（如果用户输入的都是空格就相当于没有输入）。上一节已经讲解过怎样去除字符串两边的空格了，现在我们再增加验证用户是否输入的功能。本例效果如图 2-3 和图 2-4 所示。

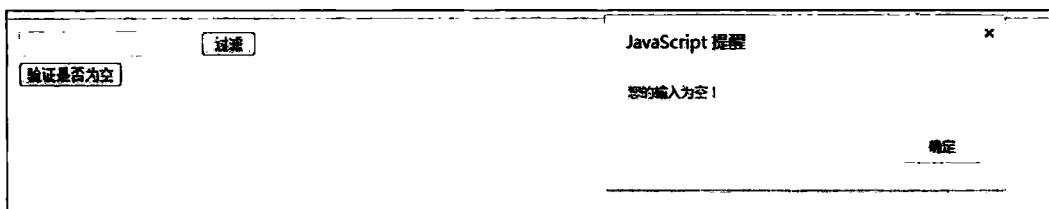


图 2-3 验证为空的效果图

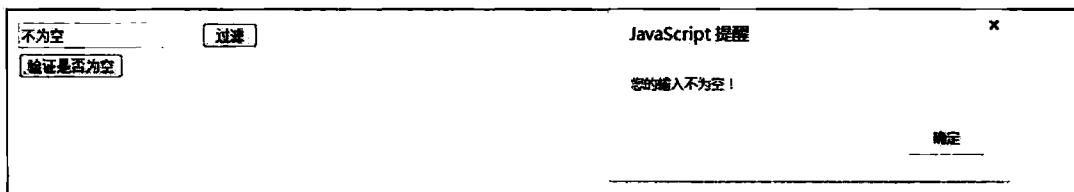


图 2-4 验证不为空的效果图

验证用户是否输入，首先创建一个 isContent() 函数，包含两种写法，参见下面的代码。第一种与第二种风格上比较相似，只是在比较算法时有些不同，第一种相对更简洁，性能更好。

```
01 function isContent(chars){ //验证是否为空，chars 为过滤后的字符串
02 return !chars ? true : false; //第一种写法
03 return chars.length == 0 ? true : false; //第二种写法
```

04 }

本例 HTML 代码如下：

```
01 <h2>验证是否输入</h2>
02 <input type='text' id='strs' value=' 需要过滤空格 '>
03 <input type='button' id='isContent' value='验证是否为空'>

```

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 //获取验证是否为空的响应元素
04 var _isContent = document.getElementById("isContent"),
05 _strs = document.getElementById("strs"); //获取待验证的文本对象
06
07 _isContent.onclick = function(){ //绑定按钮事件
08 if(!_strs.value.replace(/^(\s|\u00A0)+|(\s|\u00A0)+$/g, "")){//判断是否为空
09 alert("您的输入为空！");
10 }else{
11 alert("您的输入不为空！");
12 }
13 }
14 };
15 </script>
```

#### 【代码说明】

JavaScript 代码第 08 行采用 isContent() 函数中的第一种写法验证 “\_strs.value” 是否为空。首先，利用 replace() 与正则结合过滤空字符，然后，再用 “!” 符号判断是否为空，最后弹出对应的提示框。

## 2.3 禁止输入

我们都知道验证码的存在就是为了防范恶意机器人重复注册或频繁登录，当网站后台程序判定此注册是恶意操作时，会禁止文本框输入。目前比较流行的禁止表单输入方案，不外乎两种：

- 通过浏览器对表单元素中特殊属性的支持来禁止输入。
- 通过 JavaScript 控制输入的方式。

#### 第一种方案：通过表单元素的特殊属性来控制输入

HTML 代码如下：

```
01 <!—其余代码略—>
02 <div>表单元素特殊属性 一<input type="text" value="禁止输入" disabled /> </div>
03 <div>表单元素特殊属性 二<input type="text" value="禁止输入" disabled="disabled"/> </div>
04 <div>表单元素特殊属性 三<input type="text" value="禁止输入" readonly /> </div>
```

图 2-5 和图 2-6 是本例的效果图。

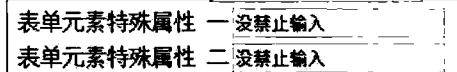


图 2-5 没有禁止的元素，区域白色，可输入

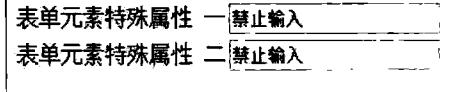


图 2-6 禁止后的元素，区域变灰，不可输入

### 【代码说明】

- `disabled` 属性规定禁用 `input` 元素，第 02 行和第 03 行是两种写法。
- `disabled` 属性无法与`<input type="hidden">`一起使用。
- 被禁用的 `input` 元素既不可用，也不可单击。可以设置 `disabled` 属性，直到满足某些其他条件（比如选择了一个复选框等），然后再通过 JavaScript 来删除 `disabled` 值，将 `input` 元素的值切换为可用。
- 代码第 04 行应用了 `readonly`。`readonly` 属性规定输入字段为只读，只读字段是不能修改的。不过，用户仍然可以使用 Tab 键切换到该字段，还可以选中或复制其中的文本。
- `readonly` 属性可与`<input type="text">`或`<input type="password">`配合使用。

### 第二种方案：通过 JavaScript 控制输入的方式

通过 JavaScript 控制输入的方式也有多种。目前比较流行的方式是，通过事件来控制文本输入或控制焦点。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var _inhibitingInput = document.getElementById("inhibitingInput");//获取响应元素
04 _inhibitingInput.onfocus = function(){ //第一种写法 控制失去焦点
05 _inhibitingInput.blur();
06 }
07
08 /*var noText = function (){ //第二种写法 通过 keyup 与 blur 组合使用
09 _inhibitingInput.value = ""; //设置值为空
10 }*/
11
12 _inhibitingInput.onkeyup =
13 _inhibitingInput.onblur = noText; */
14 };
15 </script>

```

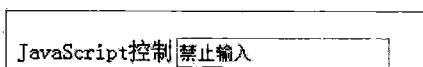
本例 HTML 代码如下：

```

01 <!—其余代码略→
02 <div>JavaScript 控制<input type="text" value="禁止输入" id='inhibitingInput' /></div>

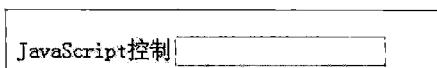
```

图 2-7、图 2-8 是本例的效果图。



JavaScript 控制 禁止单输入

图 2-7 JavaScript 方法一，被禁止的元素，区域白色，但不可输入



JavaScript 控制 禁止单输入

图 2-8 JavaScript 方法二，被禁止的元素，区域白色，但不可输入

### 【代码说明】

- 方法一，JavaScript 代码第 05 行通过让控制元素失去焦点的方式来禁止输入。这种写法是第一选择，简洁、高效。
- 方法二，JavaScript 代码第 12~13 行通过控制 keyup 事件与 blur 事件来清空文本。这种写法相较于第一种写法，明显有很多不足之处，代码量多，无法直接阻止通过鼠标复制、粘贴来的文本，虽失去焦点时可以清空内容，但还是有一定的缺陷。

**提示：**读者使用源代码中第二种方法测试时，把第一种方法注释掉，把第二种方法的注释取消。

## 2.4 关闭输入法

关闭表单元素中的输入法是很常见的需求，比如输入 E-mail、网址时用的都是英文字母，此时关闭输入法可以防止用户输入错误。目前主流的禁用（关闭）技术是利用 CSS 的特殊属性，还有就是 JavaScript 的事件文本过滤技术。下面分别讲解这两种方案：

### 方案一：植入 CSS 特殊属性

植入 CSS 特殊属性这种方案有一个弊端，就是不支持 Chrome 浏览器，但是其他主流浏览器（如 IE、FireFox 等）都支持。创建一个函数 banInputMethod()，添加为指定元素增加 CSS 特殊属性的业务逻辑，如下：

```

01 //关闭输入法
02 function banInputMethod (_elementArr){
03 var arr = _elementArr, //待处理的所有节点元素
04 self = this;
05 //判断元素是否为数组，如果不是数组将其变成数组
06 if(!_elementArr instanceof Array) {
07 arr = [_elementArr];
08 };
09 for(var i= 0,arrLen = arr.length ;i<arrLen;i++){ //遍历元素节点
10 var arrl = arr[i];
11 arrl.onfocus = function(){
12 //样式方案，只兼容除了 Chrome 浏览器之外的浏览器
13 this.styleimeMode='disabled';
14 }
}

```

```
15 }
16 }
```

#### 【代码说明】

- 第 04 行为了增加对外层域的访问，定义了一个变量 self。
- 第 02~03 行将传入的参数赋值给新的变量 arr，以便未来修改。
- 第 06 行使用 instanceof 判断参数是否为数组，如果不是数组，将参数修改为数组，方便后边代码统一处理。instanceof 用于判断一个值是否为某个对象的实例。
- 第 09~15 行遍历所有元素，在元素上绑定特殊的样式属性 imeMode，并设置为 disabled。imeMode 有 4 种属性参考——auto 表示打开输入法（默认属性）、active 代表输入法为中文、inactive 代表输入法为英文、disabled 表示关闭输入法。

本例增加 HTML 代码如下：

```
01 <h2>关闭输入法</h2>
02 <input type='text' id='banInputMethodgoogle' value='不支持谷歌浏览器'>

```

本例增加 JavaScript 代码如下：

```
01 banInputMethod(getElement("banInputMethodgoogle"));
02 function getElement (eStr){ //根据 ID 获取指定对象
03 return document.getElementById(eStr);
04 }
```

#### 【代码说明】

- 在考虑代码量、简洁、方便的基础上，通过 JavaScript 原生函数“document.getElementById()”获取 ID 对象的 API，抽象为“getElement()”。
- 在 HTML 中增加元素，为相应的元素绑定 banInputMethod() 函数。

下面为非 Chrome 浏览器的测试效果，在表单默认值后面，再次追加值，可以发现已经不能输入中文了，如图 2-9 所示。

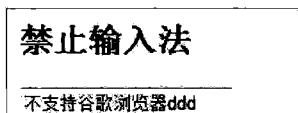


图 2-9 非 Chrome 浏览器，在默认值后边只能输入非中文

但是在 Chrome 浏览器中还可以输入中文，如图 2-10 所示。

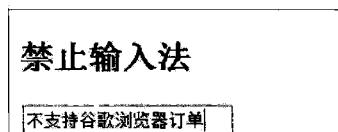


图 2-10 Chrome 浏览器，在默认值后边可以输入中文“订单”

#### 方案二：事件文本过滤

为了修复 Chrome 的不兼容性，可以利用 JavaScript 中的事件来禁止输入中文，模拟关闭输入法。结合 banInputMethod() 函数的思路，本例 JavaScript 代码如下：

```
01 var arr =[//创建节点数组
02 document.getElementById("banInputMethodgoogle"),
```

```

03 document.getElementById("banInputMethod")];
04 self = this
05 ;
06 for(var i=0,arrLen = arr.length ;i<arrLen;i++){ //遍历待处理的节点对象
07 var arrl = arr[i];
08 arrl.onfocus = function(){ //获取焦点事件
09 this.styleimeMode='disabled'; //样式方案，只兼容除了谷歌浏览器之外的浏览器
10 }
11
12 var banInputMethod = arrl.getAttribute("banInputMethod"); //获取 banInputMethod 属性
13 if(banInputMethod){ //判断是否存在 banInputMethod
14 var clearChinese = function(_this){
15 var _v = _this.value;
16 _this.value = _v.replace(/\u4e00-\u9fa5/g,""); //正则替换中文字符
17 }
18 arrl.onkeyup = function(){ //keyup 事件
19 clearChinese(this);
20 }
21 arrl.onblur = function(){ //blur 事件
22 clearChinese(this);
23 }
24 }
25 }

```

本例 HTML 代码如下：

```
<input type='text' banInputMethod='1' id='banInputMethod' value='支持主流浏览器'>
```

#### 【代码说明】

- 第 12~13 行通过获取元素节点中的 banInputMethod 属性，判断是否绑定了关闭输入法的开关。
- 第 18~23 行绑定 blur 与 keyup 事件，检测是否含有中文字符。
- 触发事件后，通过第 14~17 行的 clearChinese() 函数，清除中文字符，最终效果如图 2-11 所示。



图 2-11 使用 JavaScript 事件模拟关闭输入法，清空文本内的中文

## 2.5 禁止复制与粘贴

复制与粘贴是网民日常的一些基本操作，但有些网站为了保护版权（如小说类网站、图片类网站），禁止用户执行这些操作，这样就可以防止用户将正在浏览的文本，通过复制、粘贴的方式随意传播了。现在主流的浏览器都提供了一些新增的事件函数 API 来禁止复制与粘贴。

本例 JavaScript 代码如下：

```
01 var banCopyPaste = document.getElementById("banCopyPaste");
02 banCopyPaste.oncopy = function(){ //禁止复制事件
03 return false;
04 }
05
06 banCopyPaste.onpaste = function(){ //禁止粘贴事件
07 return false;
08 }
```

在 index.html 中增加 input 元素：

```
01 <h2>禁止复制与粘贴</h2>
02 <input type="text" name="banCopyPaste" id="banCopyPaste"/>

```

#### 【代码说明】

- JavaScript 代码第 02~08 行利用 JavaScript 的 copy 与 paste 事件禁用复制与粘贴，速度相对比较快。
- 本例的原理是为元素添加 copy 与 paste 事件，并在事件中返回 false。

## 2.6 限制只能输入数字

当我们在一些网站注册账号、填写信息时，不小心将电话号码填写为汉字或其他英文字母了，这显然是不正确的。为了帮助用户更好地纠正输入时的错误，在表单中填写信息时，需要限制手机号、邮编、电话号码这类文本框不能输入其他字符，只能是数字。本例效果分别如图 2-12、图 2-13、图 2-14 所示。

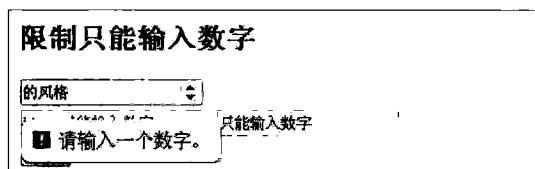


图 2-12 第一个 input 基于 HTML 5 的 number

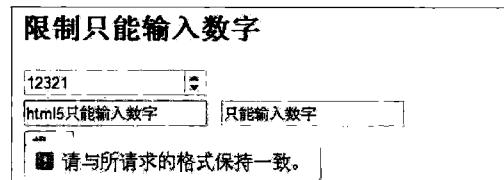


图 2-13 第二个 input 基于 HTML 5 的 pattern

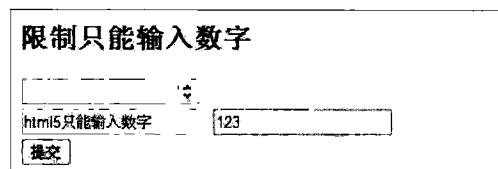


图 2-14 第三个 input 基于 JavaScript

本例有两种实现方案：一种是使用 HTML 5，一种是使用 JavaScript。

#### 方案一：使用 HTML 5

(1) 最新的 HTML 5 API 对表单中的 input 类型进行了增强与扩展，只有支持 HTML 5 及此特殊属性的浏览器才能使用，另外也新加入了 pattern，代码如下：

```

01 <form>
02 <input type="number" name="numberHTML 5" value='HTML 5 只能输入数字'
03 id='banNumberHTML 5'>
04 <input type="text" value='HTML 5 只能输入数字' pattern=[0-9] id='patternHTML 5' />
05 <input type="text" name="number" value='只能输入数字' id='banNumber' />

06 <input type='submit' value='提交'>
07 </form>

```

### 【代码说明】

- HTML 5 API 开放标准中有很多新的类型，其中 `number` 表示只能输入数字，在提交表单时会执行验证。
- HTML 5 中还增加了 `pattern` 属性，用来规定验证输入字段的正则匹配模式，适用于 `text`、`search`、`url`、`telephone`、`email`、`password` 等元素。

### 方案二：使用 JavaScript

利用 JavaScript 的事件处理来限制只能输入数字，聪明的读者估计已经猜到了，可以用“正则+事件”的方式。本例 JavaScript 代码如下：

```

01 var banNumber = document.getElementById("banNumber"),
02 clearNonumber = function(tThis){ //过滤数字
03 var _v = tThis.value;
04 tThis.value = _v.replace(/\D/g,"");
05 }
06 banNumber.onfocus = function(){ //绑定获取焦点事件
07 clearNonumber(this);
08 }
09 banNumber.onkeyup = function(){ //绑定键盘事件
10 clearNonumber(this);
11 }
12 banNumber.onblur = function(){ //失去焦点事件
13 clearNonumber(this);
14 }

```

在 `index.html` 文件中增加一个新的测试元素：

```
<input type="text" name="number" value='只能输入数字' id='banNumber' />

```

### 【代码说明】

第 06~14 行为元素绑定 `focus`、`blur`、`keyup` 这 3 个事件，在用户触发这 3 个事件时，调用第 02~05 行实现的 `clearNonumber()` 函数，清除非数字字符。其中的清除方法是 JavaScript 内置的 `replace()`，它使用正则替换不符合规范的字符。

当然限制输入数字还有很多其他的解决方法，如数字匹配提取、遍历所有字符校检等。读者可以开动脑筋，自己动手编写一些新的替代方案。

## 2.7 限制只能输入中文

只能输入中文是国内常用的功能，例如有的网站需要实名认证（如百合网），那么对

于中文的验证与限制就变得相当重要了。限制中文与限制数字的实现方法类似，因为上一节代码已经有过解释，以下代码不再做详细说明。

本例 JavaScript 代码如下：

```

01 window.onload = function(){
02 var chineseStr = document.getElementById("chineseStr"),
03 clearNonumber = function(tThis){ //过滤字符
04 var _v = tThis.value;
05 tThis.value = _v.replace(/[\u4e00-\u9fa5]/g,""); //正则替换
06 }
07 chineseStr.onfocus = function(){ //获取焦点事件
08 clearNonumber(this);
09 }
10 chineseStr.onkeyup = function(){ //键盘事件
11 clearNonumber(this);
12 }
13 chineseStr.onblur = function(){ //失去焦点事件
14 clearNonumber(this);
15 }
16 };

```

本例 html 代码如下：

```

01 <h2>限制只能输入中文</h2>
02 <input type="text" name="number" value='只能输入中文' id='chineseStr'/>


```

#### 【代码说明】

第 07~15 行为元素绑定 focus、keyup、blur 这 3 个事件，在用户触发这些事件时，调用第 03~06 行实现的 clearNonumber() 函数，清除非中文的字符。

## 2.8 限制字符串长度

在开发 Web 页面时，表单内字符过长而超出规定长度会导致一些不必要的麻烦，比如：用户注册时系统限制的用户名只有 8 个英文字符，但是为用户输入了 10 个甚至更多的字符时，就可能造成昵称页面显示错行，或者昵称被截断的问题。在 Web 前端，优先的方案是选择 CSS 或浏览器内置的 API 来控制，JavaScript 只是候选之策。因浏览器或 CSS 缺少而达不到用户体验的级别要求时才会选择 JavaScript 方式。

下面给出两种限制字符串长度的解决方案：

#### 方案一：maxlength 属性

在 HTML 的 input 中，maxlength 属性可以限制输入元素的字符长度，代码如下：

```

01 <h2>限制字符串的长度</h2>
02 通过“maxlength”限制：<input type="text" name="lname" maxlength="5" />


```

#### 【代码说明】

并不是所有的浏览器都支持 maxlength 属性，它有平台的局限性。图 2-15、图 2-16 是

这个属性的限制方式，它不会区分中英文字符和数字的编码占位，统一计算，不太友好。

### 限制字符串的长度

通过“maxlength”限制：

图 2-15 数字与英文字符按 1 计算

### 限制字符串的长度

通过“maxlength”限制：

图 2-16 中文字符按 1 计算

## 方案二：使用 JavaScript

通过 JavaScript 也可以很好地模拟以上的效果，而且兼容所有主流浏览器。代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 var limitLength = document.getElementById("limitLength"); //获取限制对象
04 clearNonumber = function(tThis){ //清除数字
05 var _v = tThis.value,
06 _vLen = _v.length,
07 dataLength = tThis.getAttribute("data-length");//获取长度属性
08 dataModel = tThis.getAttribute("data-model"),
09 subLen = dataLength;
10 if(_vLen > dataLength) tThis.value = _v.substr(0, subLen); //判断长度
11 if(remainingCharacters){
12 self.showRemainingCharacters(!_vLen ? dataLength :_vLen >
13 dataLength ? 0 :dataLength - _vLen), remainingCharacters);
14 }
15 };
16 limitLength.onfocus = function(){ //获取焦点事件
17 clearNonumber(this);
18 }
19 limitLength.onkeyup = function(){ //键盘事件
20 clearNonumber(this);
21 }
22 limitLength.onblur = function(){ //失去焦点事件
23 clearNonumber(this);
24 }
25 };
26 </script>
```

本例 HTML 代码如下：

```
01 <h2>限制字符串的长度</h2>
02 通过“maxlength”限制：<input type="text" name="lname" maxlength="5" />

03 通过“JavaScript 事件”限制：<input type="text" data-length='5' id="limitLength"
04 data-model='Ch' name="lname"/>

```

### 【代码说明】

- 在 id 为 limitLength 的 input 中，增加属性绑定“data-length='5'"”。
- 获取指定的元素，计算元素的长度（不区分中英文）。JavaScript 代码第 07 行获取指定元素绑定的长度值 data-length，然后判断实际元素的长度是否超出范围，如果

超出，在 JavaScript 代码第 10 行调用 substr() 函数截断字符。

- JavaScript 中的 substr(start, length) 函数，可以截取从 start 到 length 指定长度的字符。

**注意：**浏览器各个版本对 onpaste 事件支持的程度不一样，但是主流的浏览器基本都支持。

## 2.9 限制字符串长度（区分中英文）

微博有一个比较醒目的功能，最多输入 140 字，超出长度则禁止发布，这种限制是区分中英文的。本例在上一节的基础上，增加对中英文的支持。首先创建一个区分中英文字符的函数，在区分的基础上计算中英文单个字符的占位数。

在源代码同级目录下，新建一个文件夹 extend，在其中创建一个 strLen.js 文件，文件结构如图 2-17 所示。



图 2-17 extend 文件夹与 strLen.js 文件

extend 代表着可扩展的 JavaScript 模块文件，后续章节中，将可以单独作为一个模块的文件放在这个文件夹内（例如本例的文件 strLen.js），这样做可以让项目及文件更易于管理。

在 strLen.js 中，创建如下模块：

```

01 var strLen = (function(){
02 var trim = function(chars){
03 return (chars || "").replace(/^(\s|\u00A0)+|(\s|\u00A0)+$/g, "");
04 }
05
06 return function(_str, _model) {
07 _str = trim(_str),
08 _model = _model || "Ch"; //默认是中文
09 var _strLen = _str.length; //获取字符长度
10 if(_strLen == 0){ //如果字符为 0 直接返回
11 return 0;
12 }
13 else{
14 var chinese = _str.match(/\u4e00-\u9fa5/g); //匹配中文
15 //判断是什么模式
16 return _strLen + (chinese && _model == "Ch" ? chinese.length: 0);
17 }
18 };
19 })();

```

### 【代码说明】

- 本例的总体思路是，利用正则将中文的字符数计算出来，再加以统计。有两种模式

切换用于统计字符串的长度，“En”英文主计算模式，将每个中文算作1个字符；“Ch”中文主计算模式，将每个中文算作两个字符。

- 用模块模式构建 strLen() 函数，第 02~04 行将 trim() 作为私有函数，与外界隔离。参数 \_str 表示被统计的字符串，\_model 表示统计字符串的模式。
- 第 14 行用 match() 函数过滤中文字符，计算有多少个中文。如果为“Ch”模式则加上中文的“chinese”，如果为“En”则不加。

在 index.html 文件中，增加“data-model='Ch'”，代码如下：

```
01 <input type="text" data-length='5' id='remainingCharacters' data-model='Ch'
02 name="remainingCharacters"/>

```

将新创建的函数引入进来，代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 var
04 forElementArr = function(_elementArr, callBack){//遍历节点
05 var arr = _elementArr, //所有节点对象
06 self = this //外层环境
07 ;
08
09 if(!_elementArr instanceof Array){ //如果不是数组，变成数组对象方便处理
10 arr = [_elementArr];
11 };
12 for(var i= 0, arrLen = arr.length ;i<arrLen;i++){//遍历数组
13 var arrl = arr[i];
14 if(typeof arrl == "string"){ //判断是否为字符串
15 arrl = document.getElementById(arrl);
16 }
17 callBack && callBack(i, arrl); //如果存在回调则执行回调
18 }
19 },
20 showRemainingCharacters = function(_nums, _remainingCharacters){
21 if(_remainingCharacters.search(",") != -1){ //是否存在,
22 _remainingCharacters = _remainingCharacters.split(",");//英文字符串分割
23 }
24 forElementArr(_remainingCharacters, function(_index, _this){ //遍历元素
25 _this.innerHTML = (_nums && _nums.toString()) || "0";
26 });
27 },
28 //获取限制对象
29 remainingCharacters = document.getElementById("remainingCharacters"),
30 clearNonumber = function(tThis){ //清除数字
31 var _v = tThis.value,
32 _vLen = _v.length,
33 dataLength = tThis.getAttribute("data-length"),
34 remainingCharacters =
```

```

35 tThis.getAttribute("data-remainingCharacters");//如果有实时显示的属性，则在指定元素上显示
36
37 /*区分中英文前 if(_v.length > dataLength) tThis.value
38 = _v.substr(0, dataLength);*/
39
40 var dataModel = tThis.getAttribute("data-model"); //区分中英文后
41 var subLen = dataLength; //获取数据长度
42 if(dataModel == "Ch"){
43 _vLen = strLen(_v, dataModel); //判断模式
44 var vv = _v.match(/\u4e00-\u9fa5/g); //匹配中文
45 subLen = dataLength - (vv ? vv.length : vv.length);
46 }
47 if(_vLen > dataLength) tThis.value = _v.substr(0, subLen);
48 if(remainingCharacters){
49 showRemainingCharacters(!_vLen ? dataLength :(_vLen >
50 dataLength ? 0 :dataLength - _vLen), remainingCharacters);
51 }
52
53 };
54 remainingCharacters.onfocus = function() //获取焦点
55 clearNonumber(this);
56 }
57 remainingCharacters.onkeyup = function() //键盘事件
58 clearNonumber(this);
59 }
60 remainingCharacters.onblur = function() //失去焦点
61 clearNonumber(this);
62 }
63 };
64 </script>

```

#### 【代码说明】

- 主要思路：HTML 代码第 01 行，当元素绑定 data-model 且 data-model = ‘Ch’时，才开启区分中英文的状态。换个角度说，默认是“En”或不区分状态。
- JavaScript 代码第 42 行开启区分中英文状态之后，strLen()函数会计算中文状态下“\_v”的真实占位字符个数，之后再计算 substr()要截取字符的长度 subLen。
- JavaScript 代码第 45 行中，subLen 的计算方式是，一个中文字符算作 2，一个非中文字符算作 1，以不超过 HTML 代码第 1 行中的 data-length 最大数字为上限。

注意：当“\_vLen > dataLength”时才截取字符。

## 2.10 实时提示可输入字符（区分中英文）

虽然发微博时系统限制了用户输入字符的个数，但用户并不知道自己当时输入了多少

字符，那用户体验岂不是降低了很多？因此，需要在页面中实时显示还可以输入多少个字符，效果如图 2-18 所示。

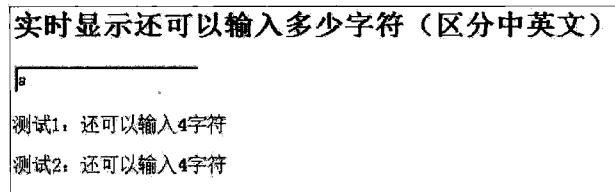


图 2-18 实时显示还可以输入多少字符

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var
04 forElementArr = function(_elementArr, callBack){ //遍历节点
05 var arr = _elementArr, //所有节点对象
06 self = this; //外层环境
07 ;
08
09 if(!_elementArr instanceof Array){//如果不是数组，变成数组对象方便处理
10 arr = [_elementArr];
11 };
12 for(var i=0,arrLen = arr.length ;i<arrLen;i++){ //遍历数组
13 var arrl = arr[i];
14 if(typeof arrl == "string"){ //判断是否为字符串
15 arrl = document.getElementById(arrl);
16 }
17 callBack && callBack(i, arrl); //如果存在回调则执行回调
18 }
19 },
20 showRemainingCharacters = function(_nums, _remainingCharacters){
21 if(_remainingCharacters.search(",") != -1){
22 _remainingCharacters = _remainingCharacters.split(",");//英文字符串分割
23 }
24 forElementArr(_remainingCharacters, function(_index, _this){
25 _this.innerHTML = (_nums && _nums.toString()) || "0";
26 });
27 },
28
29 strLen = (function() { //统计中英文字符数
30 var trim = function(chars){
31 return (chars || "").replace(/^(\s|\u00A0)+|(\s|\u00A0)+$/g, "");
32 }
33
34 return function(_str, _model) { //返回字符处理的函数

```

```

35 _str = trim(_str),
36 _model = _model || "Ch"; //默认为中文模式
37 var _strLen = _str.length; //获取字符长度
38 if(_strLen == 0){ //如果字符长度为 0 直接返回
39 return 0;
40 }
41 else{
42 var chinese = _str.match(/\u4e00-\u9fa5/g); //匹配中文
43 return _strLen + (chinese && _model == "Ch" ? chinese.length: 0);
44 }
45 };
46 },
47 //获取限制对象
48 remainingCharacters = document.getElementById("remainingCharacters"),
49 clearNonumber = function(tThis){ //清除数字字符
50 var _v = tThis.value,
51 _vLen = _v.length, //文本内的字符长度
52 dataLength = tThis.getAttribute("data-length"),
53 remainingCharacters =
54 tThis.getAttribute("data-remainingCharacters");//如果有实时显示的属性，则在指定元素上显示
55
56 /*区分中英文前 if(_v.length > dataLength) tThis.value
57 = _v.substr(0, dataLength);*/
58
59 var dataModel = tThis.getAttribute("data-model");//区分中英文后
60 var subLen = dataLength;
61 if(dataModel == "Ch"){ //如果为中文模式
62 _vLen = strLen(_v, dataModel); //获取长度
63 var vv = _v.match(/\u4e00-\u9fa5/g);
64 subLen = dataLength - (!vv ? 0 : vv.length);
65 }
66 if(_vLen > dataLength) tThis.value = _v.substr(0, subLen);
67 if(remainingCharacters){
68 showRemainingCharacters(!_vLen ? dataLength :(_vLen >
69 dataLength ? 0 :dataLength - _vLen), remainingCharacters);
70 }
71
72 };
73 remainingCharacters.onfocus = function(){ //获取焦点
74 clearNonumber(this);
75 }
76 remainingCharacters.onkeyup = function(){ //键盘事件
77 clearNonumber(this);
78 }
79 remainingCharacters.onblur = function(){ //失去焦点

```

```

80 clearNonumber(this);
81 }
82 };
83 </script>

```

本例 HTML 与 CSS 代码如下：

```

01 <style>
02 .remainingCharacters{
03 color: green;
04 font-weight: bold;
05 }
06 </style>
07 <h2>实时显示还可以输入多少字符（区分中英文）</h2>
08 <input type="text" data-length='5' id='remainingCharacters'
09 data-remainingCharacters="charActers1,charActers2" data-model='Ch'
10 name="remainingCharacters"/>

11 <p>测试 1：还可以输入5字符
12 </p>
13 <p>测试 2：还可以输入5字符
14 </p>

```

#### 【代码说明】

- HTML 代码第 08~10 行通过“`id=remainingCharacters`”的文本框来测试输入文本，CSS 代码第 02 行定义了它的样式。HTML 代码中的 `charActers1` 与 `charActers2` 用来显示测试结果。第 09 行的“`data-remainingCharacters="charActers1,charActers2"`”开启实时显示状态，并将显示结果绑定到指定的元素上。
- JavaScript 代码第 73~81 行为文本绑定响应事件。
- 变量“`_vLen`”代表文本内的文字长度，第 61~66 行计算“`_vLen`”的长度。
- JavaScript 代码第 53~70 行获取 `data-remainingCharacters` 属性，如果存在，则开启实时显示。
- JavaScript 代码第 68~69 行判断“`_vLen`”的长度，如果为 0 则显示设定的字符长度 `dataLength`，然后判断“`_vLen`”是否大于 `dataLength`，如果大于 `dataLength` 则显示 0，否则使用“`dataLength-_vLen`”计算还有多少字符。
- 调用 `showRemainingCharacters()` 函数可以显示结果。
- 为了支持“单输入多显示”，`showRemainingCharacters()` 函数的第 2 个参数“`_remainingCharacters`”表示获取待显示结果的元素 id 集合，本例有两个元素，id 分别为“`charActers1`”与“`charActers2`”。
- 第 24~26 行用 `forElementArr()` 函数遍历待显示结果的元素。

**说明：**单输入多显示是指“一个文本输入，多个元素显示”。

## 2.11 在输入框中显示提示信息

为了增加用户体验效果，告诉用户当前的表单需要填写什么信息，需要在一些表单元

素内预定义一些提示性的信息，例如：密码框中提示“请输入密码”、用户名中提示“请输入用户名”等。当获取焦点时提示性信息消失；失去焦点时，如果表单元素内所填写的内容为空，则再次显示提示性信息，否则不显示。

通过上面的需求简述，有以下3种方案可以选择。

(1) 最直接、最简单的方案就是利用HTML5技术，使用input中新增加的placeholder属性，可以预定义一些提示信息。现在的浏览器市场参差不齐，国内IE系列居多，导致HTML5推广缓慢，要想适应所有浏览器不太可能，但移动互联网（智能手机）方向的HTML5技术已经应用非常广泛。本方案代码如下：

```
<input type="text" name="info" placeholder="HTML5 提示信息" />
```

(2) 利用JavaScript脚本操作DOM。这种方案的首选方法就是动态地修改表单元素的文本值，但是在测试时发现IE浏览器有一些兼容性问题，因此，这个不是最优方案。读者可以将以下代码复制过来，放在实际测试环境中测试一下，这段代码是笔者JavaScript刚入行时写的一个小模块，基于jQuery，如果嫌麻烦，在课件的extend中有源代码。

```

01 var ContentState={
02 _Val:{},
03 _obj:null,
04 ctor:function () { //初始化的空函数
05
06 },
07 "init":function(objID){ //初始化
08 var self = this; //外层环境引用
09 var isString=typeof objID; //检测类型
10 var initAll = function(_objID){
11 self._obj=$('#'+_objID); //获取对象
12 self._Val[self._obj.attr('id')]=self._obj.val();
13
14 self.eventBind(); //事件绑定
15 }
16 if(isString == 'string'){
17 initAll(objID);
18 }else if($.isArray(objID)){ //是否为数组
19 var objIDL=objID.length;
20 for(var k=0 ; k < objIDL ; k++){
21 initAll(objID[k]);
22 }
23 }else{
24 jQuery.error('Error!Please use the string type.');//打印错误
25 }
26 },
27
28 "eventBind":function(){
29 var that=this;

```

```

30 this._obj.die();
31 this._obj.css('color','#999'); //修改颜色
32 this._obj.live('focusout focusin', function(event) {
33 that.targetEvent(event, $(this)); //绑定焦点事件
34 });
35 },
36
37 "targetEvent":function(_event, _this){
38 var that = this;
39 var valing=_this.val();
40 var _type=_this.attr("input-type");
41 if (_event.type == 'focusout') {
42 if(valing.length <= 0 && _type != "password"){ //判断是否为空或密码框
43 //是否为密码框，如果是就隐藏当前元素，创建一个与其同样属性的元素
44 _this.val(that._Val[_this.attr('id')]);
45 _this.css('color','#999'); //修改颜色
46 }
47 } else {
48 if(valing == that._Val[_this.attr('id')]){
49 if(_type && _type == "password"){
50 _this.hide();
51 _this.next().show().focus();
52 _this.next().blur(function(){
53 if($.trim($(this).val()).length <= 0){ //判断过滤后的字符长度
54 $(this).hide().unbind();
55 _this.show().val(that._Val[_this.attr('id')]);
56 }
57 });
58 }else{
59 _this.val("");
56 _this.css('color','#666');
57 }
58 }
59 }
60 },
61
62 }
63
64 }
65 },
66 },
67
68 "eventDie":function(){
69 if(this._obj){
70 this._obj.die();
71 }
72 }
73 };

```

### 【代码说明】

- 以上代码依赖jQuery库，因为仅供读者测试用，所以就没有改成JavaScript的原生写法。
- 获取待绑定行为的元素，为每个元素的值绑定事件，针对每个指定的元素值，使用“`self._Val[self._obj.attr('id')] = self._obj.val();`”将其存储在独有的对象域下，在事件触发时可以修改元素的值。
- 关键问题在于，IE不支持JavaScript修改`value`，因此在这个模块中增加了一个新的业务逻辑，当表单类型为密码时，需要模块的使用者增加一个明文的密码提示元素，来修复这一bug。换个角度来讲，当获取焦点时让明文的元素消失，显示密码域的元素，并且让密码域的元素获取焦点。当失去焦点时，如果密码域为空，则将其隐藏，显示明文的元素。

(3) 为了修复以上的bug，增加一个可复用的模块，统一采用“双元素单显示（即将2个或者2个以上的元素，在视觉上表示成一个元素的形式）”的方案，本例通过一个标签覆盖input文本框的方式，模拟一个input元素和文本输入域默认值（span元素）。不采用修改文本值的方案，JavaScript代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var setCss = function(_this, cssOption){ //设置样式
04 //检测节点类型
05 if(!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style){
06 return;
07 }
08 for(var cs in cssOption){ //遍历样式
09 _this.style[cs] = cssOption[cs];
10 }
11 return _this;
12 },
13 hintInput = document.getElementById("hintInput"), //获取目标元素
14 _span = document.createElement("span"), //创建span节点
15 dataHint = hintInput.getAttribute("data-hint");
16 _span.innerText = dataHint;
17 setCss(_span, {
18 "position":"absolute",
19 "left":hintInput.offsetLeft+2,
20 "top":hintInput.offsetTop,
21 "zIndex":2
22 });
23 _span.className = "hintInput"; //className 默认值为 hintInput
24 hintInput.value = "";
25 _span.setAttribute("id","hint0"); //设置默认 id 属性为 hint0
26 hintInput.parentNode.insertBefore(_span,hintInput); //插入元素
27

```

```

28 var onhint = function(e){
29 setCss(_span, {
30 "display":"none"
31 })
32 hintInput.focus(); //获取焦点
33 }
34 hintInput.onblur = function(e){ //失去焦点显示元素
35 //正则过滤字符判断文本是否为空
36 if(!hintInput.value.replace(/^(\s|\u00A0)+|(\s|\u00A0)+$/g, "")){ //单击事件与获取焦点事件使用同一个函数处理
37 setCss(_span, {
38 "display":"block"
39 })
40 }
41 }
42 _span.onclick = hintInput.onfocus = onhint;
43 };
44 </script>

```

本例 HTML 代码如下：

```

01 <h2>显示输入提示，获取焦点提示取消</h2>
02 <input type="text" name="info" placeholder="HTML 5 提示信息" />
03 <input id='hintInput' data-hint='我是提示信息' type='password' value="">

```

#### 【代码说明】

- 首先获取待处理的对象，然后创建 span 元素，并且设置 span 元素的位置，让其正好位于被提示元素的文本框内，并且覆盖文本框。span 的提示信息来自于待处理对象中的 data-hint。
- 第 03~12 行增加 setCss() 函数方便设置元素节点样式，为元素绑定事件处理程序 onhint。
- setCss() 函数中增加了节点类型过滤，只有符合节点类型才会被后边的程序处理。
- 第 28~33 行的 onhint() 函数获取事件类型，当为 blur 类型并且过滤空格后的值为空时，才会重新显示提示性信息（即新增的 span 元素）。获取焦点之前的效果如图 2-19 所示，获取焦点之后的效果如图 2-20 所示。

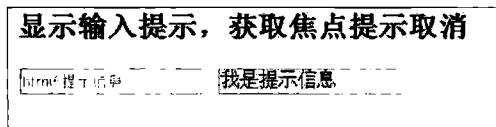


图 2-19 获取焦点之前

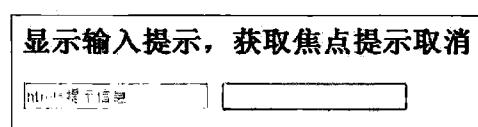


图 2-20 获取焦点之后

## 2.12 文本框内容自动滚动

显示屏的大小会限制页面的大小，在有限的面积中，怎样才能显示比较多的内容？例

如：新闻类的网站，每天有好几百条最新的标题内容，但是页面只能显示出10条。解决方案就是改变显示方式，文本自动滚动就是一个不错的选择。本例主要的思路就是修改 top 值，当然前提是要有符合结构的 CSS 与 HTML。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var rollContent = document.getElementById("rollContent"), //获取元素
04 _div = rollContent.innerHTML,
05 setCss = function(_this, cssOption){ //设置样式
06 //检测节点类型
07 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style)
08 {
09 return;
10 }
11 for(var cs in cssOption){
12 _this.style[cs] = cssOption[cs];
13 }
14 return _this;
15 };
16 rollContent.innerHTML = "<div id='rollContent_roll'>"+_div+"</div>";
17 setCss(rollContent, { //初始化样式
18 "position":"relative", //相对位置
19 "overflow":"hidden", //默认隐藏
20 "wordWrap":"break-word",
21 "wordBreak":"break-all",
22 "width":rollContent.getAttribute("data-rwidth"),
23 "height":rollContent.getAttribute("data-rheight")
24 });
25 var timeRoll = document.getElementById("rollContent_roll"),
26 _h = timeRoll.offsetHeight;
27 timeoutRoll = function(){ //修改 top 值
28 var _h = timeRoll.offsetHeight,
29 _t = parseInt(timeRoll.style.top, 10),
30 //是否将 top 设置为 0
31 _tt = _h > Math.abs(_t) || _t >= 0 ? _t-10: (_h || 0);
32 setCss(timeRoll, { //修改样式
33 "top":_tt+"px"
34 });
35 setTimeout(timeoutRoll,200); //定时调用，模拟动画
36 };
37 if(_h > rollContent.getAttribute("data-rheight")){ //判断滚动高度是否大于 data-rheight
38 timeoutRoll();
39 setCss(timeRoll, {

```

```

41 "position":"relative", //设置为相对浮动
42 "top":"0px"
43 });
44 }
45
46 };
47 </script>

```

本例 HTML 代码如下：

```

01 <h2>文本框中的文字自动滚动效果</h2>
02 <div id='rollContent' data-rwidth='100' data-rheight='100' class='rollContent'>
03 111111111111
04 222222222222
05 333333333333
06 333333333333
07 333333333333
08 333333333333
09 </div>

```

#### 【代码说明】

JavaScript 代码第 27~36 行创建 timeoutRoll() 函数处理高度变化的事件。JavaScript 代码第 29 行中 “timeRoll.style.top” 的值为空字符串时会得到 “\_t=NaN”，所以初始化时判断 \_t 是否为 NaN，如果是就设置 \_t 为 0。默认的业务逻辑是递减 \_t，当递减的值达到 \_h 临界点时，就修改 \_t 为 \_h。调用 setTimeout() 不断地循环这个事件，用户就会看到字幕滚动的效果了。

**说明：**NaN 是 JavaScript 中的一种特殊数据类型。

## 2.13 密码强度实时验证

在网络服务中，为了保证用户的私密信息足够安全，会要求用户输入具有一定安全级别的密码，这样可以更好地防止他人盗用。比如注册一些游戏账号时，如果输入纯数字或纯英文字符低于 6 位，就会提示密码强度太低，请重新输入。一般密码强度验证的方式，都是计算字符的类型，然后分类加权累算。权重越高，相应的强度也就越高。具体的验证写法及实现方式有很多种，本节只介绍比较流行的方案。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function setCss(_this, cssOption){ //设置样式
04 //判断节点类型
05 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
06 return;
07 }
08 for(var cs in cssOption){
09 _this.style[cs] = cssOption[cs];

```

```

10 }
11 return _this;
12 }
13
14 function trim(chars){ //去除字符串左右两边的空格
15 return (chars || "").replace(/^\s|\u00A0)+|(\s|\u00A0)+$/g, "");
16 }
17
18 function passwordStrength(passwordStrength, showStrength){
19 var self = this;
20
/*字符权重：
21 数字 1，字母 2，其他字符为 3
22 当密码长度小于 6 时不符合标准；
23 长度>=6，强度小于 10，强度弱；
24 长度>=6 强度>=10 且 <15，强度中；
25 长度>=6 强度>=15 强*/
26 passwordStrength.onkeyup = function(){
27 var _color = ["red", "yellow", "orange", "green"],
28 msgs = ["密码太短","弱","中","强"],
29 _strength= 0,
30 _v = trim(passwordStrength.value)
31 _vL = _v.length,
32 i = 0;
33
34 var charStrength = function(char){ //计算单个字符强度
35 if (char>=48 && char <=57){ //数字
36 return 1;
37 }
38 if (char>=97 && char <=122) { //小写
39 return 2;
40 }else{
41 return 3; //特殊字符
42 }
43 }
44
45 if(_vL < 6){ //计算模式
46 showStrength.innerText = msgs[0];
47 setCss(showStrength, {
48 "color":_color[0]
49 })
50 }else{
51 for(; i < _vL ; i++){ //遍历字符
52 _strength+=charStrength(_v.toLocaleLowerCase().charCodeAt(i));
53 }
54 if(_strength < 10){ //小于 10 的强度

```

```

55 showStrength.innerText = msgs[1];
56 setCss(showStrength, {
57 "color":_color[1] //设置颜色
58 })
59 }
60 if(_strength >= 10 && _strength < 15){ //大于 10 且小于 15 的强度
61 showStrength.innerText = msgs[2];
62 setCss(showStrength, {
63 "color":_color[2]
64 })
65 }
66 if(_strength >= 15){ //大于 15 的强度
67 showStrength.innerText = msgs[3];
68 setCss(showStrength, {
69 "color":_color[3]
70 })
71 }
72 }
73 }
74 }
75 passwordStrength(
76 document.getElementById("passwordStrength"),
77 document.getElementById("showStrength"));
78);
79 </script>

```

本例 HTML 代码如下：

```

01 <h2>密码强度实时验证</h2>
02 <input id="passwordStrength" data-hint='请输入密码' type="password">

```

#### 【代码说明】

- 本例主要涉及两个元素：被绑定元素和显示元素，JavaScript 代码第 75~78 行调用 `passwordStrength()` 实现两个元素之间的绑定。
- 在 `passwordStrength()` 函数中，JavaScript 代码第 34~43 行将单个字符占的比重定义为权重，权重越高相应的累加结果就越高，强度也就越高。
- `keyup` 事件中，JavaScript 代码第 45~71 行首先判断字符长度，小于 6 为不符合规定的最小长度，显示“密码太短”；符合条件的字符采用遍历的方式，用 `charCodeAt()` 返回字符的 Unicode 编码，根据编码范围返回对应权重。将字符的所有权重累加，计算最后的强度范围，显示对应的颜色及值，效果如图 2-21 所示。

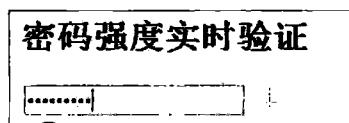


图 2-21 密码强度验证

## 2.14 回车提交表单

填写完表单数据后，很多用户喜欢直接按回车键提交，感觉速度比较快，省去了拿鼠标找“提交”按钮再单击的时间。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 document.getElementById("enterSubmit").onkeyup = function(e){
04 e = e || window.event; //获取事件对象
05 var keycode = e.keyCode || e.which || e.charCode; //获取键码
06 if(keycode === 13){ //判断键码是否为 13(回车键)
07 alert("回车提交成功");
08 }
09 }
10 };
11 </script>

```

本例 HTML 代码如下：

```

01 <h2>回车提交</h2>
02 <input type="text" id="enterSubmit" value="回车提交">

```

### 【代码说明】

- JavaScript 代码第 03~09 行为元素 `enterSubmit` 绑定 `keyup` 事件。
- JavaScript 代码第 04 行检测用户的浏览器响应事件是否含有 `e`，如果没有则赋值“`window.event`”。
- 因为不同的浏览器对键码的处理方式不同，因此 JavaScript 代码第 05 行依次检测“`e.keyCode || e.which || e.charCode`”，当响应的键码为 13 时就表示用户按下了回车键。本例为了检测是否准确，用 `alert()` 输出结果。

## 2.15 光标停留在文字最后

在文本框中，有个一闪一闪的竖行“|”标志，这就是光标。在微博、QQ 空间中，当我们选择要@的人之后，系统会让光标停留在被@人的名字后面，这样方便继续输入其他文本。目前，几乎所有的主流浏览器都提供了控制光标位置的 API。本例效果如图 2-22 所示。

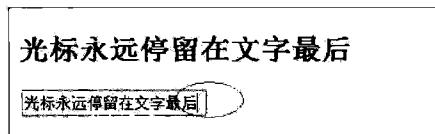


图 2-22 画圈部分为光标位置

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){ //页面载入后执行
03 //绑定两种事件，以达到最大兼容性
04 var cursorPos = document.getElementById("cursorPos");
05
06 cursorPos.onclick = cursorPos.onkeyup = function(){//绑定处理事件的函数
07 var _vLen = this.value.length;
08 if(this.setSelectionRange){ //非 IE
09 this.setSelectionRange(_vLen,_vLen);
10 }else{ //IE
11 var a =this.createTextRange();
12 a.moveStart('character',_vLen);
13 a.collapse(true);
14 a.select(); //选中操作
15 }
16 }
17 };
18 </script>

```

本例 HTML 代码如下：

```

01 <h2>光标永远停留在文字最后</h2>
02 <input type="text" id="cursorPos" value="光标永远停留在文字最后">

```

#### 【代码说明】

- 有两种交互状态会触发光标的位置显示：第 1 种是获取焦点时；第 2 种是敲击键盘上的按键时。直接绑定 focus（获取焦点事件）会有浏览器的兼容性问题，所以 JavaScript 代码第 06 行采用了绑定 click 和 keyup 的方式，click 事件针对的是鼠标操作，keyup 事件针对的是键盘操作。
- IE 浏览器支持一个不常用的对象 createTextRange，可用来设置光标的 position；在非 IE 浏览器中，通过对对象的 setSelectionRange() 也可以设置光标的 position。

## 2.16 禁止文本框的记忆功能

IE 及一些其他的主流浏览器，会自动记忆表单中输入过的文本，当用户再次输入类似文本时，会自动完成输入并提示用户。这种设计让用户可以更快速地输入文本。但有时在一些特殊情况下，并不需要自动完成的功能，如输入银行账号、游戏账号等敏感或安全性较高的文本信息时。

通过以下 3 种方式可以实现禁用文本框自动完成的功能：

(1) 利用浏览器本身的设置。这里以 IE 为例，其他浏览器有类似设置，请以此参考。单击“工具”→“Internet 选项”菜单，打开“Internet 选项”对话框，在“内容”选项卡中可以设置自动完成功能，如图 2-23 所示。

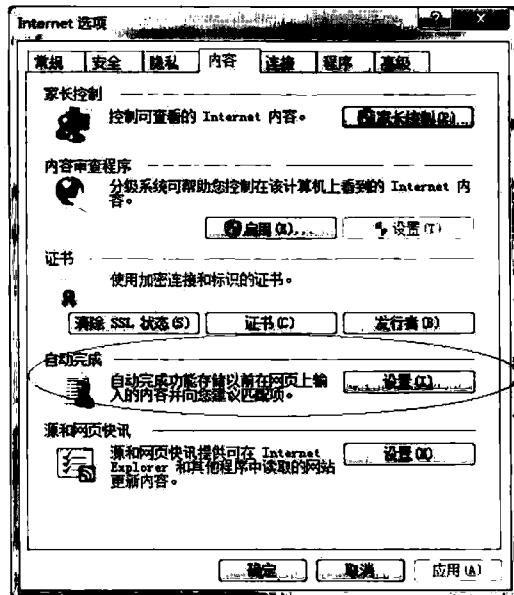


图 2-23 “内容”选项卡

(2) 利用 HTML 元素的属性控制 autocomplete, 代码如下:

```
01 <input type="text" id="banAutocomplete2" autocomplete="off" />
```

(3) 使用 JavaScript 控制, 本质上也是设置 autocomplete, JavaScript 代码如下:

```
01 <script type="text/javascript">
02 window.onload = function(){ // 页面载入后执行
03 // 设置属性
04 document.getElementById("banAutocomplete").setAttribute("autocomplete", "off");
05 };
06 </script>
```

本例 HTML 代码如下:

```
01 <h2>不允许文本框自动完成</h2>
02 <form action="">
03 <input type="text" id="banAutocomplete" value="禁止自动完成">
04 <input type="text" id="banAutocomplete2" autocomplete="off" />
05 </form>
```

#### 【代码说明】

JavaScript 代码首先获取相关元素, 然后设置 autocomplete 属性为 off。另外, 如果想禁用表单自动完成, 也可以在 form 元素上设置 autocomplete。

## 2.17 自动选定文本内容

有时网页想让用户默认选中一些文本内容, 方便用户操作。例如: 鸡汤类网站提供了“每日一句”, 这句话可以复制或分享, 如果让我们手动复制那就太麻烦了, 而如果不用

做任何操作就能直接选定内容，岂不美哉？

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 //调用浏览器内置的文本选定函数
04 document.getElementById("autoSelected").select();
05 };
06 </script>
```

本例 HTML 代码如下：

```
01 <h2>自动选定 TextArea 内容</h2>
02 <textarea id='autoSelected' rows="10" cols="50">
03 默认被选择的文本
04 </textarea>
```

#### 【代码说明】

JavaScript 代码首先获取待处理的元素对象，然后调用 `select()` 函数。目前所有主流浏览器，都有一个 `select()` 函数，可以选取 `textarea` 中的文本。

## 2.18 获取和失去焦点时改变样式

随着大数据的到来，网站希望能保存用户更多的信息，所以一些表单上的各种输入框真是百花齐放，亮瞎了用户的眼。为了让用户能够清楚地知道现在输入的是哪一项，我们可以改变焦点所在的输入框的样式。一般有两种方案来修改元素的样式，即修改“`class`”与“行内样式”。本例开发一个函数支持这两种方式。

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 var
04 strToJson = function(str){ //将字符转换为 JSON 对象
05 return typeof JSON == "object" ? JSON.parse(str) : (new Function("return " +
06 str))();
07 },
08 setCss = function(_this, cssOption){ //设置样式
09 //检测节点类型
10 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
11 return;
12 }
13 for(var cs in cssOption){ //遍历所有样式并设置
14 _this.style[cs] = cssOption[cs];
15 }
16 return _this;
17 },
18
```

```

18 autoUpdateCss = document.getElementById("autoUpdateCss"),
19 fCss = autoUpdateCss.getAttribute("data-fCss"),
20 fClass = autoUpdateCss.getAttribute("data-fClass"),
21 bClass = autoUpdateCss.getAttribute("data-bClass"),
22 bCss = autoUpdateCss.getAttribute("data-bCss");
23
24 autoUpdateCss.onfocus = function(){ //获取焦点之后的样式
25 fCss && setCss(this, strToJson(fCss));
26 fClass && (this.className = fClass);
27 }
28
29 autoUpdateCss.onblur = function(){ //失去焦点之后的样式
30 bCss && setCss(this, strToJson(bCss));
31 bClass && (this.className = bClass);
32 }
33 };
34 </script>

```

本例 HTML 代码如下：

```

01 <h2>获取、失去焦点时改变样式</h2>
02 <input type="text" value='修改样式' data-fClass='fboder' data-bClass='bboder'
03 data-fCss='{"color":"red"}' data-bCss='{"color":"green"}' id='autoUpdateCss' />

```

#### 【代码说明】

- JavaScript 代码第 04~07 行使用 `strToJson()` 将字符串转换为 JSON 对象，默认会检测浏览器是否支持 JSON，如果支持就直接调用内置的转换函数，否则利用函数的特性转换。
- 在响应元素上绑定数据。JavaScript 代码第 19~22 行的 `data-fCss` 表示获取焦点时的样式，`data-bCss` 表示失去焦点时的样式；`data-fClass` 是获取焦点类，`data-bClass` 是失去焦点类。
- 为元素绑定事件，当事件触发时直接修改 `className`，调用 `setCss()` 修改行内样式。

## 2.19 常见的验证规则

所有的数据都会保存在服务器端，这些数据都有一定的规则，如：姓名不能超过 4 个字、密码不能小于 6 位数等等。用户在输入姓名后，我们需要判断用户的输入是否符合要求，如果能在用户输入完一个字段后就给出提示，用户可以及时修改。验证用户输入的这个工作一直以来都由 JavaScript 完成。本节开始探讨一些常见的验证规则，例如：验证用户输入的是否为中文、是否为数字、是否为邮件等等。

在验证领域，正则是非常重要的基础知识，JavaScript 为开发人员提供了一个强大的正则模式匹配对象 `RegExp`。创建正则的方式有两种，如下简述。

(1) 直接表达式法，代码如下：

```
01 //样板：/模式/扩展属性
```

02 \d\*/g

(2) 创建 RegExp, 代码如下:

01 new RegExp("ab", "l"); //样板: new RegExp(模式, 扩展属性)

正则的特殊符号, 不再一一讲解, 读者可以参考其他学习资料。JavaScript 包含很多正则函数, 其中 test() 函数是本例用到的, 用于检测字符串的匹配模式。

本例 JavaScript 代码如下:

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var getRegular = function(rstr){
04 var regData={}; //正则数据存储域
05 regData.trim = /^(\s|\u00A0)+|(\s|\u00A0)+$/g; //去除空格的正则
06 regData.Chinese = /[\u4e00-\u9fa5]/g; //中文
07 regData.nonumber = /\D/g; //数字
08 regData.nochniese = /^[^\u4e00-\u9fa5]/g; //非中文
09 regData.email=/^[\s*[a-zA-Z0-9]+(([._\-\?][a-zA-Z0-9]+)*[\s*$]/; //邮件
10 *@[a-zA-Z0-9]+([_\-][a-zA-Z0-9]+)*([a-zA-Z0-9]+([_\-][a-zA-Z0-9]+)*)+[\s*$]/; //电话
11 regData.phone = /^((\d{1,3})\d{2,3})-(\d{7,8})(-(\d{3,}))\d{0,}$/; //带小数位的数字
12 regData.decimalNumber = /\d+(\.\d+)$/; //带小数位的数字
13 regData.htmlTags = /<[\!\!]*["<>]*>/ig; //html
14
15 return regData[rstr];
16
17 },
18 forElementArr = function(_elementArr, callBack){ //遍历节点
19 var arr = _elementArr, //所有节点对象
20 self = this; //外层环境
21 ;
22
23 if(!_elementArr instanceof Array){ //如果不是数组, 变成数组对象方便处理
24 arr = [_elementArr];
25 };
26 for(var i= 0,arrLen = arr.length ;i<arrLen;i++){ //遍历数组
27 var arrl = arr[i];
28 if(typeof arrl == "string"){ //判断是否为字符串
29 arrl = document.getElementById(arrl);
30 }
31 callBack && callBack(i, arrl); //如果存在回调则执行回调
32 }
33 },
34 verification = function(str, reg){
35 return getRegular(reg).test(str);
36 },
37 setCss = function(_this, cssOption){
38 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {

```

```

39 return;
40 }
41 for(var cs in cssOption){
42 _this.style[cs] = cssOption[cs];
43 }
44 return _this;
45 };
46
47 forElementArr([
48 document.getElementById("regUser"),
49 document.getElementById("regEmail"),
50 document.getElementById("regPhone"),
51 document.getElementById("regNumber")
52], function(index, _this){
53
54 _this.onkeyup = function(){
55 //获取被处理的元素值
56 var _v = this.value.replace(/^(\s|\u00A0)+|(\s|\u00A0)+$/g, "");
57 _reg = this.getAttribute("data-reg"), //获取正则
58 //如果含有“,”则将其转换成数组
59 _reg = _reg.indexOf(",") > 0 ? _reg.split(",") : [_reg],
60 _regLen = _reg.length, //数组长度
61 _emsg = this.getAttribute("data-emsg"), //错误信息显示
62 _smsg = this.getAttribute("data-smsg"), //正确信息显示
63 //获取显示信息的元素
64 _target =
65 document.getElementById(this.getAttribute("data-tmsg")),
66 i = 0;
67 for(; i < _regLen ; i++){
68 if(!verification(_v, __reg[i])){
69 _target.innerHTML = _emsg ;
70 setCss(_target, {
71 "color":"red"
72 })
73 return;
74 }
75 }
76 _target.innerHTML = _smsg ;
77 setCss(_target, {
78 "color":"green"
79 })
80 }
81 });
82
83 </script>

```

本例 HTML 代码如下：

```

01 <h2>常见的验证规则</h2>
02 <p><input type="text" value='姓名验证' data-reg='Chinese' data-smsg='通过√' data-emsg='
03 请输入中文' id='regUser' data-tmsg='msgU' /></p>
04 <p><input type="text" value='邮箱验证' data-reg='email' data-smsg='通过√' data-emsg='请
05 输入邮箱' id='regEmail' data-tmsg='msgE' /></p>
06 <p><input type="text" value='电话验证' data-reg='phone' data-smsg='通过√' data-emsg='请
07 输入电话' id='regPhone' data-tmsg='msgP' /></p>
08 <p><input type="text" value='带小数位的数字验证' data-reg='decimalNumber' data-smsg='
09 通过√' data-emsg='请输入小数数字' id='regNumber' data-tmsg='msgN' /></p>
```

#### 【代码说明】

- 本例演示 4 种验证规则供读者参考，如果读者感兴趣，可以自己增加一些正则试验一下，只需要获取指定的元素绑定 `keyup` 事件即可。
- JavaScript 代码第 34~36 行增加正则验证函数 `verification()`。
- 在事件业务处理中，首先初始化一些需要的信息变量，然后遍历正则数组。

## 2.20 对文本内容进行关键词过滤

网络中的信息有一些是有害的，比如：一些危害国家安全的舆论、不良的色情信息等，因此我们经常需要对网络信息进行屏蔽或过滤。过滤信息一般有禁止输入、信息替换（如用“\*”号替换）、直接删除等方式。这些信息过滤业务的处理一般在后台完成，如果将其转移到前端来完成，就可以降低后台的压力。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var _keyWordsFiltering = document.getElementById("keyWordsFiltering");
04 _keyWordsFiltering.onclick =function(){
05 var //关键词库
06 keyWordsLibs = [
07 "JavaScript",
08 "美女",
09 /[外]{1}.{0,3}[挂]{1}/
10],
11 keyWordsLibsLen = keyWordsLibs.length; //关键词长度
12 for(var i = 0 ; i < keyWordsLibsLen; i++){//正则过滤
13 _keyWordsFiltering.value =
14 _keyWordsFiltering.value.replace(keyWordsLibs[i], "****")
15 }
16 }
17 };

```

```
18 </script>
```

本例 HTML 代码如下：

```
01 <h2>对文本内容进行关键词过滤</h2>
```

```
02 <textarea id='keyWordsFiltering' rows="10" cols="50">
```

```
03 关键词过滤：你好，我是席新亮，非常喜欢 JavaScript，希望和大家切磋交流。业余爱好：
```

```
04 玩游戏，但是不使用外挂，游戏里面有很多美女，咱也不贪色，嘿嘿~~~
```

```
05 </textarea>
```

```
06 获取焦点执行过滤
```

#### 【代码说明】

对于关键词过滤的大致思路是，首先 JavaScript 代码第 06~10 行建立被过滤的“关键词词库”，然后以“词库”为基础，进行关联性匹配替换或删除。本例代码创建了 keyWordsLibs 词库，将文本内容中的待过滤关键词，统一替换为“\*\*\*”，效果如图 2-24 与图 2-25 所示。

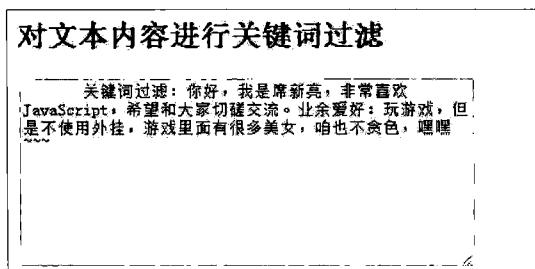


图 2-24 过滤内容之前

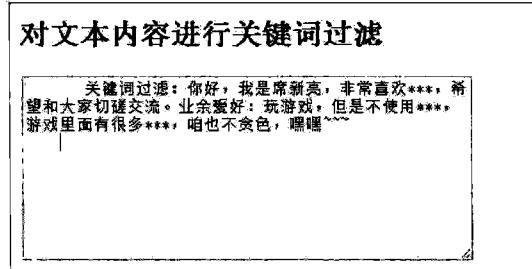


图 2-25 过滤内容之后

## 2.21 从字符串中剔除所有 HTML 代码

一些 HTML 代码会干扰程序地处理和显示，例如抓取微博页面内指定标签的数据时，有时候会把一些 HTML 代码也抓取过来，那么这些数据就需要进一步过滤，要剔除 HTML 代码的干扰。剔除字符串中的 HTML 代码与上节中的过滤关键词有很大的相似性，只要对上节代码稍作改动就可以剔除 HTML 代码了。

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 var _delHtmlTags = document.getElementById("delHtmlTags"); //获取元素对象
04 _delHtmlTags.onblur =function()//失去焦点
05 this.value = this.value.replace(/<[^\!]*[^<>]*>/ig, ""); //过滤字符
06 }
07 };
08 </script>
```

本例 HTML 代码如下：

```
01 <h2>从字符串中剔除所有 HTML 代码</h2>
```

```
02 <input type="text" id='delHtmlTags' value="剔除所有 HTML 代码">
```

```
03 失去焦点时执行操作
```

### 【代码说明】

- JavaScript 代码第 05 行剔除字符串中的 HTML 代码，重点在于正则。HTML 代码的开头与结尾都有类似的标记“<”或“>”，因此不难写出与其匹配的正则式“/<[\\!]\*[^>]\*>/”。为了兼容大小写，再在正则后边增加 ig，修改后的正则为“/<[\\!]\*[^>]\*>/ig”。
- 为了方便查看效果，JavaScript 代码第 04~06 行为响应元素绑定 blur 事件，当事件被触发时，利用 JavaScript 的 replace() 替换 HTML 标签为空，效果如图 2-26、图 2-27 所示。

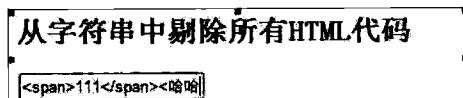


图 2-26 剔除 HTML 代码之前

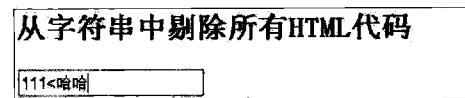


图 2-27 剔除 HTML 代码之后

## 2.22 检测是否为数值型

数值类型是 JavaScript 中的基本数据类型之一，在进行一些数学运算时非常有用。例如：计算购物车中的商品总共多少钱，玩游戏时充值多少游戏币等。typeof 是个非常有用的一元运算符，可以用来检测数据类型。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var isNumber = function(_number){
04 return typeof _number == "number";
05 },
06 _number1 = "1",
07 _number2 = 1,
08 _number3 = 1.22,
09 _number4 = null,
10 _html = "";
11 _html += isNumber(_number1) ? "_number1 是数值型
" : "_number1 不是数值型
12
";
13 _html += isNumber(_number2) ? "_number2 是数值型
" : "_number2 不是数值型
14
";
15 _html += isNumber(_number3) ? "_number3 是数值型
" : "_number3 不是数值型
16
";
17 _html += isNumber(_number4) ? "_number4 是数值型
" : "_number4 不是数值型
18
";
19 document.getElementById("isNumber").innerHTML = _html;
20 };
21 </script>

```

本例 HTML 代码如下：

```
01 <h2>测试是否为数值型</h2>
02 <div id='isNumber'>
03 </div>
```

#### 【代码说明】

- `typeof` 属于一元运算符，可以检测数据的任意类型，返回 `number`、`boolean`、`string`、`function`、`object`、`undefined`。也可以使用 `typeof` 来判断变量是否存在，防止浏览器解析报错。
- JavaScript 代码第 03~05 行使用 `typeof` 检测输入值是否为 `number`。为了验证检测是否准确，本例列举了 4 种输入类型，检测效果如图 2-28、图 2-29 所示。

```
var _number1 = "1",
 _number2 = 1,
 _number3 = 1.22,
 _number4 = null,
```

图 2-28 被测试的数据

**测试是否为数值型**

```
_number2不是数值型
_number2是数值型
_number3是数值型
_number4不是数值型
```

图 2-29 输出的结果

## 2.23 TextArea 自适应文字行数

为了节约页面空间，TextArea 的文字行数一般会随着文字数量的变化而变化，例如，默认的 TextArea 为 1 行，随着输入文字的增多，文本框的行数也会增多。

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 var autoRow = document.getElementById("autoRow");
04 autoRow.style.overflowY = "hidden"; //Y 轴是否超过隐藏
05 autoRow.onkeyup= function(){ //键盘事件
06 autoRow.style.height = autoRow.scrollHeight;
07 };
08 };
09 </script>
```

本例 HTML 代码如下：

```
01 <h2>TextArea 自适应文字行数多少</h2>
02 <textarea rows=1 name=s1 cols=27 id="autoRow"></textarea>
```

#### 【代码说明】

- JavaScript 代码第 04 行通过 CSS 的 `overflowY` 可以很好地控制元素的高度与滚轴地显示。
- JavaScript 代码第 05~07 行绑定 `keyup` 事件，当事件被触发时，动态修改元素的高度，让其等于元素的卷轴高度 `scrollHeight`。

## 2.24 判断单选框是否选中

注册账号时，一般情况下需要选择性别，选择的 UI 表现形式就是单选框。那么，JavaScript 怎样判断单选框是否选中呢？单选框有一个特殊属性 `checked`，使用 JavaScript 检测该属性值的变化就能判断单选框是否被选中。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var sexMan = document.getElementById("sexMan"), //获取元素的节点对象
04 sexWoman = document.getElementById("sexWoman");
05 if(sexMan.checked){ //判断是否被选中
06 console.log("sexMan 被选中");
07 }else{
08 console.log("sexMan 未被选中");
09 }
10
11 if(sexWoman.checked){ //判断是否被选中
12 console.log("sexWoman 被选中");
13 }else{
14 console.log("sexWoman 未被选中");
15 }
16 };
17 </script>
```

本例 HTML 代码如下：

```

01 <h2>判断单选框是否选中</h2>
02 <input type="radio" name='sex' id='sexMan' checked="checked">男

03 <input type="radio" name='sex' id='sexWoman'>女
```

### 【代码说明】

JavaScript 代码第 05~15 行通过判断元素的 `checked` 属性值来判断单选框是否被选中。本例在控制台中打印的测试结果如图 2-30 所示。



图 2-30 输出的结果

## 2.25 判断复选框至少选中一项

电子邮箱的容量是有限的，有时需要选择、删除一些垃圾邮件，如果没有选中要删除

的邮件，单击“删除”按钮时，就会提示用户至少要选中一项，这是怎么实现的呢？结合上一节判断是否选中的思路，本例通过遍历节点的方式来实现需求。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 //获取待选择的对象
04 var _checkSelects = document.getElementsByName("checkSelects");
05 for(var i in _checkSelects){ //遍历所有 check 元素
06 if(_checkSelects[i].checked){ //判断是否选中
07 console.log("至少选中了一项");
08 return;
09 };
10 }
11 console.log("没有选中");
12 };
13 </script>
```

本例 HTML 代码如下：

```

01 <h2>判断复选框至少选中一项</h2>
02 <input type="checkbox" name='checkSelects' checked='checked'>1

03 <input type="checkbox" name='checkSelects'>2

04 <input type="checkbox" name='checkSelects'>3

05 <input type="checkbox" name='checkSelects'>4

```

#### 【代码说明】

JavaScript 代码第 05~10 行遍历元素节点，因为判断单选框是否被选中与判断复选框是否被选中的方法一样，所以利用上一节的函数“`_checkSelects[i].checked`”来判断是否有任何一个元素被选中。如果有就停止遍历，这样可以减少性能开支。

## 2.26 限制复选框最多选择几项

在一个招聘网站中，通过限制用户选择职位标签的个数，可以精确定位用户的职位。例如，以复选框的形式为用户提供一些备选职位标签，限制用户最多选取 3 个，当超过 3 个时禁止用户继续选择。本例通过交互事件来控制复选框的选择。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var _forbidcheckSelects = document.getElementsByName("forbidcheckSelects"),
04 banNums = 3; //限制复选框最多选择 3 项
05 for(var i in _forbidcheckSelects){ //遍历所有元素
06 _forbidcheckSelects[i].onclick = function(){
07 var
08 _forbidcheckSelects =
```

```

09 document.getElementsByName("forbidcheckSelects"),
10 selectNum = 0; //初始化选择的数目为 0
11 for(var i in __forbidcheckSelects){
12 if(i == "length") break;
13 if(__forbidcheckSelects[i].checked){如果选中++
14 selectNum++;
15 }
16 }
17 //如果选中的复选项超过限制的最大数，将当前选项设置为未选中
18 if(selectNum > banNums) {
19 this.checked = false;
20 }
21 }
22 }
23
24 }
25 };
26 </script>

```

本例 HTML 代码如下：

```

01 <h2>限制复选框最多选择几项</h2>
02 <input type="checkbox" name='forbidcheckSelects'>1

03 <input type="checkbox" name='forbidcheckSelects'>2

04 <input type="checkbox" name='forbidcheckSelects'>3

05 <input type="checkbox" name='forbidcheckSelects'>4

06 <p>最多选择 3 项</p>

```

#### 【代码说明】

使用 JavaScript 中的 “`document.getElementsByName()`” 获取指定的复选框，当触发单击事件时，JavaScript 代码第 18~20 行检测被选中的选项卡是否超过了限制的最大数，如果超过最大数，则将当前的响应元素设置为未被选中状态。

## 2.27 复选框全选、取消全选、反选

越来越多的人有了选择恐惧症，“治疗”的最好方式就是全选或全不选。常用的邮箱每页可以显示 20 封邮件，如果要删除全部邮件，一个一个的去选是不是有点太逆天了？为了降低操作成本、提升用户体验，所有邮箱都提供了“全选”功能。本例要实现 3 个简易操作：全选、取消全选、反选。

前面介绍过，通过操控复选框的属性 `checked` 可以设置复选框的选中状态。结合本例的需求，JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var targets = document.getElementsByName("actionSelects"), //获取目标元素

```

```

04 targetsLen = targets.length, //目标元素的个数
05 i = 0;
06
07 document.getElementById("allSelect").onclick = function(){ //绑定单击事件，全选
08 for(i = 0 ;i < targetsLen ; i ++){
09 targets[i].checked = true;
10 }
11 }
12
13 document.getElementById("cancelallSelect").onclick = function(){//绑定单击事件，取消
14 for(i = 0 ;i < targetsLen ; i ++){
15 targets[i].checked = false;
16 }
17 }
18
19 document.getElementById("_select").onclick = function(){ //绑定单击事件，反选
20 for(i = 0 ; i < targetsLen ; i ++){
21 targets[i].checked = !targets[i].checked; //选择取反即可
22 }
23 }
24 }
25 };
26 </script>

```

本例 HTML 代码如下：

```

01 <h2>Checkbox 全选、取消全选、反选</h2>
02 <p><input type="button" id='allSelect' value="全选"><input type="button" id='cancelallSelect'
03 value="取消全选"><input type="button" id='_select' value='反选'> </p>
04 <input type="checkbox" name='actionSelects'>1

05 <input type="checkbox" name='actionSelects'>2

06 <input type="checkbox" name='actionSelects'>3

07 <input type="checkbox" name='actionSelects'>4


```

#### 【代码说明】

整体思路：当单击“全选”按钮时，遍历所有的元素，将元素的 checked 设置为 true；当单击“取消全选”按钮时，将所有元素的 checked 设置为 false；当单击“反选”按钮时，对元素的 checked 属性执行取反操作。

## 2.28 根据指定内容选中复选框

本例的目的是根据内容中的关键词选中符合要求的复选框。例如当用户在文本中输入的内容与指定的复选框关键词匹配时，就会选中该复选框，本例效果如图 2-31 所示。

## 根据指定内容选中复选框

如果内容中含有关键词：“JavaScript”、“席新亮”、“游戏”，则相应的关键词复选框会被选中

The screenshot shows a text area with the text "席新亮". Below it are three checkboxes:

- JavaScript (checked)
- 席新亮
- 游戏

图 2-31 被选中的复选框在内容中存在对应关键词

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 //获取内容元素
04 var contentCheckbox = document.getElementById("contentCheckbox"),
05 _targets =
06 document.getElementsByName(contentCheckbox.getAttribute("data-target")),//待选中的元素
07 targetsLen = _targets.length, //元素个数
08 i = 0;
09 contentCheckbox.onkeyup = function(){
10 for(i = 0 ; i < targetsLen ; i ++){
11 var _t = _targets[i],
12 _v = this.value;
13 /*如果内容与复选框的关键词匹配，选择当前的复选框*/
14
15 /*_v.search(_t.getAttribute("data-k")) != -1 ?
16 _t.checked = true : //三元运算写法
17 _t.checked = false;*/
18
19 //单链式条件写法
20 _t.checked = _v.search(_t.getAttribute("data-k")) != -1 && true || false;
21 }
22 }
23 };
24 </script>
```

本例 HTML 代码如下：

```
01 <h2>根据指定内容选中复选框</h2>
02 <p>如果内容中含有关键词：“JavaScript”、“席新亮”、“游戏”，则相应的关键词复选框
03 会被选中</p>
04 <textarea id='contentCheckbox' data-target='autoKeywordSelect'>
05
06 </textarea>
07

08 <input type="checkbox" name='autoKeywordSelect' data-k='JavaScript'>JavaScript

09 <input type="checkbox" name='autoKeywordSelect' data-k='席新亮'>席新亮

10 <input type="checkbox" name='autoKeywordSelect' data-k='游戏'>游戏

```

### 【代码说明】

- JavaScript 代码首先获取响应元素，在响应元素中绑定目标元素 `data-target` 作为响应的复选项卡。当触发 JavaScript 代码第 09~22 行的 `keyup` 事件时，遍历目标复选项，如果内容中存在与当前复选项匹配的关键词，则将当前复选框变为选中状态。
- 与传统的 if 式语句相比，“三元”式写法也可以实现同样的功能，这最终取决于项目代码的风格，相对而言，“单链”式写法更干净一些。

**提示：**给读者留份小小的练习，如何根据指定内容选定下拉框。

## 2.29 获取选中的复选框值

在新浪微博注册引导流程中，会要求初次注册的用户选择一些关注用户，当提交表单数据时，系统获取所有被选中的值。要想获取被选中“复选框”的“值”，必须在两者之间建立关联性。本例效果如图 2-32 所示。



图 2-32 选中两个“复选框”后显示的“值”

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var selectContents = "", //待填充的文本
04 _selectContent = document.getElementsByName("getSelectContent"),
05 i = 0,
06 sl = _selectContent.length; //获取元素的长度，即个数
07
08 for(; i < sl ; i++){ //限制复选框最多选择 3 项
09 _selectContent[i].onclick = function(){
10 var _t = this.nextSibling.innerText;
11 if(this.checked){ //如果内容被选中，则填充文本，否则删除
12 selectContents += "
" + _t;
13 }else{
14 selectContents = selectContents.replace("
" + _t, "");
15 }
16 document.getElementById("selectedContents").innerHTML = "被选择的内
17 容：" + selectContents; //填充 HTML 文本
18 }
19 }
```

```
20 }
21);
22 </script>
```

本例 HTML 代码如下：

```
01 <h2>获取复选框所有选中内容</h2>
02
03
04 <input type="checkbox" name='getSelectContent'><div class="contentCheckbox">
05 等待选择的内容 · · · 1</div>
06
07
08 <input type="checkbox" name='getSelectContent'><div class="contentCheckbox">
09 等待选择的内容 · · · 2</div>
10
11
12 <input type="checkbox" name='getSelectContent'><div class="contentCheckbox">
13 等待选择的内容 · · · 3</div>
14
15
16 <p id='selectedContents'>被选择的内容：选择内容为空</p>
```

本例 CSS 代码如下：

```
01 <style>
02 .contentCheckbox{
03
04 text-align: center;
05
06 }
07 ul li{
08 list-style: none;
09 display: block;
10 border: 1px #ccc solid;
11 width: 300px;
12 height: 20px;
13 margin-top: 1px;
14 }
15 ul li input,ul li div{
16 position: relative;
17 float: left;
18 }
19 </style>
```

### 【代码说明】

JavaScript 代码第 08~20 行获取指定的复选框元素，为每个复选框元素绑定 click 事件。

JavaScript 代码第 03 行定义的变量 selectContents 用来存储内容。当 click 事件触发时，首先判断复选框是否被选中，如果选中则追加内容，否则删除内容。

**提示：**给读者留份小小的练习，如何获取单选框选中的内容。

## 2.30 判断下拉框中的值是否被选中

当用户需要选择的信息比较多时，展示所有信息的最好方式便是下拉框。例如：全国有好多省，通过下拉框的形式让用户选择目标省。如何判断下拉框是否被选择了呢？下拉框的选中状态，是通过下拉框中的 value 属性来判断的。本例效果如图 2-33 所示。

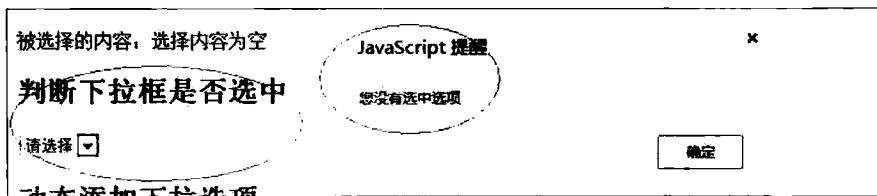


图 2-33 下拉框没有被选择的效果

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var _selectOptios = document.getElementById("selectOptios");
04 _selectOptios.onchange = function(){
05 if(this.value === ""){ //判断选项是否为空
06 alert("您没有选中选项"); //为空，弹出提示
07 }else{
08 alert(this.value); //不为空，弹出被选中的值
09 }
10 };
11 };
12 </script>

```

本例 HTML 代码如下：

```

01 <h2>判断下拉框是否选中</h2>
02 <select id="selectOptios">
03 <option value="">请选择</option>
04 <option value="1">选项 1</option>
05 <option value="2">选项 2</option>
06 <option value="3">选项 3</option>
07 <option value="4">选项 4</option>
08 <option value="5">选项 5</option>
09 </select>

```

### 【代码说明】

JavaScript 代码第 04 行为 select 元素绑定了 change 事件，该事件被触发时判断下拉框的选项值是否为空，如果为空就表示没有被选择。

## 2.31 动态添加下拉选项

在实际的网站运营中，很多数据都是动态的，包括一些下拉选项，今天是 3 个选项明天可能就变成 9 个选项了。例如：一些求职招聘的网站，每天都有新的公司加入，那么普通个人用户如果在注册时想在下拉框中选择自己的所属公司，这就需要下拉框内的选项也是变动的，此时就需要利用 JavaScript 程序来动态地添加选项了。所谓的动态添加下拉选项，实际上就是动态地操作 DOM。本例最终效果如图 2-34 所示。

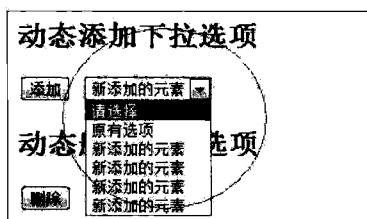


图 2-34 添加了 4 个新元素

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var _addOptions = document.getElementById("addOptions"),
04 _addOption = document.getElementById("addOption"),
05 addOptions = function(target, optons){//添加 option
06 var _option = null,
07 ol = optons.length,
08 i = 0,
09 _v = "",
10 _t = "";
11 for(; i < ol ; i++){
12 _v = optons[i].value; //设置 value 值
13 _t = optons[i].text; //设置文本值
14 //创建空 option 引用
15 _option = document.createElement("OPTION");
16 _option.value = _v; //添加值
17 _option.text = _t; //添加文本
18 target.options.add(_option); //增加 option
19 }
20 };
21 _addOptions.onclick = function(){ //单击事件，增加下拉选项
22 addOptions(_addOption, [
23 {
24 "value":"新添加的元素",
25 "text":"新添加的元素"
26 }
27);
28 }
29 }
30 }

```

```

27])
28
29 }
30);
31 </script>

```

本例 HTML 代码如下：

```

01 <h2>动态添加下拉选项</h2>
02 <input type="button" value='添加' id='addOptions' data-target='addOptios' />
03 <select id="addOption">
04 <option value="">请选择</option>
05 <option value="1">原有选项</option>
06 </select>

```

#### 【代码说明】

- JavaScript 代码第 05~20 行的 addOptions() 函数中参数 1 是被添加元素的 select，参数 2 是数组数据，为创建 option 提供数据基础。
- JavaScript 代码第 15~18 行，首先使用 document.createElement() 创建一个新的 option 元素，然后设置该元素的相关属性，最后将其追加到目标元素内。

## 2.32 动态删除下拉选项

在一些自定义下拉框列表中，用户可以删除下拉列表框的选项。JavaScript 提供了删除元素的 API——remove。本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var _removeOptions = document.getElementById("removeOptions");//获取元素对象
04 removeOptions = function(target, optons){ //删除 option
05 var ol = optons.length,
06 i = 0;
07 for(; i < ol ; i++){
08 //如果存在 option 对象，则调用 remove() 删除
09 target.options[i] && target.options.remove(target.options[i]);
10 }
11 };
12 _removeOptions.onclick = function(){ //绑定事件，执行删除
13 removeOptions(document.getElementById("removeOption"), [1, 2]);//删除 option
14 }
15 };
16 </script>

```

在 index.html 中绑定响应元素，相信读者对这个流程已经比较熟悉了，本例 HTML 代码如下：

```
01 <h2>动态删除下拉选项</h2>
```

```

02 <input type="button" value='删除' id='removeOptions' data-target='removeOptions' />
03 <select id="removeOption">
04 <option value="">请选择</option>
05 <option value="1">原有选项</option>
06 <option value="2">原有选项</option>
07 <option value="3">原有选项</option>
08 <option value="4">原有选项</option>
09 <option value="5">原有选项</option>
10 <option value="6">原有选项</option>
11 </select>

```

**【代码说明】**

- JavaScript 代码第 04 行的 removeOptions 参数说明：参数 1 绑定被删除 option 的目标元素；参数 2 绑定删除的序列。
- JavaScript 代码第 07~10 行首先遍历 select 目标元素的 option 子集，然后按照 removeOptions() 函数中第 2 个参数指定的数组序列，删除指定的选项。

## 2.33 下拉框二级联动效果

下拉框二级联动效果在日常应用场景中经常会碰到，尤其是涉及地区、品种等有多级选项时。例如：常见的省市联动下拉框，在选择省份时，城市列表也会跟随改变。本例效果如图 2-35 所示。

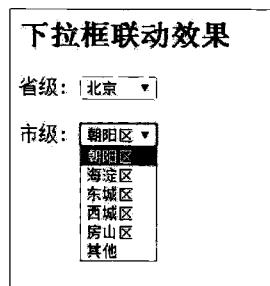


图 2-35 省市联动效果

本例只介绍二级联动的实现原理，其他的联动效果与此类似，JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 /*
04 * 联动效果，地区数据字典
05 * 为了方便用户查阅代码，将数据属性写在此处
06 * code 代表被绑定的二级菜单标示符
07 */
08 var linkDatas = { //地区字典
09 provinces:[//省级数据

```

```

10 {
11 "code":"0",
12 "name":"请选择"
13 },
14 {
15 "code":"1",
16 "name":"北京"
17 },
18
19 /*其他省级数据略 . . . */
20],
21 citys:{//城市数据
22 0:[
23 "请选择"
24],
25 1:[
26 "朝阳区",
27 "海淀区",
28 "东城区",
29 "西城区",
30 "房山区",
31 "其他"
32],
33 2:[
34 "天津"
35],
36 3:[
37 "沧州",
38 "石家庄",
39 "秦皇岛",
40 "其他"
41],
42
43 /*其他市级数据略……*/
44 }
45 };
46 function addOptions(target, optons){ //添加 option
47 var _option = null,
48 ol = optons.length,
49 i = 0,
50 _v = "",
51 _t = "";
52 for(; i < ol ; i++){
53 _v = optons[i].value; //获取 value
54 _t = optons[i].text; //获取 text

```

```
55 _option = document.createElement("OPTION");//创建 option
56 _option.value = _v; //设置 value
57 _option.text = _t; //设置 text
58 target.options.add(_option);
59 }
60
61 }
62
63 function linkage(parents, child){ //联动效果的业务处理
64 var _linkDatas = linkDatas,
65 _parents = _linkDatas.provinces, //获取省级数据
66 _childs = _linkDatas.cities, //获取市级数据
67 _initCity = _childs[0],
68 _p = [];
69 /*初始化数据*/
70
71 for(var i in _parents){ //省
72 _p.push({
73 "text" : _parents[i].name, //增加数据
74 "value" : _parents[i].code
75 });
76 }
77
78 addOptions(parents, _p); //添加选项
79
80 addOptions(childs,[{ //城市
81 "value":_initCity,
82 "text":_initCity
83 }]);
84
85 parents.onchange = function(){ //联动事件绑定及具体业务处理
86 var __childs = _childs[this.value], //寻找与省关联的市级数据
87 __childsLen = __childs.length,
88 i = 0,
89 __p = [];
90 childs.innerHTML = "";
91 for(; i < __childsLen ; i++){
92 __p.push({
93 "value" : __childs[i],
94 "text" : __childs[i]
95 });
96 }
97 addOptions(childs, __p); //添加联动的城市数据 option 选项
98 }
99 }
```

```

100 }
101
102 linkage(
103 document.getElementById("provinces"), //绑定一级菜单与二级菜单
104 document.getElementById("citys"));
105);
106 </script>

```

本例 HTML 代码如下：

```

01 <h2>下拉框联动效果</h2>
02 <p>
03 省级：
04 <select id="provinces">
05 </select>
06 </p>
07 <p>
08 市级：
09 <select id="citys">
10 </select>
11 </p>

```

#### 【代码说明】

- 所谓联动效果，是指触发父级的数据变化时，会影响到关联性子级数据元素的变化。
- 本例 JavaScript 代码第 08~44 行创建了一份“省级”与“城市”的关联性数据字典，其中的 code 表示“省级”指向“城市”的标示符，当省级的数据变更时，change 事件被触发，JavaScript 代码第 71~76 行就会依据 code 寻找二级菜单“城市”的相关数据。

## 2.34 可输入的下拉框

在一些要求比较高的应用场景中，下拉框不但可以选择还可以输入，这样可以让用户自己扩展可选择的下拉选项。例如：在用户注册时，选择自己的所属公司，但是网站提供的下拉选项中不存在公司名字，那么用户可以自己填写一个新的下拉选项，一方面帮网站扩展了公司数据库，另一方面让用户获得了更好的体验。本例结果如图 2-36 所示。

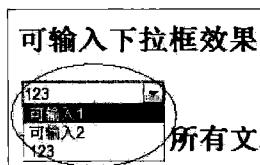


图 2-36 输入的值被添加进 select

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){

```

```

03 var _inOption = document.getElementById("inOption"),
04 _inInput = document.getElementById("inInput"),
05 addOptions = function(target, options){ //添加 option
06 var _option = null,
07 ol = options.length, //待添加的 option 个数
08 i = 0,
09 _v = "",
10 _t = "";
11 for(i < ol ; i++){
12 _v = options[i].value; //获取 value
13 _t = options[i].text;//获取文本值
14 _option = document.createElement("OPTION");//创建空 option
15 _option.value = _v; //设置 value
16 _option.text = _t; //设置文本值
17 target.options.add(_option);
18 }
19 };
20 _inInput.onkeyup = function(){ //绑定 keyup 事件
21 var key=event.keyCode,
22 _v = _inInput.value;
23 if(key == 13){ //如果为回车键，则添加值
24 addOptions(_inOption, [//添加新 option
25 {
26 "value":_v,
27 "text":_v
28 }
29])
30 }
31 }
32 }
33 _inOption.onchange = function(){ //值变更事件
34 _inInput.value = this.value;
35 }
36 }
37 };
38 </script>

```

本例 HTML 代码如下：

```

01 <h2>可输入下拉框效果</h2>
02 <select id="inOption" style="position:absolute;width: 118px;left:12px">
03 <option value="可输入 1">可输入 1</option>
04 <option value="可输入 2">可输入 2</option>
05 </select>
06 <input type="text" id='inInput' style='position: absolute;left: 11px;width: 100px;border-right:
07 0px'/'>


```

### 【代码说明】

- 本例包含了 input 元素和 select 元素，JavaScript 代码通过样式控制，调节 input 的位置与 select 重合并且覆盖 select，用户表面看上去，就如同可输入的 select。
- JavaScript 代码第 21~33 行为 input 绑定 keyup 事件，当按回车键时响应业务处理，为 select 添加新的 option 元素。
- JavaScript 代码第 34~36 行为 select 增加 change 事件，当 change 事件被触发时，修改 input 的值。

## 2.35 简单的数字及字符操作

在日常的程序中，经常会涉及数值与字符的操作，例如将一些英文标题转换为大写、将字符型转换为数值型等。本节介绍一些简单的处理方法，读者可以拿来即用。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 getLowerCase = function(str, type){ //小写转大写
04 type = type || "locale"; //是否采取本地转换格式
05 //返回转换后的值
06 return type === "locale" && str.toLocaleUpperCase() || str.toUpperCase();
07 }
08
09 getNumbers = function(s){ //字符型转换为数值型
10 var n = parseInt(s,10) //获取转换值
11 if(isNaN(n)) return 0; //如果为 NaN 则转换结果为 0
12 return n;
13 }
14
15 getString = function(n){ //数值型转换为字符型
16 return n.toString();
17 }
18 };
19 </script>
```

用 console.log() 检测以上函数是否有效，JavaScript 代码如下：

```

01 console.log(getLowerCase("ASDsssss2asdA")); //小写转大写
02
03 /*以下是字符与数字的转换*/
04 console.log(getNumbers("ASDsssss2asdA"));
05 console.log(getNumbers("123123ASDsssss2asdA"));
06 console.log(getNumbers("ASDsssss2asdA123"));
07
08 /*以下是数字转换成字符型*/
```

```
09 console.log(getString(123123)+"类型："+typeof getString(123123));
10 console.log(getString(234324)+"类型："+typeof getString(234324));
```

#### 【代码说明】

- 本例介绍了 3 个处理函数：小写转大写、字符转数字、数字转字符。
- JavaScript 代码第 03~07 行是小写转大写，首先会检测 type 参数，然后根据参数来选择相应的内置函数。
- JavaScript 代码第 09~13 行是将字符转换成数字，调用 JavaScript 的 parseInt() 函数进行转换，当被转换值为 NaN 时返回值为 0。
- JavaScript 代码第 15~17 行是将数字转换成字符，调用 JavaScript 中的 toString() 函数进行转换。

## 2.36 清空所有文本型输入框

表单中有两种文本型元素：text 与 textarea。假设已经填写完表单信息，但是所有信息都填写错误，一个一个的来修改文本框是不是很费时间？既然如此麻烦，何不创建一个单击按钮，一键清空所有文本型输入框？本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 var _clearText = document.getElementById("clearText");
04 _clearText.onclick = function() { //为清空的文本元素绑定事件
05 //设置表单中所有文本型成员的值为空
06 var _elements = document.getElementById("clearFrom").elements,
07 _elementsLen = _elements.length, //元素个数
08 _ei = null,
09 i = 0;
10 for (; i < _elementsLen ; i++){ //遍历元素
11 _ei = _elements[i];
12 //如果是文本类型，将其值修改为空
13 (_ei.type == "text" || _ei.type == "textarea") && (_ei.value = "");
14 }
15 };
16 };
17 </script>
```

本例 HTML 代码如下：

```
01 <h2>设置表单中所有文本型成员的值为空</h2>
02 <form id='clearFrom'>
03 <input id='clearText' type="button" value='清空所有文本' />

04 <input type="text" value='文本 1' />

05 <textarea>文本 2</textarea>
06 </form>
```

### 【代码说明】

JavaScript 代码第 10~14 行遍历 form 内的所有子节点元素，判断它们的 type 类型，当类型为 text 或 textarea 时，清空该元素的值。

## 2.37 JavaScript 对上传文件的相关操作

大多数网站都有很多涉及文件上传的需求，例如：一些 SNS 网站为了让用户更加形象化，会让用户自定义上传头像文件。本章的最后介绍一些关于 JavaScript 操作上传文件的方法，包括限制上传文件的格式、获取上传文件的大小、清空上传文件、多文件上传等。

浏览器内置的 file 型 input 元素在表单提交时，会完成向后台传输文件的整个过程，本例不关注后台逻辑，只关注前端业务，但是记得不要丢掉 enctype="multipart/form-data"。

本例 JavaScript 代码如下：

```
01 <h2>JavaScript 对上传文件的相关操作</h2>
02 <form id='fileUpload' enctype="multipart/form-data">
03 <p><input type="file" name='file1' id='filedemo1'> </p>
04 <input type="button" id='addFileUpload' value="添加">
05 <input type="button" id='clearFileUpload' value="清空">
06 <input type="button" id='fileUploadSub' value="上传">
07 </form>
```

本例 HTML 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 uploadCount = 1; //增加的文件 input 标示符
04
05 isIE = function(){ //是否是 IE
06 return document.all ? true : false;
07 }
08
09 getFileSize = function(_f){ //获取文件大小
10 var _fileSize = 0;
11 if (this.isIE() && !_f.files) { //判断是否为 IE，是否支持 files
12 var filePath = _f.value;
13 var FileSystem = new ActiveXObject("Scripting.FileSystemObject");
14 //是否可以调用系统存在的 FileSystem.FileExists()
15 if(!FileSystem.FileExists(filePath)){
16 return 0;
17 }
18 var file = FileSystem.GetFile(filePath);
19 _fileSize = file.Size;
20 } else {
21 _fileSize = (_f.files[0] && _f.files[0].size) || 0;
22 }
23 }
24 </script>
```

```

23 return _fileSize;
24 }
25
26 fileType = function(_eForm, types){ //限制上传文件的格式
27 var _elements = _eForm.elements,
28 _elementsLen = _elements.length, //元素个数
29 _ei = null,
30 _v = "",
31 _contentType = "",
32 i = 0;
33 for (; i < _elementsLen ; i++){
34 _ei = _elements[i];
35 if(_ei.type == "file"){
36 //var _i = _ei.value.lastIndexOf("\\");
37 _v = _ei.value,
38 _contentType = _v && _v.match(/^\.(.*)(\.(.{1,8})$)/)[3];
39 //如果文件上传为空或者类型为限制类型，则返回 false
40 if(!_v || types.search(_contentType) != -1){
41 return false;
42 }
43 }
44 }
45 return true;
46 }
47
48 clearFile = function(_eForm){ //清空 input 为 file 的元素内容
49 var _elements = _eForm.elements,
50 _elementsLen = _elements.length,
51 _ei = null,
52 i = 0;
53 for (; i < _elementsLen ; i++){
54 _ei = _elements[i];
55 (_ei.type == "file") && ((isIE() && (_ei.outerHTML = _ei.innerHTML)) ||
56 (_ei.value = ""));//如果为 IE，利用其特性清空 file 文本数据，否则直接将 value 设置为空值
57 }
58 }
59
60 fileUpload = function(){ //上传文件的相关业务
61 var _fileUpload = document.getElementById("fileUpload"),
62 _addFileUpload = document.getElementById("addFileUpload"),
63 _clearFileUpload = document.getElementById("clearFileUpload"),
64 _filedemo1 = document.getElementById("filedemo1"),
65 _fileUploadSub = document.getElementById("fileUploadSub");
66 _fileUploadSub.onclick =function(){
67 alert("测试获取文件大小：" + getFileSize(_filedemo1));//获取上传文件的大小

```

```

68 //限制上传文件的格式 ---参数（上传文件的表单对象，被限制的格式）
69 if(!_fileType(_fileUpload, "text")){
70 return;
71 }
72 }
73
74 _clearFileUpload.onclick =function(){ //清除上传文件的内容
75 clearFile(_fileUpload);
76 }
77
78 _addFileUpload.onclick =function(){ //动态多文件上传，添加上传文件元素
79 uploadCount++;
80 var _input=document.createElement("input"),
81 _pE=document.createElement("p");
82 _input.name="filedemo"+uploadCount;
83 _input.id="filedemo"+uploadCount;
84 _input.type="file";
85 _pE.appendChild(_input);
86 _fileUpload.insertBefore(_pE,document.getElementById("addFileUpload"));
87 }
88 }
89 fileUpload();
90 };
91 </script>

```

### 【代码说明】

- 上传文件的大小，可以通过浏览器对 JavaScript 开放的 API 来获取。JavaScript 代码第 13 行调用 IE 中的 ActiveXObject 控件，使用 GetFile() 获取到一个和指定路径中文件相对应的 file 对象。JavaScript 代码第 21 行表示其他的浏览器可以通过文本对象的 size 属性获取文件大小。
- 限制上传文件的格式，首先是获取 file 文件的路径，用 “\_v.match(^(.\*)\.(.)({1,8})\$)[3]” 正则过滤出文件的扩展名，再与限制的字符参数 types 匹配，最后判断文件类型是否为限制格式。
- 清空上传的文件，如果是 IE 浏览器，可以利用 IE 的特色清空文件，代码为 “\_ei.outerHTML = \_ei.outerHTML”；如果是其他浏览器则直接设置 value 为空。
- 动态上传文件，通过 Javascript 创建新的 file 类型 input 元素，追加到表单中。
- 函数 isIE() 判断当前浏览器是否为 IE，其中 document.all 在 IE 中为 true，在其他浏览器中为 false。
- 本例兼容 IE 6、IE 7、IE 8、Google Chrome、FF 等浏览器。

**说明：**如果遇到 IE 8 不支持本例，请在安全设置中进行修改。设置的过程是，单击“工具” => “Internet 选项” => “安全” => “Internet” => “自定义级别”，找到“将文件上载到服务器时包含本地目录路径”这个复选框，设置为“启用”即可。

# 第3章 图片控制常用代码

网页中随处可见各种有趣的动画，以前大部分动画都是 Flash 实现的，现在随着前端技术的发展，使用 JavaScript 也能实现这类动画效果了。

本章主要涉及的知识点有：

- 动画模块的管理。
- 图片处理的一些常见效果及操作。
- 类似 2014 版本 QQ 空间的个性相册。

**提示：**本章涉及的样式比较多，在不同浏览器上表现效果也不太一样，请在 Chrome、火狐、IE 8 下进行测试，所有代码在这 3 种环境下都要运行正确，如果代码运行失败，请检测是否符合以上标准。

## 3.1 动画管理模块

在日常的 JavaScript 开发中有很多的动画效果，如图片轮播、广告移动等。本书因为有很多的效果要实现，所以本节首先构建一个动画管理模块，有助于为以后的动画效果打下良好的基础。

由于动画管理模块服务于整个项目，是独立的模块，因此将其放在目录“extend”下面，在此目录下面创建 animateManage.js 文件，代码如下：

```
01 ;(function (window, document, undefined) {
02
03 var
04 _aniQueue = [],
05 _baseUID = 0,
06 _aniUpdateTimer = 13,
07 _aniID = -1,
08 isTicking = false;
09
10 /*
11 * options 参数
12 * context --- 被操作的元素上下文
13 * effect --- 动画效果的算法
14 * time --- 效果的持续时间
```

```

15 * startCss --- 元素的起始偏移量
16 * css --- 元素的结束值偏移量
17 */
18 window.animateManage = function(optios) {
19 this.context = optios; //当前对象
20 }
21
22 animateManage.prototype = {
23
24 init : function(){
25 this.start(this.context); //初始化方法
26 },
27
28 stop : function(_e){ //停止动画
29 clearInterval(_aniID);
30 isTicking = false;
31 },
32
33 start : function(optios){ //开始动画
34 if(optios) this.pushQueue(optios); //填充队列属性
35 if(isTicking || _aniQueue.length === 0) return false;
36 this.tick();
37 return true;
38 },
39
40 tick : function(){ //动画检测
41 var self = this;
42 isTicking = true;
43 _aniID = setInterval(function(){
44 if(_aniQueue.length === 0) {
45 self.stop();
46 }
47 else{
48 var i = 0,
49 _aniLen = _aniQueue.length;
50 for(; i < _aniLen ; i++){
51 _aniQueue[i] && self.go(_aniQueue[i], i);
52 }
53 }
54 }, _aniUpdateTimer)
55
56 },
57
58 go : function(_options, i){ //执行具体的动画业务
59 var n = this.now(),

```

```

60 st = _options.startTime,
61 ting = _options.time,
62 e = _options.context,
63 t = st + ting,
64 name = _options.name,
65 tPos = _options.value,
66 sPos = _options.startValue,
67 effect = _options.effect,
68 scale = 1;
69
70 if(n >= t){ //如果当前的时间 > 开始时间+结束时间，则停止当前动画
71 _aniQueue[i] = null;
72 this.delQueue();
73 }
74 else{
75 tPos = this.aniEffect({
76 e:e,
77 ting :ting ,
78 n :n ,
79 st :st ,
80 sPos:sPos,
81 tPos:tPos
82 }.effect)
83 }
84 e.style[name] = name == "zIndex" ? tPos : tPos + "px";
85 this.goCallBack(_options.callback, _options.uid); //是否执行回调函数
86 },
87
88 aniEffect : function(_options, effect){ //动画效果
89 effect = effect || "linear";
90 var _effect ={
91 "linear":function(_options){ //线性运动
92 var scale = (_options.n - _options.st)/_options.ting,
93 tPos = _options.sPos + (_options.tPos - _options.sPos)*scale;
94 return tPos;
95 }
96 }
97 return _effect[effect](_options);
98 },
99
100 goCallBack : function(callback, u){ //回调
101 var i = 0,
102 _aniLen = _aniQueue.length,
103 isCallback = true;
104 for(; i < _aniLen ; i++){

```

```

105 if(_aniQueue[i].uid == u){
106 isCallback = false;
107 }
108 }
109 if(isCallback){
110 callback && callback();
111 }
112 },
113
114 pushQueue : function(options){ //压入执行动画队列
115 var con = options.context,
116 t = options.time || 1000,
117 callback = options.callback,
118 effect = options.effect,
119 starCss = options.starCss,
120 c = options.css,
121 name = "",
122 u = this.setUID(con);
123 for(name in c){
124 _aniQueue.push({
125 "context": con,
126 "time": t,
127 "name": name,
128 "value":parseInt(c[name], 10),
129 "startValue":parseInt((starCss[name] || 0)),
130 "effect":effect,
131 "uid": u,
132 "callback":callback,
133 "startTime": this.now()
134 })
135 }
136 },
137
138 delQueue : function(){ //删除动画队列中指定的动画
139 var i = 0, //寻找到指定动画队列，将其删除
140 l = _aniQueue.length;
141 for(; i < l; i++){
142 if(_aniQueue[i] === null) _aniQueue.splice(i, 1);
143 }
144 },
145
146 now : function(){ //获取当前时间
147 return new Date().getTime();
148 },
149

```

```

150 getUID : function(_e){ //获取元素的 UID
151 return _e.getAttribute("aniUID");
152 },
153
154 setUID : function(_e, _v){ //设置元素的 UID
155 var u = this.getUID(_e);
156
157 if(u) return u; //如果存在 UID 则直接返回
158
159 u = _v || _baseUID++;
160 _e.setAttribute("aniUID", u);
161 return u;
162 }
163 };
164
165 })(window, document)

```

#### 【代码说明】

- 动画模块整体分为 3 部分：①第 40~56 行通过一个定时调用的“帧函数 tick()”维护整个动态效果，默认为 13 毫秒；②当帧函数调用时，会循环一系列的“动画队列”，并且执行每一个具体动画业务；③动画队列的压入与删除维护。
- 在此模块中，通过闭包封装整个模块，将 animateManage 放在 window 域下，当用户使用时，直接“new animateManage(参数列表)”传入相应的哈希参数列表即可。
- 将一些公用的函数，在 JavaScript 原型链上进行扩展。tick()为动画检测函数，当动画开始执行后，会不断地检测更新动画的效果；第 44~53 行当检测到存在需要执行的动画时，会调用 go()函数执行具体的动画业务；在具体的动画业务中，aniEffect()担任实现动画算法的角色。

如果读者对此模块感兴趣，可以完善一下，很简单的，拿来练习一下吧。

为了验证以上的动画模块是否可运行正常，写一段测试代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 //动画管理测试，单击元素会触发闪烁式动画
04 document.getElementById("focusPointSource").onclick = function(){
05 new animateManage({
06 "context":this, //被操作的元素
07 "effect":"linear", //动画的效果，linear 线性运动
08 "time": 500, //持续时间
09 "starCss":{ //元素的起始值偏移量
10 "left":this.left
11 },
12 "css":{ //元素的结束值偏移量
13 "left":200
14 },

```

```

15 "callback":function(){ //结束之后的回调函数
16 alert("动画结束的回调函数执行");
17 }
18 }).init();
19 }
20 };
21 </script>

```

本例 HTML 代码如下：

```

01 <h2>动画管理——单击元素会触发元素向右移动，移动结束后会触发回调</h2>
02

```

#### 【代码说明】

在 index.html 中增加响应元素（id 为 'focusPointSource'），当单击元素时元素会向右移动，移动结束时触发回调函数，出现提示框，效果如图 3-1 所示。

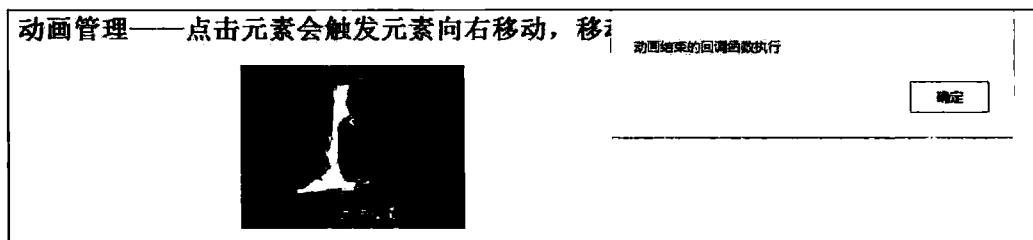


图 3-1 动画模块测试元素移动的效果

## 3.2 实时预览上传的图片

一般的图片上传，是将图片上载到服务器后才能够被预览，但那样会占用服务端的资源，降低用户体验，网速慢的话用户就更崩溃了。例如：丰富个人基本资料时，需上传个人头像，如果网络不给力，则需要等待好久才能预览上传图片的效果。如果能够在客户端直接预览上传的图片，那真是太好了。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var
04 isIE = function(){ //是否是 IE
05 return !window.ActiveXObject;
06 },
07
08 isIE6 = function(){ //是否是 IE 6
09 return isIE() && !window.XMLHttpRequest;
10 },
11
12 isIE7 = function(){ //是否是 IE 7

```

```
13 return isIE() && !isIE6() && !isIE8();
14 },
15
16 isIE8 = function(){ //是否是 IE 8
17 return isIE() && !document.documentElement;
18 },
19
20 setCss = function(_this, cssOption){ //设置元素样式
21 if(!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
22 return;
23 }
24 for(var cs in cssOption){
25 _this.style[cs] = cssOption[cs];
26 }
27 return _this;
28 },
29
30 ****
31 * 实时预览上传图片 start *
32 ****
33
34 upPreviewImg = function(options){
35 var _e = options.e,
36 preloadImg = null;
37 _e.onchange = function(){
38 var _v = this.value,
39 _body = document.body;
40 //图片正则
41 picReg = /(JPEG|jpeg|JPG|jpg|GIF|gif|BMP|bmp|PNG|png){1}/;
42
43 if(!picReg.test(_v)){ //简单的图片格式验证
44 alert("请选择正确的图片格式");
45 return false;
46 }
47
48 if(typeof FileReader == 'undefined'){ //不支持 FileReader
49 if(this.files){
50 options.previewImgSrc.setAttribute("src",
51 this.files[0].getAsDataURL());
52 options.previewImgSrc.style.display = "block";
53 }
54 else if(isIE6()){
55 //IE 6 支持
56 options.previewImgSrc.setAttribute("src", _v);
57 options.previewImgSrc.style.display = "block";
58 }
59 }
60 }
61 }
```

```

58 }
59 else{
60 /*IE 7、IE 8 等支持 filters 滤镜的浏览器*/
61 _v = _v.replace(/[\]"%]/g, function(str){ return
62 escape(escape(str)); });
63 setCss(options.previewImgSrc, {
64 "filter":"progid:DXImageTransform.Microsoft.AlphaImageLoader(
65 sizingMethod='scale',src='"+ 90 + _v + "'",
66 "display":"block"
67);
68 options.previewImgSrc.setAttribute("src", (isIE6() || isIE7() ?
69 "!blankImage" : "data:image/gif;base64,R0lGODlhAQABAAEAA
70 //wAAACH5BAEAAAALAAAAAABAAEAA
71 AICRAEAOw=="));
72 }
73 }
74 else{ //支持 FileReader
75 var reader = new FileReader(),
76 _file = this.files[0]; //读取被加载的文件对象
77
78 reader.onload = (function(file) { //监听 load 事件
79 return function() {
80 options.previewImgSrc.setAttribute("src", this.result);
81 options.previewImgSrc.style.display = "block";
82 };
83 })(_file);
84
85 reader.onerror = function(){ //监听文件读取的错误处理
86 alert("文件读取数据出错");
87 }
88
89 reader.readAsDataURL(_file); //读取文件内容
90 }
91 }
92 },
93
94 upPreviewImg({ //图片预览上传
95 "e":document.getElementById("upPreviewImg"),
96 "previewImgSrc":document.getElementById("previewImgSrc")
97 });
98 };
99 </script>

```

本例 HTML 代码如下：

```

01 <h2>实时预览上传图片</h2>
02 <form>
03 <div id='previewImg'></div>

```

```

04 <input type="file" id='upPreviewImg' name='fileimg'>
05 </form>
```

本例 CSS 代码如下：

```

01 <style>
02 /*=====图片预览上传=====*/
03 #previewImg{
04 height: 100px;
05 }
06 #previewImgSrc{
07 display: none;
08 height: 100px;
09 }
10 </style>
```

#### 【代码说明】

- 一些浏览器（如 Chrome）为了安全，对文件的操作进行了限制，因此不能直接访问本地的文件，但是在新型的 W3C 标准中，增加了 FileReader API，可以对文件进行一些常规的操作。
- JavaScript 代码第 75 行主要利用了 FileReader API，它是 HTML 5 新增的成员。此 API 中有 onload 事件，只要启动异步加载，传入正确的 file 对象，在读取文件内容时会读取文件的相关信息。
- JavaScript 代码第 48 行对于不支持 FileReader API 的浏览器 IE 6 直接修改图片 src 的原始属性；IE 7/IE 8/IE 9 等采用 AlphaImageLoader 滤镜技术获取加载本地图片的路径。
- JavaScript 代码第 64~65 行的原型为 filter:progid:DXImageTransform.Microsoft.AlphaImageLoader( enabled=bEnabled , sizingMethod=sSize , src=sURL )。其中 enabled 为可选参数，表示是否激活滤镜；sizingMethod 为可选参数，设置或检测滤镜对象容器边界内的显示方式，它包括 3 个参数 crop、image 和 scale。参数 crop 表示剪切图片适配对象大小；参数 image 是默认参数，通过增大或减小对象尺寸边界来适配对象大小；参数 scale 表示缩放图片以适配对象大小。
- 使用滤镜的解决方案会带来一个明显的 bug，默认的图片会显示一个小的图片 icon。为了解决这个问题，JavaScript 代码第 68~71 行利用 Data URI 技术，将默认的 icon 替换为一个透明图片，就不会影响预览效果了。

本例运行效果如图 3-2 所示。



图 3-2 图片上传预览效果

### 3.3 鼠标移入/移出时改变图片样式

通过改变图片的样式，可以让用户感知图片处于什么操作状态。例如：鼠标移入图片时图片的边框是一种颜色，鼠标移出时边框又是另一种颜色，这样用户就知道鼠标究竟是在图片内还是在图片外了。实际上这种变化是通过设置 CSS 样式来实现的。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var imgChangeStyle = document.getElementById("imgChangeStyle"),
04 setCss = function(_this, cssOption){ //设置元素样式
05 //判断节点类型
06 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
07 return;
08 }
09 for(var cs in cssOption){ //遍历设置所有样式
10 _this.style[cs] = cssOption[cs];
11 }
12 return _this;
13 };
14
15 imgChangeStyle.onmouseover = function(){ //鼠标移入事件
16 setCss(this, {
17 border:"2px solid red"
18 })
19 }
20
21 imgChangeStyle.onmouseout = function(){ //鼠标移出事件
22 setCss(this, {
23 border:"0px"
24 })
25 }
26 };
27 </script>
```

本例 HTML 代码如下：

```

01 <h2>鼠标移入移出，改变图片样式</h2>
02
```

#### 【代码说明】

- 原理：为特定的元素绑定响应事件，当 mouseover 事件与 mouseout 事件被触发时，分别修改对应的样式。
- JavaScript 代码第 15~19 行当 mouseover 被触发时，设置元素的 border 为“2px solid red”。
- JavaScript 代码第 21~25 行当 mouseout 被触发时，设置元素的 border 为 0px，即清空样式边框。

### 3.4 图片放大镜效果

在一些购物类型的网站中，经常会看到放大镜效果。例如：淘宝网为了让买家更加了解产品的细节，当买家将鼠标指针移动到产品图片上时，会有一个局部放大的效果。所谓放大镜效果，本质上就是准备两张图：一张小图和一张一模一样的高像素大图。当鼠标在小图上移动时，计算出对应的大图应该显示的位置及宽、高。本例效果如图 3-3 所示。

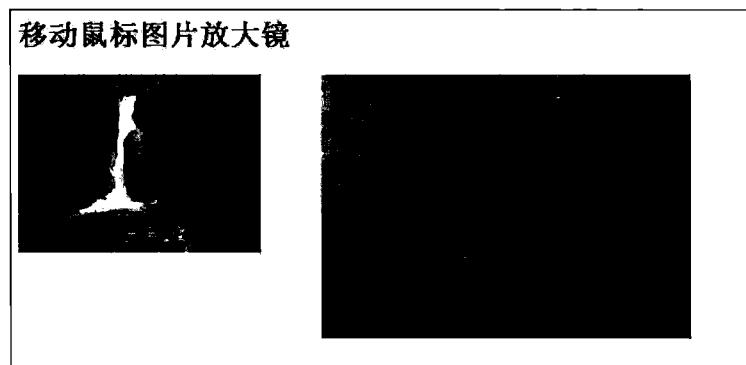


图 3-3 左边小图，右边大图

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 //*****
04 * 移动鼠标图片放大镜 start *
05 *****/
06
07 var isMoveFocus = false, //是否移动焦点
08 focusElement = null, //焦点对象
09 magnifierElement = null, //放大镜
10 magnifierWidth = 800, //放大镜宽度
11 focusZindex = 100, //焦点的 Z 轴
12 magnifierScale = 0, //比例尺
13 magnifierZindex = 101, //放大镜的 Z 轴
14 eMagnifierMages = null, //放大镜的对象
15
16 focusArae ={ //焦点的面积
17 "width":50,
18 "height":50
19 },
20
21 setCss = function(_this, cssOption){ //设置元素样式
22 //判断节点类型
23 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {

```

```
24 return;
25 }
26 for(var cs in cssOption){ //遍历节点与设置样式
27 _this.style[cs] = cssOption[cs];
28 }
29 return _this;
30 }
31
32 initMagnifierMages = function(_e){ //初始化图片管理相关元素
33 /*
34 * 初始化放大镜效果的一些数据
35 */
36 //焦点对象
37 focusElement = setCss(document.getElementById("focusPoint"), {
38 "z-index":focusZindex,
39 "width":focusArae.width,
40 "height":focusArae.height
41 });
42
43 initMagnifierPos(_e);
44
45 magnifierScale = magnifierWidth / _e.offsetWidth; //比例尺换算
46
47 var _img = _e.getAttribute("data-maxImg"); //设置大图
48 document.getElementById("magnifierImg").setAttribute("src", _img);
49
50
51 mouseMagnifier = function(_e){ //放大镜业务处理
52 this.initMagnifierMages(_e); //初始化图片管理的元素
53 this.eMagnifierMages = _e; //移动
54 },
55
56 _mousepos = { //鼠标在页面上的位置
57 "top":0,
58 "left":0
59 }
60 /**
61 * 获取鼠标在页面上的位置
62 * _e: 触发的事件
63 * left: 鼠标在页面上的横向位置, top: 鼠标在页面上的纵向位置
64 */
65 getMousePoint = function (_e) {
66 var _body = document.body,
67 _left = 0,
68 _top = 0;
```

```

69 /*浏览器如果支持 pageYOffset，那么可以使用 pageXOffset 和 pageYOffset
70 获取页面和视窗之间的距离*/
71
72 if(typeof window.pageYOffset != 'undefined') {
73 _left = window.pageXOffset;
74 _top = window.pageYOffset;
75 }
76 else if(typeof document.compatMode != 'undefined' &&
77 document.compatMode != 'BackCompat') {//如果浏览器指定了 DOCTYPE 并且支持
78 compatMode
79 _left = document.documentElement.scrollLeft;
80 _top = document.documentElement.scrollTop;
81 }
82 else if(typeof _body != 'undefined') { //其他的浏览器，如果支持
83 document.body
84 _left = _body.scrollLeft;
85 _top = _body.scrollTop;
86 }
87
88 _left += _e.clientX;
89 _top += _e.clientY;
90 _mousepos.left = _left;
91 _mousepos.top = _top;
92
93 return _mousepos;
94 },
95 pointCheck = function(_event,_e, options){
96 var _pos = getMousePoint(_event),
97 _w = options && options.width || _e.offsetWidth, //获取元素的宽度
98 _h = options && options.height || _e.offsetHeight, //获取元素的高度
99 _left = getAbsoluteLeft(_e),
100 _top = getAbsoluteTop(_e);
101 _pos.left += options && options.left || 0;
102 //计算鼠标的 top 与 left 值是否落入元素的 left 与 top 内
103 if(_pos.left < (_left+_w) && _left < _pos.left && _pos.top > _top && _pos.top <
104 (_top+_h)){
105 return true;
106 }
107 return false;
108 },
109
110 bodyMagnifiermousemove = function(event){
111 var _event = _event || window.event,
112 _e = eMagnifierMages;
113 if(pointCheck(_event, _e)){

```

```

114 isMoveFocus = true;
115 focusStatus();
116 if(!isMoveFocus) return; //是否关闭放大镜效果
117 focusPos(_e, _event); //计算焦点的位置
118 magnifierPos(_e, _event); //显示放大镜效果
119 }else{
120 isMoveFocus = false;
121 focusStatus();
122 }
123 },
124
125 focusPos = function(_e, _event){ //计算聚焦点位置
126 var _pos = getMousePoint(_event),
127 _top = _pos.top - focusArae.height/2,
128 _left = _pos.left - focusArae.width/2
129 ;
130 setCss(focusElement, {
131 "top" : _top,
132 "left" : _left
133 })
134 },
135
136 focusStatus = function(){ //焦点的状态
137 isMoveFocus &&
138 (setCss(focusElement, {
139 "display":"block"
140 }) && setCss(magnifierElement, {
141 "display":"block"
142 })) || (setCss(focusElement, {
143 "display":"none"
144 }) && setCss(magnifierElement, {
145 "display":"none"
146 }));
147 },
148
149 initMagnifierPos = function(_e){ //初始化放大镜位置
150 //位置
151 magnifierElement = setCss(document.getElementById("magnifier"), {
152 "z-index":magnifierZindex,
153 "top":getAbsoluteTop(_e),
154 "left":getAbsoluteLeft(_e) + _e.offsetWidth + focusArae.width
155 });
156 },
157
158 magnifierPos = function(_e, _event){ //计算放大镜中图片位置

```

```

159 var _pos = getMousePoint(_event),
160 _top = magnifierScale * (_pos.top - getAbsoluteTop(_e) - focusArae.height/2),
161 _left = magnifierScale * (_pos.left - getAbsoluteLeft(_e) - focusArae.width/2);
162 if(_top < 0 || _left < 0) return;
163 setCss(document.getElementById("magnifierImg"), {
164 "top": "-" + _top,
165 "left": "-" + _left
166 });
167 },
168
169 getAbsoluteLeft = function (_e){ //获取元素的绝对 left
170 var _left = _e.offsetLeft,
171 _current = _e.offsetParent;
172 while (_current != null){
173 _left += _current.offsetLeft;
174 _current = _current.offsetParent;
175 }
176 return _left;
177 },
178
179 getAbsoluteTop = function (_e){ //获取元素的绝对 top
180 var _top = _e.offsetTop,
181 _current = _e.offsetParent;
182 while (_current != null){
183 _top += _current.offsetTop;
184 _current = _current.offsetParent;
185 }
186 return _top;
187 }
188
189 eMagnifierMages = document.getElementById("imagesSource");
190 initMagnifierMages(eMagnifierMages);
191
192 document.body.onmousemove = function(e){ //body 移动事件
193 bodyMagnifiermousemove(e);
194 }
195
196 };
197 </script>

```

本例 HTML 代码如下：

```

01 <h2>移动鼠标图片放大镜</h2>
02 <div id='imagesSource' data-maxImg='./images/psu2.jpg'>
03 <!--原图-->
04

```

```
05 <!-- 焦点点-->
06 <div class='focusPoint' id='focusPoint'></div>
07 </div>
08
09 <!-- 放大镜-->
10 <div class='magnifier' id='magnifier'>
11 <!-- 放大效果图-->
12
13 </div>
```

本例 CSS 代码如下：

```
01 <style>
02 /*=====放大镜效果=====*/
03 .minImages{
04 width: 200px;
05 position: relative;
06 }
07 .focusPoint{
08 display: none;
09 border: 2px #ccc outset;
10 position: absolute;
11 top: 80px;
12 z-index: 100;
13 filter: alpha(opacity=50);
14 -moz-opacity:0.5;
15 opacity: 0.5;
16 cursor: move;
17 }
18 #imagesSource{
19 width: 200px;
20 height: auto;
21 }
22 .magnifier{
23 width: 304px;
24 height: 222px;
25 position: absolute;
26 display: none;
27 top: 300px;
28 overflow: hidden;
29 margin: 0px auto 0px;
30 }
31 .maxImage{
32 width: 800px;
33 position: absolute;
34 }
35 </style>
```

### 【代码说明】

- 本例的重点是计算小图与大图的位置，JavaScript 代码第 45 行 “`magnifierScale = magnifierWidth / e.offsetWidth;`”，首先换算小图与大图的比例关系，当小图的移动事件触发时，会应用该比例换算出大图的位置，参考 `magnifierPos()` 函数。
- 小图的移动事件并没有直接绑定在被触发元素上，JavaScript 代码第 95~108 行是通过简单的碰撞检测算法实现的，只要计算鼠标的位置位于被绑定的元素内，就触发放大效果，具体算法请参考 `pointCheck()` 函数。
- JavaScript 代码第 169~187 行使用检测元素绝对位置的 `getAbsoluteLeft()` 与 `getAbsoluteTop()` 函数，在 JavaScript 中，可以通过获取元素的 `offsetLeft` 与 `offsetTop` 属性来获取元素的绝对属性。当然，如果需要获取准确的绝对定位，还要遍历父级节点的这些属性进行累加运算。
- 本例还涉及如何获取鼠标在页面上的位置，这个主要是检测浏览器对不同事件 API 的支持，然后获取其对应的属性值坐标，最后累加 `clientX` 与 `clientY`，具体请参考 `getMousePoint()` 函数。

**说明：**碰撞检测是检测两个或者多个物体是否发生碰撞或产生接触，页面中的碰撞检测是检测元素之间是否有交集。

## 3.5 水中倒影效果

在一些相册类应用中有很多有趣的展示效果，例如将用户上传的照片变白、美化。如果能将用户的照片处理成水中倒影的效果，是不是很酷？水中倒影的效果用 JavaScript 就可以实现，如图 3-4 所示。

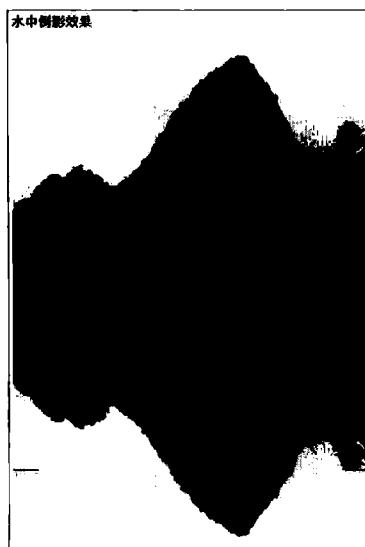


图 3-4 水中倒影效果

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var
04 setCss = function(_this, cssOption){ //设置元素样式
05 //判断节点类型
06 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
07 return;
08 }
09 for(var cs in cssOption){ //遍历并且设置样式
10 _this.style[cs] = cssOption[cs];
11 }
12 return _this;
13 },
14
15 isIE = function(){ //是否是 IE
16 return !window.ActiveXObject;
17 },
18
19 isIE6 = function(){ //是否是 IE 6
20 return isIE() && !window.XMLHttpRequest;
21 },
22
23 isIE7 = function(){ //是否是 IE 7
24 return isIE() && !isIE6() && !isIE8();
25 },
26
27 isIE8 = function(){ //是否是 IE 8
28 return isIE() && !document.documentElementMode;
29 },
30
31 /*****
32 * 水中倒影效果 start *
33 *****/
34
35 shadows = null, //初始化
36 shadowsLen = 0,
37 shadowInWater = function(){
38 //水中倒影效果
39 shadowsSource = document.getElementById("shadowInWaterSrurce");
40 shadows =
41 document.getElementById(shadowsSource.getAttribute("data-water"));
42 if (isIE()) { //IE
43 updateShadow();

```

```

44 return;
45 }
46 else{ //非 IE
47 canvasShadowInWater();
48 }
49 },
50
51 canvasShadowInWater = function(){ //canvas 的水中倒影
52 /* (1) 配置及初始化数据，创建 canvas*/
53 var settings = {
54 'speed': 1, //速度
55 'scale': 1, //比例
56 'waves': 10 //波幅度
57 },
58 waves = settings['waves'],
59 speed = settings['speed']/ 4,
60 scale = settings['scale']/ 2,
61 ca = document.createElement('canvas'),
62 c = ca.getContext('2d'), //获取画布的句柄
63 img = shadowsSource;
64 img.parentNode.insertBefore(ca, img); //canvas 覆盖源图片
65
66 var w,
67 h,
68 dw,
69 dh,
70 offset = 0,
71 frame = 0,
72 max_frames = 0, //最大数据帧率
73 frames = []; //图片的帧数据
74
75 c.save();
76 c.canvas.width = shadowsSource.offsetWidth;
77 //因为要基于原图进行倒影投射，所以必须是原图的 2 倍高度
78 c.canvas.height = shadowsSource.offsetHeight*2;
79
80 /*
81 *绘制图形 — context.drawImage(img,sx,sy,sw,sh,x,y,w,h)
82 * (必需) img 要使用的图片、视频、画布
83 * (可选) sx 开始剪切的 x 坐标
84 * (可选) sy 开始剪切的 y 坐标
85 * (可选) sw 被剪切图形的宽度
86 * (可选) sh 被剪切图形的高度
87 * (必需) x 在画布上放置图形的 x 位置
88 * (必需) y 在画布上放置图形的 y 位置

```

```

89 * (可选) w 将要使用图形的宽度
90 * (可选) h 将要使用图形的高度
91 * 在画布上绘制原图
92 */
93 c.drawImage(shadowsSource, 0, 0);
94 c.scale(1, -1); //垂直镜像转换
95 c.drawImage(shadowsSource, 0, -shadowsSource.offsetHeight*2);
96 c.restore(); //返回之前保存过的路径状态和属性
97 w = c.canvas.width;
98 h = c.canvas.height;
99 dw = w;
100 dh = h/2;

101 /*
102 * 复制画布上指定的矩形的像素数据 — context.getImageData(x,y,w,h);
103 * 以左上角为 (0, 0) 原点
104 * x 代表开始的 x 位置
105 * y 代表开始的 y 位置
106 * w 欲复制的矩形区域宽度
107 * h 欲复制的矩形区域高度
108 * 在被创建的第一个原图的基础上，绘制倒影
109 */
110
111 var id = c.getImageData(0, h/2, w, h).data,
112 end = false;
113
114 c.save(); //将状态保存起来
115 while (!end) { //预先计算缓存的帧
116 var odd = c.getImageData(0, h/2, w, h),
117 od = odd.data,
118 pixel = 0;
119 for (var y = 0; y < dh; y++) {
120 for (var x = 0; x < dw; x++) {
121 var displacement = (scale * 10 * (Math.sin((dh/(y/waves)) +
122 (-offset)))) | 0,
123 j = ((displacement + y) * w + x + displacement)*4;
124
125 if (j < 0) { //修复倒影与原图的水平线闪烁问题
126 pixel += 4;
127 continue;
128 }
129
130 var m = j % (w*4), //修复边缘波纹问题
131 n = scale * 10 * (y/waves);
132 if (m < n || m > (w*4)-n) {
133 var sign = y < w/2 ? 1 : -1;

```

```
134 od[pixel] = od[pixel + 4 * sign];
135 od[++pixel] = od[pixel + 4 * sign];
136 od[++pixel] = od[pixel + 4 * sign];
137 od[++pixel] = od[pixel + 4 * sign];
138 ++pixel;
139 continue;
140 }
141
142 if (id[j+3] != 0) { //水影阵列计算
143 od[pixel] = id[j];
144 od[++pixel] = id[++j];
145 od[++pixel] = id[++j];
146 od[++pixel] = id[++j];
147 ++pixel;
148 } else {
149 od[pixel] = od[pixel - w*4];
150 od[++pixel] = od[pixel - w*4];
151 od[++pixel] = od[pixel - w*4];
152 od[++pixel] = od[pixel - w*4];
153 ++pixel;
154 }
155 }
156 }
157
158 if (offset > speed * (6/speed)) {
159 offset = 0;
160 max_frames = frame - 1;
161 // frames.pop();
162 frame = 0;
163 end = true;
164 } else {
165 offset += speed;
166 frame++;
167 }
168 frames.push(odd);
169 }
170
171 setCss(shadows, { //隐藏原图
172 "display":"none"
173 })
174 setCss(shadowsSource, { //显示原图
175 "display":"none"
176 })
177 c.restore(); //返回上一个状态
178 }
```

```

179 setInterval(function() { //更新视图
180 c.putImageData(frames[frame], 0, h/2);
181 // c.putImageData(frames[frame], 0, h/2);
182 if (frame < max_frames) {
183 frame++;
184 } else {
185 frame = 0;
186 }
187 }, 50);
188 };
189
190 updateShadow = function(){ //IE 中动态更新倒影视图
191 if(isIE6()){
192 return;
193 }
194 shadows.filters.wave.phase+=10;
195 setTimeout("updateShadow()", 150);
196 }
197 shadowInWater();
198
199 };
200 </script>

```

本例 HTML 代码如下：

```

01 <h2>水中倒影效果</h2>
02 <p></p>
04 <p></p>

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====水中倒影效果=====*/
03 .shadowInWater{
04 position:relative;
05 top:-20px;
06 filter:wave(strength=3,freq=3,phase=0,lightstrength=30)blur()flipv()
07 }
08 </style>

```

#### 【代码说明】

- 水中倒影的效果主要利用了 HTML 5 的 Canvas 与 IE 浏览器的滤镜技术。
- 针对 IE 滤镜，最关键的是 CSS 代码“filter:wave(strength=3,freq=3,phase=0,lightstrength=30)blur()flipv()”与 JavaScript 脚本“iManage.shadows.filters.wave.phase+=10;”。
- Canvas 稍微复杂点，首先创建 Canvas 的执行上下文，将 Canvas 的高度设置为图片高度的 2 倍，宽度与图片一样。然后采用 Canvas 的图形处理函数进行绘制，预缓

存一些帧，并且对帧进行像素级处理，最后调用 `setInterval()` 更新帧。

- Canvas 有两个需要学习的地方：`APIcontext.drawImage()`与 `context.getImageData()`，关于两个函数的参数说明在代码中已有注释，此处不再赘述。

## 3.6 横向图片轮播

在所有的图片展示效果中，图片的横向轮播是目前比较流行的一种展示方案，可以更全面、生动地向用户展示所引导的内容。例如当我们打开一些新闻类的网站时，有一些带图片的新闻，就会以横向图片轮播的形式展示。本例效果如图 3-5 所示。

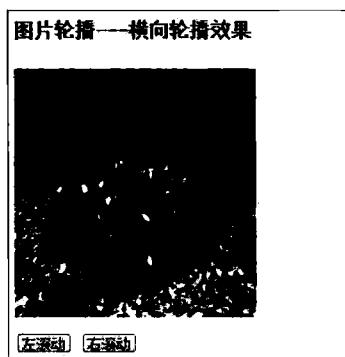


图 3-5 图片横向轮播

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 setCss = function(_this, cssOption){ //设置元素样式
04 //判断节点类型
05 if(!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
06 return;
07 }
08 for(var cs in cssOption){
09 _this.style[cs] = cssOption[cs];
10 }
11 return _this;
12 }
13
14 function getTypeElement(es, type){ //获取指定类型的节点
15 var esLen = es.length,
16 i = 0,
17 eArr = [],
18 esl = null;
19 for(; i < esLen ; i++){
20 esl = es[i];

```

```

21 if(esl.nodeName.replace("#", "").toLowerCase() == type){
22 eArr.push(esl);
23 }
24 }
25 return eArr;
26 }
27 function horizontalShuffling(options){
28 var e = options.e;
29 var child = getTypeElement(e.childNodes, "li"),
30 childLen = child.length,
31 w = 300,
32 _w = childLen * w;
33
34 this.setCss(e, { //初始化样式
35 "width": _w
36 });
37
38 var move = function(type, callback){ //节点移动
39 var v = 0,
40 _left = parseInt((e.style.left || e.offsetLeft), 10);
41
42 if(type == "l"){ //向左移动
43 v = w;
44 if(_left <= -(w-w)){
45 return ;
46 }
47 } else{ //向右移动
48 v = -w;
49 if(_left >= 0){
50 return;
51 }
52 }
53 }
54 //防止产生不是 300 的整数，导致横向轮播不准确
55 var __left = Math.ceil((_left - v)/300)*300;
56
57 if(__left > 0){ //修正左偏向值
58 __left = 0;
59 }
60 new animateManage({
61 "context":e, //被操作的元素
62 "effect":"linear",
63 "time": 200, //持续时间
64 "startCss":{ //元素的起始值偏移量
65 "left":__left

```

```

66 },
67 "css":{
68 "left":_left
69 },
70 "callback":function(){
71 callback && callback();
72 }
73 }).init();
74 }
75
76 var direction = "l", //横向轮播，定时调用
77 horizontalID = -1,
78 closeHorizontal = function(){
79 horizontalID != -1 && clearInterval(horizontalID);
80 },
81 openHorizontal = function(){
82 horizontalID = setInterval(function(){ //循环调用
83 var _left = parseInt((e.style.left || e.offsetLeft), 10);
84 if(_left == -(w-w)){
85 direction = "r";
86 }
87 if(_left == 0){
88 direction = "l";
89 }
90 move(direction);
91 }, 2000)
92 };
93 openHorizontal();
94 options.left.onclick = function(){ //左按钮单击，执行左轮播
95 move("l");
96 }
97
98 options.left.onmouseover = function(){ //左按钮移入，停止轮播
99 closeHorizontal();
100 }
111
112 options.left.onmouseout = function(){ //左按钮移出，开始轮播
113 openHorizontal();
114 }
115
116 options.right.onclick = function(){ //右按钮单击，执行左轮播
117 move("r");
118 }
119
120 options.right.onmouseover = function(){ //右按钮移入，停止轮播

```

```

121 closeHorizontal();
122 }
123
124 options.right.onmouseout = function(){ //右按钮移出，开始轮播
125 openHorizontal();
126 }
127 }
128 //图片轮播---一般轮播效果
129 horizontalShuffling({
130 "e":document.getElementById("horizontalShuffling"),
131 "left":document.getElementById("horizontalShufflingBtnLeft"),
132 "right":document.getElementById("horizontalShufflingBtnRight")
133 });
134
135 };
136 </script>

```

本例 HTML 代码如下：

```

01 <h2>图片轮播—横向轮播效果</h2>
02 <div class="horizontalShuffling">
03 <ul id="horizontalShuffling">
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19
20 </div>
21 <div id="horizontalShufflingBtn">
22 <input id="horizontalShufflingBtnLeft" type="button" value="左滚动">
23 <input id="horizontalShufflingBtnRight" type="button" value="右滚动">
24 </div>

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====图片横向轮播=====*/

```

```

03 .horizontalShuffling{
04 position: absolute;
05 z-index: 111111111111;
06 top: 45px;
07 height: 333px;
08 overflow: hidden;
09 width: 300px;
10 margin: 0px auto 0px;
11 }
12 #horizontalShuffling{
13 position: relative;
14 width: 1000px;
15 height: auto;
16 padding: 0;
17 }
18 #horizontalShuffling li{
19 float: left;
20 z-index: 0;
21 position: relative;
22 height: auto;
23 width: 300px;
24 cursor: pointer;
25 list-style-type: none;
26 border-radius: 4px;
27 box-shadow: 1px 1px 12px rgba(200, 200, 200, 1);
28 margin: 0;
29 +left:-40px; /* IE7 */
30 }
31 #horizontalShuffling li img{
32 width: 300px;
33 height: 333px;
34 }
35 #horizontalShufflingBtn{
36 position: relative;
37 top: 350px;
38 }
39 </style>

```

**【代码说明】**

- HTML 代码第 02 行在外层包装了一个 div，并限制其大小（宽高），这样子层超出部分会被隐藏，子层以 ul 标签构建，设置 li 为左浮动，这样可以使用 JavaScript 代码控制 left 值。
- JavaScript 代码第 42~53 行向左移动减少 li 的 left 数值，向右移动增加 li 的 left 数值。CSS 代码中设置了包装层 div 的宽度是 300，因此必须保持移动步长为 300 的整数。JavaScript 代码第 55 行 Math.ceil() 向上求整，修正浮动值。

### 3.7 图片层叠轮播

轮播的形式并不局限于横向轮播，还包括其他形式的展示方式。例如：层叠轮播相对于横向轮播展示面更广，用户看到的不但是被展示的主图片还有其余图片的部分界面，更有立体感觉。层叠轮播比图片轮播会复杂一点，为了便于定位，本例采用绝对定位。本例效果如图 3-6 所示。

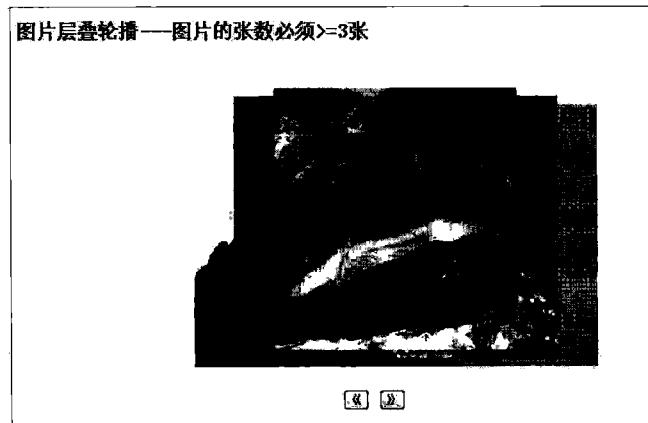


图 3-6 图片层叠轮播

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 getTypeElement = function(es, type){ //获取指定类型的节点集合
04 var esLen = es.length,
05 i = 0,
06 eArr = [],
07 esl = null;
08 for(; i < esLen ; i++){
09 esl = es[i];
10 if(esl.nodeName.replace("#", "").toLowerCase() == type){
11 eArr.push(esl);
12 }
13 }
14 return eArr;
15 }
16
17 leftPics = [], //左侧数据图片堆叠
18 rightPics = []; //右侧数据图片堆叠
19
20 function cascadingShuffling(_options){
21 //获取指定的节点数据

```

```

22 var child = this.getTypeElement(_options.e.childNodes, "li"),
23 //待缓存的一份初始化的数据，用于轮播层叠元素更新位置用
24 _child = [],
25 childlen = child.length, //节点个数
26 i = 0,
27 baseLeft = 220, //距左边的基准参考值
28 center = Math.floor((childlen-1)/2), //中心界点值
29 vt = 50, //自由变量基准
30 cvt = center * vt,
31 centerPic = null; //中间的图片
32
33 for(; i < childlen ; i++){ //左、右及中间的堆叠数据初始化
34 var childl = child[i];
35
36 if(i === 0){
37 centerPic = childl;
38 _child[i] =
39 {"style":{ //初始化样式
40 "left":baseLeft + center*vt,
41 "top" :(childl.offsetTop-vt),
42 "zIndex":childlen
43 }};
44 }else if(i <= center){
45 leftPics.push(childl);
46 _child[i] = {"style":{
47 "left":baseLeft + cvt - vt*i,
48 "top" :(childl.offsetTop-vt*(childlen - i)/childlen),
49 "zIndex":center - i
50 }};
51 }else{
52 rightPics.push(childl);
53 _child[i] = {"style":{
54 "left":baseLeft + cvt + vt*(i - center),
55 "top" :(childl.offsetTop-vt*(childlen - i + center)/childlen),
56 "zindex":childlen - (i - center)
57 }};
58 }
59 }
60
61 //ui 动画变换 参数—target 为被变换的元素， _target 变换的目标属性
62 var updateUI = function(target,_target, callback){
63
64 new animateManage({ //动画管理测试，单击元素会触发闪烁式动画
65 "context":target, //被操作的元素
66 "effect":"linear",

```

```

67 "time": 200, //持续时间
68 "starCss":{ //元素的起始值偏移量
69 "left":target.style.left || target.offsetLeft,
70 "top":target.style.top || target.offsetTop,
71 "zIndex":target.style.zIndex
72 },
73 "css":{ //元素的结束值偏移量
74 "left":_target.style.left || target.offsetLeft,
75 "top":_target.style.top || target.offsetTop,
76 "zIndex":_target.style.zIndex
77 },
78 "callback":function(){
79 callback && callback();
80 }
81 }).init();
82
83 }
84
85 /*
86 *所有的元素层叠变换
87 *o1: 参考对象 1, 当 o1 为 leftPics, 方向向左旋转
88 *o2: 参考对象 2
89 *type: 与前面的第一个参数对应, 当 o1 为 leftPics 时, 对应的值必须为 "l" ,
90 * 反之为 "r"
91 */
92 var rotate = function(o1, o2, type){
93 type = type || "l";
94 o1.unshift(centerPic);
95 var li = 0,
96 leftLen = o1.length-1;
97 _center = type == "r" && (center) || 0;
98 for(; li < leftLen ; li++){
99 if(li == 0){
100 updateUI(o1[li], _child[1+_center]);
101
102 }else{
103 updateUI(o1[li], _child[li+1+_center]);
104 }
105 }
106 o2.push(o1.pop());
107 var ri = o2.length-1;
108 for(; ri >= 0 ; ri--){
109 if(ri == 0){
110 updateUI(o2[ri], _child[0]);
111 }else{

```

```

112 updateUI(o2[ri], _child[center+ri-_center]);
113 }
114 }
115 }
116 centerPic = o2.shift();
117 }
118
119 var rotateID = -1, //定时调用的线程
120 closeRotate = function(){ //关闭定时调用
121 clearInterval(rotateID);
122 },
123 openRotate = function(){ //开启定时调用
124 rotateID = setInterval(function(){//循环调用
125 rotate(leftPics , rightPics);
126 }, 2000);
127 }
128 rotate(leftPics , rightPics); //初始化所有层叠节点的位置样式
129 openRotate(); //开启轮播
130
131 _options.left.onclick = function(){ //单击左按钮，左旋转
132 rotate(leftPics , rightPics);
133 }
134 _options.left.onmousemove = function(){ //移入左按钮，停止旋转
135 closeRotate()
136 }
137 _options.left.onmouseout = function(){ //移出左按钮，开启旋转
138 openRotate()
139 }
140
141 _options.right.onclick = function(){ //单击右按钮，右旋转
142 rotate(rightPics, leftPics, "r");
143 }
144 _options.right.onmousemove = function(){ //移入右按钮，停止旋转
145 closeRotate()
146 }
147 _options.right.onmouseout = function(){ //移出右按钮，开启旋转
148 openRotate()
149 }
150 }
151
152 cascadingShuffling({ //图片层叠轮播初始化
153 "e":document.getElementById("cascadingShuffling"), //待旋转的父节点
154 "left":document.getElementById("cascadingBtnLeft"), //向左旋转
155 "right":document.getElementById("cascadingBtnRight")//向右旋转
156 });

```

```
157 };
158 </script>
```

本例 HTML 代码如下：

```
01 <h2 class="shufflingcss">图片层叠轮播---图片的张数必须≥3 张</h2>
02 <ul id='cascadingShuffling'>
03
04
05
06
07
08
09
10
11
12
13
14
15
16
17
18
19 <div id='cascadingBtn'>
20 <input id='cascadingBtnLeft' type="button" value="《">
21 <input id='cascadingBtnRight' type="button" value="》">
22 </div>
```

本例 CSS 代码如下：

```
01 <style>
02 /*=====图片层叠轮播=====*/
03 #cascadingShuffling{
04 position: absolute;
05 z-index: 1;
06 top: 100px;
07 left: -180px;
08 }
09 #cascadingShuffling li{
10 z-index: 0;
11 position: absolute;
12 top: 20px;
13 height: auto;
14 width: 300px;
15 cursor: pointer;
16 list-style-type: none;
17 left: 377px;
18 border-radius: 4px;
```

```

19 box-shadow: 1px 1px 12px rgba(200, 200, 200, 1);
20 margin: 0;
21 }
22 #cascadingShuffling li img{
23 width: 300px;
24 height: 333px;
25 }
26 #cascadingBtn{
27 position: absolute;
28 top: 508px;
29 left: 210px;
30 }
31 .shufflingcss{
32 position: absolute;
33 top: 0px;
34 }
35 </style>

```

**【代码说明】**

初始化数据之后，以中心节点的图片为参考点，分为左堆叠数据图片、右堆叠数据图片，以及中间的数据图片，计算每个图片的位置。JavaScript 代码第 33~58 行缓存到 \_child 变量中一份数据，作为位置变换的目标点参考值，运行 rotate() 进行第一次的位置校准，抽象出 updateUI() 函数，管理动画过渡的效果。

## 3.8 单击图片逐渐放大

假如一个页面有很多图片，因为担心用户看不清楚，所以所有的图片都被设置得很大，但页面的空间不够该怎么办？可以采用折中的办法，默认是小图，当用户单击图片后再把图片缓缓放大，供图片浏览者查看大图。新浪微博信息流中的图片就采用类似的处理方案，默认是小图，单击后变成大图。通过改变图片的 width 或 height，可以实现这个效果。因为在 3.1 节中已经抽象出了一个动画模块，所以本例在动画模块的基础上实现这个缓缓放大的效果。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var e = document.getElementById("slowlyEnlarge");
04 e.onclick = function(){
05 if(e.offsetWidth + 50 >= 300){
06 return;
07 }
08 new animateManage({
09 "context":e,

```

//被操作的元素

```

10 "effect": "linear",
11 "time": 200, //持续时间
12 "starCss": { //元素的起始值偏移量
13 "width": e.offsetWidth
14 },
15 "css": { //元素的结束值偏移量
16 "width": e.offsetWidth + 50
17 }
18 }).init();
19 }
20);
21 </script>

```

本例 HTML 代码如下：

```

01 <h2>单击图片缓缓放大--图片 Width 范围 <= 300px</h2>
02 <div class="slowlyEnlarge">
03
04 </div>

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====单击图片缓缓放大=====*/
03 #slowlyEnlarge{
04 width: 100px;
05 }
06 .slowlyEnlarge{
07 height: 220px;
08 }
09 </style>

```

#### 【代码说明】

- 本例的效果限制了图片的最大值，当单击图片时会触发动放大的业务。
- JavaScript 代码第 5~18 行调用 3.1 节讲解的动画模块，设定“200 毫秒”的定时器，不间断地修改图片的 width 值，用户就会看到图片缓慢放大的效果了。
- 修改图片大小时只修改 width 或 height，这样不会让图片产生拉伸、变形的效果。

## 3.9 图片旋转

在常见 Web 应用中，上传图片后需要用户自己进行一些效果图的处理。例如：用户上传个人头像后，要把图片裁剪、旋转才能得到想要的效果。图片旋转效果目前有两种方案：一种是 IE 的滤镜；另一种是 HTML 5 解决方案 Canvas 或 transform:rotate(degree)。本例效果如图 3-7 所示。

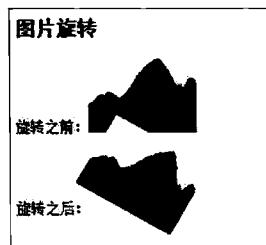


图 3-7 旋转前与旋转后的图片对比

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var
04 isIE = function(){ //是否是 IE
05 return !!window.ActiveXObject;
06 },
07
08 isIE6 = function(){ //是否是 IE 6
09 return isIE() && !window.XMLHttpRequest;
10 },
11 //先检测支持哪一种 CSS 3 变换属性，如果为空则用 IE 的私有方法变换
12 transform = (function(){
13 var _transform = "",
14 _transforms = [//待变换样式属性库
15 "transform",
16 "MozTransform",
17 "webkitTransform",
18 "OTransform",
19 "msTransform"
20],
21 _transformsLen = _transforms.length,
22 i = 0,
23 _styles = document.createElement("div").style
24 ;
25 for(; i<_transformsLen; i++){
26 if(_transforms[i] in _styles){
27 _transform = _transforms[i];
28 break;
29 }
30 }
31
32 return _transform;
33 })();
34
35 //旋转图片

```

```

36 var rotateImg = function (img, degree){
37 if(isIE6()){
38 return;
39 }
40 //设置矩阵变换数据
41 var cosa = (degree == 90 || degree == 270) ? 0 : Math.cos(degree*Math.PI/180),
42 sina = (degree == 180) ? 0 : Math.sin(degree*Math.PI/180),
43 newMatrix = {M11: cosa, M12: (-1*sina), M21: sina, M22: cosa},
44 name;
45
46 if(transform == ""){ //if IE
47
48 /*
49 *IE 的滤镜语法：
50 * filter: progid:DXImageTransform.Microsoft.Matrix(enabled= bEnabled ,
51 SizingMethod= sMethod , FilterType= sType , Dx= fDx , Dy= fDy ,
52 M11= fM11 , M12= fM12 , M21= fM21 , M22= fM22)
53 * 如果只是简单的实现一些旋转，可以用
54 filter:progid:DXImageTransform.Microsoft.BasicImage(rotation=degree)语法结构;
55 */
56 img.style.filter =
57 "progid:DXImageTransform.Microsoft.Matrix(
58 SizingMethod='auto expand')";
59 for (name in newMatrix)
60 img.filters.item("DXImageTransform.Microsoft.Matrix")[name] =
61 newMatrix[name];
62 }
63 else
64 {
65 /*
66 *在最新的 CSS3 中，新增加了 transform 属性，允许对元素应用 2D 或 3D
67 转换，该属性允许我们对元素进行旋转、缩放、移动或倾斜
68 * matrix(n,n,n,n,n,n)对元素进行 2D 矩阵变换总共需要设置 6 个值
69 */
70
71 img.style[transform] = "matrix(" + newMatrix.M11 + "," + (-newMatrix.M12) +
72 + ","
73 //保持旋转的一致性，修正方向， matrix 设置旋转属性
74 + (-newMatrix.M21) + "," + newMatrix.M22 + ",0,0)";
75
76 /*transform:rotate(degree); 只要设置相应的旋转角度即可
77 img.style[transform] = "rotate("+degree+"deg);*/
78 }
79 };
80 rotateImg(document.getElementById("rotated"), 30);

```

```
81 };
82 </script>
```

本例 HTML 代码如下：

```
01 <h2>图片旋转</h2>
02 旋转之前：

04 旋转之后：
```

本例 CSS 代码如下：

```
01 <style>
02 /*=====单击图片缓缓放大=====*/
03 #slowlyEnlarge{
04 width: 100px;
05 }
06 .slowlyEnlarge{
07 height: 220px;
08 }
09 </style>
```

#### 【代码说明】

- 首先初始化 `transform()`，检测不同的浏览器是否支持`_transforms`属性库中定义的标准。
- 如果支持库中的标准，直接运用 CSS 3 中的 `matrix` 特性设置，如果不支持则采用 IE 的滤镜技术，即 JavaScript 代码第 48~58 行的“`progid:DXImageTransform.Microsoft.Matrix(SizingMethod='auto expand')`”。
- `matrix` 矩阵中，IE 的滤镜版本与 CSS 3 中的类似，是一种 2D 变换  $3 \times 3$  矩阵，就是本例中所应用的场景技术，具体的参数代码注释中已经列明。图形的“缩放”、“放大”、“平移”、“旋转”等，都会用到矩阵的知识。在本例中，读者只要记住如何进行旋转就可以了，这里给出一个通用的旋转公式“`matrix(cosθ,sinθ,-sinθ,cosθ,0,0)`”。

## 3.10 类似 QQ 相册效果

如果读者单击 QQ 2014 版相册中的个性化相册进行预览，就会看见一幅有趣的动画图片效果。有的小图会自由地移动，拖动小滚轴后中间的图片也会跟着移动，不错吧！下面一起做一个类似的效果，如图 3-8 所示。

**说明：**本例代码经过测试的浏览器环境有 Chrome、IE 8、IE 9，其他的浏览器可能样式会有点区别。



图 3-8 类似 2014 版 QQ 相册的个性化效果

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var
04 _mousepos = {
05 "top":0,
06 "left":0
07 },
08
09 /**
10 * 获取鼠标在页面上的位置
11 * _e 触发的事件
12 * left: 鼠标在页面上的横向位置, top: 鼠标在页面上的纵向位置
13 */
14 getMousePoint = function (_e) {
15 var _body = document.body,
16 _left = 0,
17 _top = 0;
18 /*浏览器如果支持 pageYOffset, 可以使用 pageXOffset 和 pageYOffset 获取页
19 面和视窗之间的距离*/
20 if(typeof window.pageYOffset != 'undefined') {
21 _left = window.pageXOffset;
22 _top = window.pageYOffset;
23 }
24 //如果浏览器指定了 DOCTYPE 并且支持 compatMode
25 else if(typeof document.compatMode != 'undefined' &&
26 document.compatMode != 'BackCompat') {
27 _left = document.documentElement.scrollLeft;
28 _top = document.documentElement.scrollTop;
29 }

```

```

30 //如果浏览器支持 document.body
31 else if(typeof _body != 'undefined') {
32 _left = _body.scrollLeft;
33 _top = _body.scrollTop;
34 }
35 _left += _e.clientX;
36 _top += _e.clientY;
37 _mousepos.left = _left;
38 _mousepos.top = _top;
39
40 return _mousepos;
41 },
42
43 getTypeElement = function(es, type){ //获取指定类型的节点
44 var esLen = es.length,
45 i = 0,
46 eArr = [],
47 esl = null;
48 for(; i < esLen ; i++){
49 esl = es[i];
50 if(esl.nodeName.replace("#", "").toLowerCase() == type){
51 eArr.push(esl);
52 }
53 }
54 return eArr;
55 },
56
57 getAbsoluteLeft = function (_e){ //获取元素的绝对 left
58 var _left = _e.offsetLeft,
59 _current = _e.offsetParent;
60 while (_current != null){
61 _left += _current.offsetLeft;
62 _current = _current.offsetParent;
63 }
64 return _left;
65 },
66
67 QQPhotoAlbum = function(options){
68
69 var scrollLine = options.scrollLine, //线轴运动
70 minLeft = getAbsoluteLeft(scrollLine),
71 maxLeft = options.width - 100,
72 childDivs = getTypeElement(options.photoStreamMain.childNodes,
73 "div"),
74 childDivsLen = childDivs.length,

```

```

75 cW = 0,
76 ve = [], //横向浮动元素
77 initChilds = (function(){
78 var i = 0,
79 imge = null;
80 while(childDivs[i]){
81 imge = getTypeElement(childDivs[i].childNodes, "img")[0];
82 if(i%2 == 0){
83 ve.push(imge);
84 cW += 150;
85 childDivs[i].style.width = "150px";
86 }else{
87 cW += 300;
88 childDivs[i].style.width = "300px";
89 }
90 i++;
91 }
92 })(),
93 s = cW/options.width;
94
95 scrollLine.onmousedown = function(e){ //开启
96 bodyscrollLineingE.scrollLineing = true;
97 e = e || window.event;
98 var _pos = getMousePoint(e);
99 bodyscrollLineingE.vx = _pos.left - minLeft - (parseInt(this.style.left) || 0);
100
101 }
102
103 document.body.onmouseup = function(){ //关闭
104 bodyscrollLineingE.scrollLineing = false;
105 }
106
107 options.photoStreamMain.style.width = cW; //主流宽度
108
109 mainScrollLine = function(ml){ //主流的 left
110 var _ml = -1 * ml * s;
111 options.photoStreamMain.style.left = _ml+"px";
112 }
113
114 bodyscrollLineingE = { //移动
115 minLeft:minLeft,
116 maxLeft:maxLeft,
117 scrollLine:scrollLine,
118 mainScrollLine:mainScrollLine
119 }

```

```

120
121 setInterval(function(){
122 var veLen = ve.length,
123 l = 0;
124 for(; l < veLen ; l++){
125 new animateManage({
126 "context":ve[l], //图片动态浮动
127 "effect":"linear",
128 "time": 5000, //持续时间
129 "starCss":{ //元素的起始值偏移量
130 "left":ve[l].style.left || 0
131 },
132 "css":{ //元素的结束值偏移量
133 "left":(parseInt(ve[l].style.left, 10) == -150 ? 0 : -150)
134 }
135 }).init();
136 }
137 },6000)
138 },
139 bodyscrollLineingE = {
140 scrollLineing:false,
141 minLeft:0,
142 maxLeft:0,
143 scrollLine:null,
144 vx:0,
145 mainScrollLine:0
146 },
147
148 bodyscrollLineing = function(e){
149 var _ee = bodyscrollLineingE;
150 if(_ee.scrollLineing){
151 e = e || window.event;
152 var _pos = getMousePoint(e),
153 _l = _pos.left - _ee.minLeft- _ee.vx;
154 if(_l < 0) _l = 0;
155 if(_l > _ee.maxLeft) _l = _ee.maxLeft;
156 _ee.scrollLine.style.left = (_l || 0)+"px";
157 _ee.mainScrollLine(_l); //横向布局 left 转换
158 }
159 };
160
161 document.body.onmousemove = function(e){ //body 移动事件
162 bodyscrollLineing(e);
163 }
164

```

```

165 QQPhotoAlbum({
166 "scrollLine":document.getElementById("scrollLine"),
167 "photoStreamMain":document.getElementById("photoStreamMain"),
168 /**"minLeft":580,*/
169 "width":334
170 })
171 };
172 </script>

```

本例 HTML 代码如下：

```

01 <!--占位层-->
02 <div class='QQPhotoAlbum'>
03 <div class="phototitle">
04 <h2>类似 QQ 相册效果</h2>
05 </div>
06 <div class="photoname">
07 <h5>相册名称</h5>
08 </div>
09
10 <!--横向流主体-->
11 <div class="photoStreamMain">
12 <div id='photoStreamMain'>
13 <div>
14
15 </div>
16 <div>
17
18 </div>
19 <div>
20
21 </div>
22 <div>
23
24 </div>
25 <div>
26
27 </div>
28 <div>
29
30 </div>
31 <div>
32
33 </div>
34
35

```

```
36 </div>
37 </div>
38
39 <!--滚动线轴-->
40 <div class="rollLine" style="margin-left:-150px">
41 <div id='scrollLine' class="line"></div>
42 </div>
43
44 </div>
```

本例 CSS 代码如下：

```
01 <style>
02 /*=====QQ 相册横向流效果=====*/
03 .QQPhotoAlbum{
04 position: relative;
05 top: 0px;
06 width:100%;
07 height:783px;
08 background-color: #232429;
09 }
10
11 .QQPhotoAlbum .phototitle{
12 position: relative;
13 text-align: center;
14 font-size: 35px;
15 font-weight: normal;
16 top: 100px;
17 +top:90px; /* IE7 */
18 /* _top:value; /*IE6 */
19 color: #fff;
20 }
21
22 .QQPhotoAlbum .photoname{
23 position: relative;
24 text-align: center;
25 font-weight: normal;
26 top: 41px;
27 +top:80px; /* IE7 */
28 }
29 .QQPhotoAlbum .photoname h5{
30 font-weight: normal;
31 width: 200px;
32 line-height: 34px;
33 height: 34px;
34 color: #fff;
```

```
35 display: inline-block;
36 font-size: 19px;
37 background-color: rgba(0, 0, 0, .3);
38 filter: none;
39 border: 1px solid rgba(119, 119, 119, .7);
40 }
41 .photoStreamMain{
42 border-left-width: 0;
43 border-right-width: 0;
44 padding: 1px 0;
45 border-radius: 0;
46 border: 1px solid rgba(119, 119, 119, .7);
47 border-radius: 3px;
48 width: 100%;
49 height: 300px;
50 overflow: hidden;
51 padding: 1px;
52 filter: progid:DXImageTransform.Microsoft.gradient(startColorstr="#B2777777",
53 endColorstr="#B2777777");
54 position: relative;
55 top: 100px;
56 }
57 #photoStreamMain{
58 position: relative;
59 }
60 #photoStreamMain div{
61 float: left;
62 position: relative;
63 height: 300px;
64 overflow: hidden;
65 }
66 #photoStreamMain div img{
67 height: 280px;
68 margin-top: 10px;
69 position: relative;
70 }
71 .rollLine{
72 position: relative;
73 top: 135px;
74 left: 50%;
75 +left: 40%;
76 margin-left: -152px;
77 width: 334px;
78 background-position: 0 -36px;
79 font-size: 0;
```

```
80 height: 6px;
81 _background-image: none;
82 _background-color: transparent;
83 _height: 6px;
84 border: 1px solid #535353;
85 border-radius: 4px;
86 }
87 .rollLine .line{
88 width: 100px;
89 left:0px;
90 height: 6px;
91 background: #FFF;
92 background: rgba(255,255,255,.8);
93 border-radius: 4px;
94 position: absolute;
95 top: 0;
96 font-size: 0;
97 }
98 body {
99 margin: 0px;
100}
101 </style>
```

#### 【代码说明】

- 本例代码主要包括 4 部分：主流部分的横向滚动、小滚轴部分的滚动、小滚轴部分与主流部分的协调、小图的动画移动效果。
- JavaScript 代码第 95~163 行小滚轴的滚动主要是判断鼠标是否进入小滚轴的范围。当进入小滚轴范围且鼠标按下时，才开始随着鼠标的移动进行拖动，当鼠标抬起，关闭小滚轴的移动。
- 计算主流部分的图片移动，主要是通过小滚轴与主流层的宽度比计算出来的比例来进行移动的。
- 小图的动画移动效果，间隔“6000 毫秒”进行一次动画 left 样式操作。

# 第4章 内容展示常用代码

互联网中信息无处不在，而内容是传达信息的基础要素。在页面中，层（div）、表格、文本段落都是常见的内容容器。本章讲解目前网页流行的内容展示代码段。

本章主要涉及的知识点有：

- 表格及单元格的内容处理。
- 常见内容的格式化处理、效果处理。
- 一些提示框、消息框效果。

## 4.1 单元行上的鼠标悬停提示

用户浏览网页时经常会看到鼠标悬停提示，例如：当鼠标移到用户头像时，提示该用户的详细昵称或注册时间。本例就来实现这样的效果，如图 4-1 所示。

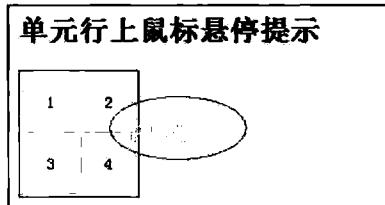


图 4-1 鼠标悬停时提示信息

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 var
04 setCss = function(_this, cssOption){ //设置元素样式
05 //判断节点
06 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style){
07 return;
08 }
09 for(var cs in cssOption){
10 _this.style[cs] = cssOption[cs];
11 }
12 return _this;
13 }
14 }
15 }
```

```
13 },
14
15 _mousepos = { //鼠标在页面上的位置
16 "top":0,
17 "left":0
18 },
19
20 /**
21 * 获取鼠标在页面上的位置
22 * _e: 触发的事件
23 * left: 鼠标在页面上的横向位置, top: 鼠标在页面上的纵向位置
24 */
25 getMousePoint = function (_e) {
26 var _body = document.body,
27 _left = 0,
28 _top = 0
29 ;
30 /*如果浏览器支持 pageYOffset, 那么可以使用 pageXOffset 和 pageYOffset
31 获取页面和视窗之间的距离*/
32 if(typeof window.pageYOffset != 'undefined') {
33 _left = window.pageXOffset;
34 _top = window.pageYOffset;
35 }
36 //如果浏览器指定了 DOCTYPE 并且支持 compatMode
37 else if(typeof document.compatMode != 'undefined' &&
38 document.compatMode != 'BackCompat') {
39 _left = document.documentElement.scrollLeft;
40 _top = document.documentElement.scrollTop;
41 }
42 //如果浏览器支持 document.body
43 else if(typeof _body != 'undefined') {
44 _left = _body.scrollLeft;
45 _top = _body.scrollTop;
46 }
47
48 _left += _e.clientX;
49 _top += _e.clientY;
50
51 _mousepos.left = _left;
52 _mousepos.top = _top;
53
54 return _mousepos;
55 },
56
57 //参数—e 被绑定的节点; tooltipMsg 显示信息的节点
```

```

58 tooltip = function(e, tooltipMsg){
59 var trE = e.rows, //获取被遍历的 tr 节点对象
60 trLen = trE.length, //获取被遍历的节点长度
61 i = 0;
62
63 for(; i < trLen ; i++){ //遍历被提示的对象
64 var trEi = trE[i],
65 dataTooltip = trEi.getAttribute("data-tooltip");//获取数据
66
67 if(dataTooltip){//如果存在 data-tooltip 数据则绑定响应事件
68 trEi.onmousemove = function(event){ //显示提示信息
69 event = event || _W.event;
70 var _pos = getMousePoint(event);
71 //修正提示信息的坐标
72 tooltipMsg.innerHTML = this.getAttribute("data-tooltip");
73 setCss(tooltipMsg,{
74 "left":_pos.left + "px",
75 "top": (_pos.top+18) + "px",
76 //显示信息
77 "display":"inline"
78 })
79 }
80
81 trEi.onmouseout = function(){ //隐藏提示信息
82 setCss(tooltipMsg,{ //隐藏信息
83 "display":"none"
84 })
85 }
86 }
87
88 }
89
90 };
91 tooltip(document.getElementById("tooltip"),
92 document.getElementById("tooltipMsg"));
93 };
94 </script>
```

本例 HTML 代码如下：

```

01 <h2>单元行上鼠标悬停提示</h2>
02 <table id='tooltip' border="1" width="100">
03 <tr data-tooltip="第 1 行提示">
04 <td>1</td>
05 <td>2</td>
06 </tr>
07 <tr data-tooltip="第 2 行提示">
```

```

08 <td>3</td>
09 <td>4</td>
10 </tr>
11 </table>
12 <div id='tooltipMsg'>我是提示信息</div>

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 /*===== 单元行上鼠标悬停提示 =====*/
14 #tooltipMsg{
15 position: absolute;
16 border: 1px solid #ccc;
17 display: none;
18 color: #ccc;
19 font-size: 12px;
20 padding: 2px;
21 -moz-border-radius: 2px;
22 -webkit-border-radius: 2px;
23 border-radius: 2px;
24 z-index: 1;
25 }
26 </style>

```

#### 【代码说明】

- 鼠标悬停的主要场景动作包括：移动鼠标进入元素 → 显示提示信息 → 移动鼠标离开元素 → 隐藏提示信息。
- JavaScript 代码第 68~85 行针对以上的场景为元素绑定鼠标移入事件 `onmousemove` 与移出事件 `onmouseout`。当鼠标移入时，将提示信息显示出来，当鼠标离开时，直接隐藏提示信息。

## 4.2 表格光棒效果

在用户选择表格中的某一行时，为了突出选中的行，引入了表格光棒效果。例如：当

用户移动到表格的第2行时，改变第2行颜色，设置凸显效果，如图 4-2 所示。

表格光棒效果				
1	2	3	4	5
6	7	8	9	10
6	7	8	9	10

图 4-2 光棒效果，第 2 行高亮

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var trE = document.getElementById("lightBar").rows,
04 trLen = trE.length, //获取被遍历的节点长度
05 i = 0;
06
07 for(; i < trLen ; i++){ //遍历被提示的对象
08 var trEi = trE[i];
09
10 trEi.onmousemove = function(event){ //设置光棒效果的样式
11 //光棒样式 background-color:#a5e5aa
12 this.style.backgroundColor = "#a5e5aa";
13 }
14
15 trEi.onmouseout = function(){ //还原初始的样式
16 this.style.backgroundColor = "#fff";
17 }
18 }
19 };
20 </script>
```

本例 HTML 代码如下：

```

01 <h2>表格光棒效果</h2>
02 <table id='lightBar' border="1" width="500">
03 <tr>
04 <td>1</td>
05 <td>2</td>
06 <td>3</td>
07 <td>4</td>
08 <td>5</td>
09 </tr>
```

```

10 <tr>
11 <td>6</td>
12 <td>7</td>
13 <td>8</td>
14 <td>9</td>
15 <td>10</td>
16 </tr>
17<!—省略部分行 -->
18 </table>

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 </style>

```

#### 【代码说明】

- 本效果主要利用了背景样式的处理，当 onmousemove 事件触发时修改选中行为光棒效果的背景颜色，参见 JavaScript 代码第 10~13 行。
- 当 onmouseout 事件触发时恢复元素初始的样式，即还原样式，参见 JavaScript 代码第 15~17 行。

## 4.3 让表单没有凹凸感

在苹果的 iOS7 系统下流传着一种说法：隐藏式按钮，在这之前的 iOS 系统中，所有的按钮都清晰可见，因为按钮设计了凸显效果，让用户知道这里可以单击。据说设计人员觉得这样很突兀，所以现在使用了隐藏式按钮，按钮不再凸显。本例使用 JavaScript 来控制样式，改变表单指定元素的外观，让表单没有凹凸感。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){ //页面载入
03 cleanConcaveConvex = function(e){
04 e.style.border = "1 solid #000000"; //设定样式
05 }

```

```

06 cleanConcaveConvex(document.getElementById("cleanConcaveConvex"));
07 };
08 </script>

```

本例 HTML 代码如下：

```

01 <h2>让表单没有凹凸感</h2>
02 <input id="cleanConcaveConvex" value="没有凹凸感" />

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 </style>

```

#### 【代码说明】

- 文档载入完毕时获取被绑定的元素，利用 JavaScript 代码第 06 行更新元素的样式。
- 要想让元素没有凹凸感，只需设置表单元素的 border 样式。如果读者想实现其他效果，如虚线效果，则只要略微修改 JavaScript 代码第 04 行的样式即可。

## 4.4 动态插入和删除单元行

动态插入和删除单元行是比较常用的 DOM 操作，在日常开发中也是比较重要的需求。

例如：用户填写完自己的基本信息后，又想增加一些备注信息，或者想删除一些多余的信息。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 /*
04 * 如果 tr 存在则是新增操作，否则为删除操作
05 * 当为删除操作时，参数--- table 为表格的 table 对象， num 为被删除的单元行
06 * 序列数
07 * 当为新增操作时，参数--- table 为表格的 table 对象， num 是新增单元行
08 * 的位置， tr 为新增行的字符串型数组
09 */
10 trAct = function(table, num, tr){
11 if(!tr){ //如果 num 不存在则执行删除操作

```

```

12 var _num = table.rows[num];
13 if(_num){ //如果被删除的行对象存在，则删除，返回 true
14 table.deleteRow(_num); //用 JavaScript 的原生函数删除行
15 return true;
16 }
17 else{
18 return false; //如果删除的对象不存在，则删除失败，返回 false
19 }
20 }
21 else{
22 var r = table.insertRow(num), //在指定的位置创建行对象
23 i = 0,
24 l = tr.length; //待插入的数据长度
25 for(; i < l ; i++){ //遍历待插入数据
26 r.insertCell(i).innerHTML = tr[i]; //插入新单元格数据
27 }
28 return true; //新增成功返回 true
29 }
30 }
31 }
32
33 /*动态插入和删除单元行*/
34 var _tableAct = document.getElementById("tableAct");
35
36 document.getElementById("deleteRow").onclick = function(){//删除第一行
37 trAct(_tableAct, 0);
38 }
39
40 document.getElementById("addRow").onclick = function(){ //新增一行
41 trAct(_tableAct, 0, [
42 "新增单元格 1",
43 "新增单元格 2"
44]);
45 }
46 };
47 </script>

```

本例 HTML 代码如下：

```

01 <h2>动态插入和删除单元行</h2>
02 <table id="tableAct" border="1" width="500">
03 <tr>
04 <td>1</td>
05 <td>2</td>
06 </tr>
07 </table>

```

```
08 <input value='删除第一行' type="button" id='deleteRow' />
09 <input value='新增一行' type="button" id='addRow' />
```

本例 CSS 代码如下：

```
01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 </style>
```

#### 【代码说明】

- 在 JavaScript 中，可以使用 rows 访问表格中的每一行，没有兼容性问题。
- 利用 JavaScript 代码第 14 行的 deleteRow() 删除指定的行。
- 创建行的基本逻辑是首先创建行的引用对象 “r = table.insertRow(num)”，然后再填充内容 “r.insertCell(i).innerHTML = tr[i];”。

## 4.5 表格内容的展开和折叠

表格的折叠与展开，可以增加网页空间的有效利用率。例如：用户收藏的购物清单中有很多要购买的产品，但页面无法全部显示，不太重要的就采用默认折叠的形式提示用户还有购物信息，具体是什么购物信息，需要用户展开后才可以查看。本例效果如图 4-3 所示。

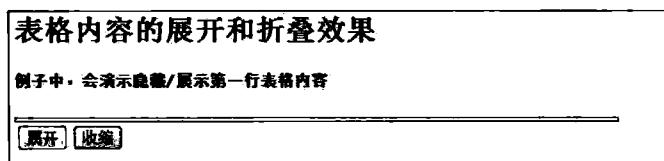


图 4-3 内容收缩的状态

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 ****
04 * 表格内容的展开和折叠效果 start *
05 ****
06 }
```

```

07 tableOutIn = function(e, type){
08 if(type != "open"){
09 e.style.display = "none" //隐藏指定的行元素
10 }
11 else{
12 e.style.display = "table-row" //table-row 设置此元素会作为一个表格行显示
13 }
14 }
15
16 var _tableOutIn = document.getElementById("tableOutIn");
17
18 document.getElementById("openRow").onclick = function(){ //展开一行 openRow
19 tableOutIn(_tableOutIn.rows[0], "open")
20 }
21
22 document.getElementById("inRow").onclick = function(){ //收缩一行 inRow
23 tableOutIn(_tableOutIn.rows[0])
24 }
25 };
26 </script>

```

本例 HTML 代码如下：

```

01 <h2>表格内容的展开和折叠效果</h2>
02 <h5>例子中，会演示隐藏/展示第一行表格内容</h5>
03 <table id='tableOutIn' border="1" width="500">
04 <tr>
05 <td>1</td>
06 <td>2</td>
07 </tr>
08 </table>
09 <input value='展开' type="button" id='openRow' />
10 <input value='收缩' type="button" id='inRow' />

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }

```

13 </style>

#### 【代码说明】

- 本例的动画效果没有利用动画模块处理，相信读者已经注意到了。那么，为什么还会看到收缩或展开的过程呢，因为人会有视觉暂留现象，本例利用这一特点，制造了“假象”动画。
- 针对上述现象，JavaScript 代码第 12 行在单击展开时设置“e.style.display = "table-row"”，在收缩时设置“e.style.display = "none"”，就可以实现本例效果了。

## 4.6 表格内容拖曳效果

为了更大限度地让用户自由操控表格中的内容，实现内容的拖曳是一个不错的想法。例如：一个用户在表格 1 输入了一些内容，但是输入完后才意识到输入的表格位置不对，常规处理是删除旧有信息，再在新的表格位置重新输入，这样是不是比较麻烦？如果能将已输入的内容直接拖曳进目标表格的位置，岂不是更快、更有效吗？本例效果如图 4-4 所示。

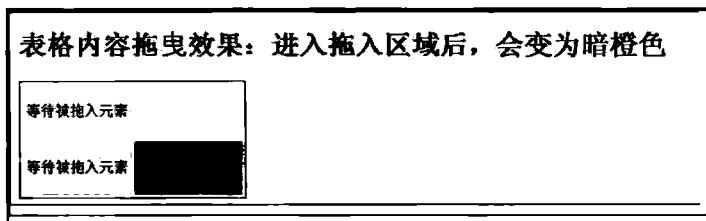


图 4-4 目标位置变色

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var
04 //body 事件，将所有的 body 事件都放在一个函数内——拖曳事件
05 bodyEvents = function(){
06
07 document.body.onmousemove = function(e){ //body 的鼠标移动事件
08 bodyTableDrawmove(e); //表格内容拖曳效果
09 }
10
11 document.body.onmouseup = function(e){ //body 的 onmouseup 事件
12 bodyTableDrawmouseup(e);
13 }
14 },
15
16 getAbsoluteLeft = function (_e){ //获取元素的绝对 left
17 var _left = _e.offsetLeft,
18 _current = _e.offsetParent;

```





```

109 "top" : 0
110 };
111
112 drawTds = [];
113 drawTdsLen = 0;
114 var tableDraw = function(table, tableDrawContent){ //拖曳的所有 td 元素
115 drawContent = tableDrawContent;
116
117 var r = table.rows,
118 rl = r.length,
119 i = 0,
120 c = [],
121 cl = 0,
122 l = 0;
123 for(; i < rl; i++){
124 c = r[i].cells;
125 cl = c.length;
126 l = 0;
127 for(; l < cl; l++){
128 drawTds.push(c[l]);
129 }
130 }
131 drawTdsLen = drawTds.length;
132 tableDrawing();
133 },
134
135 openTableDraw = function(event){ //开启拖曳
136 event = event || window.event;
137 var _pos = getMousePoint(event);
138 this.drawing = open;
139
140 startDrawPos = { //起始偏移值
141 "left" : _pos.left-this.getAbsoluteLeft(drawContent),
142 "top": _pos.top-this.getAbsoluteTop(drawContent)
143 };
144 },
145
146 closeTableDraw = function(){ //关闭拖曳
147 this.drawing = false;
148 setCss(drawContent, {
149 "left" : "0px",
150 "top" : "0px",
151 "position" : "static"
152 });
153 drawTd.style.backgroundColor = "#ffff";

```

```

154 },
155
156 tableDrawing = function(){
157
158 drawContent.onmousedown = function(e){
159 startDrawTd = this.parentNode;
160 openTableDraw(e);
161 }
162 },
163
164 bodyTableDrawmoveTd = function(event){
165 var i = 0,
166 _drawTdsI = null;
167 for(; i < drawTdsLen; i++){
168 _drawTdsI = drawTds[i];
169
170 if(pointCheck(event, _drawTdsI)){ //检测是否选择当前元素
171 if(drawing){ //进入元素
172 drawTd = _drawTdsI; //设置选中的元素
173 }
174 }else{
175 _drawTdsI.style.backgroundColor = "#fff"; //恢复没有被选中的颜色
176 }
177 }
178 drawTd.style.backgroundColor = "#E7AB83"; //被选中的显示颜色
179 },
180
181 bodyTableDrawmove = function(event){ //表格拖曳 body 的 move 事件
182 event = event || window.event;
183 var _pos = getMousePoint(event);
184 //如果不存在被拖曳的对象，禁止拖曳
185 if(!drawContent || !drawing) return false;
186
187 //进入哪一个 td --- 由于拖曳的元素会覆盖 td，所以事件绑定由碰撞检测担任
188 bodyTableDrawmoveTd(event);
189 setCss(drawContent, { //设置元素的位置
190 "left" : (_pos.left - startDrawPos.left) + "px",
191 "top" : (_pos.top - startDrawPos.top) + "px"
192 });
193
194 if(drawContent.style.position != "absolute"){ //修改元素的定位方式
195 drawContent.style.position = "absolute";
196 }
197
198 },

```

```

199 //表格拖曳 body 的 mouseup 事件
200 bodyTableDrawmouseup = function(){
201 //如果不存在被拖曳的对象，禁止拖曳
202 if(!drawContent || !drawTd || !drawing) return false;
203 drawTd.innerHTML = ""; //设置被拖入的区域为空
204 drawTd.appendChild(drawContent); //将被拖曳的内容追加在拖入的区域元素内
205
206 var _html = startDrawTd.innerHTML; //内容替换，防止被拖曳的内容覆盖
207 if(_html.search("等待被拖入元素") == -1 && _html.search("被拖曳的内容") ==
208 -1){
209 startDrawTd.innerHTML = "等待被拖入元素";
210 }
211
212 closeTableDraw(); //关闭拖曳
213 };
214
215 tableDraw(
216 document.getElementById("tableDraw"),
217 document.getElementById("tableDrawContent"));
218
219 bodyEvents();
220 };
221 </script>

```

本例 HTML 代码如下：

```

01 <h2>表格内容拖曳效果：进入拖入区域后，会变暗橙色</h2>
02 <table id='tableDraw' border="1">
03 <tr>
04 <td>
05 <div class="draw" id="tableDrawContent">被拖曳的内容</div>
06 </td>
07 <td>等待被拖入元素</td>
08 </tr>
09 <tr>
10 <td>等待被拖入元素</td>
11 <td>等待被拖入元素</td>
12 </tr>
13 </table>

```

### 【代码说明】

- 为了增加用户体验，让用户知道自己拖曳进哪一个方格，设置了拖入区域变暗的样式。
- 主要的业务流：JavaScript 代码第 158 行当触发 onmousedown 事件时开启拖曳效果，记录初始的位置，因为如果直接让元素的位置等于坐标的位置是不符合实际的，所以先记录初始位置，再计算拖曳移动时的坐标位置与初始位置之间的差值，就是真

实的拖曳位置。当开始移动时，利用 `bodyTableDrawmove()` 函数处理移动逻辑，并将这个函数放置在“`document.body.onmousemove`”事件中进行移动响应，JavaScript 代码第 164~179 行在移动的过程中调用 `bodyTableDrawmoveTd()` 函数，判断鼠标是否进入了特定的格子内，如果进去则改变格子的颜色，并记录最后进入的格子对象。当关闭拖曳时调用 `closeTableDraw()` 函数处理关闭业务，关闭拖曳状态“`this.drawing = false;`”，还原样式。

## 4.7 表格分页

通过表格的方式展示信息是一个不错的选择，但是如果表格内的信息非常多，一个 Web 页面展示不完怎么办？那么只有采用分页的形式展示更多信息了。使用 JavaScript 对表格进行分页，本质就是在指定的页数下显示对应的数据。通常情况下，分页的数据是来自服务端的，本例抽象了一个函数 `_CM.getPageData()`，模拟服务端的数据。如果读者想用服务端的真实数据，可以将这个函数中的返回数据方式改为 Ajax 或者缓存的数据，其实本质上只要数据的结构一样，至于用什么手段获取数据，对于显示的内容来说是一样的结果。本例效果如图 4-5 所示。

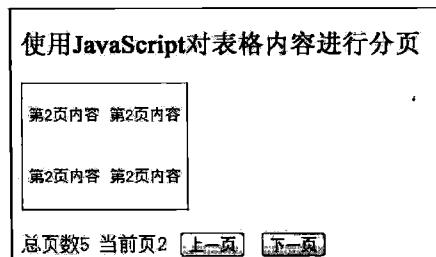


图 4-5 分页显示第 2 页数据内容

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 /*
04 * 如果 tr 存在则是新增操作，否则为删除操作
05 * 当为删除操作时，参数 --- table 为表格的 table 对象，num 为被删除的单元行
06 *
07 * 当为新增操作时，参数 --- table 为表格的 table 对象，num 是新增单元行
08 * 的位置，tr 为新增行的单元格的字符串型数组
09 */
10 var
11 trAct = function(table, num, tr){
12
13 //如果 num 不存在则执行删除操作
14 if(!tr){

```

```
15 var _num = table.rows[num];
16 //如果被删除的行对象存在，则删除，返回 true
17 if(_num){
18 //JavaScript 的原生函数删除行
19 table.deleteRow(_num);
20 return true;
21 }
22 else{
23 //如果被删除的对象不存在，则删除失败，返回 false
24 return false;
25 }
26 }
27 else{
28 //在指定的位置创建行对象
29 var r = table.insertRow(num),
30 i = 0,
31 //待插入的数据长度
32 l = tr.length;
33 //遍历待插入数据
34 for(; i < l ; i++){
35 //插入新单元格数据
36 r.insertCell(i).innerHTML = tr[i];
37 }
38
39 //新增成功返回 true
40 return true;
41 }
42
43 },
44
45 //当前的页数
46 currentPage = 1,
47 //table 对象
48 table = null,
49 //当前页的 UI 显示
50 currentPageUi = null,
51 //所有页的 UI 显示
52 allPage = null,
53 //更新表格 UI
54 updateUi = function(){
55 //更新表格的内容 Ui
56 tableUi();
57 //设置当前页数
58 currentPageUi.innerHTML = currentPage;
59 //设置总页数
```

```
60 allPage.innerHTML = allPages;
61 },
62 //模拟后端返回的数据结构，以供分页使用
63 getPageData = function(){
64 return [
65 [
66 "第"+currentPage+"页内容",
67 "第"+currentPage+"页内容"
68],
69 [
70 "第"+currentPage+"页内容",
71 "第"+currentPage+"页内容"
72]
73]
74 },
75 //模拟所有页数
76 allPages = 5,
77 //初始化分页
78 tablePaging = function(args){
79 table = args.tablePaging;
80 currentPageUi = args.currentPage;
81 allPage = args.allPage;
82
83 nextPaging(args.nextPaging);
84 prevPaging(args.prevPaging);
85
86 updateUi();
87 },
88
89 //分页的下一页
90 nextPaging = function(e){
91 e.onclick = function(){
92 currentPage++;
93 //如果页数高于最大页，则修改页数为最大页，并阻止程序流
94 if(currentPage > allPages){
95 currentPage = allPages;
96 return;
97 }
98 //更新 UI
99 updateUi();
100 }
101 }
102 },
103 //分页的上一页
104 }
```

```
105 prevPaging = function(e){
106 e.onclick = function(){
107 currentPage--;
108 //如果页数小于 1，则修改页数为 1，并阻止程序流
109 if(currentPage < 1){
110 currentPage = 1;
111 return;
112 }
113 //更新 UI
114 updateUi();
115 }
116 },
117 //更新当前页数的表格数据
118 tableUi = function(){
119 //返回模拟的后台数据
120 var d = getPageData(),
121 _data1 = null,
122 l = d.length,
123 i = 0;
124
125 //清空表格数据，由于数据结构没变，所有清空数据与插入数据都是 2
126 for(i < l; i++) {
127 //删除的节点永远是第一个
128 trAct(table, 0);
129 }
130
131 //插入表格数据
132 for(i = 0; i < l; i++) {
133 _data1 = d[i];
134 //增加 td 内容
135 trAct(table, 0, [
136 _data1[0],
137 _data1[1]
138]);
139 }
140 };
141
142 //表格分页*****
143 tablePaging({
144 "tablePaging":document.getElementById("tablePaging"),
145 "currentPage":document.getElementById("currentPage"),
146 "allPage":document.getElementById("allPage"),
147 "nextPaging":document.getElementById("nextPaging"),
148 "prevPaging":document.getElementById("prevPaging")
149 });
```

```

150 });
151 };
152 </script>

```

本例 HTML 代码如下：

```

01 <h2>使用 JavaScript 对表格内容进行分页</h2>
02 <table id='tablePaging' border="1">
03 <tr>
04 <td>第 1 页内容</td>
05 <td>第 1 页内容</td>
06 </tr>
07 <tr>
08 <td>第 1 页内容</td>
09 <td>第 1 页内容</td>
10 </tr>
11 </table>
12 <p class="paging">
13 总页数1
14 当前页1
15 <input value='上一页' id='prevPaging' type="button" />
16 <input value='下一页' id='nextPaging' type="button" >
17 </p>

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13
14 </style>

```

### 【代码说明】

- 使用 JavaScript 分页主要涉及 3 个方面：视图的渲染、当前页的数据、处理过程。即当页数发生改变时，首先获取指定页的数据结构内容，然后将当前页的数据按一定的结构进行处理，渲染为视图。
- 本例中的页数变换主要采用了“上一页”与“下一页”的交互，请参考 JavaScript 代码第 91~116 行。
- 当页数发生变换时，调用 `getPageData()` 更新当前页的数据。

- 请求回来的最新数据调用 `tableUi()` 进行视图渲染更新。
- 本例中的视图更新，首先会清空所有的视图，然后再利用 4.4 节讲的 `trAct()` 函数，遍历数据增加“行”内容。

## 4.8 英文字符串自动换行

语言是沟通的基本元素之一，也是比较复杂的信息载体。例如：英文字符有各种格式、各种组合形式，为了在有限的版面中展示更多信息，可以对超出元素宽度的字符进行换行处理。换行处理方案有两种：样式和 JavaScript。

样式的处理通过 CSS 中的 “`word-break: break-all; word-wrap: break-word`” 属性实现，但是有一些兼容性问题。通过 JavaScript 实现这一效果，会损失一点性能，但不会有兼容性问题。本例效果如图 4-6 所示。

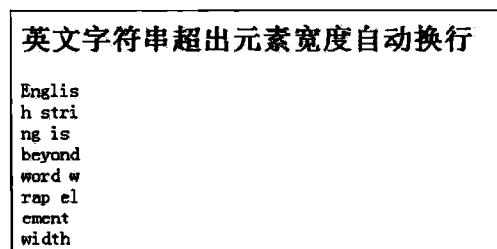


图 4-6 英文字符串自动换行

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function setCss(_this, cssOption){ //设置元素样式
04 //判断节点类型
05 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
06 return;
07 }
08 for(var cs in cssOption){
09 _this.style[cs] = cssOption[cs];
10 }
11 return _this;
12 }
13
14 function autoNewline(e){
15 var str = "", //初始化接收字符串对象
16 strContent = e.innerHTML, //被切割的字符
17 allWidth = getTextWidth(e), //被绑定元素的所有字体宽度
18 fontWidth = allWidth/strContent.length, //每个字体的宽度
19 rowWidth = Math.floor(e.offsetWidth/fontWidth); //每行最多放多少字

```

```

20
21 while(strContent.length > rowWidth){ //切割字符
22 str+=strContent.substr(0,rowWidth)+"
";
23 strContent=strContent.substr(rowWidth,strContent.length);
24 }
25 str+=strContent;
26
27 e.innerHTML=str; //设置元素的字符结果
28
29 }
30
31 function getTextWidth(e){ //获取文字的宽度
32 e = e.cloneNode(true); //深度克隆文字节点
33 var textWidth = 0,
34 _body = document.body;
35 _body.appendChild(e); //追加在 body 元素上
36
37 setCss(e, { //设置样式
38 "width" : "auto",
39 "position": "absolute",
40 "zIndex": -1
41 });
42 textWidth = e.offsetWidth; //获取宽度
43 _body.removeChild(e); //释放节点
44 return textWidth; //返回文字宽度
45 }
46 //英文字符串超出元素宽度时自动换行
47 autoNewline(document.getElementById("autoNewline"));
48 };
49 </script>

```

本例 HTML 代码如下：

```

01 <h2>英文字符串超出元素宽度自动换行</h2>
02 <div id='autoNewline'>English string is beyond word wrap element width</div>

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;

```

```

11 padding: 2px;
12 }
13 /*==英文字符串超出元素宽度自动换行==*/
14 #autoNewline{
15 display: block;
16 width: 50px;
17 /*样式处理换行，有兼容性问题
18 word-break: break-all;
19 word-wrap: break-word;*/
20 }
21 </style>

```

**【代码说明】**

- **业务流：**首先获取被绑定的元素，再获取元素内文字的总宽度，接着根据元素的宽度计算单个字的宽度，然后计算出每行最多放多少字。然后遍历所有字的长度，如果此长度大于每行字符最大值，则执行截取字符操作，并在后边增加<br />换行标签。直到结束循环，最后将所有截取的字符，追加到 str 变量中。
- **JavaScript 代码第 31~45 行的 getTextWidth()**可以获取字符串的宽度，主要是利用了 JavaScript 中的“e = e.cloneNode(true);”来深度克隆一份节点，设置样式“"width": "auto"，”保证宽度的准确性，然后再利用“e.offsetWidth;”获取元素的真实宽度，最后删除节点，释放节点。

## 4.9 内容超过元素宽度显示省略号

上一节中提到过语言的复杂性，实际的项目开发中，所有的内容都难免会出现溢出 div 的问题。为了让用户知道内容并没有显示完整，显示一段“...”是个不错的方案。本例效果如图 4-7 所示。

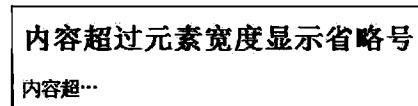


图 4-7 显示省略号

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function getAbsoluteLeft(_e) // 获取元素的绝对 left
04 var _left = _e.offsetLeft,
05 _current = _e.offsetParent;
06 while (_current !== null){ // 遍历父层，累加 left
07 _left += _current.offsetLeft;
08 _current = _current.offsetParent;

```

```

09 }
10 return _left;
11 }
12
13 function getAbsoluteTop(_e){ //获取元素的绝对 top
14 var _top = _e.offsetTop,
15 _current = _e.offsetParent;
16 while (_current !== null){ //遍历父层，累加 top
17 _top += _current.offsetTop;
18 _current = _current.offsetParent;
19 }
20 return _top;
21 }
22
23 function setCss(_this, cssOption){ //设置元素样式
24 //判断节点类型
25 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
26 return;
27 }
28 for(var cs in cssOption){
29 _this.style[cs] = cssOption[cs];
30 }
31 return _this;
32 }
33
34 function getTextWidth(e){ //获取文字的宽度
35 e = e.cloneNode(true); //深度克隆文字节点
36 var textWidth = 0,
37 _body = document.body;
38
39 _body.appendChild(e); //追加在 body 元素上
40
41 setCss(e, { //设置样式
42 "width" : "auto",
43 "position": "absolute",
44 "zIndex": -1
45 });
46
47 textWidth = e.offsetWidth; //获取宽度
48 _body.removeChild(e); //释放节点
49 return textWidth; //返回文字宽度
50 }
51
52 function contentApostrophe(e){

```

```

53 var _left = getAbsoluteLeft(e),
54 _top = getAbsoluteTop(e),
55 _w = e.offsetWidth;
56 e.style.overflow = "hidden";
57 /*循环节点，比较文字与目标元素的长度，如果文字的长度大于元素宽度，
58 则继续循环处理*/
59 while(getTextWidth(e) > _w){
60 //提取文字片段
61 e.innerHTML = e.innerHTML.substring(0, e.innerHTML.length-4);
62 e.innerHTML = e.innerHTML + "..."; //添加省略号
63 }
64
65 }
66 contentApostrophe(document.getElementById("contentApostrophe"));
67 };
68 </script>
```

本例 HTML 代码如下：

```

01 <h2>内容超过元素宽度显示省略号</h2>
02 <div id='contentApostrophe'>内容超过元素宽度显示省略号</div>
```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 /*=====内容超过元素宽度显示省略号=====*/
14 #contentApostrophe{
15 white-space: nowrap;
16 display: block;
17 width: 100px;
18 }
19 </style>
```

#### 【代码说明】

- 业务流：获取元素的宽度，当宽度大于元素的真实宽度时，截取字符串，并增加“...”。
- JavaScript 中的 `substring(start,stop)` 返回一个新的字符串。

## 4.10 调整字体大小

阅读是人类学习知识的有效途径，阅读时如果字体不合适可能会导致眼睛不舒服。例如：每个人的阅读习惯不一样，对字体大小有一些个人需求，那么在一些博客、新闻类的页面文章中加入字体设置功能，是一个不错的方案。修改字体的大小，主要是修改 CSS 中的 font-size 样式。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var _fontSize = document.getElementById("fontSize"); //获取元素对象
04 fontSize = function(e, unit){ //设置元素字体大小
05 e.style.fontSize = unit;
06 };
07
08 document.getElementById("fontSizeBig").onclick = function(){ //大字体设置
09 fontSize(_fontSize, "16px");
10 }
11
12 document.getElementById("fontSizeMedium").onclick = function(){ //中字体设置
13 fontSize(_fontSize, "14px");
14 }
15
16 document.getElementById("fontSizeSmall").onclick = function(){ //小字体设置
17 fontSize(_fontSize, "12px");
18 }
19 };
20 </script>
```

本例 HTML 代码如下：

```

01 <h2>调整字体大小</h2>
02 <p id='fontSize'>字体的大小变化展示</p>
03 <p>
04 <input id='fontSizeBig' value="大" type="button" />
05 <input id='fontSizeMedium' value="中" type="button" />
06 <input id='fontSizeSmall' value="小" type="button" />
07 </p>
```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
```

```

07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 </style>

```

**【代码说明】**

- 为变换字体大小的元素绑定 3 个按钮交互事件。
- 当单击按钮时，JavaScript 代码第 04~06 行调用 `fontSize()` 函数改变元素的 `font-size`，则对应的字体就会发生变化。

## 4.11 实现打字机效果

一些特殊的应用场景会引入打字机的效果，来模拟真实的打字过程。本例效果如图 4-8 所示。

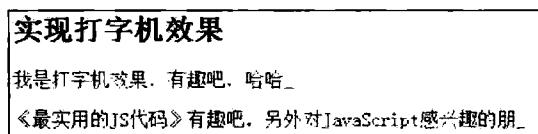


图 4-8 打字机效果

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var
04 typewriterArr = [], //打字的数据库队列
05 typewritering = false, //打字机的线程是否开启
06 typewriterID = -1, //打字机的线程 ID
07 typewriterTime = 1000, //定时调用的时间
08
09 typewriterEffect = function(e, str, color){ //增加显示的元素
10 typewriterArr.push({
11 "context" : e, //目标元素上下文
12 "str" : str, //显示的元素
13 "lening":0, //截取的进度
14 "maxLength":str.length //最大进度
15 });
16
17 e.style.color = color || "#000000"; //设置元素颜色
18 },
19

```

```

20 closeTypewriter = function(){
21 clearTimeout(typewriterID);
22 typewritering = false;
23 },
24
25 typewriterUi = function(){
26 var i = 0,
27 l = typewriterArr.length,
28 eing = null;
29 for(; i < l; i++){
30 eing = typewriterArr[i]
31
32 /*判断中英文进行+2 或+1 操作
33 递增，获取最新截取的长度*/
34
35 eing.lening++;
36 //如果截取的长度超过最大长度，则截取的长度设置为 1
37 if(eing.lening > eing.maxLength) eing.lening = 0;
38 //显示截取的字符
39 eing.context.innerHTML = eing.str.substring(0, eing.lening) + "_";
40 }
41 typewriterID = setTimeout(typewriterUi, typewriterTime);
42 },
43 //开启定时调用，参数为设置定时调用的时间
44 startTypewriter = function(typewriterTime){
45
46 if(!typewritering){ //如果没有开启，则开启
47 typewriterTime = typewriterTime || typewriterTime;
48 typewriterUi(); //开始定时调用
49 }
50 };
51
52 typewriterEffect(
53 document.getElementById("typewriterEffect"),
54 "我是打字机效果，有趣吧，哈哈！！！",
55 "red");
56 typewriterEffect(
57 document.getElementById("typewriterEffect2"),
58 "《最实用的 JavaScript 代码》有趣吧，另外对 JavaScript 感兴趣的朋友，请加 QQ
59 群：296811675，欢迎您的加入！！！",
60 "green");
61 startTypewriter(100);
62 };
63 </script>

```

本例 HTML 代码如下：

```
01 <h2>实现打字机效果</h2>
02 <p id='typewriterEffect'></p>
03 <p id='typewriterEffect2'></p>
```

本例 CSS 代码如下：

```
01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13
14 </style>
```

#### 【代码说明】

- JavaScript 代码第 41 行开启一个定时器 `setInterval`，不断地更新视图。
- JavaScript 代码第 10~15 行为每个元素保存一些初始属性值并存入 `typewriterArr`，等待定时调用业务处理时用。
- 在定时调用业务处理中，每次更新时都递增一次 “`eing.length++`”，用以截取新的字符长度，并且在字符后边追加 “`_`”，用来模拟打字效果，当递增的值超过最大长度时，截取长度重新设置为 0。

## 4.12 文本段落的展开和折叠

4.5 节讲过类似的效果，在文章展示类的 Web 页面中，经常会有信息量比较大的文章，但是页面的面积资源是有限的，因此一些新闻类网站，首先会展示文章的部分内容，如果用户对某文章比较感兴趣，可以展开全部查看。之前的演示没有加入动画模块，本例加入动画模块，实现展开和折叠效果。

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 //参数--context 是被绑定操作的元素；out 表示展开的元素；roundin 表示收缩的元素
04 function textOutIn(options){
05 var e = options.e;
06 }
```

```

07 options.out.onclick = function(){
08 if(e.style.height == "") return;
09 new animateManage({
10 "context": e,
11 "effect":"linear",
12 "time": 100,
13 "starCss":{
14 "height":e.style.height || 35
15 },
16 "css" :{
17 "height":203
18 }
19 }).init();
20 }
21
22 options.roundin.onclick = function(){
23 new animateManage({
24 "context": e,
25 "effect":"linear",
26 "time": 100,
27 "starCss":{
28 "height":e.style.height || 203
29 },
30 "css" :{
31 "height":35
32 }
33 }).init();
34 }
35
36 }
37
38 textOutIn({ //初始化对象
39 "e": document.getElementById("textOutIn"),
40 "out":document.getElementById("textOut"),
41 "roundin":document.getElementById("textRoundin")
42 });
43 };
44 </script>

```

本例 HTML 代码如下：

```

01 <h2>文本段落的展开和折叠效果</h2>
02 <div id='textOutInParent'>
03 <div id='textOutIn'>
04 <p>今天写的代码挺有趣，自学并思考很重要！我是文本段落的展开和折叠效果，我是
05 文本段落的展开和折叠效果我是文本段落的……

```

```

06 </p>
07 </div>
08 </div>
09 <input value="展开" type="button" id='textOut' />
10 <input value="收缩" type="button" id='textRoundin' />
```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 /*=====文本段落的展开和折叠效果=====*/
14 #textOutInParent{
15 display: block;
16 width: 500px;
17 height: 203px;
18 }
19 #textOutIn{
20 overflow: hidden;
21 height: 203px;
22 width: 500px;
23 display: block;
24 border: 1px #ccc solid;
25 }
26 #textOutIn p{
27 white-space: pre-wrap;
28 }
29 </style>
```

#### 【代码说明】

JavaScript 代码第 07~34 行实现文本段落的展开和折叠效果，主要是通过修改 `height` 高度来实现的。当展开时，增加 `height` 高度，当折叠时，减少 `height` 高度。

## 4.13 关键字的高亮显示

关键字的高亮有很多应用场景，例如中奖公告中会突显一些人名，强调谁中奖了。高

亮功能主要是通过颜色的变化来区别显示一些关键字，效果如图 4-9 所示。

### 关键字的高亮显示

我是关键字的高亮显示文本，高亮的文字有 JavaScript，还有 CSS，以及 HTML5！

图 4-9 关键字的高亮

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 //参数—e 代表被绑定的元素，keys 是关键字数组列表，color 设置关键词的高亮颜色
04 function keyWordsHighlight(e, keys, color){
05 var
06 i = 0,
07 l = keys.length, //关键词的长度
08 k = "";
09 for(; i < l ; i++){
10 k = keys[i]; //获取关键词的对象
11 //替换关键词的数据
12 e.innerHTML = e.innerHTML.replace(k, "<span style='color:" + (color ||
13 "#000") + "'>" + k + "")
14 }
15 }
16
17 keyWordsHighlight(document.getElementById("keyWordsHighlight"),
18 [
19 "JavaScript",
20 "CSS",
21 "HTML5"
22], "red")
23 };
24 </script>
```

本例 HTML 代码如下：

```

01 <h2>关键字的高亮显示</h2>
02 <p id="keyWordsHighlight">我是关键字的高亮显示文本，高亮的文字有 JavaScript，还有 CSS，
03 以及 HTML5！</p>
```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
```

```

09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 </style>

```

#### 【代码说明】

- 在 JavaScript 代码第 04~15 行的函数 keyWordsHighlight() 中，第 1 个参数是显示高亮文字的容器，第 2 个参数是关键字，第 3 个是关键字的颜色。
- 获取所有的字符，利用 “e.innerHTML.replace(k, "<span style='color:" + (color || "#000") + ">" + k + "</span>")” 将指定的关键字替换，并加上颜色 color。

## 4.14 字幕上下滚动

字幕上下滚动的效果，在公告栏或实时新闻中经常出现，它可以很大程度地节约空间，增加 Web 页面的使用率。本例效果如图 4-10 所示。



图 4-10 字幕滚动效果

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 //获取指定类型的节点
04 function getTypeElement(es, type){
05 var esLen = es.length,
06 i = 0,
07 eArr = [],
08 esl = null;
09 for(; i < esLen ; i++){
10 esl = es[i];
11 if(esl.nodeName.replace("#", "").toLowerCase() == type){
12 eArr.push(esl);
13 }
14 }
15 return eArr;
16 }

```

```

17 var
18 llis = [],
19 rollingE = null,
20 rollingID = -1,
21 rolling = "up";
22
23 function rollingSubtitles(e){ //初始化滚动层的数据
24 rollingE = e;
25 llis = getTypeElement(e.childNodes, "li"); //绑定滚动的元素
26
27 e.onmouseover = function(){ //当进入滚动层，停止滚动
28 clearInterval(rollingID);
29 }
30
31 e.onmouseout = function(){ //当离开滚动层，开启滚动
32
33 if(rolling == "down"){ //根据滚动的形式，选择 up/down
34 downSubtitles();
35 } else{
36 upSubtitles();
37 }
38 }
39 return this;
40
41 }
42 //向上滚动，参数--manualOperation 如果为 true，表示是单击按钮操作
43 function upSubtitles(manualOperation){
44 clearInterval(rollingID);
45 rolling = "up";
46 var up = function(){
47 new animateManage({
48 "context": rollingE, //被操作的元素
49 "effect": "linear",
50 "time": 200, //持续时间
51 "startCss": { //元素的起始值偏移量
52 "marginTop": rollingE.style.marginTop || 0
53 },
54 "css": { //元素的结束值偏移量
55 "marginTop": -103
56 },
57 "callback": function(){
58 llis = getTypeElement(rollingE.childNodes, "li");
59 rollingE.style.marginTop = "0px";
60 var cloneE = null;
61 for(var i = 0; i < 4; i++){

```

```

62 cloneE = llis[i].cloneNode(true);
63 rollingE.removeChild(llis[i]);
64 rollingE.appendChild(cloneE);
65 }
66
67 }
68 }).init();
69 }
70 if(manualOperation){
71 up();
72 }
73 rollingID = setInterval(function(){
74 up();
75 }, 3000)
76 }
77
78 //向下滚动，参数—manualOperation 如果为 true，表示是单击按钮操作
79 function downSubtitles(manualOperation){
80 clearInterval(rollingID);
81 rolling = "down";
82 var down = function(){
83 llis = getTypeElement(rollingE.childNodes, "li");
84 var len = llis.length - 1, //获取最大循环位置
85 l = len - 4, //获取最小循环位置
86 cloneE = null; //克隆的 li 对象
87
88 for(var i = len; i > l; i--) //递减方式遍历，并填充节点
89 //动态获取每次变更的 li
90 llis = getTypeElement(rollingE.childNodes, "li");
91 cloneE = llis[len].cloneNode(true); //克隆节点
92 rollingE.removeChild(llis[len]); //删除原有的节点
93 rollingE.insertBefore(cloneE,llis[0]); //在新的 0 点位置，插入节点
94 }
95
96 rollingE.style.marginTop = "-103px";
97
98 new animateManage({
99 "context": rollingE, //被操作的元素
100 "effect": "linear",
101 "time": 200, //持续时间
102 "starCss": {
103 "marginTop": rollingE.style.marginTop || 0
104 },
105 "css": { //元素的结束值偏移量
106 "marginTop": -2

```

```
107 },
108 "callback" : function(){
109
110 }
111 }).init();
112 }
113 if(manualOperation){
114 down();
115 }
116 rollingID = setInterval(function(){
117 down();
118 }, 3000)
119 }
120 var _downSubtitles = document.getElementById("downSubtitles");
121
122 /*字幕上下滚动*/
123 rollingSubtitles(document.getElementById("rollingSubtitles"))
124 downSubtitles();
125
126
127 document.getElementById("upSubtitles").onclick = function(){ //向上滚动
128 upSubtitles(true)
129 }
130
131 document.getElementById("upSubtitles").onmouseover = function(){//移入清除定时
132 clearInterval(rollingID);
133 }
134
135 document.getElementById("upSubtitles").onmouseout = function(){//移出开启定时
136 upSubtitles();
137 }
138
139 _downSubtitles.onclick = function(){ //向下滚动
140 downSubtitles(true)
141 }
142
143 _downSubtitles.onmouseover = function(){ //移入清除定时
144 clearInterval(rollingID);
145 }
146
147 _downSubtitles.onmouseout = function(){ //移出开启定时
148 downSubtitles();
149 }
150 };
151 </script>
```

本例 HTML 代码如下：

```
01 <h2>字幕上下间歇滚动</h2>
02 <div class="rollingSubtitles">
03 <ul id='rollingSubtitles' style="margin-top: -2px;">
04
05 QingJs 框架—1 页
06
07
08 <a
09 href="http://wpa.qq.com/msgrd?V=1&Uin=296811675&Site=chat&Menu=yes"
10 target="_blank">作者 QQ: 296811675
11
12
13 单击
14 作者新浪微博
15
16
17 <a
18 href="http://wpa.qq.com/msgrd?V=1&Uin=296811675&Site=chat&Menu=yes"
19 target="_blank">作者 QQ: 296811675
20
21
22 QingJs 框架—2 页
23
24
25 <a
26 href="http://wpa.qq.com/msgrd?V=1&Uin=296811675&Site=chat&Menu=yes"
27 target="_blank">作者 QQ: 296811675
28
29
30 单击
31 作者新浪微博
32
33
34 QingJs 框架
35
36
37 <a
38 href="http://wpa.qq.com/msgrd?V=1&Uin=296811675&Site=chat&Menu=yes"
39 target="_blank">作者 QQ: 296811675—3 页
40
41
42 单击
43 作者新浪微博
```

```

44
45
46
47 </div>
48

49 <input type="button" id='upSubtitles' value='向上滚动' />
50 <input type="button" id='downSubtitles' value='向下滚动' />

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 /*=====字幕上下间歇滚动=====*/
14 .rollingSubtitles{
15 width: 350px;
16 height: 94px;
17 min-height: 25px;
18 line-height: 25px;
19 border: #cc1233 1px solid;
20 overflow: hidden;
21 }
22 </style>

```

#### 【代码说明】

- 本例默认是向下滚动的。
- JavaScript 代码第 79~112 行的 `downSubtitles()` 处理向下滚动业务，在滚动的过程中，总会面临节点不够的情景，所以在每次向下滚动之前，在节点的前面追加几个节点，补齐缺少的元素，然后再进行动画业务处理，设置“`marginTop`”:-2。
- 向上滚动是先进行动画业务，当动画业务结束后，再进行追加节点业务，补全节点。
- 当鼠标移入节点时关闭自动循环，移除时开启自动循环。

## 4.15 弹出层

弹出层效果的应用场景很多，例如确认按钮、警告框等。本例效果如图 4-11 所示。

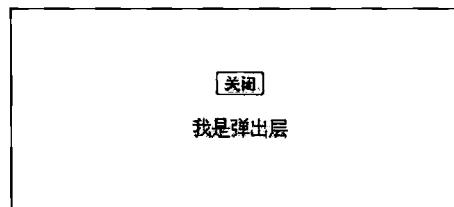


图 4-11 弹出层效果

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function setCss(_this, cssOption){ //设置元素样式
04 //判断节点类型
05 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
06 return;
07 }
08 for(var cs in cssOption){
09 _this.style[cs] = cssOption[cs];
10 }
11 }
12
13 function setPopup(e, openPop, closePop){
14 setCss(e, { //初始化样式
15 "position":"absolute",
16 "zIndex":100,
17 "backgroundColor":"#EBEDF3"
18 });
19 openPop.onclick = function(){
20 e.style.display = "block";
21 setCss(e, { //修改弹出层的位置，将其定位于屏幕可见区域的中间位置
22 "left": "50%",
23 "marginLeft": -e.offsetWidth/2 + "px",
24 "top":((document.body.scrollTop ||
25 document.documentElement.scrollTop) + window.screen.availHeight/2- e.offsetHeight) + "px"
26 });
27 }
28
29 closePop.onclick = function(){
30 e.style.display = "none";
31 }
32 }
33 //弹出层*****setPopup(
34 setPopup(
35 document.getElementById("popupDiv"),

```

```

36 document.getElementById("popupOpen"),
37 document.getElementById("popupClose")
38);
39 }
40 </script>

```

本例 HTML 代码如下：

```

01 <h2>弹出层</h2>
02 <div id='popupDiv' style="border:1px solid #ccc;display: none;height: 100px;width: 300px;
03 text-align: center;">
04 <p><input type="button" id='popupClose'value='关闭' /></p>
05 <p>我是弹出层</p>
06 </div>
07 <input type="button" id='popupOpen'value='弹出' />

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 </style>

```

#### 【代码说明】

- JavaScript 代码第 13~32 行初始化绑定元素的样式，修改为绝对定位，这样可以自由地在整个页面定位元素的位置，方便脚本控制。
- 绑定一个 click，当发生事件交互时，将弹出层的 display 修改为 block，即可见状态，并且设置坐标为可见区域的中间位置。

## 4.16 用层模拟的提示消息框

学完上一节的内容后，读者或许也猜到了本例效果的大概原理。一般情况下只要调用 JavaScript 的 API 就可以弹出一些系统提示框，但是那种提示框比较难看，如果要让弹出框 UI 风格与网站的风格一致，就需要用层来模拟提示消息框了。本例效果如图 4-12 所示。

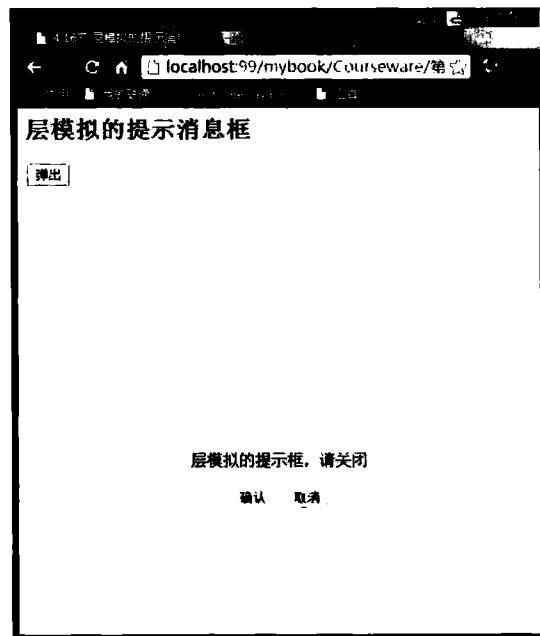


图 4-12 用层模拟提示消息框

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 function promptMsgBox(options){
04 var e = options.promptMsgBox, //获取元素
05 setCss = function(_this, cssOption){ //设置元素样式
06 //判断节点类型
07 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
08 return;
09 }
10 for(var cs in cssOption){
11 _this.style[cs] = cssOption[cs];
12 }
13 return _this;
14 };
15
16 setCss(e, { //初始化基本样式
17 "position":"absolute",
18 "zIndex":100,
19 "top":((document.body.scrollTop || document.documentElement.scrollTop)
20 + window.screen.availHeight/2- e.offsetHeight) + "px",
21 "backgroundColor":"#EBEDF3"
22 });
23
24 options.promptMsgBoxOpen.onclick = function(){ //打开确认按钮
25 //处理逻辑
26 }
27 }
28 }
29 }
```

```

25 e.style.display = "block";
26
27 setCss(e, { //设置位置
28 "left": "50%" ,
29 "marginLeft": -e.offsetWidth/2 + "px",
30 /*计算目标元素的 top 值：（页面的整体高度）减去（文档可见的高一
31 半高度）减去（元素的一半高度）*/
32 "top":((document.body.scrollTop ||
33 document.documentElement.scrollTop) + window.screen.availHeight/2- e.offsetHeight) + "px"
34 });
35 }
36
37 options.promptMsgBoxAgree.onclick = function(){ //确认按钮
38 e.style.display = "none"; //隐藏层
39 //如果存在确认的回调，则执行
40 if(options.agreeCallBack) options.agreeCallBack();
41 }
42
43 options.promptMsgBoxCancel.onclick = function(){//取消按钮
44 e.style.display = "none"; //隐藏层
45 //如果存在取消的回调，则执行
46 if(options.cancelCallBack) options.cancelCallBack();
47 }
48 }
49
50 promptMsgBox({ //层模拟的提示消息框
51 "promptMsgBox":document.getElementById("promptMsgBox"),//响应的元素
52 //打开确认按钮
53 "promptMsgBoxOpen": document.getElementById("promptMsgBoxOpen"),
54 //确认按钮
55 "promptMsgBoxAgree": document.getElementById("promptMsgBoxAgree"),
56 "agreeCallBack":function(){ //确认之后的回调函数
57 alert("确认的回调函数！")
58 },
59 //取消按钮
60 "promptMsgBoxCancel": document.getElementById("promptMsgBoxCancel"),
61 "cancelCallBack":function(){ //取消之后的回调函数
62 alert("取消的回调函数！")
63 }
64 });
65 }
66 };
67 </script>

```

本例 HTML 代码如下：

```

01 <h2>层模拟的提示消息框</h2>
02 <div id='promptMsgBox' style="border:1px solid #ccc;display: none;height: 100px;width:
03 300px; text-align: center;">
04 <p>层模拟的提示框, 请关闭</p>
05 <p>
06 <input type="button" id='promptMsgBoxAgree'value='确认' />
07 <input type="button" id='promptMsgBoxCancel'value='取消' />
08 </p>
09 </div>
10 <input type="button" id='promptMsgBoxOpen' value='弹出' />

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 </style>

```

### 【代码说明】

- 用层模拟提示框的主要构成：提示信息、确认按钮、取消按钮。
- JavaScript 代码第 03 行调用 `promptMsgBox()` 为元素节点绑定交互。
- 初始化层的基本样式，设置为绝对定位，方便脚本控制。
- JavaScript 代码第 24~35 行触发 “`options.promptMsgBoxOpen.onclick`” 事件时，设置目标元素的样式 “`e.style.display = "block";`”，并且设置坐标为可见区域的中间位置。
- 当单击“确认”按钮时，关闭目标元素。如果存在“确认”按钮的回调函数，则执行回调。
- 当单击“取消”按钮时，关闭目标元素。如果存在“取消”按钮的回调函数，则执行回调。

## 4.17 隐藏层

隐藏层是 Web 应用中最常见的需求了，例如：初次进入某课程网站的用户，会看到一些执行教学任务的元素层，教学完毕后，层自动隐藏。为了方便演示隐藏层的效果，增加一个交互功能：单击按钮隐藏上边的层。本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 document.getElementById("hidePrev").onclick = function(){ //绑定隐藏单击事件
04 document.getElementById("hide").style.display = "none"; //隐藏元素
05 }
06 };
07 </script>
```

本例 HTML 代码如下：

```
01 <h2>隐藏层</h2>
02 <div id="hide" style="position:relative; border: 1px solid #ccc; width: 200px; height:
03 100px; display: block">
04 我是将要被隐藏的层！</div>
05 <input type="button" id="hidePrev" value="隐藏上边的层" />
```

本例 CSS 代码如下：

```
01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 </style>
```

## 【代码说明】

- JavaScript 代码第 04 行实现了隐藏层的功能，通过修改元素的 `display` 属性实现，关键是“`e.style.display = "none";`”。
  - 当单击交互元素时，直接修改元素的样式，隐藏节点。

## 4.18 用层实现滚动条

在一些项目中，浏览器自带的滚动条不一定能满足需求。如果想让滚动条的样式看起来更加美观，与网站的 UI 风格最好一致，但是浏览器的滚动条不可以自定义，也不可以随意用 JavaScript 处理滚动条，那么模拟滚动条就有用武之地了。本例效果如图 4-13 所示。

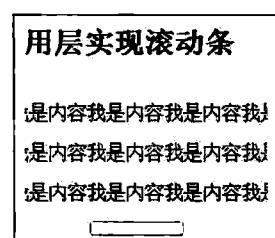


图 4-13 用层实现滚动条

```

02 window.onload = function(){
03 /**
04 * 获取鼠标在页面上的位置
05 * _e: 触发的事件
06 * left: 鼠标在页面上的横向位置, top: 鼠标在页面上的纵向位置
07 */
08 var
09 _mousepos = { //鼠标在页面上的位置
10 "top":0,
11 "left":0
12 },
13
14 getMousePoint = function (_e) {
15 var _body = document.body,
16 _left = 0,
17 _top = 0
18 ;
19 /*如果浏览器支持 pageYOffset, 那么可以使用 pageXOffset 和 pageYOffset
20 获取页面和视窗之间的距离*/
21 if(typeof window.pageYOffset != 'undefined') {
22 _left = window.pageXOffset;
23 _top = window.pageYOffset;
24 }
25 //如果浏览器指定了 DOCTYPE 并且支持 compatMode
26 else if(typeof document.compatMode != 'undefined' &&
27 document.compatMode != 'BackCompat') {
28 _left = document.documentElement.scrollLeft;
29 _top = document.documentElement.scrollTop;
30 }
31 else if(typeof _body != 'undefined') {//其他浏览器如果支持 document.body
32 _left = _body.scrollLeft;
33 _top = _body.scrollTop;
34 }
35
36 _left += _e.clientX;
37 _top += _e.clientY;
38 _mousepos.left = _left;
39 _mousepos.top = _top;
40
41 return _mousepos;
42 },
43
44 getAbsoluteLeft = function (_e){ //获取元素的绝对 left
45 var _left = _e.offsetLeft,
46 _current = _e.offsetParent;

```

```

47 while (_current !== null){
48 _left += _current.offsetLeft;
49 _current = _current.offsetParent;
50 }
51 return _left;
52 },
53
54 getAbsoluteTop = function (_e){ //获取元素的绝对 top
55 var _top = _e.offsetTop,
56 _current = _e.offsetParent;
57 while (_current !== null){
58 _top += _current.offsetTop;
59 _current = _current.offsetParent;
60 }
61 return _top;
62 },
63 isScrollBar = false, //是否开启滚动
64 scrollBarLeft = 0, //拖曳的修正值
65 scrollBarMaxLeft = 0, //left 最大拖曳值
66 scrollBarObj = null, //被拖曳的滚轴元素的对象
67 scrollBarContent = null, //被拖动的内容元素
68 scrollBarScale = 1, //转换比例
69
70 scrollBar = function(options){ //初始化滚动条
71 var _parent = options.scrolling.parentNode,
72 parentWidth = _parent.offsetWidth,
73 contentWidth = options.contentScroll.scrollWidth,
74 _scrolling = options.scrolling,
75 _scale = scrollBarScale = parentWidth / contentWidth,
76 _scrollingWidth = parentWidth * _scale;
77
78 scrollBarContent = options.contentScroll; //初始化内容元素对象
79 scrollBarObj = _scrolling; //初始化变量差值
80 _scrolling.style.width = _scrollingWidth + "px"; //初始化滚动条长度
81 scrollBarMaxLeft = getAbsoluteLeft(_parent) + _parent.offsetWidth -
82 _scrolling.offsetWidth - 10; //初始化最大 left 值
83
84 _scrolling.onmousedown = function(event){ //开启滚动
85 //获取拖曳对象的坐标
86 event = event || window.event;
87 var _pos = getMousePoint(event);
88 isScrollBar = true;
89 scrollBarLeft = _pos.left - getAbsoluteLeft(this);
90 }
91

```

```

92 },
93
94 closeScrollBar = function(){ //关闭滚动条
95 isScrollBar = false;
96 },
97
98 moveScrollBar = function(event){ //移动滚动条
99 if(isScrollBar){ //如果开启滚轴
100 event = event || window.event; //获取拖曳对象的坐标
101 var _pos = getMousePoint(event),
102 _left = _pos.left - scrollBarLeft,
103 cLeft = _left;
104 if(_left < 0) {
105 _left = 0;
106 cLeft = 0;
107 }
108 //如果滚轴的 left 坐标大于最大值则修正
109 if(_left > scrollBarMaxLeft) _left = scrollBarMaxLeft;
110
111 if(cLeft > scrollBarMaxLeft + 10) cLeft = scrollBarMaxLeft + 10;
112
113 scrollBarObj.style.left = _left + "px"; //设置滚轴层的 left
114
115 //设置内容层 left 变化
116 scrollBarContent.style.left = -1 * cLeft / scrollBarScale + "px"
117 }
118 };
119
120 scrollBar({
121 "contentScroll":document.getElementById("contentScroll"),
122 "scrolling":document.getElementById("scrolling")
123 });
124
125 document.body.onmousemove = function(e){ //拖动层移动
126 moveScrollBar(e); //用层实现滚动条
127 }
128
129 document.onmouseup = function(e){ //拖动结束
130 closeScrollBar(e);
131 }
132};
133</script>

```

本例 HTML 代码如下：

01 <h2>用层实现滚动条</h2>

```

02 <!--最外层-->
03 <div style="position: relative; background-color: #F6F6F6; width: 200px; border: 1px solid #ccc
04 solid; overflow: hidden">
05 <!--内容包-->
06 <div id="contentScroll" style="position: relative; left: 0px">
07 <p>我是内容我是内容我是内容我是内容我是内容我是内容我是内容我
08 123456789<p>
09 <p>我是内容我是内容我是内容我是内容我是内容我是内容我是内容我
10 123456789<p>
11 <p>我是内容我是内容我是内容我是内容我是内容我是内容我是内容我
12 123456789<p>
13 </div>
14 <!--横向滚动条-->
15 <div id="scrolling" style="position: relative; display: block; width: 100px; height:
16 10px; background-color: #fff; border: 1px solid #979797; -moz-border-radius: 3px;
17 -webkit-border-radius: 3px;"></div>
18 </div>

```

本例 CSS 代码如下：

```

01 <style>
02 h2, h5, #tooltipMsg, p{
03 white-space: nowrap;
04 }
05 td{
06 border: 1px solid #ccc;
07 height: 50px;
08 text-align: center;
09 font-size: 10pt;
10 padding: 2px;
11 }
12 </style>

```

### 【代码说明】

- 先来看滚动条的长度计算，JavaScript 代码第 75 行首先计算内容宽度与外层元素的比例“\_scale = scrollBarScale = parentWidth / contentWidth”，然后计算滚动条的宽度“\_scrollingWidth = parentWidth \* \_scale”，并且初始化鼠标按下的初始位置。
- 滚动条随着鼠标移动，当鼠标按下时，开启滚动条的移动状态。当鼠标移动时，JavaScript 代码第 98~118 行调用 moveScrollBar() 处理滚动条的移动。因为滚动条只是 left 值变化，所以只要修改 left 即可。让滚动条的 left 等于鼠标指针的 left 并不符合实际的位置，因此减去鼠标按下的差值来修正位置“\_left = \_pos.left - scrollBarLeft”，当 left 值小于 0 或大于最大值时，修正其实际位置。
- 最后按比例计算修改内容的移动值：“scrollBarContent.style.left = -1 \* cLeft / scrollBarScale + "px"”。

## 4.19 让层可以随意拖动

让层可以随意拖动是一个有趣、有用的需求，例如：在 QQ 空间中，用户可以随意地拖动 QQ 空间内的一些模块，根据自己的爱好来摆放位置。本例效果如图 4-14 所示。

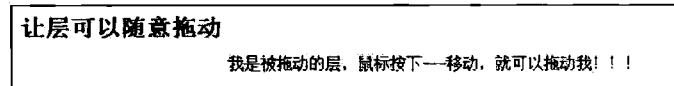


图 4-14 被拖动的层向右移动了一部分

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 /**
04 * 获取鼠标在页面上的位置
05 * _e: 触发的事件
06 * left: 鼠标在页面上的横向位置， top: 鼠标在页面上的纵向位置
07 */
08
09 _mousepos = { //鼠标在页面上的位置
10 "top":0,
11 "left":0
12 }
13
14 function getMousePoint(_e){ //获取鼠标的坐标点
15 var _body = document.body,
16 _left = 0,
17 _top = 0
18 ;
19 /*如果浏览器支持 pageYOffset, 那么可以使用 pageXOffset 和 pageYOffset
20 获取页面和视窗之间的距离*/
21 if(typeof window.pageYOffset != 'undefined') {
22 _left = window.pageXOffset;
23 _top = window.pageYOffset;
24 }
25 //如果浏览器指定了 DOCTYPE 并且支持 compatMode
26 else if(typeof document.compatMode != 'undefined' &&
27 document.compatMode != 'BackCompat') {
28 _left = document.documentElement.scrollLeft;
29 _top = document.documentElement.scrollTop;
30 }
31 //如果浏览器支持 document.body
32 else if(typeof _body != 'undefined') {
33 _left = _body.scrollLeft;

```

```
34 _top = _body.scrollTop;
35 }
36
37 _left += _e.clientX;
38 _top += _e.clientY;
39
40 _mousepos.left = _left;
41 _mousepos.top = _top;
42
43 return _mousepos;
44 }
45
46 function setCss(_this, cssOption){ //设置元素样式
47 //判断节点类型
48 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
49 return;
50 }
51 for(var cs in cssOption){ //遍历设置样式
52 _this.style[cs] = cssOption[cs];
53 }
54 return _this;
55 }
56
57 function getAbsoluteLeft(_e){ //获取元素的绝对 left
58 var _left = _e.offsetLeft,
59 _current = _e.offsetParent;
60 while (_current !== null){ //遍历父节点累加 left
61 _left += _current.offsetLeft;
62 _current = _current.offsetParent;
63 }
64 return _left;
65 }
66
67 function getAbsoluteTop(_e){ //获取元素的绝对 top
68 var _top = _e.offsetTop,
69 _current = _e.offsetParent;
70 while (_current !== null){ //遍历父节点累加 top
71 _top += _current.offsetTop;
72 _current = _current.offsetParent;
73 }
74 return _top;
75 }
76
77 drawElement = null; //是否可拖动元素
78 isDrawElement = false;
```

```
79 startMousePos = {
80 "left": 0,
81 "top": 0
82 }
83 //绑定要滚动的元素
84 function bindDrawElement(e){
85 drawElement = e;
86 var absoluteLeft = getAbsoluteLeft(e),
87 absoluteTop = getAbsoluteTop(e);
88
89 setCss(e, {
90 "position": "absolute",
91 "left": absoluteLeft + "px",
92 "top": absoluteTop + "px",
93 "cursor": "move",
94 "zIndex": 101
95 });
96 e.onmousedown = function(event){
97 event = event || window.event;
98 var _pos = getMousePoint(event);
99 isDrawElement = true; //开启拖曳
100
101 startDrawPos = {
102 "left": _pos.left - getAbsoluteLeft(this),
103 "top": _pos.top - getAbsoluteTop(this)
104 }
105 }
106
107 e.onmouseup = function(){
108 isDrawElement = false;
109 }
110 }
111
112 function moveDraw(event){
113 if(isDrawElement){
114 event = event || window.event; //如果开启滚轴
115 var _pos = getMousePoint(event); //获取拖曳对象的坐标
116 setCss(drawElement, {
117 "left": (_pos.left - startDrawPos.left) + "px",
118 "top": _pos.top - startDrawPos.top + "px"
119 })
120 }
121 }
122 }
123 //让层可以随意拖动
```

```

124 bindDrawElement(document.getElementById("bindDrawElement"));
125
126 document.body.onmousemove = function(e){
127 moveDraw(e); //让层可以随意拖动
128 }
129
130
131 };
132 </script>

```

本例 HTML 代码如下：

```

01 <h2>让层可以随意拖动</h2>
02 <div id='bindDrawElement'>我是被拖动的层，鼠标按下一移动，就可以拖动我！！！</div>
03 <div style="position: relative; top:50px; height: 10px"> </div>

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 /*=====让层可以随意拖动=====*/
14 #bindDrawElement{
15 border: 1px solid #ccc;
16 }
17 </style>

```

#### 【代码说明】

- 当鼠标按下时开启拖曳，计算鼠标指针的坐标与元素的差值，参见 JavaScript 代码第 101~104 行。
- 当元素被拖动时，首先判断是否开启拖曳，如果开启，则执行拖曳的逻辑。
- 修改被拖曳对象的 left 与 top，并减去差量，不断地移动，重复此业务就看见被拖曳的效果了，参见 JavaScript 代码第 112~122 行。
- 鼠标键抬起时，关闭拖动，修改“isDrawElement = false;”。

## 4.20 遮罩层效果

遮罩层是为了限制用户的一些行为或者突显局部页面，例如：在需要实名认证的网页

中，弹出提示框要求用户填写一些认证信息，如果用户不填写认证信息，就通过遮罩的形式限制用户操作页面内的其他元素。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function setCss(_this, cssOption){ //设置元素样式
04 //判断节点类型
05 if(!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
06 return;
07 }
08 for(var cs in cssOption){ //遍历设置样式
09 _this.style[cs] = cssOption[cs];
10 }
11 return _this;
12 }
13
14 document.getElementById("showMaskLayer").onclick = function(){//获取响应元素
15 var b = document.body.parentNode,
16 maskLayer = document.getElementById("maskLayer");
17
18 setCss(maskLayer, { //初始化遮罩的样式
19 "position":"absolute", //绝对
20 "left":"0px",
21 "display":"block",
22 "top":"0px",
23 "zIndex":1000, //Z 层级
24 "backgroundColor":"#ccc",
25 "height": b.scrollHeight + "px",
26 "width": b.offsetWidth + "px",
27 /*为了兼容各种浏览器的透明层效果*/
28 "filter":"alpha(Opacity=60)",
29 "opacity" : "0.6",
30 "-moz-opacity":"0.6",
31 "filter": "progid:DXImageTransform.Microsoft.Alpha(Opacity=60)",
32 "-MS-filter":"progid:DXImageTransform.Microsoft.Alpha(Opacity=60)"
33 })
34 }
35
36
37 };
38 </script>
```

本例 HTML 代码如下：

```

01 <h2>遮罩层效果</h2>
02 <input type="button" value='显示遮罩层' id='showMaskLayer' />
03 <div id='maskLayer' style="display: none"></div>
```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13
14 </style>
```

#### 【代码说明】

- 遮罩层的实现，主要是通过层或图片区来模拟遮罩。
- 以上例子通过使用 JavaScript 修改遮罩层 div 的样式来模拟遮罩。
- 设置定位为绝对定位，透明度为 0.6，并设置遮罩层的宽度与高度。

## 4.21 Tab 选项卡切换

Tab 选项卡可以帮助用户自由切换自己想要的内容及模块，例如：购物网站有很多产品，每个产品都有自己的种类，如果将所有的产品名称都显示出来显然是不太科学的。而如果将每个产品分类处理，采用 Tab 切换的形式，展示指定种类的产品，不但引导了用户选择产品的过程，还节约了页面空间。本例效果如图 4-15 所示。

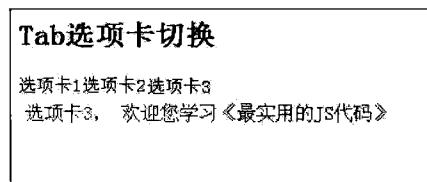


图 4-15 选中选项卡 3，并显示其内容

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function getTypeElement(es, type){ //获取指定类型的节点
04 var esLen = es.length,
05 i = 0,
06 eArr = [],
07 esl = null;
```

```

08 for(; i < esLen ; i++){
09 esl = es[i];
10 if(esl.nodeName.replace("#", "").toLowerCase() == type){
11 eArr.push(esl);
12 }
13 }
14 return eArr;
15 }
16
17 function tabSwitch(e){
18 var divs = getTypeElement(e.childNodes, "div"),
19 l = divs.length,
20 i = 0;
21 for(; i < l;i++){
22 divs[i].onclick = function(){ //单击进行切换
23 for(var ii = 0; ii < l; ii++){ //改为未被选中状态
24 divs[ii].className = ""; //删除选项卡的边框
25 //隐藏内容
26 document.getElementById("tabSwitch" + (ii+1)).style.display
27 ="none";
28 }
29 this.className = "on"; //设置当前元素的选中样式
30 //获取指定内容的对象并显示
31
32 document.getElementById(this.getAttribute("data-targent")).style.display = "block";
33 }
34 }
35 }
36
37 tabSwitch(document.getElementById("tabSwitch")); //Tab 选项卡切换
38 };
39 </script>

```

本例 HTML 代码如下：

```

01 <h2>Tab 选项卡切换</h2>
02 <div class="tabSwitchParent">
03 <!--选项卡-->
04 <div id='tabSwitch'>
05 <div data-targent="tabSwitch1' class="on">选项卡 1</div>
06 <div data-targent="tabSwitch2">选项卡 2</div>
07 <div data-targent="tabSwitch3">选项卡 3</div>
08 </div>
09 <!-- 内容-->
10 <div class="tabSwitchTarget" style="display: block;color: #123323" id='tabSwitch1'>
11 选项卡 1,

```

```
12 欢迎您学习《最实用的 JavaScript 代码》
13 </div>
14 <!-- 内容 -->
15 <div class="tabSwitchTarget" style="color: #abcdee" id="tabSwitch2">
16 选项卡 2,
17 欢迎您学习《最实用的 JavaScript 代码》
18 </div>
19 <!-- 内容 -->
20 <div class="tabSwitchTarget" style="color: #444321" id="tabSwitch3">
21 选项卡 3,
22 欢迎您学习《最实用的 JavaScript 代码》
23 </div>
24 </div>
```

本例 CSS 代码如下：

```
01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 /*=====Tab 选项卡切换=====*/
14 .tabSwitchParent{
15 position: relative;
16 height: 100px;
17 width: 200px;
18 }
19 .tabSwitchParent div{
20 position: relative;
21 float: left;
22 }
23 .tabSwitchTarget{
24 display: none;
25 left: 0;
26 top: 0px;
27 z-index: 1;
28 width: 324px;
29 height: 54px;
30 padding: 5px;
```

```

31 border: solid 1px #ccc;
32 color: #666;
33 }
34 #tabSwitch{
35 position: relative;
36 float: left;
37 z-index: 2;
38 top:1px;
39 }
40 #tabSwitch{
41 font-size: 15px;
42 margin: 0px;
43 text-align: center;
44 cursor: pointer;
45 }
46 #tabSwitch .on{
47 border: 1px solid #ccc;
48 border-bottom: none;
49 background-color: #fff;
50 }
51 </style>

```

**【代码说明】**

- JavaScript 代码第 22~33 行为每一个特定的选项卡绑定 click 事件。
- 当单击元素时，修改被单击元素的样式为选中状态，其他元素修改为未选中状态，在被选中的元素节点上，获取将要显示的选项内容 id，将所有的选项内容隐藏，即“display = "none";”，然后显示选取内容的对象，设置样式“display = "block";”。

## 4.22 对联浮动广告

浏览一些门户网站时，经常会看到有类似对联效果的浮动广告，如图 4-16 所示，是不是在新浪、搜狐网站上经常看到？本例就来实现这样一个对联浮动广告。



图 4-16 对联浮动广告效果

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){

```

```

03 //对联广告单击显示
04 document.getElementById("openAbs").onclick = function(){
05 var
06 coupletAdsLeft = null,
07 coupletAdsRight = null,
08
09 coupletAds = function(absLeft, absRight){
10 coupletAdsLeft = absLeft;
11 coupletAdsRight = absRight;
12
13 this.coupletAdsLeft.style.display = "block";
14 this.coupletAdsRight.style.display = "block";
15 absTop();
16 },
17
18 absTop = function(){ //两个对联层的 top 位置
19 if(!this.coupletAdsLeft) return;
20 /*如果 document.body.scrollTop == 0 选择
21 document.documentElement.scrollTop 值， */
22 var top = ((document.body.scrollTop ||
23 document.documentElement.scrollTop) + window.screen.availHeight/2 -
24 this.coupletAdsLeft.offsetHeight) + "px";
25 this.coupletAdsLeft.style.top = top;
26 this.coupletAdsRight.style.top = top;
27 },
28
29 scrollEvent = (function(){ //滚轴事件
30 window.onscroll = function(){
31 absTop(); //重新修正广告 top
32 }
33 })();
34
35 coupletAds(
36 document.getElementById("coupletAdsLeft"),
37 document.getElementById("coupletAdsRight")
38);
39 }
40 };
41 </script>

```

本例 HTML 代码如下：

```

01 <h2>对联浮动广告</h2>
02 <div class="coupletAds" id='coupletAdsLeft'>浮动广告左边</div>
03 <div class="coupletAds" id='coupletAdsRight'>浮动广告右边</div>
04 <input type="button" id='openAbs' value='显示对联广告'>

```

```

05 <p>测试浮动广告`~~~~~`</p>
06

07 <p>测试浮动广告`~~~~~`</p>
08

09 <!--其余重复代码略-->
本例 CSS 代码如下：
01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 /*=====对联浮动广告=====*/
14 .coupletAds{
15 position: absolute;
16 width: 100px;
17 border: 1px #ccc solid;
18 height: 200px;
19 z-index: 2000;
20 background-color: #ccc;
21 white-space: pre-wrap;
22 display: none;
23 }
24 #coupletAdsLeft{
25 left: 5px;
26 }
27 #coupletAdsRight{
28 left:100%;
29 margin-left: -107px;
30 }
31 </style>

```

**【代码说明】**

- 单击“显示对联广告”按钮，创建左浮动广告与右浮动广告的元素。
- 初始化时，JavaScript 代码第 18~27 行修正两个广告节点的 top 值。
- 当触发 scroll 事件时更改坐标，继续调用 absTop() 函数修正 top 值。

## 4.23 类似 QQ 消息窗口提示

QQ 的消息提示方式众所周知，现在的腾讯公司将 QQ 消息窗口运营的淋漓尽致。例如：当发生重大新闻时会通知用户、有其他用户在当前用户发表评论时也以 QQ 消息的形式提醒。本例开发一个类似的模块，效果如图 4-17 所示。

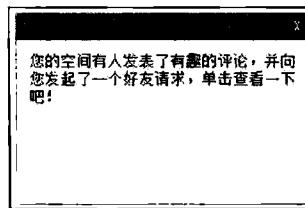


图 4-17 类似 QQ 消息的弹窗

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 //类似 QQ 消息窗口提示
04 document.getElementById("showQQMsg").onclick = function(){
05 var
06 setCss = function(_this, cssOption){ //设置元素样式
07 //判断节点类型
08 if (!_this || _this.nodeType === 3 || _this.nodeType === 8 || !_this.style) {
09 return;
10 }
11 for(var cs in cssOption){
12 _this.style[cs] = cssOption[cs];
13 }
14 return _this;
15 },
16 QQMsgE = null,
17 QQMsgTop = function(){ //设置 QQ 窗口的 top
18 if(QQMsgE && QQMsgE.style.display != "none"){ //判断是否隐藏
19 var bodyHeight = (document.body.scrollTop ||
20 document.documentElement.scrollTop) + window.screen.availHeight;
21 this.setCss(QQMsgE, {
22 "top": (bodyHeight - 260) + "px",
23 "left": "100%",
24 "marginLeft": "-244px"
25 });
26 }
27 },
28 QQMsg = function (e, closeQQMsg){
29 QQMsgE = e;

```

```

30 //top 的值 = 滚轴的高度 top+窗口的高度 , 目的是隐藏窗口
31 var bodyHeight = (document.body.scrollTop ||
32 document.documentElement.scrollTop) + window.screen.availHeight;
33 QQMsgE.style.display = "block";
34 setCss(QQMsgE, {
35 "top": bodyHeight+ "px",
36 "left": "100%",
37 "marginLeft": "-244px"
38 });
39 new animateManage({
40 "context" : e, //被操作的元素
41 "effect": "linear",
42 "time": 200, //持续时间
43 "starCss":{ //元素的起始值偏移量
44 "top": bodyHeight
45 },
46 "css" :{ //元素的结束值偏移量
47 "top": bodyHeight - 260
48 },
49 "callback" : function(){
50
51 }
52 }).init();
53 closeQQMsg.onclick = function(){
54
55 new animateManage({
56 "context" : e, //被操作的元素
57 "effect": "linear",
58 "time": 200, //持续时间
59 "starCss":{ //元素的起始值偏移量
60 "top": bodyHeight - 260
61 },
62 "css" :{ //元素的结束值偏移量
63 "top": bodyHeight
64 },
65 "callback" : function(){
66 e.style.display = "none";
67 }
68 }).init();
69 }
70
71 };
72
73 QQMsg(
74 document.getElementById("QQMsg"),

```

```

75 document.getElementById("closeQQMsg")
76);
77 }
78 };
79 </script>

```

本例 HTML 代码如下：

```

01 <h2>类似 QQ 消息窗口提示</h2>
02 <div id='QQMsg'>
03 <!--title-->
04 <div class="title">
05 <div class="titles">qq 消息 - 您的空间有新的评论</div>
06 <div class="close" id='closeQQMsg'>X</div>
07 </div>
08 <div style="clear: both"></div>
09 <!--内容-->
10 <div class="content">
11 <p>您的空间有人发表了有趣的评论，并向您发起了一个好友请求，单击查看一下吧！
12 </p>
13 </div>
14 </div>
15 <input type="button" id='showQQMsg' value='显示类似 QQ 消息窗口' />

```

本例 CSS 代码如下：

```

01 <style>
02 /*=====公共=====*/
03 h2,h5,#tooltipMsg,p{
04 white-space: nowrap;
05 }
06 td{
07 border: 1px solid #ccc;
08 height: 50px;
09 text-align: center;
10 font-size: 10pt;
11 padding: 2px;
12 }
13 /*=====类似 QQ 消息窗口提示=====*/
14 #QQMsg{
15 display: none;
16 position: absolute;
17 width: 242px;
18 height: 160px;
19 border: 1px #4F9BD0 solid;
20 }
21 #QQMsg div{

```

```

22 width: 242px;
23 }
24 #QQMsg .title{
25 background-color: #4F9BD0;
26 height: 22px;
27 line-height: 21px;
28 }
29 #QQMsg .title div{
30 background-color: #4F9BD0;
31 position: relative;
32 float: left;
33 }
34 #QQMsg .title div.titles{
35 margin: 0px;
36 font-size: 13px;
37 width: 200px;
38 }
39 #QQMsg .title div.close{
40 font-size: 12px;
41 color: #fff;
42 margin: 0px;
43 background-color: #58A0D3;
44 width: 4px;
45 left: 30px;
46 cursor: pointer;
47 }
48 #QQMsg .content p{
49 white-space: normal;
50 margin: 10px;
51 font-size: 13px;
52 color: #0C4E7C;
53 }
54 </style>

```

### 【代码说明】

- 类似 QQ 的消息提示窗口主要包括：窗口右下角动画弹出、弹框动画关闭、窗口随 scroll 事件的触发而移动。
- 调用 QQMsg() 弹出窗口，设置元素的样式 “display = "block"” ，并且修正 top 值为滚轴的高度与窗口的高度之和，导致弹出的窗口在可视区域之外，用户看不见，参见 JavaScript 代码第 30~38 行。然后再利用第 55~68 行的代码将窗口的 top 值递减，用户就可以看到一个缓缓上升的动画效果了。
- 当单击“关闭”按钮时，调用 click 事件，将 top 值修正为隐藏元素时的高度。

## 4.24 修改浏览器的主题

浏览器的主题用来告诉用户当前正在查看的页面是哪一个。这个主题默认是用页面 head 中的<title>标签来设置的，可以在程序中动态修改。本例效果如图 4-18 所示。

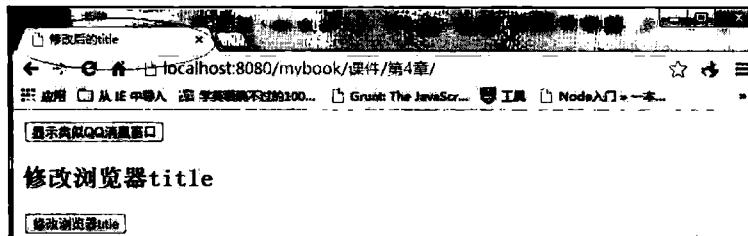


图 4-18 浏览器修改后的主题

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 //修改浏览器 title
04 document.getElementById("titleNewline").onclick = function(){
05 document.title = this.getAttribute("data-title"); //修改为 target title
06 }
07 };
08 </script>
```

本例 HTML 代码如下：

```
01 <h2>修改浏览器 title</h2>
02 <input id='titleNewline' data-title='修改后的 title' type="button" value='修改浏览器 title'/>
```

### 【代码说明】

JavaScript 代码第 04~06 行实现了浏览器主题的修订，主要是使用 JavaScript 中的 document.title 来修改，它是可读写属性。

## 4.25 打开链接时弹出确认框

当用户单击某未知链接时，提醒用户是否真的打开，帮助用户提高对未知网页风险地控制。本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 document.getElementById("linkConfirmation").onclick = function(){ //单击链接
04 if(window.confirm(this.getAttribute("data-msg"))){ //弹出确认框
05 window.open(this.getAttribute("data-target")); //如果确认再跳转
06 }
07 }
08 }
09 </script>
```

```

08 };
09 </script>

```

本例 HTML 代码如下：

```

01 <h2>打开链接时弹出确认框</h2>
02 <a id='linkConfirmation' data-msg='您真的要打开 "Qingjs" 连接吗?'
03 data-target='http://www.qingjs.com/' href='javascript:void(0)'>Qingjs

```

#### 【代码说明】

- 将 a 链接的 href 修改为 “javascript:void(0)”，这样可以阻止链接进程。
- JavaScript 代码第 05 行为 a 链接绑定一个指向目标地址的数据，并绑定一个弹出信息。
- 当单击 a 链接时触发 click 事件，调用 JavaScript 的 window.confirm() 弹出对话框，让用户确认是否跳转至目标链接，并显示绑定的数据信息。

## 4.26 删除时弹出确认对话框

用户在执行一些删除操作，有时是非理性的，如果在情急之下误删一些重要文件，会造成无法挽回的错误，岂不很痛心？如果在用户执行删除操作之前，善意的警告提醒，就可能避免这样的损失。在 JavaScript 中，window.confirm() 会弹出一个系统对话框，用户做任何业务都可以调用这个系统对话框来处理。本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 //删除时弹出确认对话框 *****
04 document.getElementById("deleteConfirmation").onclick = function(){ //删除单击
05 if(document.getElementById("deleteElement")){ //是否存在被删除的元素
06 if(window.confirm(this.getAttribute("data-msg"))){ //弹出删除确认框
07
08 document.body.removeChild(document.getElementById("deleteElement")); //删除执行
09 alert("删除成功！");
10 }
11 }else{
12 alert("元素已经删除！");
13 }
14
15 }
16 };
17 </script>

```

本例 HTML 代码如下：

```

01 <h2>删除时弹出确认对话框</h2>
02 <div id='deleteElement' style="height: 100px; background-color: #ccc">将要被删除的元
03 素……</div>
04 <input id='deleteConfirmation' data-msg='警告，您真的要删除上边的元素吗?' type="button"

```

```
05 value="删除" />
```

#### 【代码说明】

- JavaScript 代码第 04 行在单击“删除”按钮时，判断将要被删除的元素是否存在，如果不存在，则弹出“元素已经删除”。
- 如果存在，则调用 window.confirm() 弹出系统对话框，如果确认则删除元素，否则不执行删除。

# 第 5 章 页面控制常用代码

在 JavaScript 系统中，除了文档对象模型（DOM），还有浏览器对象模型（BOM）。BOM 可以操作浏览器的各个部件接口，所以页面的控制离不开 BOM。

本章主要涉及的知识点有：

- 页面、浏览器窗口、`iframe` 的打开、关闭、刷新等操作。
- IE 浏览器中一些特殊接口的操作，如 `window.external`、`document.execCommand`。
- 浏览器的一些部件局部效果，如添加收藏夹、修改地址栏图标等。
- 其他的一些常见效果，如禁止查看源代码、检测某个网站的链接速度、脚本永不出错等。

## 5.1 打开新页面

在日常的项目中，最常用的效果肯定是打开新的页面。例如：在当当网上陈列着很多书籍封面，单击封面时会打开新的页面显示书籍详情。在 HTML 代码中，打开新页面是通过`<a>`标签来实现的，那么如何使用 JavaScript 来实现此效果呢？

本例 JavaScript 代码如下：

```
01 <script type="text/JavaScript">
02 window.onload = function(){
03 var openNewPage = function(e){
04 /*
05 ①在原来的窗口中，直接跳转至目标页面
06 window.location.href="将要跳转的页面";
07
08 ②在新窗口中打开页面
09 window.open('将要跳转的页面');
10 window.history.back(-1);返回上一页
11
12 ③通过指定的 URL 替换当前的页面及缓存内容，导致跳转页面后没有后退功能
13 window.location.replace("将要跳转的页面")
14
15 ④通过文档头 META 跳转：CONTENT 后面的阿拉伯数字是代表几秒后转入
16 目标网页，URL 是目标地址
17 <META HTTP-EQUIV="Refresh" CONTENT="0;URL= "将要跳转的页面">
```

```

18 */
19 e.onclick = function(){
20 window.location.href = this.getAttribute('data-href');//第①种方式跳转
21 }
22 }
23 openNewPage(document.getElementById("openNewPage"))
24 };
25 </script>

```

本例 HTML 代码如下：

```

01 <h2>打开新页面</h2>
02 <input id='openNewPage' data-href='http://www.qingjs.com/' type="button" value="打开
03 QingJs 网站" />

```

#### 【代码说明】

- JavaScript 代码第 04~18 行介绍了 4 种常用的跳转页面的方法。本例第 20 行中的“`window.location.href`”是比较常见的跳转方法，其中“`window.location`”对象为 `window` 对象的子对象，用于获取当前页面的 URL 及一些其他相关参数信息，它也可以通过修改 URL 的形式，重定向到新的页面。
- “`window.location.replace`”是“`window.location`”对象的一个方法，用来替换当前页面及缓存的 URL，使用该方法重定向到新的 URL 后，无法后退到被替换之前的 URL。
- “`window.open`”用于打开新的窗口，或查找已命名的窗口，其第 1 个参数是可选参数，如果设置为 URL，就可以重定向到此 URL，如果省略参数或设置为空，则打开的新窗口不会显示任何文档。
- “`window.history.back`”为“`window.history`”的方法。“`window.history`”对象包含了浏览器的历史，可以通过此对象的一些接口方法，控制历史列表中的 URL。
- “`<META HTTP-EQUIV="Refresh" CONTENT="0;URL="将要跳转的页面">`”，不属于 JavaScript 的范畴，但是也要向读者介绍一下，其中 `CONTENT` 表示几秒后跳转，`URL` 是要跳转的目标页。

## 5.2 打开指定大小的窗口

在腾讯视频网站观看视频时，在设置里面可以选择小窗口播放，这个小窗口是按指定大小来显示的。此功能一般是通过“`window.open`”来实现的，这个方法的参数比较多，本例来演示一下，效果如图 5-1、图 5-2 所示。

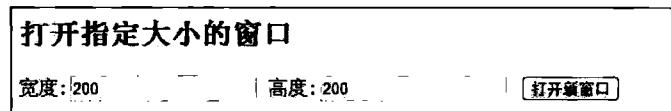


图 5-1 修改弹出窗口的宽、高

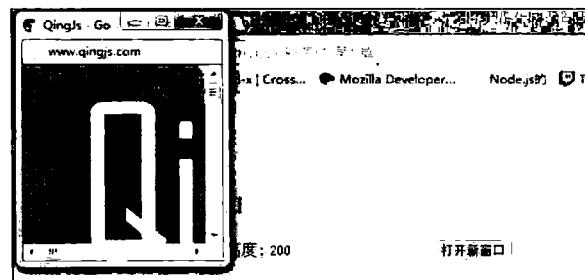


图 5-2 左上角是弹出的窗口（宽 200、高 200）

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 var openwindow = function(w, h, href){
04 w = w || "";
05 h = h || "";
06 /*window.open(
07 "可选，将要跳转的页面",
08 "可选，新窗口的名称",
09 "可选，设置新窗口的一些特征，默认为浏览器的标准特征，可选参数
10 ---channelmode、directories、fullscreen、height、width 等"
11 "可选，规定装载到窗口的 URL，是替换浏览器当前历史条目还是创建一个新
12 条目，true 为替换，false 为创建新条目"
13) */
14 window.open(href,'width='+ w+',height=' + h); //打开指定尺寸的新页面
15 }
16 document.getElementById("openwindow").onclick = function(){//跳转页面的响应按钮
17 openwindow(
18 document.getElementById("windowWidth").value,
19 document.getElementById("windowHeight").value,
20 this.getAttribute("data-href")
21);
22 }
23 };
24 </script>
```

本例 HTML 代码如下：

```

01 <h2>打开指定大小的窗口</h2>
02 宽度:<input id='windowWidth' />
03 高度:<input id='windowHeight' />
04 <input id='openwindow' type="button" data-href="http://www.qingjs.com/" value="打开新窗口"
05 />
```

### 【代码说明】

- JavaScript 代码第 14 行包含了“`window.open`”的参数，代码注释中已做详细说明。本例主要是应用了第 1 个参数和第 2 个参数。

- 在“window.open”的第2个参数中，通过修改width和height的大小，就可以控制新窗口的宽和高。

### 5.3 打开模式子窗口

除了标准的新窗口之外，还有一些特殊窗口，如“模式”子窗口，也可以称作“模态”子窗口，这类窗口一般是在子窗口需要与父窗口交互时用到。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //打开模式子窗口=====
04 document.getElementById("openChildWindow").onclick = function(){
05 /*
06 * showModalDialog 方法用来创建一个显示 HTML 内容的模态对话框
07 * object.showModalDialog(
08 *'必选参数，类型：字符串
09 *,'可选，'
10 *,'可选')
11 *3个参数分别为：对话框加载的 HTML 页面的 URL、传入对话框页面的参数，
12 *控制对话框展现效果的参数*/
13
14 window.showModalDialog("../index.html");
15 }
16 };
17 </script>
```

本例 HTML 代码如下：

```

01 <h2>打开模式子窗口</h2>
02 <input id='openChildWindow' type="button" data-href='http://www.qingjs.com/' value="打开模
03 式子窗口" />
```

#### 【代码说明】

- JavaScript 代码第14行的“window.showModalDialog”用来创建一个显示HTML的模式（模态）对话框，目前IE 4+、Firefox 13.0、Safari 5.1都支持。除此之外的浏览器中，有的也有此方法，但是实现此API的方法不一样，例如Chrome也有此方法，但不会有“模态”，而是打开了一个新窗体。Opera 12.0不支持该方法。
- 既然有“模式”，就有“非模式”。“非模式”的API是“window.showModelessDialog”，用法与“模式”方法类似，此处不再赘述。

### 5.4 获得子窗口的返回值

上一节介绍了打开“模式”子窗口的方法，那么在打开“模式”子窗口后，能不能获

取子窗口的返回值呢？如果能获取返回值，这个值可以帮助网站处理很多需求。例如用户在子窗口修改完信息后，会将子窗口的信息显示在父窗口中，方便用户查看修改之后的效果，如图 5-3、图 5-4 所示。

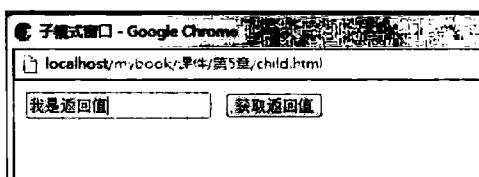


图 5-3 单击“获取返回值”按钮之前

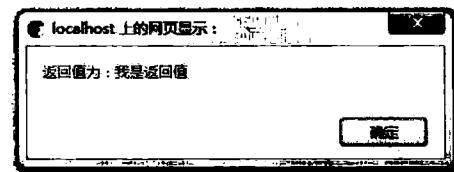


图 5-4 单击“获取返回值”按钮之后

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 document.getElementById("openChildWindowValue").onclick = function(){
04 var returnV = window.showModalDialog("../child.html"); //子窗口的引用
05 if(returnV){
06 alert("返回值为：" + returnV);
07 }
08 }
09 };
10 </script>
```

本例 HTML 代码如下：

```

01 <h2>获取子窗口的返回值</h2>
02 <input id='openChildWindowValue' type="button" value="打开模式子窗口" />
```

在第 5 章的源代码文件夹下增加 child.html 文件，代码如下：

```

01 <html>
02 <head>
03 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
04 <title>子模式窗口</title>
05 </head>
06 <body>
07 <input type="text" id='setReturnValue' value="填写返回值" name="btn">
08 <input type="button" value="获取返回值" name="btn"
09 onclick="getReturnV(document.getElementById('setReturnValue').value)">
10 <script language="JavaScript">
11 /*父窗体子模式窗口的简单模拟，为了简单、直观，将 JavaScript 脚本直接写在 html
12 页面中*/
13 function getReturnV(returnValue){
14 /*window.returnValue 用于返回 window.showModalDialog() 打开的模式
15 窗口的值*/
16 window.returnValue = returnValue;
17 window.close();
18 }
```

```

19 </script>
20 </body>
21 </html>

```

#### 【代码说明】

- JavaScript 代码第 04 行的“window.showModalDialog”可以打开子模式窗口，上节已介绍。此方法在打开子窗口时，会返回子模式窗口句柄，用来监听返回值。
- 在 child.html 文件中，代码第 16 行的“window.returnValue”用于返回模式窗口的值。当调用 getReturnValue() 函数时，就会将参数赋给“window.returnValue”，并关掉当前窗口，父窗体就会监听到返回的值，并且打印该值。

## 5.5 刷新 iframe 窗口

有一些 Web 应用比较大，可能是几个网页嵌套组合成的，当完成一个操作时，对应的 iframe 需要随时刷新，以保持信息同步。例如：在 A 页面与 B（iframe 窗口）页面都显示一样的用户信息，B 页面是载入时初始化信息，当通过 A 页面修改完用户的部分信息后，需要 B 页面也同步更新，那么只要刷新 B 页面重新载入就可以了。如何刷新 iframe 窗口呢？

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //刷新 iframe 窗口=====
04 document.getElementById("updateiframeBtn").onclick = function(){
05 /*获取 iframe --- 用 iframe 的 id 属性定位，用 location.href 可以解决跨域
06 的问题，网上流传用 location.reload()，这个会有域的限制*/
07 updateiframe.location.href = "../child.html";
08 }
09 };
10 </script>

```

本例 HTML 代码如下：

```

01 <h2>刷新 iframe 窗口</h2>
02 <input id='updateiframeBtn' type="button" value="刷新 iframe 窗口" />
03 <iframe src="../child.html" width=850 height=300 id="updateiframe"
04 name="updateiframe"></iframe>

```

#### 【代码说明】

- “location.reload”也可以让页面或 iframe 刷新，但是必须在一个“域”下才行，很多浏览器为了安全起见，禁止用此方法“跨域”控制。
- JavaScript 代码第 07 行的“location.href”可以解决因跨域问题而不能刷新的页面或 iframe 框架。

## 5.6 刷新当前页面

刷新当前页面与刷新 iframe 页面作用类似，在 JavaScript 中，有很多刷新当前页面的方法，本节介绍 3 种常用的方法。

本例 JavaScript 代码如下：

```
01 <script type="text/JavaScript">
02 window.onload = function(){
03 document.getElementById("updatePage").onclick = function(){
04 /* window 为当前的窗口对象 */
05 window.location.reload(); //方法 1 直接调用 reload()
06 /*方法 2 让页面跳转至原有的页面，也是刷新的一种实现方案
07 window.location.href = window.location.href;
08 方法 3 利用新的页面替换当前的页面，会替换历史中的痕迹
09 window.location.replace(window.location.href);
10 */
11 }
12 };
13 </script>
```

本例 HTML 代码如下：

```
01 <h2>刷新当前页面</h2>
02 <input id='updatePage' type="button" value="刷新"/>
```

### 【代码说明】

- JavaScript 代码第 05 行的方法 1 可以重新加载当前的文档。
- 方法 2 可以设置或返回完整的 URL，只要将 URL 设置为当前页面的 URL 即可实现刷新效果。
- 方法 3 可以用新的 URL 替换当前的 URL，但是会覆盖 History 对象的当前页。

## 5.7 不弹出提示框关闭父窗口

默认情况下关闭窗口都会出现一个提示框，询问用户是否真的关闭。如果用户进行地是恶意操作，我们可以不征询用户同意就直接关闭窗口。在 iframe 中，如果要关闭框架所在的父窗口，就不能使用直接调用 `window.close()` 方法了。要想在不弹出提示框的情况下关闭父窗口，子窗口必须获取父窗口的句柄。

本例 HTML 代码如下：

```
01 <h2>不弹出提示框关闭父窗口</h2>
02 <iframe src="../closeParent.html" width=850 height=300 name="updateIframe"></iframe>
```

在第 5 章的源代码文件夹下增加 `closeParent.html` 文件，代码如下：

```
01 <html>
02 <head>
```

```

03 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
04 <title>子模式窗口</title>
05 </head>
06 <body>
07 <input type="button" id='closeParentWindow' onclick="closeParentWindow()"
08 value="关闭父窗口" name="btn">
09 <script language="JavaScript">
10 /*父窗体子模式窗口的简单模拟，为了简单、直观，将 JavaScript 脚本直接写入 html
11 页面中*/
12 function closeParentWindow(returnValue){
13 //重新打开新的窗体，防止提示，或解决有的浏览器关闭不了的兼容性问题
14 window.parent.open("", "_self");
15 /*返回一个窗口对象。opener 属性与打开该窗口的父窗口相联系，当访问子
16 窗口中 opener 属性时，返回的是父窗口，通过该属性可以使用父窗口对象中的方法和属性。
17 设置 opener 属性为 null，父窗体就不会提示*/
18 window.parent.opener = null
19 window.parent.close();
20 }
21 </script>
22 </body>
23 </html>

```

#### 【代码说明】

- closeParent.html 文件第 12 行调用了 closeParentWindow()，用于响应“关闭父窗口”按钮的方法。
- 在 Chrome 及其他一些浏览器中，会发生关闭不了的问题，只要加入“window.parent.open("", "\_self")”，就会绕过此问题。
- “window.parent.close”用来获取父窗体句柄，并且执行关闭方法。

## 5.8 弹出窗口关闭时刷新父窗口

在一些特殊场景下，可能需要在关闭子窗口之后刷新父窗口。例如：子窗口中修改了一些用户的信息（年龄、性别等），父窗口就需要同步数据，实时改变。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //弹出窗口被关闭时，刷新父窗口---请查看 updateParentWindow.html 相关代码
04 document.getElementById("openChildWin").onclick = function(){
05
06 window.open("../updateParentWindow.html", 'openChildWin', 'width=500,height=500')
07
08 };

```

```
09 </script>
```

本例 HTML 代码如下：

```
01 <h2>弹出窗口关闭时刷新父窗口</h2>
```

```
02 <input id='openChildWin' type="button" value="打开子窗口"/>
```

在第 5 章的源代码文件夹下增加 updateParentWindow.html 文件，代码如下：

```
01 <html>
```

```
02 <head>
```

```
03 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

```
04 <title>子模式窗口</title>
```

```
05 </head>
```

```
06 <body>
```

```
07 <input type="button" id='closeParentWindow' onclick="closeChildWindow()"
```

```
08 value="关闭当前，刷新父窗口" name="btn">
```

```
09 <script language="JavaScript">
```

```
10 /*父窗体子模式窗口的简单模拟，为了简单、直观，将 JavaScript 脚本直接写入 html
```

```
11 页面中*/
```

```
12 function closeChildWindow(returnValue){
```

```
13 /*返回的是一个窗口对象。opener 属性与打开该窗口的父窗口相联系，当访问子
14 窗口中 opener 属性时，返回的是父窗口，通过该属性可以使用父窗口对象中的
15 方法和属性。*/
16 window.opener.location.reload();
17 window.close();
```

```
18 }
```

```
19 </script>updateParentWindow.htm
```

```
20 </body>
```

```
21 </html>
```

### 【代码说明】

- 在 updateParentWindow.html 文件中有一个 closeChildWindow()，用于响应“关闭当前，刷新父窗口”按钮的方法。
- 当单击“打开子窗口”按钮后，会弹出 updateParentWindow.html 页面。在此页面中，首先获取父窗口的句柄“window.opener”，然后调用 reload() 刷新父窗口页面，之后再调用子窗口的 close() 方法，关闭子窗口。

## 5.9 子窗口全屏

为了最大限度地显示屏幕内容，有时需要让打开的子窗口全屏显示，本例学习如何让子窗口全屏。

本例 JavaScript 代码如下：

```
01 <script type="text/JavaScript">
```

```
02 window.onload = function(){
```

```
03 document.getElementById("windowFullScreen").onclick = function(){
```

```

04 /*设置新打开的页面：
05 toolbar 不显示浏览器的工具栏、
06 location 不显示地址字段、
07 menubar 不显示菜单栏、
08 directories 不显示目录添加按钮、
09 scrollbars 窗体中内部超出窗口可视范围时不存在滚动条 */
10 var win =
11 window.open("../child.html","_blank","resizable=yes;status=yes;toolbar=no;
12 location=no;menubar=no;directories=no;scrollbars=no;");
13 win.moveTo(0,0); //设置新窗口的位置(0,0)
14 win.resizeTo(screen.availWidth,screen.availHeight); //设置新打开窗口的宽、高
15 }
16 };
17 </script>
```

本例 HTML 代码如下：

```

01 <h2>子窗口全屏</h2>
02 <input id='windowFullScreen' type="button" value="全屏"/>
```

#### 【代码说明】

- JavaScript 代码第 11~12 行用 “`window.open`” 打开子窗口，并设置一些基本参数。
- 因为要全屏，所以子窗口的坐标从 “`(0,0)`” 开始，如 JavaScript 代码第 13 行设置 “`win.moveTo(0,0);`”，然后让子窗口的宽度与高度等于浏览器的宽度与高度 “`win.resizeTo(screen.availWidth,screen.availHeight);`”。有的浏览器并不支持这两个方法，或者由于安全起见，这两个方法被禁用了。

## 5.10 屏蔽右键

在一些特殊 Web 应用场景中会屏蔽右键，例如：防止复制一些有版权保护的小说、禁止查看源代码、防止代码泄露等。屏蔽右键后右击网页的效果如图 5-5 所示。

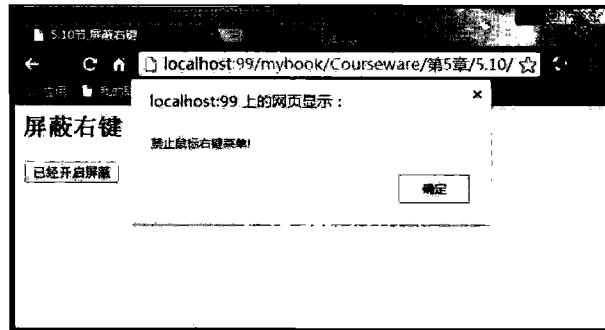


图 5-5 屏蔽右键后右击页面的效果

本例 JavaScript 代码如下：

```
01 <script type="text/JavaScript">
```

```

02 window.onload = function(){
03 //屏蔽右键
04 document.getElementById("shieldingRight").onclick = function(){
05 if(this.value == '已经开启屏蔽'){
06 return;
07 }
08 this.value = '已经开启屏蔽';
09 /*禁止右键菜单的事件,一般情况下, IE、FF、Chrome 都会支持, 其他
10 个别浏览器下会不支持*/
11 document.oncontextmenu=function(){
12 alert('禁止鼠标右键菜单!');
13 return false; //返回 false 则会禁止
14 }
15 }
16 };
17 </script>

```

本例 HTML 代码如下：

```

01 <h2>屏蔽右键</h2>
02 <input id='shieldingRight' type="button" value="开启屏蔽"/>

```

#### 【代码说明】

JavaScript 代码第 11~14 行的 oncontextmenu 事件为禁止右键单击，只要给指定的元素绑定此事件，并且返回 false 就可实现禁止功能。

## 5.11 网页防止另存为

网页不能被另存为也是一些网页为了保护页面内容的一些独特优势而采取的保护措施。例如：一些独家授权的论文类发布平台，为了防止用户通过另存网页的形式保存信息，就有这样的需求。截至本书出版时，JavaScript 还没有兼容性比较强的 API 禁止网页另存为，但是，可以采取一些曲线的方案来解决这一问题。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //网页将不能被另存为
04 document.getElementById("banOnPage").onclick = function(){
05 var _win=window.open("","");
06 _win.opener = null; //设置打开窗口的 opener = null
07
08 /*兼容性有待验证, IE 6~8, Chrome 版本也会支持
09 //向新的窗口填充 HTML: 加入 iframe 框架, 设置 src 为坏链接*/
10 _win.document.write('<html><head><meta http-equiv="Content-Type" '+
11 +'content="text/html; charset=utf-8"><title>禁止用“另存为”</title>' +

```

```

12 +'</head><body>禁止另存为的代码<noscript>禁止另存为'+
13 +'<iframe scr="*.htm"></iframe></noscript></body></html>');
14 }
15 };
16 </script>

```

本例 HTML 代码如下：

```

01 <h2>网页将不能被另存为</h2>
02 <input id='banOnPage' type="button" value="打开禁止另存网页"/>

```

#### 【代码说明】

- 本例在新的页面中，限制网页另存为。
- JavaScript 代码第 10~14 行的“\_win.document.write”是渲染新网页的 HTML。
- 在新的网页中添加“<noscript> 禁止另存为 <iframe scr="\*.htm"></iframe> </noscript>”，这是一个<iframe>的坏链接，在一些浏览器中，如果存在坏链接，网页就不会被另存，效果如图 5-6 所示。

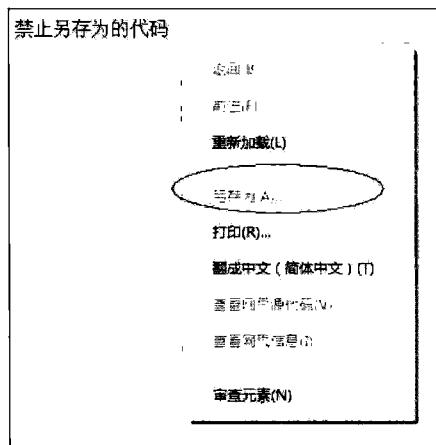


图 5-6 Chrome 下“另存为”命令是灰色状态

## 5.12 防止被人 frame

如果别人用一个程序就能把你的网站变成他自己的网站，你愿意吗？变成他自己的网站后，走你的流量，但用户不是你的。为了防止自己的劳动成果被别人窃取，只要防止被人 frame 就可以了。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //自动检测，然后跳出 --top 表示顶级的窗口，也就是最外层的窗口，self 指代当前窗口对象，属于 window 最上层的对象*
04 if (top.location != self.location){ //判断有没有 frame
05 top.location = self.location;

```

```

07 }
08 };
09 </script>

```

**【代码说明】**

- top 代表了顶级的窗口，也是最外层的窗口，self 代表当前的窗口对象。
- 第 05~07 行只要检测 top.location 与 self.location 是否相等，就可以知道是否在自己的域下，如果是在其他的域名下，则自动跳转至“self.location”。

## 5.13 永远都带着框架

与上一节相反的思路是，怎样永远带着框架呢？这样可以保证特定的服务一直保持在网页中。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //判断网页是否在框架中，如果不在框架中，自动套上框架
04 if (window.location.href == top.location.href) {
05 top.location.href = "../frames.html"; //frames.html 为框架网页
06 }
07 };
08 </script>

```

在第 5 章的源代码文件夹下增加 frames.html 文件，代码如下：

```

01 <!DOCTYPE html>
02 <html>
03 <head>
04 <title>我是框架页</title>
05 </head>
06 <body>
07 <iframe src="index.html"></iframe>
08 </body>
09 </html>

```

**【代码说明】**

- 上一例讲解了 top 与 self 的作用，本例把 self 换成 window。
- 同样的道理，在 JavaScript 代码第 04~06 行中，如果当前的 href 与最外层的 href 不一样，就跳转至有 iframe 的网页。

## 5.14 禁止滚动条

有时候为了防止用户误滚动导致内容偏离，会禁用滚动条。例如：在新闻类的 Web 应

用中，A 页面中有两个内容块 B 与 C，B 是新闻的列表，C 是单条新闻的详情页，内容超过 A、B、C 时都会显示滚动条，当鼠标移入 B 或 C 之后才允许 B 或 C 滚动条滚动。但是在浏览器中，当滚动条滚动到 B 或 C 的底部时，就会带动 A 页面的滚动，那么如何禁止 A 的滚动条滚动呢？CSS 可以实现，JavaScript 也可以实现。先简要介绍一下 CSS 的方法，代码如下：

```
01 body{ overflow-x:hidden; }
```

#### 【代码说明】

样式中的 `overflow` 是规定内容溢出时应该显示的效果，设为 `hidden`，就会隐藏溢出的内容，包括滚动条。这种实现方式的用户体验不太好，在 IE 中会出现页面抖动一下的感觉。

采用 JavaScript 也可以实现此功能，代码如下：

```
01 <script type="text/JavaScript">
02 window.onload = function(){
03 //禁止滚动条 =====
04 document.getElementById("banScrollBar").onclick = function(){
05 /*css 的方案，兼容性有问题
06 //document.body.style.overflow = "hidden";*/
07 document.onmousewheel = function(event){
08 event = event||window.event;
09 //检测浏览器是否存在，取消浏览器默认事件的接口 preventDefault
10 if (event && event.preventDefault){
11 event.preventDefault(); //取消默认事件
12 event.stopPropagation(); //阻止事件的传播
13 }else{
14 return false; //返回 false 阻止事件
15 }
16 }
17 };
18 };
19 </script>
```

本例 HTML 代码如下：

```
01 <h2>禁止滚动条</h2>
02 <input id="banScrollBar" type="button" value="禁止">
```

#### 【代码说明】

禁止滚动条的目的就是禁止滚动效果，滚动条可以通过鼠标滚动，也可以通过浏览器的单击拖动，截至本书出版之前，还没有禁止拖动的 JavaScript API，只有禁止鼠标滚轴的 API。在 JavaScript 中有滚轴事件 `onmousewheel`，JavaScript 代码第 7~16 行只要捕获这一个事件，将其绑定在 `document` 对象上，在滚动时，阻止事件即可。

## 5.15 禁止查看源代码

现在 HTML 5 游戏很流行，但是 HTML 5 游戏的很多源代码是在浏览器客户端运行的，

这样就容易暴露源码，为第三方复制游戏提供了先天条件，那么如何禁止查看源代码呢？真正地禁止查看源代码是不存在的，毕竟源代码就在客户端了，用户可以通过很多手段获取到，因此只能退而求其次，达到一定的项目目标就可以了。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 /*禁止查看源代码
04 本节讨论禁止源代码是在 JavaScript 的基础上执行的，绝对地禁止不存在，因为可
05 以采用很多其他的特殊手段获取源代码*/
06 banViewSource = function(e){
07 /*以打开新窗口的形式禁止查看源代码，关闭浏览器的工具栏等，但是一些浏览器
08 的调试工具无法屏蔽*/
09 e.onclick = function(){
10
11 window.open("../banViewSource.html","", "menubar=no,location=no,scrollbars=yes,
12 resizable=yes")
13 }
14 }
15 banViewSource(document.getElementById("banViewSource"));
16 };
17 </script>
```

本例 HTML 代码如下：

```

01 <h2>禁止查看源代码</h2>
02 <input id='banViewSource' type="button" value="禁止" />
```

在第 5 章的源代码文件夹下增加 banViewSource.html 文件，代码如下：

```

01 <html>
02 <head>
03 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
04 <title>禁止查看源代码</title>
05 </head>
06 <body>
07 <h2>禁止查看源代码</h2>
08 <script language="JavaScript">
09 document.oncontextmenu = function(){
10 window.event.returnValue=false;
11 };
12 </script>
13 </body>
14 </html>
```

### 【代码说明】

- 本例的禁止查看源代码，只要单击“禁止”按钮，即可在子窗口中试验。
- 以窗口的形式打开，可以很好地关闭浏览器的工具栏等部件。
- 在 banViewSource.html 文件中，第 10 行为 document 对象绑定禁用右键菜单事件“window.event.returnValue=false”。

## 5.16 取消选取、防止复制

很多小说网站的网页都是不能复制的，本例使用 JavaScript 实现此功能，代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 document.getElementById("deselect").onclick = function(){ //取消选区
04 /*
05 1.document.selection 不是 W3C 的标准，只是一些主流浏览器的 API，只有 IE
06 支持，它代表选中区块，之后可以调用这个 API 的一些操作方法；
07 2.window.getSelection()返回一个由用户选定的表示文本范围的对象，并可以
08 调用此对象的一些方法进行操作
09 */
10 (window.getSelection && window.getSelection().removeAllRanges())
11 || (document.selection && document.selection.empty &&
12 document.selection.empty());
13 }
14
15 var preventCopying = document.getElementById("preventCopying"); //防止复制
16 /**第一种方法*/
17
18 preventCopying.oncopy = function(){ //禁止复制
19 return false;
20 }
21
22 preventCopying.oncut = function(){ //禁止剪切
23 return false;
24 }
25 /**第二种方法*/
26
27 preventCopying.onselectstart = function(){ //禁止选取
28 return false;
29 }
30 };
31 </script>
```

本例 HTML 代码如下：

```

01 <h2>取消选取</h2>
02 <input id='deselect' type="button" value="取消" />
03 <h2>防止复制</h2>
04 <input id='preventCopying' type="text" value="防止复制" />
```

### 【代码说明】

- JavaScript 代码第 10~12 行的 “document.selection” 不是 W3C 的标准，代表了选中的区域对象，并提供了一些方法对选中内容进行控制，本例中的 “document.selection.empty()” 可以清除选中区域。

- “`window.getSelection`”是用户选中的文本范围对象，只有 IE 支持，其中`“window.getSelection().removeAllRanges()”`可以清除选中内容。
- 第一种方法是 JavaScript 代码第 18~20 行，它通过让复制事件 `oncopy` 与剪切事件 `oncut` 返回 `false` 来防止复制，因为目标是防止用户抄袭文本，所以把剪切的功能也禁止了。
- 第二种方法 JavaScript 代码第 27~29 行，它通过禁止选取来实现禁止复制功能，复制操作执行的第一步就是选中文本，如果不让选中文本也同样实现了禁止复制的效果。

## 5.17 添加到收藏夹

为了让用户下一次更快捷地访问页面，可以在页面中增加一个“添加到收藏夹”的功能。例如我们每天都要访问一下新浪微博，但又不想通过输入网址或搜索引擎的方式打开新浪微博，那么何不将新浪微博添加到收藏夹中，这样以后直接单击链接就可以进入微博了。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //添加到收藏夹=====
04 function addFavorite(fURL, fTitle){
05 try { //IE 支持的 API
06 window.external.AddFavorite(fURL, fTitle);
07 } catch(e) { //FF 支持的 API
08 try{
09 window.sidebar.addPanel(fTitle, fURL, "");
10 }catch(error){
11 //如果不支持以上两种方案，采用提示性收藏
12 alert("加入收藏失败，请用 Ctrl+D 或手动设置！");
13 }
14 }
15 }
16 document.getElementById("addFavorite").onclick = function(){
17 addFavorite("http://qingjs.com", "QingJs 框架");
18 };
19 };
20 </script>
```

本例 HTML 代码如下：

```

01 <h2>添加到收藏夹</h2>
02 <input id='addFavorite' type="button" value="添加到收藏夹" />
```

### 【代码说明】

- 一般的主流浏览器都有收藏夹的 API，只需调用即可。针对没有 API 的浏览器，也可以采用提示的形式告诉用户如何实现收藏（如 JavaScript 代码第 12 行所示）。
- 在 JavaScript 代码第 04~15 行中，IE 支持 “`window.external.AddFavorite(fURL,`

fTitle)" , 第1个参数是收藏的URL, 第2个参数是收藏的标题, 与IE收藏API类似的是“window.sidebar.addPanel(fTitle, fURL, "")” , 用法基本一样。

## 5.18 将网页设置为首页

假如经常使用百度的搜索引擎服务, 可以将百度设置为浏览器的首页, 那样打开浏览器时就会直接显示百度页面。

本例JavaScript代码如下:

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //网页设置为首页=====
04 function setHomePage(val){
05 try{
06 //设置或检索对象的 DHTML 行为
07 document.body.style.behavior='url(#default#homepage)';
08 document.body.setHomePage(val);
09 }
10 catch(e){
11 if(window.netscape) {
12 try {
13 /*netscape.security.PrivilegeManager.enablePrivileged , 权限设置,
14 有的浏览器需要用户配置浏览器安全属性才能执行*/
15
16 netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect");
17 }
18 catch (e) {
19 alert("此操作被浏览器拒绝, 请手动设置!");
20 }
21 //Components.classes 是一个被 ContractID 类索引的只读对象属性
22 var prefs =
23 Components.classes["@mozilla.org/preferences-service;1"].getService
24 (Components.interfaces.nsIPrefBranch);
25 prefs.setCharPref('browser.startup.homepage',val); //浏览器的偏好设置
26 }else{
27 alert("设置首页失败, 请手动设置!");
28 }
29 }
30 }
31
32 document.getElementById("setHomePage").onclick = function(){ //网页设置为首页
33 setHomePage("http://qingjs.com");
34 };

```

```
35 };
36 </script>
```

本例 html 代码如下：

```
01 <h2>网页设置为首页</h2>
02 <input id='setHomePage' type="button" value="网页设置为首页" />
```

#### 【代码说明】

- 很多浏览器由于各种原因，并没有为 JavaScript 提供设为首页的 API，有的即使提供了 API，也是方法不一。
- 在本例中，第 32~34 行首先检测浏览器是否直接支持“`document.body.setHomePage`”方法，如果支持就采用此方法设置，否则就验证是否为“`window.netscape`”的浏览器。如果不是则提示“设置首页失败，请手动设置！”，如果是就检测 `UniversalXPConnect` 权限“`netscape.security.PrivilegeManager.enablePrivilege("UniversalXPConnect")`”。如果无权限则提醒用户“此操作被浏览器拒绝，请手动设置！”，然后执行浏览器的个性设置，即设网页为首页“`prefs.setCharPref("browser.startup.homepage",val);`”。

## 5.19 将网页另存为

作为 JavaScript 程序员，是不是经常会寻找一些很酷的 JavaScript 效果？如果可以直接将找到的效果保存下来，是不是很赞？将网页保存到本地，可以辅助用户查看离线网页，当然也包括效果。

本例 JavaScript 代码如下：

```
01 <script type="text/JavaScript">
02 window.onload = function(){
03 isIE = function(){ //是否是 IE
04 return document.all ? true : false;
05 }
06 //将网页另存为
07 function webSave(page){
08 //document.execCommand(指令参数[,交互方式, 动态参数])
09 var save = document.execCommand('saveas','true', page);
10 if(!save && !this.isIE()){
11 alert("设置网页另存失败，请用 Ctrl + S 或手动设置！");
12 }
13 }
14 //将网页另存为
15 document.getElementById("webSave").onclick = function(){
16 webSave('../index.html');
17 }
18 };
```

```
19 </script>
```

本例 html 代码如下：

```
01 <h2>将网页另存为</h2>
```

```
02 <input id='webSave' type="button" value="将网页另存为" />
```

#### 【代码说明】

- JavaScript 代码第 09 行的 “document.execCommand” 方法提供了一些对浏览器内置命令调用的接口，暂时不属于任何规范的范围，由各个浏览器自行实现。值得一提地是，HTML 5 对其已经有了描述。
- “document.execCommand” 提供了很多命令集，包括剪切、复制等，本例只是用到网页另存的参数指令，在不支持此方法的浏览器中，同样是以提示的方式告诉用户来手动另存网页。

## 5.20 自定义 IE 地址栏图标

如果将 IE 地址栏中的图标，换成自己的图标，可以很好地为网站营造形象，增加网站的 UI 识别。实现这一目标并不难，只要修改 HTML 中的 link 标签就可以。本例效果如图 5-7 所示。



图 5-7 IE 浏览器地址栏的图标

本例 HTML 代码如下：

```
01 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
02 <head>
03 <title>IE 地址栏前换成自己的图标</title>
04 <!--IE 地址栏前换成自己的图标-->
05 <link rel="Shortcut Icon" href="../images/favicon.ico">
06 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
07 </head>
08 <body>
09
10 </body>
11 </html>
```

#### 【代码说明】

- 首先需要制作一个 ico 图标，ico 图标一般为网站的 logo，网络上有很多在线生成图标的工具或软件。
- 将第 05 行 “<link rel="Shortcut Icon" href="images/favicon.ico">” 中的 href 内容改成自己的 ico 图标即可，其中 rel 属性代表当前文档与被链接文档之间的关系。有时会因为网页缓存的关系，不能及时更新显示效果。

## 5.21 在收藏夹中显示自定义图标

收藏夹中的图标，同样也可以帮助网站在用户心目中营造一种专业、美观、统一的形象。这个功能实现起来相对比较简单，直接修改 link 属性即可。本例效果如图 5-8 所示。

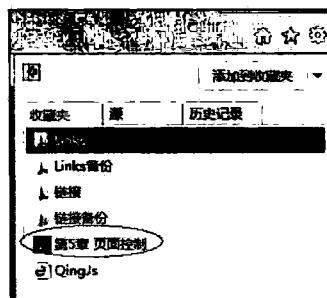


图 5-8 IE 浏览器收藏夹中的图标

本例 HTML 代码如下：

```
01 <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
02 <head>
03 <title>在收藏夹中显示出你的图标</title>
04 <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
05 <!--可以在收藏夹中显示出你的图标 -->
06 <link rel="Bookmark" href="../images/favicon.ico">
07 </head>
08 <body>
09
10 </body>
11 </html>
```

### 【代码说明】

- 第 06 行在 head 标签中添加 “`<link rel="Bookmark" href="images/favicon.ico">`”，即可在收藏夹中显示出自定义图标。
- 与上一个例子不同的是“`rel="Bookmark"`”，它表示“书签的关系”。

## 5.22 查看网页源代码

一些编程教学类的网站，会将 JavaScript 效果的源代码展示给学习者。例如：源代码爱好者网站会在 JavaScript 效果说明下面显示源代码，方便学习者查看。本例就来实现查看网页源代码的功能。

本例 JavaScript 代码如下：

```
01 <script type="text/JavaScript">
02 window.onload = function(){
03 //查看网页源代码
04 }
```

```

04 document.getElementById("viewDocumentSource").onclick = function(){
05 var source =
06 window.open("index.html","", "menubar=no,location=no,scrollbars=yes,resizable=yes");
07 //通过判断 window.XMLHttpRequest 对象是否存在来调用不同的创建方式
08 if (window.XMLHttpRequest) {
09 //FireFox、Opera 等浏览器支持的创建方式
10 xmlhttp = new XMLHttpRequest();
11 } else {
12 //IE 浏览器支持的创建方式
13 xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
14 }
15 xmlhttp.onreadystatechange = function(){
16 if (xmlhttp.readyState == 4) { //请求数据成功
17 source.document.write("<textarea cols='1000' rows='1000>" +
18 xmlhttp.responseText + "</textarea>"); //将获取到的数据放入 textarea 标签中
19 }
20 };
21 xmlhttp.open("GET", "index.html", true); //用 GET 形式请求数据
22 xmlhttp.send(null);
23 }
24 };
25 };
26 </script>

```

本例 HTML 代码如下：

```

01 <h2>查看网页源代码</h2>
02 <input id='viewDocumentSource' type="button" value="查看网页源代码" />

```

#### 【代码说明】

- 查看网页源代码，第 1 步就是要获取网页的 HTML。本例应用了异步请求 Ajax，它会针对不同的浏览器创建 xmlhttp，以 GET 的形式获取 index.html 页面。
- JavaScript 代码第 16~19 行在 “xmlHttp.readyState == 4” 时表示请求数据成功，将获取到的 responseText 数据放入 textarea 标签中，此时就能看见网页源代码了，如图 5-9 所示。

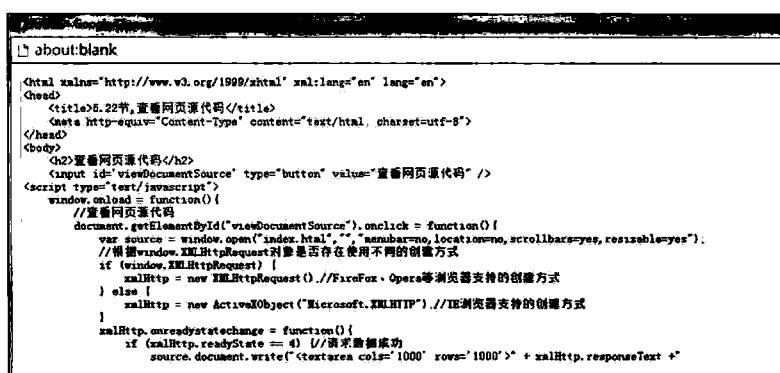


图 5-9 Chrome 浏览器显示的网页源代码

## 5.23 判断上一页的来源

获取上一页的来源，在数据监测中比较实用，可以让网站的运营人员知道网站的用户是通过什么途径进入网站的。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //判断上一页的来源
04 function getPreviousPage(){
05 /*
06 *一般情况下一些浏览器会将 document.referrer 来源处理为空字符，假设场景
07 为：由 B 页面，跳转至 A 页面
08 * 1.在地址栏中直接输入 A 地址
09 * 2.在 B 页面右击 link A，在新窗口中打开
10 * 3.在 B 页面右击 link A，在新标签页中打开
11 * 4.鼠标拖动 link A 至地址栏、标签栏
12 * 5.修改 window.location 打开 A 页面（同域）
13 * 6.利用 window.open 打开 A 页面
14 * 7.通过单击 flash 打开 A 页面
15 * 8.后台（服务器）重定向至 A 页面
16 */
17 return document.referrer;
18 }
19 //判断上一页的来源
20 document.getElementById("getPreviousPage").onclick = function(){
21 document.getElementById("getPreviousPageView").innerHTML = "上一页来源：
22 " + getPreviousPage();
23 }
24 };
25 </script>
```

本例 HTML 代码如下：

```

01 <h2>判断上一页的来源</h2>
02 <p id='getPreviousPageView'>上一页来源：</p>
03 <input id='getPreviousPage' type="button" value="获取上一页来源" />
```

### 【代码说明】

- JavaScript 代码第 17 行中返回的是 `document.referrer`，关于它的许多应用场景及注意事项在代码注释中已详细阐述，请读者注意查看。
- 值得一提地是，个别浏览器中不存在 `document.referrer` 这一属性，例如早期版本的 IE 浏览器。

## 5.24 最小化、最大化、关闭窗口

我们在日常生活中经常会改变浏览器窗口的状态，如窗口的最大化、最小化、关闭等。本例通过 JavaScript 来实现这 3 个常见操作，代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 function updateWin(str){ //最小化、最大化、关闭窗口
04 var _w = 0,
05 _h = 0;
06 //IE 支持，其他浏览器只支持新打开窗口的大小设置
07 if(str == "max"){
08 _w=screen.availWidth;
09 _h=screen.availHeight;
10 }
11
12 if(str == "min"){
13 //方法 1 updateWinMinIE.Click();
14
15 /*方法 2 各种浏览器对 window.resizeTo() 与 window.moveTo() 支
16 持程度不一样导致结果不一样，
17 出于用户体验及安全的考虑，Chrome、Firefox 不支持此方法，IE 支持*/
18
19 _w = 1;
20 _h = 1;
21 }
22
23 if(str == "close"){ //关闭窗口
24 /* 只对 IE 与 Chrome 起作用，Firefox 需要特殊配置
25 1. 在 Firefox 地址栏里输入 about:config
26 2. 在配置列表中找到 dom.allow_scripts_to_close_windows
27 3. 鼠标单击右键，弹出 UI 面板，选择切换按钮，把上面的 false 修改为 true
28 即可，默认为 false，是为了防止使用脚本乱关窗口
29 Firefox 中做如此设置以后，直接使用 window.close() 即可关闭窗口。
30 */
31 window.open("", '_self','');
32 window.close();
33 }
34 window.moveTo(0,0);
35 window.resizeTo(_w,_h);
36
37 }
38
39 document.getElementById("updateWinMin").onclick = function(){ //最小化

```

```

40 updateWin("min");
41 }
42 document.getElementById("updateWinMax").onclick = function(){ //最大化
43 updateWin("max");
44 }
45 document.getElementById("updateWinClose").onclick = function(){ //关闭窗口
46 updateWin("close");
47 }
48 }
49 </script>

```

本例 html 代码如下：

```

01 <h2>最小化、最大化、关闭窗口</h2>
02 <!--IE 最小化方法 1-->
03 <!--object id=updateWinMinIE classid="clsid:ADB880A6-D8FF-11CF-9377-00AA003B7A11">
04 <param name="Command" value="Minimize"></object-->
05 <input id='updateWinMin' type="button" value="最小化" />
06 <input id='updateWinMax' type="button" value="最大化" />
07 <input id='updateWinClose' type="button" value="关闭窗口" />

```

#### 【代码说明】

- JavaScript 代码第 03 行的 updateWin() 函数，通过判断不同字符串执行“最小”、“最大”、“关闭”窗口的任务。
- 当参数值为 min 时，将宽、高设置为 1，通过“window.resizeTo(\_w,\_h)”改变浏览器窗口的大小。
- 当参数值为 max 时，将宽高设置为显示器的宽、高，通过“window.resizeTo(\_w,\_h)”改变浏览器窗口的大小。
- 当参数值为 close 时，调用“window.open('\_self');”与“window.close();”即可关闭浏览器窗口。

**注意：**在 FireFox 及某些其他浏览器上会阻止 window.close() 行为，目的是防止脚本乱关窗口，因此需要用户手动设置，设置的详细步骤在代码注释中已介绍。

## 5.25 禁止浏览器缓存

一般的浏览器默认会缓存网页，导致一些网页的内容不能实时更新，网站的一些新改动用户就无法体验。例如：当我们浏览一些即时新闻消息时，需要手动清空缓存才能看到最新内容，那是不是太不智能了？

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //网页不会被缓存
04 function noCache (){

```

```

05 /*
06 html 标准 meta 的写法
07
08 <!--第一种写法-->
09 <META HTTP-EQUIV="pragma" CONTENT="no-cache"> 用于设定禁止浏览
10 器从本地机的缓存中调阅页面内容，设定后一旦离开网页就无法从 Cache 中再调出
11 <META HTTP-EQUIV="Cache-Control" CONTENT="no-cache,
12 must-revalidate"> 用于控制 HTTP 缓存(在 HTTP/1.0 中可能部分没实现，仅仅实现了 Pragma:
13 no-cache)
14 <META HTTP-EQUIV="expires" CONTENT="Wed, 26 Feb 1997 08:21:57
15 GMT"> 表示存在时间，允许客户端在这个时间之前不去检查(发请求)
16 <!--第二种写法-->
17 <META HTTP-EQUIV="expires" CONTENT="0">/*
18
19 //JavaScript 方法随机数 一般利用后台可以处理
20 window.location = window.location.href+"#"+new Date().getTime()
21 }
22
23 document.getElementById("noCache").onclick = function(){ //网页不会被缓存
24 noCache();
25 }
26 };
27 </script>

```

本例 HTML 代码如下：

```

01 <h2>网页不会被缓存</h2>
02 <input id='noCache' type="button" value="网页不会被缓存">

```

#### 【代码说明】

- 第一种解决方案，可以通过“`<META HTTP-EQUIV="pragma" CONTENT="no-cache">`”这种头部声明来禁止缓存。META 提供了页面中的一些元信息，可以很好地辅助浏览器解析网页。
- 第二种解决方案，JavaScript 代码第 20 行通过不断地更新网页 URL 中的尾部参数，达到重新加载页面的目的，因为只要 URL 变了，就会重新加载网页数据。

## 5.26 检测某个网站的链接速度

在一些 Web 性能测试中，经常被提起的性能指数就是某个网站的链接速度有多快，因为网站的链接速度直接影响到用户会不会使用这个网站！那么通过 JavaScript 怎么去实现测试呢？

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){

```

```
03 //检测某个网站的链接速度
04 var
05 linkSpeedTime = 0,
06 linkSpeedInterl = -1,
07 linkSpeedE = null,
08 linkSpeedURL = "",
09
10 getLinkSpeed = function() { //计算、显示测试结果
11 linkSpeedE.value = (linkSpeedTime / 10) + "秒";
12 window.clearInterval(linkSpeedInterl); //关闭线程
13 },
14 linkSpeed = function(e, t){
15 linkSpeedTime = 1;
16 window.clearInterval(linkSpeedInterl); //关闭线程
17 linkSpeedInterl = setInterval(function(){ //开启线程
18 linkSpeedTime++;
19 console.log(linkSpeedTime);
20 },100)
21 linkSpeedURL = e.value;
22 linkSpeedE = t;
23 //创建一个 Image 对象，创建一个 img 请求连接，实现图片的预下载
24 var img = new Image();
25 img.src = linkSpeedURL +"/"+Math.random();
26 //检测测试结果，图片下载完毕时异步调用 callback 函数
27 img.onerror = function (call) {
28 window.event
29 console.log(call)
30 if(linkSpeedURL){
31 getLinkSpeed();
32 }
33 };
34 };
35 //检测某个网站的链接速度
36 document.getElementById("linkSpeed").onclick = function(){
37 linkSpeed(document.getElementById("linkSpeedWeb"),
38 document.getElementById("linkSpeedTime"));
39 }
40 };
41 </script>
```

本例 HTML 代码如下：

```
01 <h2>检测某个网站的链接速度</h2>
02 时间：<input id='linkSpeedTime' type="text" value="0">

03 网站：<input id='linkSpeedWeb' type="text" value="http://www.qingjs.com">
04 <input id='linkSpeed' type="button" value="检测">
```

**【代码说明】**

- 第1步，开启“定时调用”，JavaScript代码第18行不断地递增“linkSpeedTime++”。
- 第2步，创建一个img对象，设置其src等于待测试网站地址与随机数组组合的地址，也就是一个不存在的地址。
- 第3步，当发现请求的地址为坏链时，会调用onerror回调处理，结束第1步的定时调用，并计算出结果“(linkSpeedTime / 10) + “秒””。测试QingJs网站的速度效果如图5-10所示。



图5-10 QingJs网站链接速度

## 5.27 脚本永不出错

在网站中，经常有一些莫名其妙的JavaScript错误干扰用户查看网页。例如：当用户使用IE浏览器访问一个网页时，有时候会弹出未知错误，这样会让用户觉得这个网站很烦，总是干扰他访问网站，下次用户或许就直接和网站说拜拜了。那么如何解决这个现实而又严峻的问题呢？

本例JavaScript代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 /*在打开页面时，普通用户并不希望看到弹出脚本错误的对话框，为避免这种情况
04 可以在网页中添加捕获错误的代码*/
05
06 /*脚本发生错误时触发的对象---参数：m为错误信息、f错误的文件、l错误的行号
07 */
08 window.onerror = function(m, f, l){
09 return true;
10 };
11 };
12 </script>
```

**【代码说明】**

- 在window对象中有一个onerror事件，可以捕获脚本的错误，只要在页面加载时运行第08~10行代码即可阻止脚本出错。
- 从另一个角度说，onerror事件会获取一些调试用的信息。把第08行onerror中3个参数(m、f、l)返回的错误信息、错误文件、错误行号，记录在错误日志中，对项目调试会有很大的裨益。

## 5.28 解决点击空链接返回页面顶部的问题

有时候一些空链接被点击时会让网页返回到顶部。例如：访问瀑布流式的网页过程中，当网站浏览者浏览到网页的底部时想继续向下浏览，但是点击了一个带“#”号的空连接，导致网页直接返回到顶部，会让访问者觉得莫名其妙，页面怎么返回顶部了呢？访问者的浏览欲望可能就一次性结束了。导致这些问题的原因，一般是加了一些特殊的链接，如例子中的“#”等。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //一般点击空连接，页面重置到首页是因为在链接中加入了锚点“#”或空字符
04 var as = document.getElementsByTagName("a"),
05 i = 0,
06 l = as.length,
07 h = "";
08 while(i < l){ //遍历所有 a 链接
09 h = as[i].getAttribute("href");
10 if(h == "#" || !h) as[i].href = "JavaScript:void(0)"; //修改 a 链接
11 i++;
12 }
13 };
14 </script>
```

本例 HTML 代码如下：

```

01 <h2>防止点击空链接，页面重置到页的首端</h2>
02 我是空链接

03 我是“#”链接
```

### 【代码说明】

- 本例只对“#”与空链接进行了过滤，只是为了向读者展示一下基本原理。
- “#”与空字符默认会访问本页，但不会重新加载文档，所以跳转至顶部。
- JavaScript 代码第 10 行将空链接修改为“JavaScript:void(0)”，其中 void 是一个操作符，该操作符指定要计算一个表达式，但不是返回值，在页面表现上不会发生任何效果，也就消除了 a 链接的跳转。

## 5.29 获取浏览器信息

获取浏览器的一些信息，对于分析用户的喜好、统计用户的客户端信息来说非常有用。如果知道用户是在移动设备还是在 PC 端打开浏览器，就可以知道网站用户的设备群体分布比例了。本例效果如图 5-11 所示。

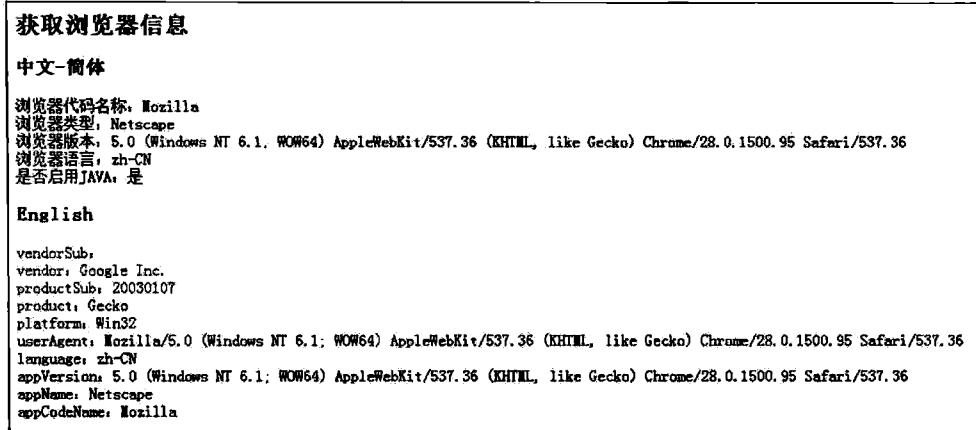


图 5-11 Chrome 的基本信息

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //来自 navigator 对象的信息不一定准确，浏览器开发商可以随意更改，没有统一性
04 var browserInfo = "";
05 nav = navigator;
06
07 browserInfo += "<h3>中文-简体</h3>"; //展示浏览器的一些基本信息
08 /*appCodeName 属性是一个只读字符串，代表浏览器的代码名，在所有以 Netscape
09 代码为基础的浏览器中，值为 Mozilla*/
10 browserInfo += "浏览器代码名称：" + nav.appCodeName + "
";
11 browserInfo += "浏览器类型：" + nav.appName + "
";
12 browserInfo += "浏览器版本：" + nav.appVersion + "
";
13 browserInfo += "浏览器语言：" + nav.language + "
";
14 browserInfo += "是否启用 JAVA：" + (nav.javaEnabled() ? "是" : "否") + "
";
15
16 browserInfo += "<h3>English</h3>";
17 //遍历所有的 navigator 浏览器对象数据，如果提供基本的字符数据，就打印出来
18 for(var n in nav){
19 if(typeof nav[n] == "string") browserInfo += n + ": " + nav[n] + "
";
20 }
21 document.getElementById("getBrowserInformation").innerHTML = browserInfo;
22 };
23 </script>
```

本例 HTML 代码如下：

```

01 <h2>获取浏览器信息</h2>
02 <div id='getBrowserInformation'></div>
```

### 【代码说明】

- 浏览器针对使用 JavaScript 获取浏览器信息的功能开放了一个 API，即 `navigator` 对象，它包含浏览器的一些信息。需要注意的是，截至本书出版之前，还没有公布的

标准，所以各个浏览器提供的信息也没有统一规范。

- 本例打印了浏览器的一些基本信息。这些基本信息大多都是通过访问 `navigator` 的属性得来。另外，验证是否支持 Java，使用 `javaEnabled()` 方法，验证浏览器是否启用了“数据污染点”，即 `data tainting`，使用 `taintEnabled()` 方法。

### 5.30 解决 URL 传递中文参数出现乱码

在 URL 传递中文参数时，有的浏览器会出现乱码现象，这个现象是正常的。因为浏览器对中文参数进行了编码，但是不同的浏览器处理编码的方式不同。例如：Chrome、IE 大多是 UTF-8 编码，FireFox 可能是 ISO-8859-1 编码。这么多编码种类，如何解决中文乱码问题呢？

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //URL 传参中文出现乱码
04 document.getElementById("getURL").onclick = function(){
05 /*为了解决中文字符传递乱码的问题，一般都将传递的参数利用
06 encodeURIComponent 进行 utf-8 格式的 url 编码，服务端可以再解码，这样
07 就解决中文乱码问题了*/
08 var cencodeStr = encodeURIComponent("我是 Qingjs");
09 alert("调用 encodeURIComponent 对 '我是 Qingjs' 编码：" + cencodeStr +"\n"
10 /*decodeURIComponent 返回统一资源标识符（URI）的一个已编码
11 组件的非编码形式*/
12 + "调用 decodeURIComponent 对解码后：" +
13 decodeURIComponent(cencodeStr));
14 }
15 };
16 </script>
```

本例 HTML 代码如下：

```

01 <h2>URL 传参中文出现乱码</h2>
02 <input type="button" id='getURL' value="显示例子"/>
```

#### 【代码说明】

- 各个浏览器的编码处理方式不同，所以统一使用 JavaScript 来处理编码。
- JavaScript 代码第 08~13 行可以使用 `encodeURIComponent` 编码，然后在服务端用 `decodeURIComponent` 解码，也可以用 `encodeURIComponent` 对字符串进行 URI 组件编码，对应的解码函数为 `decodeURIComponent`。以上两种编码，会将字符串中的某些字符替换为 16 进制的转义序列，不会对 ASCII 字母和数字进行编码，也不会对这些 ASCII 标点符号（如-\_.!~\*'()）进行编码。
- 针对以上两种处理编码的方式，后台语言也都有相对应的解码函数，如 PHP 可以使

用 iconv() 函数解码用 encodeURIComponent 编码的字符。其他的后台语言 Ruby、Java 等都有自己的解码函数，不再一一说明，读者只要参考一下相关的手册即可。

### 5.31 获取地址栏 URL 的参数

在 JavaScript 中经常要获取 URL 的参数来执行不同的业务处理。例如：在一些 Web 服务中，通过 URL 中的“? userID=1”（userID 用户的唯一识别码编号）来显示不同的网页，微博是通过“/u/用户 ID 编号”的组合形式来访问不同的微博个人空间的。JavaScript 提供的“window.location.href”可以获取地址栏 URL 的全部信息。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03
04 function getURLArgs(e){ //获取地址栏 URL 的参数
05 var args = "空", //试着在浏览器中添加参数
06 _args = [];
07 u = window.location.href,
08 s = u.indexOf("?");
09 i = 0,
10 l = 0,
11 o = null;
12
13 if(s != -1){ //如果不等于-1 表示存在参数
14 args = u.substr(s+1).split("&");
15 }
16 e.innerHTML = "参数以逗号分隔：" + args;
17 l = args.length;
18 for(; i < l; i++){
19 if(args[i]){
20 o = args[i].split("=");
21 //相同的参数以后添加的参数为主，会覆盖前面的参数
22 _args[o[0]] = o[1];
23 }
24 }
25
26 console.log(_args); //打印参数列表
27 return _args;
28 }
29 //获取地址栏 URL 的参数
30 getURLArgs(document.getElementById("getURLArgs"));
31 };
32 </script>
```

本例 html 代码如下：

```
01 <h2>获取地址栏 URL 的参数</h2>
02 <p id='getURLArgs'></p>
```

**【代码说明】**

- 第 1 步，URL 中的参数都是从“？”开始的，所以 JavaScript 代码第 08 行先计算“？”的位置，然后截取从“？”起始的所有字符，不包括“？”。
- 第 2 步，参数的分隔都是以“&”为标示符，因此以“&”为节点进行字符分隔，生成数组。
- 第 3 步，遍历用“&”分隔后的数组，再以“=”进行分隔，就可以得出参数的“键”与“值”了。

## 5.32 获得一个窗口的大小

在一些自适应的页面中，需要用 JavaScript 脚本不断地调整页面的大小，以适应不同设备的显示需求。那么如何获取窗口的大小呢？

本例 JavaScript 代码如下：

```
01 <script type="text/JavaScript">
02 window.onload = function(){
03 //获得一个窗口的大小
04 getWinSize = function(){
05 return {
06 //如果 IE7~8 不支持，采用后边获取属性的方法
07 width: window.innerWidth || document.documentElement.offsetWidth,
08 height:window.innerHeight || document.documentElement.offsetHeight
09 };
10 }
11 var winSize = getWinSize(); //获取大小
12 document.getElementById("getWinSize").innerHTML = "高：" + winSize.height + "宽：
13 " + winSize.width;
14 };
15 </script>
```

本例 HTML 代码如下：

```
01 <h2>获得一个窗口的大小</h2>
02 <p id='getWinSize'></p>
```

**【代码说明】**

- JavaScript 代码第 07 行“window.innerWidth”在浏览器 Chrome、FF、Safari、Opera 下支持，用来获取整个文档的可视区域宽，同样“window.innerHeight”获取可视区域高，注意以上这些尺寸包含滚动条的大小。
- IE 的一些旧版本不支持以上 API，所以转而采用“document.documentElement.offsetWidth”与“document.documentElement.offsetHeight”的方案来完成本例功能。

注意：在IE 8与IE 9中包含滚动条，IE 7中不包含。

### 5.33 让弹出窗口总在最前面

当打开很多页面窗口时，需要让用户优先操作的窗口可能会被其他窗口覆盖。要想让窗口总是显示在最前面，可以通过让弹出窗口获取焦点的方法来实现，但是不同的浏览器会产生不同的结果。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //让弹出窗口总在最前面
04 document.getElementById("popWinMaxTop").onclick = function(){
05 /*1 在支持模式对话框的浏览器中，用 showModalDialog()方法建立模式对话框
06 或用 showModelessDialog()方法建立无模式对话框（参数与模式对话框一样），
07 即可实现窗口在最前面*/
08
09 //2 打开新窗口
10 var self =
11 window.open("../closeParent.html","newWindow","width=100,height=100");
12 //新窗口获取焦点，即可在最上面
13 self.focus();
14 }
15 };
16 </script>
```

本例 HTML 代码如下：

```

01 <h2>让弹出窗口总是在最前面</h2>
02 <input type="button" id='popWinMaxTop' value="打开窗口"/>
```

#### 【代码说明】

- 有些支持 showModalDialog 模态对话窗口的浏览器，直接使用 showModalDialog() 方法即可实现弹出窗口总在上面。
- JavaScript 代码第 09~13 行在不支持模态对话窗口的浏览器下，只要让弹出的窗口获取焦点即可实现总在最前面的效果，但这种方式并不是所有浏览器都支持，请根据项目的需求选择具体方案。

### 5.34 屏蔽功能键 Shift、Alt、Ctrl

浏览器的功能键与其他键组合会产生很多特殊组合效果，例如复制、粘贴等，一些有版权保护的文章类网站会限制这些操作。本例学习如何屏蔽 Shift、Alt、Ctrl，效果如图 5-12 所示。

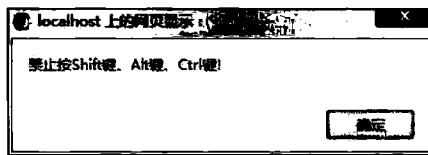


图 5-12 Chrome 屏蔽特殊键的效果

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 var shieldingFunctionKeys = (function(){ //屏蔽功能键 Shift、Alt、Ctrl
04 document.onkeydown = function(event){
05 event = event || _W.event;
06 if(event.shiftKey || event.altKey || event.ctrlKey){
07 alert("禁止按 Shift 键、Alt 键、Ctrl 键!")
08 }
09 };
10 })();
11 };
12 </script>

```

#### 【代码说明】

- 浏览器提供了一些特殊键的识别功能，JavaScript 代码第 06~08 行通过判断“Shift 键：event.shiftKey”、“Alt 键：event.altKey”、“Ctrl 键：event.ctrlKey”就知道用户按下的是否是这些键了。
- 当按下被屏蔽键时，给出弹框提示，就可以实现屏蔽这些特殊键的效果了。

## 5.35 页面慢慢变大

动画的大多数作用是为了增强用户体验。当用户单击一个链接打开新页面时，采用动画显示的方式（页面慢慢变大）会让用户觉得很亲切。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //页面慢慢变大
04 var
05 pageFillOutSize= {
06 width:100,
07 height:100
08 },
09 pageFillOutWin = null,
10 pageFillOutId = -1,
11 /*页面慢慢变大，支持 window.resizeBy()方法的浏览器都支持此效果，部分 IE、

```

```

12 Chrome 等支持*/
13 pageFillOut = function(){
14
15 pageFillOutWin =
16 window.open("../closeParent.html","", "width=100,height=100");
17 pageFillOutId = setInterval(function(){
18 pageFillOutSize.width += 6; //累加
19 pageFillOutSize.height += 6; //累加
20 pageFillOutWin.window &&
21 pageFillOutWin.window.resizeBy(pageFillOutSize.width, pageFillOutSize.height);
22 //如果此对象不存在或大于最大的宽度则停止动画
23 if(!pageFillOutWin.window || pageFillOutSize.width >= screen.availWidth){
24 clearInterval(pageFillOutId);
25 }
26 }, 50)
27
28 };
29
30 document.getElementById("pageFillOut").onclick = function(){ //页面慢慢变大
31 pageFillOut();
32 }
33 };
34 </script>

```

本例 HTML 代码如下：

```

01 <h2>页面慢慢变大</h2>
02 <input type="button" id='pageFillOut' value="页面变大" />

```

#### 【代码说明】

- 页面慢慢变大的效果主要是通过 JavaScript 代码第 20~21 行的“window.resizeBy()”方法实现，通过动态修改浏览器窗口的大小就可以实现动画效果。注意此方法在有的浏览器下会被禁止或不支持。
- 本例开启 setInterval 定时调用功能，不断地修改浏览器窗口的大小，当宽度大于浏览器最大窗口宽度或不存在“pageFillOutWin.window”时，关闭 setInterval。

## 5.36 IE 浏览器页面进入和退出的特效

在 IE 5.5~IE 8 浏览器中，有一些很酷的滤镜特效。例如：进入页面时盒装收缩、离开页面时垂直百页窗效果等，这样的特性可以很好地吸引网站访问者的眼球。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 /* 第一种方法，在<head></head>中加入以下类似代码：

```

```
04 <meta http-equiv="Page-Enter"
05 content="RevealTrans(duration=5,Transition=11)">
06 http-equiv 参数：
07 "进入网页" (Page-Enter) 、 "离开网页" (Page-Exit) 、 "进入站点" (Site-Enter) 、
08 "离开站点" (Site-Exit)
09 duration=时间
10 Transition=方式
11 Page-Enter
12 说明：duration 为页面切换的时间长度，3.000 表示 3 秒钟，一般可以直接输入 3
13 ; transition 为切换效果，从 1-23 共 22 种不同的切换效果，其中 23 为随机效果。
14 效果
15 盒状收缩 0
16 盒状展开 1
17 圆形收缩 2
18 圆形展开 3
19 向上擦除 4
20 向下擦除 5
21 向左擦除 6
22 向右擦除 7
23 垂直百叶窗 8
24 水平百叶窗 9
25 横向棋盘式 10
26 纵向棋盘式 11
27 溶解 12
28 左右向中部收缩 13
29 中部向左右展开 14
30 上下向中部收缩 15
31 中部向上下展开 16
32 阶梯状向左下展开 17
33 阶梯状向左上展开 18
34 阶梯状向右下展开 19
35 阶梯状向右上展开 20
36 随机水平线 21
37 随机垂直线 22
38 随机 23
39 */
40
41 //第二种方法 操控 IE 滤镜
42 if(!pageInOut.filters) return;
43 pageInOut.filters[0].Apply();
44 if (pageInOut.style.visibility == "visible")
45 {
46 pageInOut.style.visibility = "hidden";
47 pageInOut.filters.revealTrans.transition=23;
48 } else
```

```

49 {
50 pageInOut.style.visibility = "visible";
51 pageInOut.filters[0].transition=23;
52 }
53 pageInOut.filters[0].Play();
54 };
55 </script>

```

本例 HTML 代码如下：

```

01 <h2>页面进入和退出的特效,IE5.5~8 版本会支持此方法</h2>
02 <span id='pageInOut' style="position:absolute; Visibility:hidden;
03 Filter:revealTrans(duration=1, transition=1); width:100%; height:100%;">

```

#### 【代码说明】

- IE 浏览器提供了两种设置浏览器效果的方法；第一种便是在 META 元信息中加入特殊属性操控。http-equiv 为效果的类型：“进入网页”（Page-Enter）、“离开网页”（Page-Exit）、“进入站点”（Site-Enter）、“离开站点”（Site-Exit）。在 “content="RevealTrans(duration=5,Transition=11)"” 中，duration 是时间，Transition 代表效果参数。注释中已经将所有效果参数列出，请读者注意参考。
- 第二种，首先在页面中加入一个对象 pageInOut，通过脚本操控此对象的滤镜 duration 与 Transition 实现对应效果，参数与第一种方法类似。

## 5.37 页面全屏

在展示一些商品信息时，我们可以强制性地让用户的浏览器全屏，这样可以帮助用户更清晰地浏览网页内容。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 document.getElementById("fullscreen").onclick = function(){ //全屏交互按钮
04 window.moveTo(0,0); //定位坐标
05 window.resizeTo(screen.availWidth,screen.availHeight); //修改大小
06 }
07 };
08 </script>

```

本例 HTML 代码如下：

```

01 <h2>页面全屏</h2>
02 <input type="button" value="页面全屏" id="fullscreen" />

```

#### 【代码说明】

- JavaScript 代码第 05 行的“screen.availHeight”与“screen.availWidth”显示了除 window 窗口任务栏之外的大小。
- 通过定位坐标、修改窗口大小，就可以让浏览器全屏。要注意地是，有的浏览器不

支持或禁用了这两个方法。

### 5.38 定时关闭页面

我们在注册网站时，有些网站会弹出一个显示注册协议的网页，不管我们是否看完，几秒钟之后就自动关闭这个页面。这就用到了本例要说的“定时关闭页面”。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 //多少毫秒之后关闭
04 timeClosePageT = -1;
05 function timeClosePage(t){ //关闭页面，t 表示时间
06 timeClosePageT = t || -1;
07 if(timeClosePageT != -1){
08 setTimeout(function(){ //定时关闭
09 window.open("'",'_self',"");
10 window.close();
11 }, timeClosePageT)
12 }
13 }
14 document.getElementById("timeClosePage").onclick = function(){
15 timeClosePage(3000);
16 }
17 };
18 </script>
```

本例 HTML 代码如下：

```

01 <h2>定时关闭页面</h2>
02 <input type="button" id='timeClosePage' value="3 秒后关闭页面"/>
```

#### 【代码说明】

- 实现定时关闭，主要是通过调用 `setTimeout()` 来完成。
- 有的浏览器直接使用 “`window.close()`” 会不起作用，因此加上 “`window.open("'",'_self,"')`” 打开一个空窗口，绕过浏览器实现关闭，请参考代码第 07~12 行。
- 既然知道了如何定时关闭页面，给读者留个小练习，如何定时跳转页面呢？答案见 `pManage.js` 文件第 578 行。

### 5.39 打印页面的指定区域

页面打印经常在导出表格数据时用到。例如：电子商务网站需要打印一些购物清单、业务数据等。以下一行代码就可以实现这个功能：

```
01 window.print();
```

但是在实际的项目中，并没有这么简单，有时会打印一部分页面区域。本例就来演示如何打印指定的页面区域，在 Chrome 浏览下的效果如图 5-13 所示。

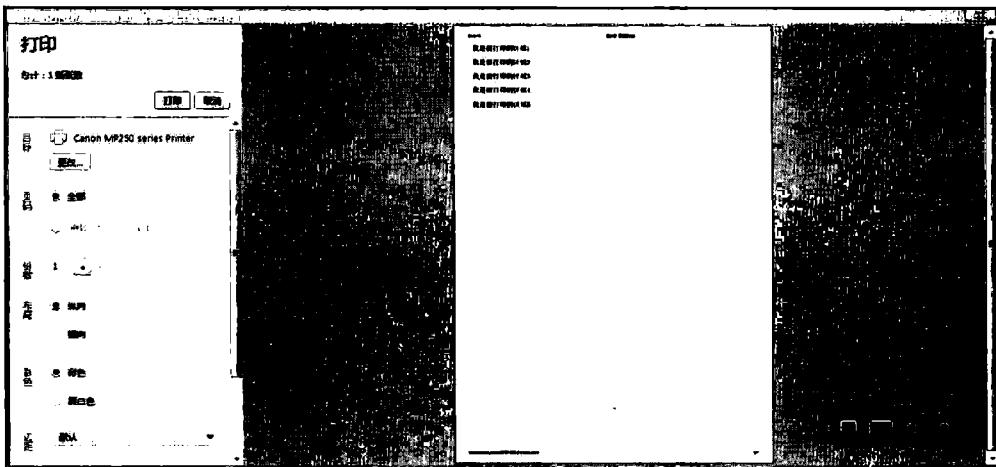


图 5-13 Chrome 打印效果

本例 JavaScript 代码如下：

```

27 }
28 }
29
30 displayFun("none"); //将不被打印的区域隐藏
31
32 if(isIE()){
33 /*注意有可能执行时，会出现没有效果的错误。
34 原因可能是你的浏览器限制了 active 对象的创建，只要取消限制就好了，
35 取消方法如下：
36 打开 IE 浏览器中“internet 选项”——“安全”——“自定义级别”
37 把“对没有标记为安全的 activex 控件进行初始化和脚本运行”设置为启用，
38 这样在打印按钮中只要加个事件触发就好了。*/
39 wb.execwb(6,6)
40 }else{
41 window.print(); //非 IE 支持的方案
42 }
43 //显示不被打印的区域
44 displayFun("block");
45 }
46
47 //页面打印及预览指定区域
48 document.getElementById("printAndpreview").onclick = function(){
49 printAndpreview();
50 }
51 };
52 </script>

```

本例 HTML 代码如下：

```

01 <div class="noprint">
02 <h2>页面打印及预览指定区域</h2>
03 </div>
04 <p>我是被打印的区域 1</p>
05 <p>我是被打印的区域 2</p>
06 <p>我是被打印的区域 3</p>
07 <p>我是被打印的区域 4</p>
08 <p>我是被打印的区域 5</p>
09 <div class="noprint">
10 <OBJECT classid="CLSID:8856F961-340A-11D0-A96B-00C04FD705A2" height=0
11 id=wb name=wb width=0></OBJECT>
12 <input type="button" id='printAndpreview' value="打印" />
13
14 </div>

```

### 【代码说明】

- 第 1 步，非 IE 浏览器内置了打印方法 “window.print()” 。IE 浏览器是先在页面植入一个对象 “<OBJECT classid="CLSID:8856F961-340A-11D0-A96B-00C04FD705A2" height=0 id=wb name=wb width=0></OBJECT>” ，然后再调用 “wb.execwb(6,6)” ，但是这些方法都是打印全页。

- 既然是打印全页，那么与我们要打印部分区域就有一定的关系了，只要在策略上改变一下即可。本例在打印之前，首先将“非打印区域”全部隐藏，然后再打印，打印完了之后再将隐藏的区域显示出来，这样就能实现区域打印了。

## 5.40 去掉打印时的页眉页脚

打印总会显示一些不需要的信息。例如：打印时的页眉与页脚会干扰打印的实际效果，不美观也浪费墨水。目前只有 IE 提供了去除页眉与页脚的 API。

本例 JavaScript 代码如下：

```

01 <script type="text/JavaScript">
02 window.onload = function(){
03 function clearPrintHeadFoot(){ //去掉打印时的页眉页脚
04 try //IE 中设置网页打印的页眉页脚为空
05 {
06 var
07 hkey_root="HKEY_CURRENT_USER",
08
09 hkey_path="\Software\Microsoft\Internet Explorer\PageSetup\",
10 hkey_ = hkey_root+hkey_path,
11 /*通过 ActiveXObject 可以访问 Windows 的本地文件系统和应用程序，创建
12 WScript.Shell 服务系统组件对象*/
13 ws=new ActiveXObject("WScript.Shell"),
14 hkey_key="header"; //设置头部为根键
15 ws.RegWrite(hkey_+hkey_key,"");
16 hkey_key="footer"; //设置底部为根键
17 ws.RegWrite(hkey_+hkey_key,"");
18 }
19 catch(e)
20 {
21 alert("您的浏览器不支持脚本去除页眉与页脚，请手动设置！");
22 }
23 }
24 };
25 </script>
```

### 【代码说明】

- IE 浏览器中有一个特殊的 API，就是 JavaScript 代码第 11~13 行的 ActiveXObject，通过它可以访问 Windows 本地文件系统和一些应用程序，所以要去除页眉与页脚不但要求是 IE 浏览器还得要求是 Windows 系统。
- 第 1 步，创建访问系统程序的对象 ws，并设置组件的访问路径等。
- 第 2 步，通过 ws.RegWrite 设置相应的参数，修改系统服务。
- 第 3 步，如果系统不支持以上方法，则给出提示。

# 第 6 章 日期处理常用代码

时间是构成世界的基本要素，在人们知道如何描述时间之后，“日期”这一名词便诞生了。日期的处理是项目中常见的需求，在 JavaScript 中，Date 对象用于处理日期和时间，它是浏览器内置对象的一员。

本章主要涉及的知识点如下。

- 获取时间的各个部分，包括：年、月、日、时、分、秒。
- 通过 Date 对象，获取当前时间及最后修改时间。
- 格式化时间：日期格式化成字符、获取短日期格式。
- 获取指定日期的结果：天数、第几周、判断是闰年还是平年。
- 其他一些常用代码：比较日期大小、时间倒计时、计算两个日期的时差、指定日期加减等。

## 6.1 获取日期的指定部分

在程序中，日期的主要构成部分包括：年、月、日、时、分、秒。JavaScript 可以获取本地浏览器的系统日期，本例就来演示如何获取本地日期的各个组成部分。

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 /*获取日期的指定部分*/
04 var d = new Date(); //获取日期对象
05 console.log(d.getFullYear() + "年"); //获取年
06 console.log((d.getMonth() + 1) + "月"); //获取月函数默认是 0~11 所以要+1
07 console.log(d.getDate() + "日"); //获取日
08 console.log(d.getHours() + "时"); //获取时
09 console.log(d.getMinutes() + "分"); //获取分
10 console.log(d.getSeconds() + "秒"); //获取秒
11 };
12 </script>
```

### 【代码说明】

- Date 对象用于处理日期和时间，JavaScript 代码第 04 行通过 new Date() 来创建 Date 对象，Date 对象会自动初始化当前时间，并返回日期格式的对象。

- Date 对象有很多种获取特定时间部分的函数，本节只介绍了获取年、月、日、时、分、秒的函数。
- 获取日期中的“年”有两种方法：getYear()与getFullYear()。由于从 ECMAScript v3 开始不再使用 getYear()，因此建议使用 getFullYear()。
- 获取日期中“月”的函数是 getMonth()，该函数返回月的范围值是“0~11”，因此按照国内需求，一般情况下需要将返回值加 1。

## 6.2 显示当前时间

本例用年、月、日、时、分、秒的形式显示当前时间，效果如图 6-1 所示。

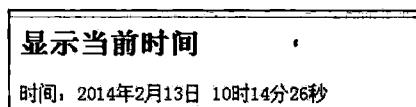


图 6-1 显示当前时间

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function getNowTime(){
04 var date =new Date(); //获取日期对象
05
06 /*获取年、月、日、时、分、秒，本地系统的时间*/
07 return date.getFullYear() + "年"
08 + (date.getMonth() + 1) + "月"
09 + date.getDate() + "日"
10 + " "
11 + date.getHours() + "时"
12 + date.getMinutes() + "分"
13 + date.getSeconds() + "秒";
14 }
15 //显示当前时间
16 document.getElementById("nowTime").innerHTML = "时间: " + getNowTime();
17 };
18 </script>
```

本例 HTML 代码如下：

```

01 <h2>显示当前时间</h2>
02 <p id='nowTime'>时间: </p>
```

### 【代码说明】

本例的关键是 JavaScript 代码第 03~14 行实现的 getNowTime() 函数，它首先利用上一节的方法获取时间的各个部分，然后将时间的各个部分用“+”号运算符按照“年月日时分秒”的顺序进行组合，就形成了当前时间格式。

### 6.3 显示最后修改时间

针对不同的项目，时间有不同的用法和算法。例如：在线招聘网站中需要记录求职者最后修改简历的时间，效果如图 6-2 所示。

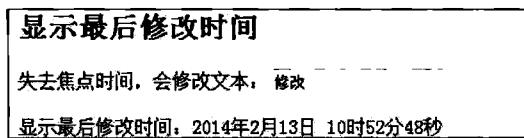


图 6-2 显示文本最后修改时间

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function getNowTime(){
04 var date =new Date(); //获取日期对象
05 /*获取年、月、日、时、分、秒，本地系统的时间*/
06 return date.getFullYear() + "年"
07 + (date.getMonth() + 1) + "月"
08 + date.getDate() + "日"
09 + " "
10 + date.getHours() + "时"
11 + date.getMinutes() + "分"
12 + date.getSeconds() + "秒";
13 }
14 //显示最后修改时间
15 document.getElementById("updateContent").onblur = function(){
16 document.getElementById("getLastMTime").innerHTML = "显示最后修改时间：" +
17 + getNowTime();
18 }
19 };
20 </script>
```

本例 HTML 代码如下：

```

01 <h2>显示最后修改时间</h2>
02 失去焦点时间，会修改文本：<input type="text" id='updateContent' />

03 <p id='getLastMTime'>显示最后修改时间：</p>
```

#### 【代码说明】

- 第 1 步，JavaScript 代码第 15 行为待修改的文本绑定 blur 事件，事件被触发时就记录当前时间。
- 第 2 步，调用 getNowTime() 获取当时的修改时间，即最后修改时间。

## 6.4 实时显示当前时间

笔者刚做开发时，觉得在网页上显示时间太简单了，不就是创建一个 Date 对象嘛！但这个时间是固定值，没法像钟表一样自动走起来，要让网页上这个时间走起来，我们还需要用到定时器。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function getNowTime (){
04 var date =new Date(); //获取日期对象
05 /*获取年、月、日、时、分、秒，本地系统的时间*/
06 return date.getFullYear() + "年"
07 + (date.getMonth() + 1) + "月"
08 + date.getDate() + "日"
09 + " "
10 + date.getHours() + "时"
11 + date.getMinutes() + "分"
12 + date.getSeconds() + "秒";
13 }
14
15 setInterval(function(){ //定时调用，不断修改为当前时间
16 document.getElementById("showNowTime").innerHTML = "时间：" +
17 getNowTime();
18 }, 1000);
19 };
20 </script>
```

本例 html 代码如下：

```

01 <h2>实时显示当前时间</h2>
02 <p id='showNowTime'>时间：</p>
```

### 【代码说明】

- 要想实时地显示时间，必须隔一段时间获取一下当前的日期，时间的间隔周期建议是 1 秒。
- 第 1 步，JavaScript 代码第 15 行开启一个定时器 setInterval。
- 第 2 步，在时间间隔调用的函数中，通过 getNowTime() 获取当前时间。
- 将以上两步综合起来，就可以看到走动的时间了，即实时显示当前时间。

## 6.5 将日期格式化成字符串

东西方的日期显示格式不同，因为语言都是别人发明的，如果要显示国内的日期格式，开发人员必须在后台进行日期格式化操作。例如：默认格式为：“MM-YYYY-DD h:m:s”，

要求转换为：“YYYY/MM/DD h-m-s”或“YYYY:MM:DD h:m:s”，效果如图 6-3 所示。

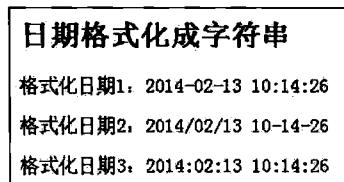


图 6-3 格式化后的日期

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 //日期格式化成字符串
04 function dateFormat(){
05 Date.prototype.format = function(f){
06 var date = { //获取对象中的日期
07 "Y" : this.getFullYear(), //获取年
08 "M" : (this.getMonth() + 1), //获取月
09 "D" : this.getDate(), //获取日
10 "h" : this.getHours(), //获取小时
11 "m" : this.getMinutes(), //获取分钟
12 "s" : this.getSeconds() //获取秒
13 },
14 d = "", //初始化接受日期变量的对象
15 r = false, //判断是否存在待替换的字符
16 reg = null, //正则
17 _d = ""; //日期
18 ;
19
20 for(d in date){ //过滤日期标示符
21 reg = new RegExp("[^" + d + "]{1,}", "g"); //判断是否有待格式化的字符
22 r = reg.test(f);
23
24 if(r) //验证是否存在
25 {
26 _d = date[d]; //被替换的日期
27 f = f.replace(reg, _d < 10 ? ("0" + _d) : _d);
28 }
29 }
30
31 return f;
32 }
33 }
34
35 var d = new Date(); //获取日期对象

```

```

36
37 /*待格式化的日期为当前日期,
38 字符格式替换规则：“Y”被替换为年，“M”替换为月，“D”替换为日期,
39 “h”替换为小时，“m”替换为分钟，“s”替换为秒
40 */
41 dateFormat();
42 document.getElementById("formatTime1").innerHTML = "格式化日期 1: " +
43 d.format("YYYY-MM-DD h:m:s");
44 document.getElementById("formatTime2").innerHTML = "格式化日期 2: " +
45 d.format("YYYY/MM/DD h-m-s");
46 document.getElementById("formatTime3").innerHTML = "格式化日期 3: " +
47 d.format("Y:M:D h:m:s");
48 };
49 </script>

```

本例 HTML 代码如下：

```

01 <h2>日期格式化成字符串</h2>
02 <p id='formatTime1'>格式化日期 1: </p>
03 <p id='formatTime2'>格式化日期 2: </p>
04 <p id='formatTime3'>格式化日期 3: </p>

```

#### 【代码说明】

- JavaScript 代码第 05 行是在 Date 原型链 prototype 上扩展。
- 第 1 步，构建当前日期对象的日期数据 date。
- 第 2 步，过滤日期标示符，检查传入的字符 f 中是否有符合待替换的日期格式，匹配规则：Y 被替换为年，M 替换为月，D 替换为日，h 替换为小时，m 替换为分钟，s 替换为秒。

## 6.6 获取短日期格式

日期一般分为：长格式与短格式。两种格式在定义上没有太明显的界限，大多情况下获取短日期格式，就是获取日期的年、月、日。例如：2014 年 5 月 1 日（短日期格式）是劳动节。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function dateFormat(){ //日期格式化成字符串
04 Date.prototype.format = function(f){
05 var date = { //获取对象中的日期
06 "Y" : this.getFullYear(), //获取年
07 "M" : (this.getMonth() + 1), //获取月
08 "D" : this.getDate(), //获取日
09 "h" : this.getHours(), //获取小时

```

```

10 "m" : this.getMinutes(), //获取分钟
11 "s" : this.getSeconds() //获取秒
12 },
13 d = "", //初始化接受日期变量的对象
14 r = false, //判断是否存在待替换的字符
15 reg = null, //正则
16 _d = ""; //日期
17
18 for(d in date){ //过滤日期标示符
19 //判断是否有待格式化的字符
20 reg = new RegExp("[^" + d + "]{1,}", "g");
21 r = reg.test(f);
22
23 if(r) //验证是否存在
24 {
25 _d = date[d]; //被替换的日期
26 f = f.replace(reg, _d < 10 ? ("0" + _d) : _d);
27 }
28 }
29
30 return f;
31 }
32 }
33 dateFormat(); //获取短日期格式
34 function getMinDate(){
35 return new Date().format("YYYY-MM-DD");
36 }
37 }
38 //获取短日期格式
39 document.getElementById("getMinDate").innerHTML = getMinDate();
40 };
41 </script>

```

本例 HTML 代码如下：

```

01 <h2>获取短日期格式—即年、月、日</h2>
02 <p id='getMinDate'>短日期：</p>

```

### 【代码说明】

- 本例首先明确一个概念，什么是短日期？一般情况下，短日期就是只有“年、月、日”没有“时、分、秒”的日期。
- JavaScript 代码第 36 行调用上一节讲解的格式化日期函数 “new Date().format("YYYY-MM-DD")”，可以直接显示短日期格式的日期。

## 6.7 获得指定日期所在月份的天数

通过计算某个月份的天数可以完成很多与日期结合的需求。例如：通过程序可以计算员工当月实际的工作天数、计算 2 月份天数判断是平年还是闰年等。如何获取指定日期的天数呢？如果不太了解 JavaScript 特性的读者，首先想到的是用多么复杂的算法实现，其实没那么麻烦。本例效果如图 6-4 所示。

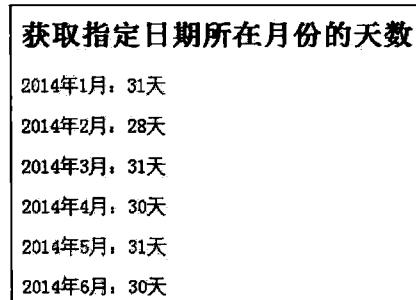


图 6-4 显示指定日期所在月份的天数

本例 JavaScript 代码如下：

```

26 _d = date[d];
27 f = f.replace(reg, _d < 10 ? ("0" + _d) : _d);
28 }
29 }
30
31 return f;
32 }
33 }
34
35 dateFormat();
36
37 //获取指定日期所在月份的天数
38 function getMonthDays(Y, M){
39 /*Y 代表年份；M 代表为月数 0~11，月份应加 1，但第 3 个参数为 0，所以不+1；
40 第 3 个参数要求最小为 1，设置为 0 就是指 M 月的最后一天*/
41 return new Date(Y, M, 0).getDate();
42 }
43 //获取指定日期所在月份的天数=====start
44 document.getElementById("getMonthDays1").innerHTML = "2014 年 1 月：" +
45 getMonthDays("2014", "1") +"天";
46 document.getElementById("getMonthDays2").innerHTML = "2014 年 2 月：" +
47 getMonthDays("2014", "2") +"天";
48 document.getElementById("getMonthDays3").innerHTML = "2014 年 3 月：" +
49 getMonthDays("2014", "3") +"天";
50 document.getElementById("getMonthDays4").innerHTML = "2014 年 4 月：" +
51 getMonthDays("2014", "4") +"天";
52 document.getElementById("getMonthDays5").innerHTML = "2014 年 5 月：" +
53 getMonthDays("2014", "5") +"天";
54 document.getElementById("getMonthDays6").innerHTML = "2014 年 6 月：" +
55 getMonthDays("2014", "6") +"天";
56 };
57 </script>

```

本例 HTML 代码如下：

```

01 <h2>获取指定日期所在月份的天数</h2>
02 <p id='getMonthDays1'>2014 年 1 月：</p>
03 <p id='getMonthDays2'>2014 年 2 月：</p>
04 <p id='getMonthDays3'>2014 年 3 月：</p>
05 <p id='getMonthDays4'>2014 年 4 月：</p>
06 <p id='getMonthDays5'>2014 年 5 月：</p>
07 <p id='getMonthDays6'>2014 年 6 月：</p>

```

#### 【代码说明】

- 在 Date 对象中，第 3 个参数默认为 1~31，那么如果写一个超过范围的数字会出现什么现象，读者可以测试一下。
- 当将第 3 个参数设置为 0 时，就会获取月份的最后一天，也就是当前月份有多少天。

## 6.8 获取指定日期是第几周

有些需求是以周为单位计算的。例如：在一个孕妇管理网站中，会帮助孕妇计算怀胎几周了、当前日期是第几周等。如果获取日期时，需要获取指定日期是当年的第几周，该怎么测算呢？计算第几周没有获取天数那么简单，需要真正地“计算”。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 //获取指定日期所在周是第几周=====
04 function getHowManyWeeks(Y, M, D){
05 var totalDays = 0, //总天数
06 i = 1; //默认开始为第 1 个月
07
08 for(; i < M ; i++){ //计算总天数
09 totalDays += this.getMonthDays(Y, M);
10 }
11
12 totalDays += D;
13
14 return Math.ceil(totalDays/7); //除以 7，向上取整，计算第几周
15
16 }
17 //获取指定日期所在周是第几周
18 document.getElementById("getHowManyWeeks").innerHTML = "第" +
19 getHowManyWeeks("2014", "1", "6") + "周";
20 };
21 </script>
```

本例 HTML 代码如下：

```

01 <h2>获取指定日期所在周是第几周</h2>
02 <p id='getHowManyWeeks'></p>
```

### 【代码说明】

- JavaScript 代码第 04~16 行是本例的关键，要想计算第几周，首先要计算指定的天是当年的第几天，然后除以 7，向上取整，就可以获取具体是第几周。
- 第 1 步，以月份 M 为最大的循环范围，计算月份 M 之前有多少天，由于上一节已经学习过取得指定月份的天数，所以这一节的算法直接调用就可以。
- 第 2 步，加上 D 天，即为所有累加的天数。
- 第 3 步，除以 7，向上取整，就可以算出指定日期为当年的第几周。

## 6.9 倒计时

中国奥运会的倒计时电子牌当年在国内各大城市让人印象深刻。在 Web 项目中，经

常需要计算距离某个节日的天数，尤其是电商狂欢日越来越多，倒计时功能都成了前端开发人员必备代码了。本例效果如图 6-5 所示。

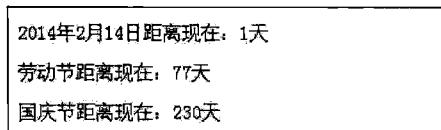


图 6-5 倒计时的时间

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function getCountDown(Y, M, D, h, m, s){ //到指定日期的倒计时
04 Y = Y || 0;
05 M = M || 0;
06 D = D || 0;
07 h = h || 0;
08 m = m || 0;
09 s = s || 0;
10 var date = new Date(Y, M-1, D, h, m, s),
11 //转换为时间戳，方便计算差值
12 times = date.getTime() - new Date().getTime();
13 return Math.ceil(times / (1000 * 60 * 60 * 24)); //返回天数
14 }
15 //到指定日期时间的倒计时
16 document.getElementById("getCountDown").innerHTML = "2014 年 2 月 14 日距离现
17 在：" + getCountDown("2014", "2", "14") +"天";
18 //节日倒计时
19 document.getElementById("LabourDay").innerHTML = "劳动节距离现在：" +
20 getCountDown("2014", "5", "1") +"天";
21 document.getElementById("NationalDay").innerHTML = "国庆节距离现在：" +
22 getCountDown("2014", "10", "1") +"天";
23 };
24 </script>
```

本例 HTML 代码如下：

```

01 <p id='getCountDown'>2014 年 2 月 14 日距离现在：</p>
02 <p id='LabourDay'>劳动节距离现在：</p>
03 <p id='NationalDay'>国庆节距离现在：</p>
```

#### 【代码说明】

- 第 1 步，要计算距指定日期的天数，必须知道两个日期：当前日期与指定日期。
- 第 2 步，JavaScript 代码第 12 行将当前日期与指定日期都转换为时间戳，然后相减，除以“ $1000 * 60 * 60 * 24$ ”，就可以得到倒计时的天数。

## 6.10 比较两个日期相差多少秒

先给读者出一个题：“2013年3月2日”与“2014-3-1”如何计算差值？需要年月日都计算一遍吗？日期的表现形式很多，3月2日能直接减去3-1吗？是不是本来觉得很简单，看到这个问题后才发现，比较两个日期不是那么容易的呀！

其实，如果转换为相同格式的时间形式就很好计算了，比如都转化为时间戳再计算。本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function getDateDifferenceValue(date1,date2){
04 //第1个时间
05 var d1 = new Date(date1.Y || 0, (date1.M-1) || 0, date1.D || 1, date1.h || 0,
06 date1.m || 0, date1.s || 0).getTime(),
07 //第2个时间
08 d2 = new Date(date2.Y || 0, (date2.M-1) || 0, date2.D || 1, date2.h || 0,
09 date2.m || 0, date2.s || 0).getTime();
10 return (d1 - d2)/1000; //计算时间差值
11 }
12 //比较两个日期的差
13 document.getElementById("getDateDifferenceValue").innerHTML = "2014 年 6 月 3
14 日与 2014 年 6 月 4 日差值" + getDateDifferenceValue(
15 {
16 "Y":"2014",
17 "M":"6",
18 "D":"3"
19 },
20 {
21 "Y":"2014",
22 "M":"6",
23 "D":"4"
24 }) + "秒";
25 };
26 </script>
```

本例 HTML 代码如下：

```

01 <h2>比较两个日期的差</h2>
02 <p id='getDateDifferenceValue'>2014 年 6 月 3 日与 2014 年 6 月 4 日差值：</p>
```

### 【代码说明】

- 第1步，将待比较的两个日期转换为相同的时间戳，加空格 JavaScript 代码第 03~11 行调用 `getTime()` 可以自动转换，`getTime()` 可以返回从 1970 年 1 月 1 日到现在的毫秒数。
- 第2步，比较计算后的时间戳，除以 1000，就可以计算出相差多少秒。

## 6.11 比较日期大小

日期之间比较大小是常见需求。例如：判断时间是否过时，只要比较指定日期与当前日期大小即可。上一节已经学会怎样计算两个日期的差值，那么，只要将计算的差值与 0 比较，就可以判断两个日期的大小了。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 //比较两个日期的差
04 function getDateDifferenceValue(date1,date2){
05 //第 1 个时间
06 var d1 = new Date(date1.Y || 0, (date1.M-1) || 0, date1.D || 1, date1.h || 0,
07 date1.m || 0, date1.s || 0).getTime(),
08 //第 2 个时间
09 d2 = new Date(date2.Y || 0, (date2.M-1) || 0, date2.D || 1, date2.h || 0,
10 date2.m || 0, date2.s || 0).getTime();
11 return (d1 - d2)/1000; //计算时间差值
12 }
13
14 function getDateSize(date1,date2){ //日期比较大小，返回 true 为大，false 为小
15 return getDateDifferenceValue(date1, date2) > 0 ? true : false;
16 }
17 //日期比较大小
18 document.getElementById("getDateSize1").innerHTML = "2014 年 6 月 3 日比 2014
19 年 6 月 4 日：" + (getDateSize(
20 {
21 "Y":"2014",
22 "M":"6",
23 "D":"3"
24 },
25 {
26 "Y":"2014",
27 "M":"6",
28 "D":"4"
29 }) ? "大" : "小");
30
31 document.getElementById("getDateSize2").innerHTML = "2014 年 7 月 5 日比 2014
32 年 6 月 4 日：" + (getDateSize(
33 {
34 "Y":"2014",
35 "M":"7",
36 "D":"5"
37 },

```

```

38 "Y": "2014",
39 "M": "6",
40 "D": "4"
41 }) ? "大": "小";
42 }
43 };
44 </script>

```

本例 HTML 代码如下：

```

01 <h2>日期比较大小</h2>
02 <p id='getDateSize1'>2014 年 6 月 3 日比 2014 年 6 月 4 日：</p>
03 <p id='getDateSize2'>2014 年 7 月 5 日比 2014 年 6 月 4 日：</p>

```

#### 【代码说明】

- 第 1 步，JavaScript 代码第 04~12 行调用上一节的函数 getDateDifferenceValue() 计算出两个日期的差值。
- 第 2 步，将两个差值相减，然后和 0 进行比较。如果大于 0，则第 1 个日期大于第 2 个日期，反之，第 1 个日期小于第 2 个日期。

## 6.12 对指定日期进行加减

学过 10 位数加减法的幼儿园学生都可以计算出 7 天后是几号，但在程序中，我们如果要获取 7 天后的日期，可不是简单的加 7 就可以的。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var d = new Date(); // 获取日期对象
04 // 日期格式化成字符串
05 function dateFormat(){
06 Date.prototype.format = function(f){
07 var date = {// 获取对象中的日期
08 "Y" : this.getFullYear(), // 获取年
09 "M" : (this.getMonth() + 1), // 获取月
10 "D" : this.getDate(), // 获取日
11 "h" : this.getHours(), // 获取小时
12 "m" : this.getMinutes(), // 获取分钟
13 "s" : this.getSeconds() // 获取秒
14 },
15 d = "", // 初始化接受日期变量的对象
16 r = false, // 判断是否存在待替换的字符
17 reg = null, // 正则
18 _d = ""; // 日期
19

```

```

20 for(d in date){ //过滤日期标示符
21 //判断是否有待格式化的字符
22 reg = new RegExp("[" + d + "]{1,}", "g");
23 r = reg.test(f);
24
25 if(r) //验证是否存在
26 {
27 _d = date[d]; //被替换的日期
28 f = f.replace(reg, _d < 10 ? ("0" + _d) : _d);
29 }
30 }
31
32 return f;
33 }
34 }
35 dateFormat();
36 function setXDate(date, xY, xM, xD, xh, xm, xs){ //对指定日期进行加减
37 xY = xY || 0;
38 xM = xM || 0;
39 xD = xD || 0;
40 xh = xh || 0;
41 xm = xm || 0;
42 xs = xs || 0;
43
44 if(xY){ //如果存在年的差值，则计算
45 date.setFullYear(date.getFullYear() + xY);
46 }
47
48 if(xM){ //如果存在月的差值，则计算
49 date.setMonth(date.getMonth() + xM);
50 }
51
52 if(xD){ //如果存在日的差值，则计算
53 date.setDate(date.getDate() + xD);
54 }
55
56 if(xh){ //如果存在时的差值，则计算
57 date.setHours(date.getHours() + xh);
58 }
59
60 if(xm){ //如果存在分的差值，则计算
61 date.setMinutes(date.getMinutes() + xm);
62 }
63
64 if(xs){ //如果存在秒的差值，则计算

```

```

65 date.setSeconds(date.getSeconds() + xs);
66 }
67
68 return date.format("YYYY-MM-DD h:m:s")
69
70 }
71 //对指定日期进行加减
72 document.getElementById("setXDate1").innerHTML = "获取当前日期, 增加 7 天: " +
73 setXDate(d, 0, 0, 7);
74 document.getElementById("setXDate2").innerHTML = "获取当前日期, 增加 7 年: " +
75 setXDate(d, 7);
76 }
77 </script>

```

本例 HTML 代码如下：

```

01 <h2>对指定日期进行加减</h2>
02 <p id='setXDate1'>获取当前日期, 增加 7 天: </p>
03 <p id='setXDate2'>获取当前日期, 增加 7 年: </p>

```

#### 【代码说明】

- Date 对象可以获取时间，也可以设置时间。设置时间的函数与获取时间的函数名字基本一样，只是将所有的 get 都换成了 set。如 JavaScript 代码第 44~66 行的 getFullYear() 改成了 setFullYear()、getMonth() 改成了 setMonth()。
- 第 1 步，第 37~42 行获取增加或减少的时间，如果参数不存在，默认设置为 0。
- 第 2 步，第 44~66 行判断是否存在对应的时间变化，数值不等于 0 就进行加减计算。

## 6.13 将字符串转换成日期格式

格式化的时间字符与日期格式是可以相互转换的，也就是说，既然日期可以按一定的格式转换成字符，那么字符格式的日期也可以转换为日期格式的对象。本例效果如图 6-6 所示。

### 字符串转换为日期格式

2014-02-19 15:56:01 转换为日期格式: Wed Mar 19 2014 15:56:01 GMT+0800 (中国标准时间)

图 6-6 日期格式的时间

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function strDate(strDate, s1, s2){ //字符串转换为日期格式
04 var d = strDate.split(" "), //以空格为界进行第一次日期分隔
05 d1 = d[0], //年月日的数组
06 d2 = d[1]; //时分秒的数组

```

```

07 D1 = d1.split(s1 || "-"), //分隔年月日为数组
08 D2 = d2.split(s2 || ":"), //分隔时分秒为数组
09 return new Date(
10 D1[0] || 0,
11 D1[1] || 0,
12 D1[2] || 1,
13 D2[0] || 0,
14 D2[1] || 0,
15 D2[2] || 0
16)
17 }
18 //字符串转换为日期格式
19 document.getElementById("strDate").innerHTML = "2014-02-19 15:56:01 转换为日
20 期格式：" + strDate("2014-02-19 15:56:01");
21);
22 </script>

```

本例 HTML 代码如下：

```

01 <h2>字符串转换为日期格式</h2>
02 <p id='strDate'></p>

```

#### 【代码说明】

- 第 1 步，JavaScript 代码第 04~08 行将传入的时间字符进行空格分隔，得到“年月日”一组，“时分秒”一组。
- 第 2 步，JavaScript 代码第 07~08 行进行再一次分隔，得到两个分组，即“D1”与“D2”。
- 第 3 步，将分隔后的所有数据传入 Date，实例化对象便可以得到日期格式的时间。

## 6.14 判断是闰年还是平年

闰年与平年是为了解决因人为历法造成的年度天数与地球公转周期的一些时差问题。当然，我们没有必要去计算地球的公转，那样既麻烦也不实际。在 JavaScript 中，判断日期是闰年还是平年的最好方法便是判断特定某年的 2 月是多少天。本例效果如图 6-7 所示。

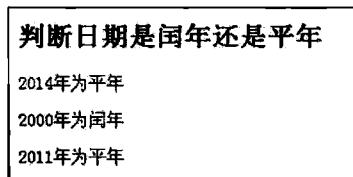


图 6-7 显示特定日期是什么年

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){

```

```

03 //日期格式化成字符串
04 function dateFormat(){
05 Date.prototype.format = function(f){
06 var date = { //获取对象中的日期
07 "Y" : this.getFullYear(), //获取年
08 "M" : (this.getMonth() + 1),//获取月
09 "D" : this.getDate(), //获取日
10 "h" : this.getHours(), //获取小时
11 "m" : this.getMinutes(), //获取分钟
12 "s" : this.getSeconds() //获取秒
13 },
14 d = "", //初始化接受日期变量的对象
15 r = false, //判断是否存在待替换的字符
16 reg = null, //正则
17 _d = ""; //日期
18
19 for(d in date){ //过滤日期标示符
20 reg = new RegExp("[^" + d + "]{1,}", "g"); //判断是否有待格式化的字符
21 r = reg.test(f);
22
23 if(r) //验证是否存在
24 {
25 _d = date[d]; //被替换的日期
26 f = f.replace(reg, _d < 10 ? ("0" + _d) : _d);
27 }
28 }
29
30 return f;
31 }
32 }
33 dateFormat();
34 function getMonthDays(Y, M){ //获取指定日期所在月份的天数
35 /*Y 代表年份； M 代表月数 0~11， 月份应加 1， 但是第 3 个参数为 0， 所以不+1;
36 第 3 个参数要求最小为 1， 但设置为 0， 就变成 M 月的最后一天了*/
37 return new Date(Y, M, 0).getDate();
38 }
39
40 function getYearType(Y){ //判断日期是闰年还是平年
41 return getMonthDays(Y, 2) == 28 ? "平年" : "闰年";
42 }
43 //判断日期是闰年还是平年
44 document.getElementById("getYearType1").innerHTML = "2014 年为 " +
45 getYearType("2014");
46 document.getElementById("getYearType2").innerHTML = "2000 年为 " +
47 getYearType("2000");

```

```

48 document.getElementById("getYearType3").innerHTML = "2011 年为" +
49 getYearType("2011");
50 };
51 </script>

```

本例 HTML 代码如下：

```

01 <h2>判断日期是闰年还是平年</h2>
02 <p id='getYearType1'></p>
03 <p id='getYearType2'></p>
04 <p id='getYearType3'></p>

```

#### 【代码说明】

- 第 1 步，JavaScript 代码第 34~38 行调用 `getMonthDays()` 函数，获取特定年份 2 月的天数。
- 第 2 步，判断 2 月份的天数是否等于 29，如果是则为“闰年”，反之为“平年”。

## 6.15 日期合法性验证

Web 应用中，用户的输入经常不按常理出牌，如果不是按规定的日期格式输入，程序在处理日期时就会发生问题，例如：日期格式要求“年-月-日”，但用户输入的是“2014.06.02”，这样的日期是不符合要求的，加大了后台程序的处理难度，因此日期格式的合法性验证也是前端比较重要的一个环节。本例效果如图 6-8 所示。

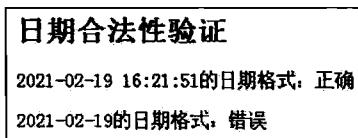


图 6-8 验证日期格式是否符合格式

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function verifyDate(vDate){ //日期合法性验证
04 /*验证格式必须为 "YYYY-MM-DD hh:mm:ss" 格式，
05 类似 "2014-02-12 16:34:57" */
06 return /^(\d{4})-(\d{2})-(\d{2}) (\d{1})(\d{2}):\d{2}:\d{2}$/.test(vDate);
07 }
08 //日期合法性验证=====start
09 document.getElementById("verifyDate1").innerHTML = "2021-02-19 16:21:51 的日期
10 格式：" + (verifyDate("2021-02-19 16:21:51") ? "正确" : "错误");
11 document.getElementById("verifyDate2").innerHTML = "2021-02-19 的日期格式：" +
12 (verifyDate("2021-02-19") ? "正确" : "错误");
13 };
14 </script>

```

本例 HTML 代码如下：

```
01 <h2>日期合法性验证</h2>
02 <p id='verifyDate1'></p>
03 <p id='verifyDate2'></p>
```

**【代码说明】**

- 第1步，获取日期 vDate，日期为字符类型。
- 第2步，JavaScript 代码第 06 行的 test() 函数采用正则验证的方式，检测日期是否与正则表达式匹配，如果匹配则返回 true，否则返回 false。

# 第 7 章 页面特效常用代码

Web 1.0 时代的网页是枯燥的，我们还记得进入某些网站后，鼠标后面会追随着一堆字母，这是早期的页面特效，现在想起来觉得挺 OUT 的。Web 2.0 时代，页面特效越来越绚丽，除了第 5 章介绍的页面处理外，还有一些好玩的特效。这些特效主要是通过动态操作 DOM 来实现的。

本章主要涉及的知识点有如下。

- 各种菜单效果：下拉式导航菜单、滑动门导航、树形菜单导航、仿 QQ 菜单。
- 各种广告效果：滑动展开/收缩广告、定时关闭背投广告。
- 有趣的效果：五颜六色的雪花。
- 其他效果：图片快速翻动的幻灯效果。

## 7.1 页面悬浮导航

在比较长的页面中，为了让 UI 总是显示在页面的可视范围内，需要不断地调节 UI 在页面的位置，比如通过调整 top 与 left 值就可以将一个 UI 定位在可视范围内。最常见的应用场景就是关于导航菜单的定位。例如：在一些很长的页面中，不管怎么滚动页面，导航菜单始终如一地显示在页面窗口顶部。

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 //获取待定位的元素
04 var suspendNavigation = document.getElementById("suspendNavigation");
05 window.onscroll = function(){ //绑定滚轴事件
06 suspendNavigation.style.top = (document.documentElement.scrollTop ||
07 document.body.scrollTop) + "px"; //将元素 top 定位
08 }
09 };
10 </script>
```

本例 HTML 代码如下：

```
01 <div id='suspendNavigation'>
02 <h2>页面悬浮导航</h2>
03 <div>导航 1</div>
```

```

04 <div class="">导航 2</div>
05 <div>导航 3</div>
06 <div>导航 4</div>
07 <div>导航 5</div>
08 </div>
09

10 <!--重复代码略-->
```

本例 CSS 代码如下：

```

01 <style>
02 a{
03 text-decoration:none;
04 }
05 /*=====页面悬浮导航=====*/
06 #suspendNavigation{
07 position: absolute;
08 top:0px;
09 left: 0px;
10 background-color: #ccc;
11 width: 100%;
12 z-index: 1000;
13 }
14 #suspendNavigation div{
15 position: relative;
16 float: left;
17 border: #cc2123 1px solid;
18 width: 19%;
19 height: 30px;
20 text-align: center;
21 line-height: 30px;
22 }
23 </style>
```

#### 【代码说明】：

- 第 1 步，在页面中增加一个导航。
- 第 2 步，为页面中的导航绑定 onscroll 事件。
- 第 3 步，JavaScript 代码第 05~08 行触发 onscroll 事件时，调整导航的 top 值，让 top 值等于滚动值，这样就能看到页面的导航一直悬浮于页面的顶部，效果如图 7-1 所示。

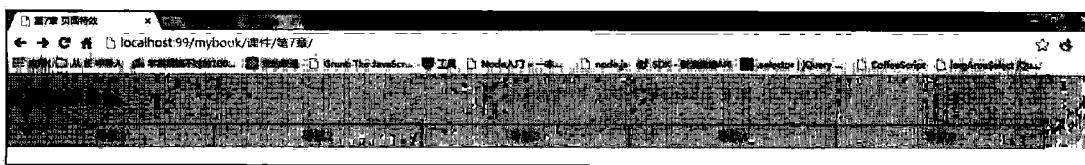


图 7-1 页面悬浮导航

## 7.2 下拉式导航菜单

有的导航分类太多，会导致子分类信息显示不全，此时可以采取其他的导航设计方案。例如：可扩展式导航、多级导航、下拉式导航等。本例演示的是下拉式导航，效果如图 7-2 所示。

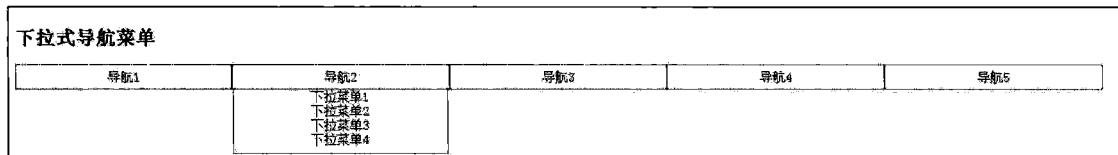


图 7-2 下拉式导航菜单

**说明：**下拉式导航菜单，是只显示主要的分类，隐藏子分类。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var
04 getTypeElement = function(es, type){ //获取指定类型的节点
05 var esLen = es.length,
06 i = 0,
07 eArr = [],
08 esl = null;
09 for(; i < esLen ; i++){
10 esl = es[i];
11 if(esl.nodeName.replace("#", "").toLowerCase() == type){
12 eArr.push(esl);
13 }
14 }
15 return eArr;
16 },
17 //获取所有下拉式导航菜单
18 navs =
19 getTypeElement(document.getElementById("pullDownNavigation").childNodes, "div"),
20 i = 0,
21 l = navs.length, //元素个数
22 targetID = null;
23
24 for(; i < l ; i++){
25 navs[i].onmousemove = function(){ //显示下拉菜单
26 targetID = this.getAttribute("data-targetID");
27 document.getElementById(targetID).style.display = "block";
28 }

```

```

29
30 navs[i].onmouseout = function(){ //隐藏下拉菜单
31 document.getElementById(targetID).style.display = "none";
32 }
33 }
34 };
35 </script>

```

本例 HTML 代码如下：

```

01 <h2>下拉式导航菜单</h2>
02 <div id='pullDownNavigation'>
03 <div class="navigation navigation1" data-targetID='pullDownNavigation1'>
04 <div class="nav">导航 1</div>
05 <!-- 导航 1 菜单-->
06 <div class="pullDownNavigationc" id='pullDownNavigation1'>
07 <div>下拉菜单 1</div>
08 <div>下拉菜单 2</div>
09 <div>下拉菜单 3</div>
10 <div>下拉菜单 4</div>
11 </div>
12 </div>
13 <div class="navigation navigation1" data-targetID='pullDownNavigation2'>
14 <div class="nav">导航 2</div>
15 <!-- 导航 2 菜单-->
16 <div class="pullDownNavigationc" id='pullDownNavigation2'>
17 <div>下拉菜单 1</div>
18 <div>下拉菜单 2</div>
19 <div>下拉菜单 3</div>
20 <div>下拉菜单 4</div>
21 </div>
22 </div>
23 <div class="navigation navigation1" data-targetID='pullDownNavigation3'>
24 <div class="nav">导航 3</div>
25 <!-- 导航 3 菜单-->
26 <div class="pullDownNavigationc" id='pullDownNavigation3'>
27 <div>下拉菜单 1</div>
28 <div>下拉菜单 2</div>
29 <div>下拉菜单 3</div>
30 <div>下拉菜单 4</div>
31 </div>
32 </div>
33 <div class="navigation navigation1" data-targetID='pullDownNavigation4'>
34 <div class="nav">导航 4</div>
35 <!-- 导航 4 菜单-->
36 <div class="pullDownNavigationc" id='pullDownNavigation4'>

```

```

37 <div>下拉菜单 1</div>
38 <div>下拉菜单 2</div>
39 <div>下拉菜单 3</div>
40 <div>下拉菜单 4</div>
41 </div>
42 </div>
43 <div class="navigation navigation1" data-targetID='pullDownNavigation5'>
44 <div class="nav">导航 5</div>
45 <!—导航 5 菜单-->
46 <div class="pullDownNavigationc" id='pullDownNavigation5'>
47 <div>下拉菜单 1</div>
48 <div>下拉菜单 2</div>
49 <div>下拉菜单 3</div>
50 <div>下拉菜单 4</div>
51 </div>
52 </div>
53 </div>

```

本例 CSS 代码如下：

```

01 <style>
02 a{
03 text-decoration:none;
04 }
05 /*-----下拉式导航菜单-----*/
06 #pullDownNavigation .navigation{
07 position: relative;
08 float: left;
09 border: #cc2123 1px solid;
10 width: 19%;
11 height: 30px;
12 text-align: center;
13 }
14 .pullDownNavigationc{
15 position: relative;
16 float: left;
17 top: -1px;
18 overflow: hidden;
19 height: 80px;
20 text-align: center;
21 width: 99%;
22 border: 1px solid #cc2123;
23 border-top: 0px;
24 display: none;
25 }
26 .navigation .nav{

```

```

27 height: 32px;
28 line-height: 30px;
29 }
30 </style>

```

#### 【代码说明】

- 第1步，创建主要分类及子分类，并且将子分类隐藏。
- 第2步，当鼠标移入主分类时，以下拉的形式将它的子分类展现出来。
- 第3步，当鼠标移出时，隐藏子类层。

## 7.3 滑动门导航

将滑动门导航放在页面的一侧，更能增加用户体验，并且不会占据页面的主空间，用户想使用的时候直接移到导航上，就可以显示导航菜单的全部UI界面。本例效果如图7-3所示。

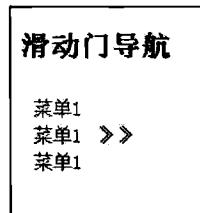


图7-3 滑动门导航划出页面

本例JavaScript代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function slideNavs(slide){// 滑动门导航=====
04 var slideld = -1;;
05 slide.onmouseover = function(){ //打开滑动门
06 //子元素会干扰事件，导致抖动，当再次进入元素时，停止隐藏滑动门的线程
07 clearTimeout(slideld);
08 new animateManage({ //播放显示元素的动画
09 "context": slide, //被操作的元素
10 "effect":"linear", ...
11 "time": 100, //持续时间
12 "startCss":{ //元素的起始值偏移量
13 "left":slide.style.left
14 },
15 "css":{ //元素的结束值偏移量
16 "left":0
17 }
18 }).init();

```

```

19 }
20
21 slide.onmouseout = function(){
22 slideId = setTimeout(function(){
23 new animateManage({
24 "context": slide,
25 "effect": "linear",
26 "time": 100,
27 "startCss": {
28 "left": slide.style.left
29 },
30 "css": {
31 "left": -72
32 }
33 }).init();
34 }, 300)
35
36 }
37 }
38 slideNavs(document.getElementById("slide"));
39 };
40 </script>

```

本例 HTML 代码如下：

```

01 <h2>滑动门导航</h2>
02 <div id='slide'>
03 <!--菜单主体-->
04 <div class="slideMain" id='slideMain'>
05 <div>菜单 1</div>
06 <div>菜单 1</div>
07 <div>菜单 1</div>
08 </div>
09
10 <!--引导卡-->
11 <div class="slideTab" title='显示菜单' id='slideTab'>
12 >
13 </div>
14 </div>

```

本例 CSS 代码如下：

```

01 <style>
02 a{
03 text-decoration:none;
04 }
05 /*=====滑动门导航=====*/
06 #slide{

```

```

07 position: relative;
08 left: -72px;
09 display: block;
10 border: 1px #ccc solid;
11 height: 82px;
12 width: 100px;
13 }
14 #slide div{
15 position: relative;
16 float: left;
17 margin-top: 5px;
18 margin-left: 5px;
19 }
20
21 .slideMain div{
22 clear: both;
23 }
24 .slideMain,.slideTab{
25 height: 80px;
26 }
27 .slideTab{
28 line-height: 74px;
29 }
30 </style>

```

**【代码说明】**

- 第1步，JavaScript代码第05~19行中，当鼠标移入元素时，播放动画，修改left值为0，就会看到菜单栏以滑动的形式滑出页面。
- 第2步，当鼠标移出元素时，播放动画，修改left值为-72。

## 7.4 树形菜单导航

在一些系统的后台分类导航中，树形菜单导航是比较常见的形式，因为它能以友好的界面方式无限地延伸、扩展菜单。例如：书籍的分类，1级分类是书籍科目，2级分类是书籍行业，3级分类是书籍专业课题等等，这种层级比较多、分类比较多的结构就可以采用树形菜单导航。本例效果如图7-4所示。

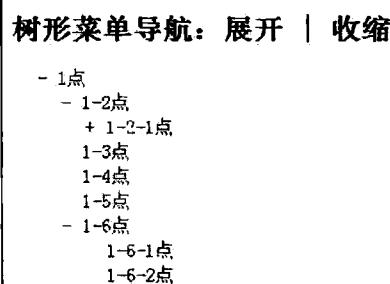


图7-4 树形菜单展示

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function getTypeElement(es, type){ //获取指定类型的节点
04 var esLen = es.length,
05 i = 0,
06 eArr = [],
07 esl = null;
08 for(; i < esLen ; i++){
09 esl = es[i];
10 if(esl.nodeName.replace("#", "").toLowerCase() == type){
11 eArr.push(esl);
12 }
13 }
14 return eArr;
15 }
16
17 function treeMenuNav(){ //树形菜单导航
18
19 var as = document.getElementsByTagName('a'), //获取所有 a 元素
20 ai = 0, //循环变量初始引导值
21 al = as.length, //a 的个数
22 ao = null; //被遍历的当前元素
23
24 for(; ai < al ; ai++){
25 ao = as[ai];
26
27 if(ao.className == "treeIcon"){ //判断是否是树形节点被单击的地方
28 ao.onclick = function(){ //绑定单击事件
29
30 var iconType = this.innerHTML, //获取展示类型
31 uls =
32 getTypeElement(this.parentNode.parentNode.childNodes, "ul"),
33 uli = 0, //元素初始值
34 ull = uls.length, //子菜单个数
35 dis = "block"; //默认显示（展开）子菜单
36
37 if(iconType == "-"){
38 this.innerHTML = "+";
39 dis = "none";
40 }
41 else{
42 this.innerHTML = "-";
43 }
44 }
45 }
46 }
47 }
48 }
49
```

```

44 for(; uli < ull ; uli++){
45 uis[uli].style.display = dis;
46 }
47 }
48 }
49 }
50 }
51 treeMenuNav();
52 }
53 </script>

```

本例 HTML 代码如下：

```

01 <h2>树形菜单导航：展开 | 收缩</h2>
02 <ul class="treeNode">
03
04 <!-- 当前项 -->
05 <div class="treeList">
06 -
07 1 点
08 </div>
09 <!-- 子菜单 -->
10 <ul class="treeNode">
11
12 <!-- 当前项 -->
13 <div class="treeList">
14 -
15 1-2 点
16 </div>
17 <!-- 子菜单 -->
18 <ul class="treeNode">
19
20 <!-- 当前项 -->
21 <div class="treeList">
22 -
23 1-2-1 点
24 </div>
25 <!-- 子菜单 -->
26 <ul class="treeNode">
27
28 <!-- 当前项 -->
29 <div class="treeList">
30
31 class="treeIconNo">
32 1-2-1-1 点
33 </div>

```

```
34 <!--子菜单-->
35
36
37
38
39
40
41
42 <!--当前项-->
43 <div class="treeList">
44
45 1-3 点
46 </div>
47
48
49 <!--当前项-->
50 <div class="treeList">
51
52 1-4 点
53 </div>
54
55
56 <!--当前项-->
57 <div class="treeList">
58
59 1-5 点
60 </div>
61
62
63 <!--当前项-->
64 <div class="treeList">
65 >-
66 1-6 点
67 </div>
68 <!--子菜单-->
69 <ul class="treeNode">
70
71 <!--当前项-->
72 <div class="treeList">
73
74 1-6-1 点
75 </div>
76
77
78 <!--当前项-->
```

```

79 <div class="treeList">
80
81 1-6-2 点
82 </div>
83
84
85
86
87
88

```

本例 CSS 代码如下：

```

01 <style>
02 a{
03 text-decoration:none;
04 }
05 /*=====树形菜单导航=====*/
06 ul{
07 list-style: none;
08 }
09 .treeNode li{
10 margin: 3px 0px 3px -20px;
11 white-space: nowrap;
12 }
13 .treeIcon{
14 border: 1px solid #ccc;
15 }
16 .treeIconNo{
17 border: 1px solid #fff;
18 }
19 </style>

```

#### 【代码说明】

- 第1步，创建树形菜单的结构UI。
- 第2步，为了遍历所有的a对象，检测哪一个属于树形菜单的单击按钮，检测到后绑定事件。
- 第3步，当单击树形菜单的触发按钮时，JavaScript代码第37~43行判断是“-”还是“+”，如果是“-”，则隐藏子元素，反之则显示子元素。

## 7.5 仿QQ菜单

QQ是我们经常使用的IM工具，里面有很多友好的交互，其中有一个功能是查看各个分类下的好友。本例就来模仿此模块，效果如图7-5所示。

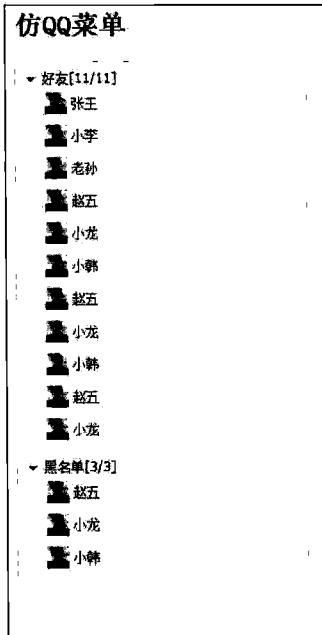


图 7-5 仿 QQ 菜单

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function getTypeElement(es, type){ //获取指定类型的节点
04 var esLen = es.length,
05 i = 0,
06 eArr = [],
07 esl = null;
08 for(; i < esLen ; i++){ //获取所有元素
09 esl = es[i];
10 if(esl.nodeName.replace("#", "").toLowerCase() == type){
11 eArr.push(esl);
12 }
13 }
14 return eArr;
15 }
16 function likeQQMenue(){ //QQ 菜单
17 var ls = document.getElementById("likeQQMenue").childNodes,
18 li = 0,
19 ll = ls.length,
20 lo = null;
21 for(li < ll ; li++){
22 lo = ls[li];
23 if(lo.className == "likeQQMenueLists"){
24 lo.onclick = function(){

```

```

25 var divs = getTypeElement(this.childNodes, "div"),
26 dis = "block",
27 classNames = divs[0].className,
28 target =
29 document.getElementById(this.getAttribute("data-targetID"));
30
31 if(classNames == "relationMenu on"){ //展开列表
32 divs[0].className = "relationMenu";
33 target.style.display = "block";
34 }
35 else{ //收缩列表
36 divs[0].className = "relationMenu on";
37 target.style.display = "none";
38 }
39 }
40 }
41 }
42 }
43 likeQQMenue();
44
45 };
46 </script>

```

本例 HTML 代码如下：

```

01 <h2>仿 QQ 菜单</h2>
02 <!--QQ 菜单最外层-->
03 <div id='likeQQMenue'>
04 <!--好友-->
05 <div class="likeQQMenueLists" data-targetID='relationList1'>
06 <div class="relationMenu"> </div>
07 <div class="relationTitle"> 好友[11/11]</div>
08 </div>
09 <!--好友-关系列表-->
10 <div class="relationList likeQQMenueLists" id='relationList1'>
11 <div class="lists">
12 <!--头像-->
13 <div class="images">
14
15 </div>
16 <!--昵称-->
17 <div class="nickname">
18 张王
19 </div>
20 </div>
21 //省略部分名单

```

```

22 <div class="lists">
23 <!--头像-->
24 <div class="images">
25
26 </div>
27 <!--昵称-->
28 <div class="nickname">
29 &nbsp小龙
30 </div>
31 </div>
32 </div>
33 <div style="clear: both"></div>
34 <!--黑名单-->
35 <div class="likeQQMenueLists" data-targetID='relationList2'>
36 <div class="relationMenu">&nbsp</div>
37 <div class="relationTitle">&nbsp黑名单[3/3]</div>
38 </div>
39 <!--好友-关系列表-->
40 <div class="relationList likeQQMenueLists" id='relationList2'>
41 <div class="lists">
42 <!--头像-->
43 <div class="images">
44
45 </div>
46 <!--昵称-->
47 <div class="nickname">
48 &nbsp赵五
49 </div>
50 </div>
51 //省略部分名单
52 </div>
53 </div>

```

本例 CSS 代码如下：

```

01 <style>
02 a{
03 text-decoration:none;
04 }
05 /*=====仿 QQ 菜单 =====start*/
06 #likeQQMenue{
07 position: relative;
08 width: 261px;
09 height: 522px;
10 border: 1px solid #9DC3E7;
11 background-color: #F5FAFD;

```

```
12 border-radius: 6px;
13 -webkit-border-radius: 6px;
14 -moz-border-radius: 6px;
15 font: 12px/1.5 tahoma,helvetica,clean,sans-serif;
16 }
17 .likeQQMenuLists{
18 width: 100%;
19 margin: 10px;
20 }
21 .relationList .lists div{
22 position: relative;
23 float: left;
24 }
25 .relationList .lists .images img{
26 width: 20px;
27 height: 20px;
28 }
29 .likeQQMenuLists .lists{
30 width: 80%;
31 position: relative;
32 left: 6%;
33 clear: both;
34 height: 30px;
35 }
36 .relationMenu{
37 background: url(..../images/icons.png) 0px 0px no-repeat;
38 zoom: 1;
39 line-height: 30px;
40 width: 10px;
41 height: 10px;
42 cursor: pointer;
43 color: #000;
44 }
45 .relationMenu.on{
46 background: url(..../images/icons.png) -16px 0px no-repeat;
47 zoom: 1;
48 line-height: 30px;
49 width: 10px;
50 height: 10px;
51 cursor: pointer;
52 color: #000;
53 }
54 .likeQQMenuLists div{
55 position: relative;
56 float: left;
```

```

57 }
58 .relationTitle{
59 top: -5px;
60 }
61 </style>

```

#### 【代码说明】

- 第 1 步，创建一个 QQ 面板，模拟好友分类与好友列表。
- 第 2 步，JavaScript 代码第 24~39 行为每个分类绑定事件，当单击分类时，判断所有分类是否展开，如果处于展开状态则收缩，如果处于收缩状态则展开。

## 7.6 漂浮广告

广告，是最常见的品牌传播形式。目前，广告是很多互联网公司的主要利润来源。因此，广告的经营及表现形式对所有互联网公司来说都是非常重要的。漂浮广告属于一种硬性广告，虽然很令普通用户讨厌，但是推广效果非常有效。本例效果如图 7-6 所示。

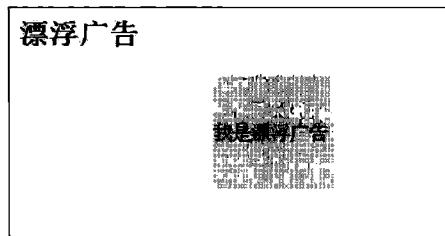


图 7-6 漂浮广告

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function floatingAd(){
04
05 var animateFloat = function(){ // 运行动画
06
07 var floatingAd = document.getElementById("floatingAd"), // 浮动的动画
08 bodyW = window.innerWidth || // 浮动的最大范围
09 document.documentElement.offsetWidth, // 浮动的最大范围修正
10 maxLeft = bodyW - 120, // 浮动的最大范围修正
11 thisLeft = parseInt(floatingAd.style.left); // 元素的 left 值
12
13 new animateManage({
14 "context": floatingAd, // 被操作的元素
15 "effect": "linear", // 持续时间
16 "time": 10000, // 元素的起始值偏移量
17 "starCss": {

```

```

18 "left":thisLeft
19 },
20 "css" :{ //元素的结束值偏移量
21 //检测是否达到最左边或最右边，开始反向移动
22 "left":thisLeft >= maxLeft ? 0 : maxLeft
23 }).init();
24 }
25
26 animateFloat();
27
28 setInterval(function(){ //更新动画
29 animateFloat();
30 }, 10100);
31 }
32 floatingAd();
33
34 };
35 </script>

```

本例 HTML 代码如下：

```

01 <h2>漂浮广告</h2>
02 <div id='floatingAd'>
03 我是漂浮广告
04 </div>

```

本例 CSS 代码如下：

```

01 <style>
02 a{
03 text-decoration:none;
04 }
05 /*=====漂浮广告=====start*/
06 #floatingAd{
07 position: absolute;
08 background-color: #ccc;
09 width: 100px;
10 line-height: 100px;
11 height: 100px;
12 }
13 </style>

```

#### 【代码说明】

- 广告的展示形式多种多样，本例中漂浮广告的关键是 JavaScript 代码第 13~23 行，它主要是通过动态修改元素的 left 值来实现的。
- 第 1 步，广告默认开始运行位置为左边，然后不断地让其向右移动。
- 第 2 步，当 left 值大于页面宽度时，修改 left 值向左移动，即目标动画位置是 0。

## 7.7 滑动展开/收缩广告

在一些网站上，经常看到有的页面弹出一个广告，过几秒后慢慢收缩直至隐形，这种广告在京东商城、天猫等电商类网站比较常见。这种广告的广告效果很吸引眼球，而且很多用户会误操作（不小心点击）进入广告引导的页面。本例效果如图 7-7 所示。

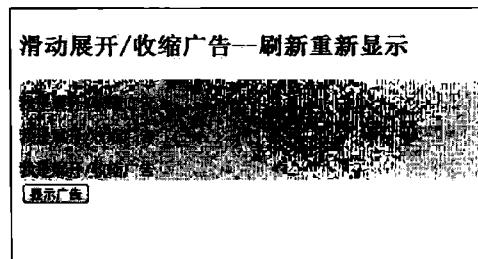


图 7-7 展开/收缩广告

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function adOpenOrShrink(e){ //滑动展开/收缩广告
04
05 new animateManage({ //滑动展开广告
06 //被操作的元素
07 "context" : e,
08 "effect":"linear",
09 "time": 1000, //持续时间
10 "startCss":{ //元素的起始值偏移量
11 "height":0
12 },
13 "css" :{ //元素的结束值偏移量
14 "height":210
15 },
16 "callback":function(){ //滑动收缩广告
17 new animateManage({ //被操作的元素
18 "context" : e,
19 "effect":"linear",
20 "time": 1000, //持续时间
21 "startCss":{ //元素的起始值偏移量
22 "height":210
23 },
24 "css" :{ //元素的结束值偏移量
25 "height":0
26 }
27 }).init();
28 }
29 }
30 }
31 }
32 }
33
```

```

29 }).init();
30 }
31 //滑动展开/收缩广告
32 document.getElementById("updateAdOpenOrShrink").onclick = function(){
33 adOpenOrShrink(document.getElementById("adOpenOrShrink"));
34 }
35
36 };
37 </script>

```

本例 HTML 代码如下：

```

01 <h2>滑动展开/收缩广告</h2>
02
03 <div id='adOpenOrShrink'>
04 <p>我是展开/收缩广告</p>
05 <p>我是展开/收缩广告</p>
06 <p>我是展开/收缩广告</p>
07 <p>我是展开/收缩广告</p>
08 <p>我是展开/收缩广告</p>
09 <p>我是展开/收缩广告</p>
10 </div>
11 <input value='显示广告' id="updateAdOpenOrShrink" type="button" />

```

本例 CSS 代码如下：

```

01 <style>
02 a{
03 text-decoration:none;
04 }
05 /*=====滑动展开/收缩广告=====start*/
06 #adOpenOrShrink{
07 position: relative;
08 height: 0px;
09 overflow: hidden;
10 background-color: #ccc;
11 }
12 </style>

```

#### 【代码说明】

- 本例中广告的滑动效果主要是通过修改元素的 `height` 值来实现的。
- 第 1 步，JavaScript 代码第 05~29 行实现一个动画，慢慢地修改元素的 `height`，值慢慢变大，视觉上会看到元素慢慢展开。
- 第 2 步，当展开到最大值时，再逐渐减少 `height`，视觉上会看到元素慢慢收缩。

## 7.8 定时关闭的背投广告

定时关闭广告，可以很好地显示和隐藏广告。例如：打开当当网时，会有一些较大版

面的广告在首页展示，过几秒后就自动关闭了。本例效果如图 7-8 所示。



图 7-8 显示 3 秒后关闭背投广告

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function backAd(e){ //定时关闭的背投广告
04
05 var time = 5000, //间隔多久关闭
06 timeID = -1; //定时调用线程
07
08 timeID=setInterval(function(){
09
10 if(time == 0){ //如果时间等于 0，则关闭背投广告，停止调用
11 e.style.display = "none"; //隐藏元素
12 clearInterval(timeID); //关闭调用
13 }
14
15 time-=1000; //递减时间
16 e.innerHTML = (time/1000) + "秒后关闭背投广告"; //多少秒关闭广告
17 },1000)
18 }
19 document.getElementById("backAd").onclick = function(){ //定时关闭的背投广告
20 backAd(document.getElementById("backAd"));
21 }
22 };
23 </script>
```

本例 HTML 代码如下：

```

01 <h2>定时关闭的背投广告</h2>
02 <div id='backAd'>
03 单击我，5 秒后关闭背投广告
04 </div>
```

本例 CSS 代码如下：

```

01 <style>
02 a{
03 text-decoration:none;
04 }
05 /*=====滑动展开/收缩广告=====start*/
06 #adOpenOrShrink{
07 position: relative;
08 height: 0px;
09 overflow: hidden;
10 background-color: #ccc;
11 }
12 </style>
```

#### 【代码说明】

- 定时关闭广告主要是调用 `setInterval()` 定时器来（JavaScript 代码第 08~17 行）实现的。
- 第 1 步，设置间隔多久关闭。
- 第 2 步，当间隔 1 秒时，递减 1000。
- 第 3 步，当时间等于 0 时，隐藏元素。

## 7.9 页面五颜六色的雪花

如果在电脑中模拟出一片片五颜六色的雪花是不是很有趣呢？冬天，在网站的商品推广中加入雪花效果，更能够吸引用户的眼球。本例效果如图 7-9 所示。

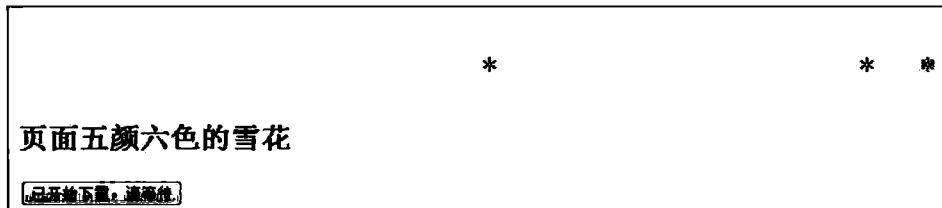


图 7-9 五颜六色的雪花

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 var isOpen = false; //雪花启动的开关
04
05 function flutterChar(){
06 if(isOpen){ //是否开启过
07 return;
08 }
09 }
```

```

10 isOpen = true;
11
12 var allChar = [],
13 maxSnowflake = 90,
14 maxleft = (document.body.clientWidth ||
15 document.documentElement.clientWidth)-100,
16 maxTop = -1,
17 i = 0,
18 snowflake = [
19 '*',
20 '*',
21 '',
22 '*',
23 '@',
24 '*'
25],
26 snowflakeColor = [//颜色库
27 "red",
28 "green",
29 "#ccc123",
30 "#345232",
31 "#231111",
32 "#ab2322"
33],
34 s = 0, //计数雪花—类型与颜色
35
36 createCharr = function(){ //创建雪花
37 var d = document.createElement("div");
38 s++;
39 s = s > 5 ? 0 : s;
40 d.innerHTML = snowflake[s]; //填充值
41 //设置雪花的 left 值
42 d.style.left = Math.round(Math.random()*maxleft+0) + "px";
43 //设置雪花的 top 值
44 d.style.top = (-1 * Math.round(Math.random()*100+0)) + "px";
45 d.style.position = "absolute"; //绝对定位
46 d.style.zIndex = "999"; //Z 轴设置
47 d.style.color = snowflakeColor[s]; //设置颜色
48 //随机雪花速度
49 d.setAttribute("data-v", Math.round(Math.random()*5+2));
50 d.setAttribute("data-time", "0"); //雪花的漂浮时间
51 document.body.appendChild(d);
52 return d; //返回雪花对象
53 },
54

```

```

55 moveChar = function(e){ //移动雪花
56 maxTop = document.body.scrollHeight-50;
57 var l = parseInt(e.style.left, 10),
58 t = parseInt(e.style.top, 10),
59 v = parseInt(e.getAttribute("data-v"), 10), //当时速度
60 time = parseInt(e.getAttribute("data-time"), 10), //时间
61 _time = time + 50,
62 _l = l + v,
63 //落体位移的路程，加入修正值 v
64 _t = 0.5 * 9 * _time * _time * 0.001 * 0.001 * v,
65 top = _t >= maxTop ? 0 : _t,
66 _time = _t >= maxTop ? 0 : _time;
67
68 e.style.top = top + "px";
69 e.style.left = (_l >= maxleft ? 0 : _l) + "px";
70 e.setAttribute("data-time", _time);
71 };
72
73
74
75 var createS = setInterval(function(){ //雪花移动
76 //创建很多雪花
77 var length = allChar.length,
78 l = length + 10;
79 for(; i < l; i++){
80 allChar.push(createCharr());
81 }
82 if(allChar.length > maxSnowflake){
83 clearInterval(createS);
84 }
85 }, 1000)
86
87 setInterval(function(){ //雪花移动
88 var ll = allChar.length;
89 for(i = 0 ; i < ll ; i++){
90 moveChar(allChar[i]);
91 }
92 }, 50)
93 }
94 //页面五颜六色的雪花
95 document.getElementById("startFlutterCharacter").onclick = function(){
96 this.value = "已开始下雪，请等待";
97 flutterChar();
98 }
99

```

```
100 ...};
101 </script>
```

本例 html 代码如下：

```
01 <h2>页面五颜六色的雪花</h2>
```

```
02 <input id='startFlutterCharacter' value="开始" type="button"/>
```

**【代码说明】**

- 第 1 步，单击开启下雪按钮，首先判断是否单击过下雪按钮，如果单击过，则禁止继续单击，如果没单击过，允许单击。
- 第 2 步，初始化一些基本的数据，包括：雪花的最大数、用来存储待创建雪花的数据对象、雪花的类型及雪花的颜色、创建雪花的方法、移动雪花的方法。
- 第 3 步，运行创建雪花的线程，间隔 1 秒，保证视图中初始阶段有间隔不断的雪花飘落下来。创建雪花的过程，主要是设置雪花的颜色、位置、随机地修正速度值。
- 第 4 步，启动雪花运行的线程，间隔 50 毫秒，这样就有比较好的动画效果，当然设置其他值也可以。`moveChar()` 函数主要是计算下一次的目标坐标值，其中重力匀速运动需要注意一下是怎么计算的，参考 JavaScript 代码第 64 行可以回味一下重力位移公式 “ $s = 1/2 * g * t * t$ ”（ $s$  位移、 $g$  重力加速度、 $t$  时间）。

# 第 8 章 移动开发常用代码

随着移动浪潮的到来，编程语言也在波动。目前移动的市场状态是：设备参差不齐、平台版本种类繁多。为了适应不同设备的要求，在移动端还要做许多工作，任重道远。当今移动互联网的发展趋势已经达到了一个井喷态势，布局移动端是现今很多企业、公司的重要战略方向。JavaScript 不仅能开发 PC 端的网页应用，也能开发智能手机上的网页应用。

本章主要涉及的知识点有如下。

- 区分移动设备平台。
- 检测设备的方向。
- 让 Web 应用近似本地 APP：移除浏览器地址。
- 使用移动地图：百度地图。

## 8.1 区分平台类型

目前主流的设备平台是 iOS、Android、Windows Phone。截至 2014 年 6 月份，移动设备全球市场占有率 Android 排第一，iOS 排第二，Windows Phone 排第三。Windows Phone 设备在至少 24 个国家中领先苹果 iPhone 智能手机，这些国家包括智利、哥伦比亚、捷克共和国、埃及、厄瓜多尔、芬兰、希腊、匈牙利、印度、意大利等，因此区分平台也将 Windows Phone 考虑进来。本例效果如图 8-1 所示。

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
 <head></head>
 <body>
 <script type="text/javascript">...</script>
 </body>
</html>
```

图 8-1 显示平台类型

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 /*
04 * 区分平台类型
05 * 在主体元素中添加 iPhone、Android 等标示符
06
```

```

06 * */
07
08 function mobPlatformType(){
09
10 var platformType = "others", //平台类型模式是其他
11 platforms = [//一些主流设备类型库
12 "iphone",
13 "ipod",
14 "ipad",
15 "android",
16 "windows phone os"
17],
18 reg = null, //正则匹配的设备字符
19 userAgent = navigator.userAgent.toLowerCase(),//设备字符串
20 i = 0, //初始值
21 l = platforms.length, //设备的数量上限
22 p = ""; //当前设备
23
24 for(; i < l;i++){
25 p = platforms[i];
26 reg = new RegExp(p, "i"); //正则设备匹配
27 if (userAgent.match(reg)) //正则判断是否为 iPhone
28 {
29 platformType = p;
30 }
31 }
32
33 document.body.setAttribute("mob-platformType", platformType);
34 }
35 mobPlatformType();
36 };
37 </script>

```

**【代码说明】**

- 第 1 步，增加设备类型库，为检测设备类型提供依据。
- 第 2 步，JavaScript 代码第 19 行利用 `navigator.userAgent` 获取用户代理头的值，其中 `userAgent` 是只读属性，声明了浏览器 HTTP 请求的用户代理头。
- 第 3 步，遍历设备类型库，并判断遍历的当前设备是否与代理头匹配，如果匹配，将平台类型添加到 `body` 属性中，如果没有匹配的类型，就设置为 `others`。

## 8.2 判断设备方向变更

当今的移动设备为了适应人们在各种应用场景下的需要，增加了屏幕旋转的功能。例

如：有的人竖着拿手机、有的人横着拿手机，程序会自动检测设备的方向，并根据设备的不同方向来显示不同的页面效果。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 /*
04 * 方向变更捕获
05 */
06
07 var orientationCall = [], //方向改变事件中待执行的函数队
08 orientation = (function(){ //移动设备方向改变事件
09 var o = ""; //待遍历的对象键
10 //检测浏览器的方向改变
11 window.addEventListener("orientationchange", function(event) {
12 for(o in orientationCall){
13 orientationCall[i](); //运行待执行的函数
14 }
15 }, false);
16 })();
17
18 var
19 addOrientation = function(callFun){ //增加方向变换的回调队列
20 orientationCall[orientationCall.length] = callFun; //推入回调
21 },
22
23 /*
24 *区分移动设备屏幕的竖/横 屏
25 */
26 orientation = "vertical", //默认垂直 (vertical) , 横向 (Horizontal)
27 isOrientation = (typeof window.orientation == "number" && typeof
28 window.onorientationchange == "object"),//检测是否支持 window.orientation
29
30 getOrientation = function(){ //获取设备的方向
31 if(this.isOrientation){ //如果支持 window.orientation
32 //0 表示竖屏, 正负 90 表示横屏 (向左与向右) 模式
33 orientation = window.orientation == 0 ? "vertical" : "Horizontal";
34 }
35 else{
36 this.orientation = window.innerWidth > window.innerHeight ? "Horizontal" :
37 "vertical"; //根据高度与宽度判断是横屏还是竖屏
38 }
39 //为 body 添加判断方向属性
40 document.body.setAttribute("mob-orientation", this.orientation);
41 },

```

```

42 updateorientation = function () { //更新屏幕的方向值
43 getOrientation(); //更新方向
44 addOrientation(getOrientation()); //方向变更时，更新屏幕方向数值
45 };
46 };
47 </script>

```

#### 【代码说明】

- 针对移动设备，HTML 5 新增了一个事件 `orientationchange`，当设备方向发生变更时触发该事件，参见 JavaScript 代码第 11~15 行。
- 本例主要的思想是，将所有的变更事件统一管理，变更事件触发回调队列。
- 在一些设备浏览器中，`window.orientation` 接口可以获取设备的方向，因此检测浏览器是否支持 `window.orientation`，如果支持，则采用此 API 提供的数值判断方向：0 表示竖屏，正负 90 表示横屏（向左与向右）模式；如果不支持，则通过判断宽、高的相对值来区分横屏与竖屏。

**说明：**因为方向变更会处理一堆的业务逻辑，所以将所有待处理的函数，都存放在一个队列中，发生方向变更时就处理这些在队列中的函数。

## 8.3 移除移动浏览器地址栏

在移动端，有时候需要让 Web APP 与 Native App 达到一个近似的融合。这个需求提出了很多新的挑战，首先就是如何隐藏地址栏，地址栏属于浏览器的行为，因此只能利用浏览器的特性去实现，但是就目前的形式来看，实现不会特别完美，因为 Web APP 总是会受制于浏览器。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03
04 var orientationCall = [], //方向改变事件中待执行的函数队
05
06 orientation = (function(){
07 var o = ""; //移动设备方向改变事件
08 //检测浏览器的方向改变
09 window.addEventListener("orientationchange", function(event) {
10 for(o in orientationCall){ //遍历的对象键
11 orientationCall[o](); //运行待执行的函数
12 }
13 }, false);
14
15 })();
16 function addOrientation(callFun){ //增加方向变换的回调队列

```

```

17 orientationCall[orientationCall.length] = callFun;//推入回调
18 }
19 /*
20 *移除浏览器地址栏
21 */
22
23 function clearUrl(){
24
25 /*
26 * 页面内容太多，超过屏幕高度时，会自动隐藏地址栏，
27 * 目前网络上流行的解决方案是将当前页面的屏幕向上滚动，
28 * 造成地址栏超出视野范围，就不会看到地址栏了。
29 */
30
31 var scroll = function(){
32 window.scrollTo(0, 1); //将屏幕滚动到指定的位置
33 };
34 scroll();//页面载入时运行
35 addOrientation(scroll); //方向变更时修改屏幕滚轴
36 }
37 };
38 </script>

```

**【代码说明】**

- 现在大部分浏览器的页面都有一个特殊的行为，当内容超过设备的高度时，在向下移动高度时会自动隐藏地址栏，可以利用这个特点，隐藏地址栏。
- 核心代码是第 32 行的“`window.scrollTo(0, 1)`”，当页面加载完或方向变更时修正这个值。

## 8.4 防止网页触摸滚动

有时候，为了禁止用户的一些操作，需要屏蔽 touch（触摸）事件。例如：初次访问网站时，为了帮助用户更好地了解网站的功能，新手教程中，有执行教学引导的需求，需禁止用户的一些触摸行为，强制让用户看完网站说明之类的文章。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 /*防止网页触摸滚动*/
04 function notouchmove(event) {
05 document.body.ontouchmove = function(event){}; //在 body 中增加触摸事件
06 //该函数将通知浏览器不要执行与事件关联的默认动作
07 event.preventDefault();

```

```

08 }
09 }
10 notouchmove();
11 };
12 </script>

```

#### 【代码说明】

- 第 05~08 行主要展示了 touchmove 的屏蔽方式，其他的事件 touchstart、touchend 也可以用类似的方法屏蔽。
- 第 07 行的 preventDefault() 函数，可以取消事件的默认动作。

## 8.5 使用 JavaScript 调用百度地图

在当今的移动和 Web 领域，很多新颖的应用层出不穷。其中 LBS（基于位置服务 Location Based Service）是众多火热应用之一。

既然谈到了位置，那么地图就是不可缺少的基本条件，这也是阿里买下高德地图的原因所在。在这个业务领域，发展比较早的是“Google 地图”，但是由于后来 Google 相关事业部撤离北京，Google 的好多业务就不太稳定了，有时候不能访问，有时候失效。因此，本节向读者介绍一个后起之秀——“百度地图”：<http://developer.baidu.com/map/>。另外，对“Google 地图”感兴趣的读者可以参考“Google 地图”的 API 官网：<https://developers.google.com/maps/>。

“百度地图”针对 JavaScript 开放了一套应用程序层的接口，可以帮助第三方开发者构建交互性强、功能丰富的产品，而且为了适应最新的发展趋势，它引入了对 HTML 5 的支持。

成本方面，API 免费对外开放，“百度地图”采用了两种合作方式，任何非盈利性的应用都可以直接使用，商业应用则会有相关限制。

百度地图 API 从 1.5 版本起，要申请“密钥”才可使用。因此第一步便是申请“密钥”，“百度地图”的“密钥”申请步骤相对比较简单，只要按照网站的引导即可申请成功。

在成功申请完“密钥”之后，开始构建“百度地图”，效果如图 8-2 所示。



图 8-2 百度地图

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 /*
04 使用 JavaScript 代码调用百度地图
05 */
06 var
07 baiduMap = null, //百度地图对象
08 intBaiduMap = function(m){
09 baiduMap = new BMap.Map(m); //创建 BMap 实例
10 var point = new BMap.Point(116.404, 39.915); //创建点坐标
11 baiduMap.centerAndZoom(point,15); //初始化地图，设置中心点坐标和地图级别
12 baiduMap.enableScrollWheelZoom(); //启用滚轮放大缩小，默认禁用
13 baiduMap.disableDragging(); //禁用地图拖拽
14 };
15 intBaiduMap("baiduMap");
16 };
17 </script>
```

本例 HTML 代码如下：

```

01 <h2>使用 JavaScript 代码调用百度地图</h2>
02 <div id="baiduMap"></div>
```

本例 CSS 代码如下：

```

01 <style type="text/css">
02 /*=====使用 JavaScript 代码调用百度地图=====*/
03 #baiduMap {
04 width: 500px;
05 height: 500px;
06 overflow: hidden;
07 margin:0;
08 }
09 </style>
```

#### 【代码说明】

- 第1步，在index.html文件的head内引入“百度地图”的JavaScript文件，并且加上申请的key，代码如下：

```
<script type="text/javascript" src="http://api.map.baidu.com/api?v=2.0&ak=D874c49a728d*****">
</script>
```

- 第2步，实例化“百度地图”对象，用于以后对“百度地图”的操作，参见JavaScript代码第9行。

- 第3步，初始化地图对象API，参见JavaScript代码第10~11行。

- 第4步，根据实际需要，定制化一些地图模块，代码如下：

```
baiduMap.enableScrollWheelZoom(); //启用滚轮放大缩小，默认禁用
baiduMap.disableDragging(); //禁用地图拖曳
```

提示：“百度地图”的其他 API 请参考地址：<http://developer.baidu.com/map/reference/index.php?title=Class:%E6%80%BB%E7%B1%BB/%E6%A0%B8%E5%BF%83%E7%B1%BB>。

## 8.6 获取当前地理坐标

HTML 5 引入了一个新的 API，可以通过浏览器获取位置坐标。这个新 API 可以更好地服务于 LBS 类的移动需求，例如基于位置的社交、本地餐饮。本例效果如图 8-3 所示。

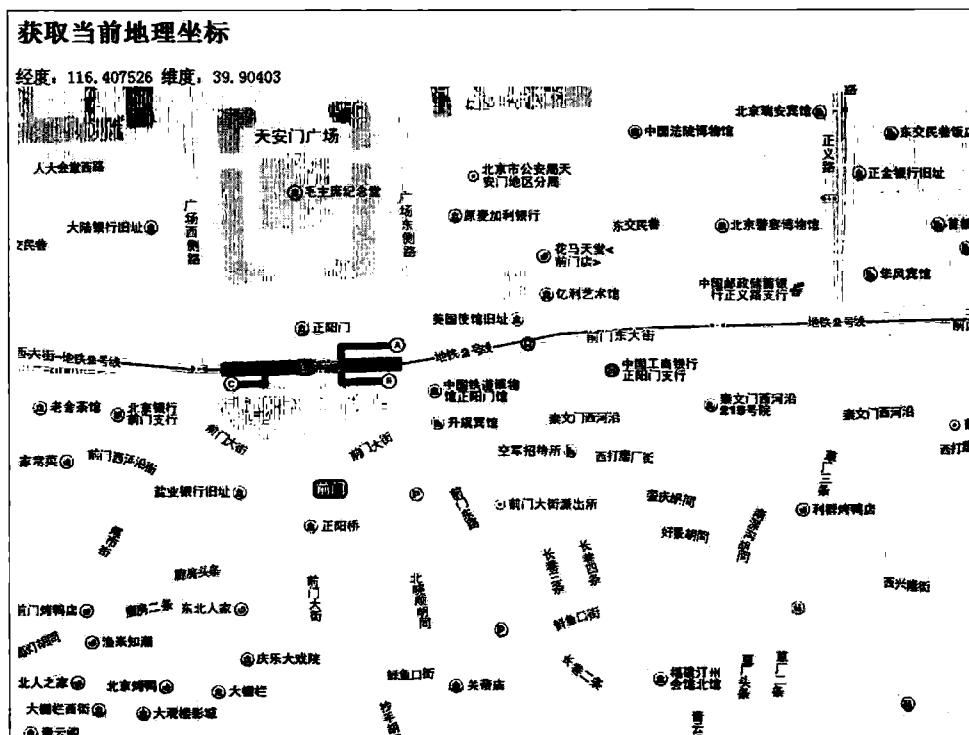


图 8-3 获取经度与维度后定位的百度地图

本例 JavaScript 代码如下：

```

12 latitude = coords.latitude; //维度
13 document.getElementById("geolocationMap").innerHTML =
14 经度: " + longitude + " 纬度: " + latitude;
15
16 if(success){ //成功后的回调
17 success(longitude, latitude);
18 }
19 },
20 function(error){ //失败后的回调
21 var msg = "";
22 switch (error.code){
23 case 1:
24 msg = "用户拒绝了位置服务";
25 break;
26 case 2:
27 msg = "获取不到位置信息";
28 break;
29 case 3:
30 msg = "获取信息超时错误";
31 break;
32 default :
33 msg = "code: " + error.code + " msg: " +
34 error.message
35 }
36 //显示错误信息
37 document.getElementById("geolocationMap").innerHTML =
38 msg;
39 },
40 null); //getCurrentPosition(成功回调, 错误回调, 更多选项)
41 } else {
42 document.getElementById("geolocationMap").innerHTML = "获取当前地理
43 坐标失败";
44 }
45
46 }
47 /*=====获取当前地理坐标=====*/
48 geolocation(function(longitude, latitude){
49 var viewMap = new BMap.Map("viewMap"); //创建 BMap 实例
50 //初始化地图,设置中心点坐标和地图级别。
51 viewMap.centerAndZoom(new BMap.Point(longitude, latitude), 17);
52 viewMap.enableScrollWheelZoom(); //启用滚轮放大缩小, 默认禁用
53 });
54 };
55 };
56 </script>

```

本例 JavaScript 代码如下：

```
01 <h2>获取当前地理坐标</h2>
02 <div id='geolocationMap'>正在获取地理位置 . . . </div>
03 <div id='viewMap'></div>
```

本例 CSS 代码如下：

```
01 <style type="text/css">
02 /*===== 获取当前地理坐标 =====*/
03 #viewMap{
04 width: 900px;
05 height: 900px;
06 margin:0;
07 }
08 </style>
```

#### 【代码说明】

- geolocation 接口可以通过浏览器获取用户端的经度与维度，但是这个接口目前尚不稳定。
- 第 1 步，代码第 06 行检测是否存在 geolocation，如果存在则调用此接口的 getCurrentPosition() 获取位置，不存在则提示获取位置失败。
- 第 2 步，支持 geolocation 接口的浏览器也会有调用失败的时候，目前提供了几种错误码：“1”代表用户拒绝服务、“2”获取不到位置信息、“3”获取信息超时、“4”其他位置的错误。
- 第 3 步，成功获取位置之后，调用百度地图的 API 定位地图位置。

## 8.7 判断当前浏览器是否为移动浏览器

因为移动设备屏幕大小有限，在移动设备上显示的网页效果应该和在 PC 端显示的效果不一致。要让程序知道该显示哪一套方案，就得能让程序判断当前浏览器是不是移动浏览器。

本例 JavaScript 代码如下：

```
01 var userAgent = navigator.userAgent; //获取 user agent 信息
02 if(userAgent.indexOf('AppleWebKit') > -1){ //判断是否为移动浏览器
03 alert('您使用的是移动浏览器');
04 }else{
05 alert('您使用的是普通浏览器');
06 }
```

#### 【代码说明】

浏览器和客户端的信息都保存在第 01 行的 navigator.userAgent 里，通过对 userAgent 进行关键词的匹配，就知道用户当前正在使用什么浏览器了。

# 第 9 章 其他常用代码

除前面介绍的 JavaScript 代码外，还有一些比较零碎但又不得不介绍的常用代码段，如跨浏览器的 Ajax、解析 XML 和 JSON 数据等。

本章主要涉及的知识点有：

- Ajax 技术的一些日常使用，包括解析 XML、解析 JSON、跨浏览器请求。
- HTML 5 的一些富媒体应用，例如播放 MP3。
- 其他一些有趣的效果，例如百度自动完成下拉显示、星级投票效果等。

## 9.1 网页图片较多时分批次加载图片

前端页面中，加载耗时最长的当属图片了，如果页面中图片很多，将会浪费很长的加载时间，用户可不是很有耐心。最好的解决方案是用户需要查看图片的时候再单独加载图片。本节简单介绍一下图片分批加载的基本原理，效果如图 9-1 与图 9-2 所示。



图 9-1 加载图片（前）

网页图片较多时，分批次加载图片

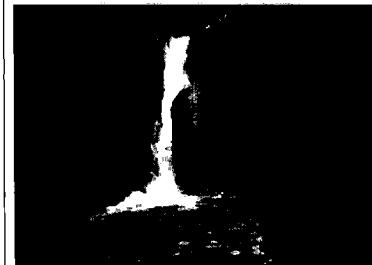


图 9-2 加载图片（后）

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 /*网页图片较多时，分批次加载图片*/
04
05 var preImg =
06 "data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAAAAEAAAABCAYAAAAfFcSJAA
07 AAAXNSR0lArs4c6QAAAARnQU1BAACxjwv8YQUAAAJcEhZcwAADsQAAA7EAZUrDhsA
```

```

08 AAANSURBVhXYzh8+PB/AAffA0nNPuCLAAAAEIFTkSuQmCC";//预设图片
09 var needLoadImgs = (function(loads){
10 /*
11 * 1.待加载元素的集合
12 * 2.待加载的元素，必须采用替换或其他方法停止图片加载
13 * 3.加载图片的触发条件，容器在可视区域内
14 * 4.加载图片的影响因素，窗体滚动、大小改变时
15 * 5.重复以上动作，直到所有图片加载完毕为止
16 */
17
18 var needLoadImgs = function(){
19 var that = this;
20 this._imgs = [];
21 this.initImgs = function(){
22 var l = document.images.length,
23 i = 0,
24 imgs = document.images,
25 _i = null;
26
27 if(l > 0){ //是否启用事件
28 for(;i < l ; i++){
29 _i = imgs[i];
30 this._imgs[i] = _i; //待载入的图片数组
31 //检测是否存在损坏的图片
32 if (_i.src === undefined || _i.src === false) {
33 _i.src = preImg;
34 }
35 }
36 that.updateImg();
37 that.initEvent();
38 }
39 }
40 }
41 }
42
43 needLoadImgs.prototype.initEvent = function(){ //绑定事件检测
44 var that = this;
45 window.onscroll = function(){ //滚轴事件
46 that.updateImg();
47 }
48 }
49
50 needLoadImgs.prototype.updateImg = function(){
51 var i = 0,
52 l = this._imgs.length,

```

```

53 imgs = this._imgs,
54 cimg = "",
55 _i = null,
56 windowHeight=document.all ?
57 document.getElementsByTagName("html")[0].offsetHeight : window.innerHeight,
58 scrollY = document.documentElement.scrollTop ||
59 document.body.scrollTop; //滚动条距离顶部高度
60 for(i<1 ; i++) {
61 _i = imgs[i];
62 cimg = _i.getAttribute("data-source");
63
64 if (cimg){ //判断图片是否在可视区域，如果在可视区域就加载
65 if(getAbsoluteTop(_i)+_i.offsetHeight/4 < windowHeight + scrollY){
66 _i.src = cimg; //设置源图片地址
67 imgs.splice(i, 1); //删除已经设置过图片的对象
68 i = this._imgs.length;
69 }
70 }
71 }
72 }
73 return new needLoadImgs(loads);
74 });
75
76 function getAbsoluteTop(_e){ //获取元素的绝对 top
77 var _top = _e.offsetTop,
78 _current = _e.offsetParent;
79 while (_current != null){
80 _top += _current.offsetTop;
81 _current = _current.offsetParent;
82 }
83 return _top;
84 }
85
86 /*=====图片分批加载=====*/
87 needLoadImgs.initImgs();
88 };
89 </script>

```

本例 HTML 代码如下：

```

01 <h2>网页图片较多时，分批次加载图片—移动至底部看到待加载的图片</h2>
02

03

04

```

【代码说明】

- 分批加载图片又名延迟加载，主要的流程是：①JavaScript 代码第 24 行获取所有待加载的图片集合；②默认采取替换或不加载的方式减少 http 加载图片；③是图片的触发条件，一般是通过“滚轴”与“窗体大小改变”事件（scroll/resize）；④在 JavaScript 代码第 43~48 行中，检测事件触发判断条件，判断是否进入可视区域内，如果进入可视区域就载入图片。
- 判断图片是否进入可视区域的方法，主要是判断待加载图片元素的绝对高度与本身高度的“1/4”之和，是否小于窗口的高度与滚轴的高度之和。

## 9.2 使用 JavaScript 解析 XML 数据

XML 是一种用来存储数据的可扩展标记语言，之所以要学习使用 JavaScript 来解析 XML 数据，是因为 XML 数据非常适合网络传输，它提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。本例效果如图 9-3 所示。



图 9-3 XML 解析结果（控制台打印）

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 /*=====JavaScript 解析 XML 数据 start=====*/
04 var xmlStr = "<bookstore>" +
05 "<book>" +
06 "<title>Hello</title>" +
07 "<author>J. K. Rowling</author>" +
08 "<year>2005</year>" +
09 "</book>" +
10 "</bookstore>" +
11 /*=====JavaScript 解析 XML 数据 end=====*/
12 parseXML = function(xmlData){ //解析函数
13 var
14 /*
15 * 初始化数据
16 */
17 xml = null, //默认为 null
18 }
19 }

```

```

20 domP;
21 if (!xmlData || typeof xmlData !== "string") {
22 return xml;
23 }
24 try {
25 /*
26 * 检测浏览器支持哪种 XML 解析方法
27 * DOMParser 对象解析 XML 文本并返回一个 XML Document 对象,
28 * 浏览器支持: Firefox、Mozilla、Opera 等
29 * 目前在全球市场份额中, IE 的整体占有率在降低, 因此将这个方法放在
30 * 第一检测位置
31 */
32
33 if (window.DOMParser) {
34 domP = new DOMParser(); //构建加载对象
35 //执行解析, 加载文件
36 xml = domP.parseFromString(xmlData , "text/xml");
37 } else { //IE 支持的方法
38 //构建 IE 浏览器的 xml 加载对象
39 xml = new ActiveXObject("Microsoft.XMLDOM");
40 //关闭异步加载可确保在文档完整加载之前, 解析器不会继续执行脚本
41 xml.async = "false";
42 xml.loadXML(xmlData); //执行解析, 加载文件
43 }
44 } catch(e) {
45 xml = "Parse failure"; //解析失败, 返回提示性信息
46 }
47 if (!xml || !xml.documentElement ||
48 xml.getElementsByTagName("parsererror").length) {
49 new Error("损坏的 XML: " + xmlData);
50 }
51 return xml;
52 };
53
54 console.log(parseXML(xmlStr));
55 };
56 </script>

```

### 【代码说明】

- 某些浏览器会有内置的 XML 解析函数, JavaScript 代码第 33 行首先检测是否支持 `DOMParser()` 函数, 如果不支持则调用 IE 浏览器的 “`ActiveXObject("Microsoft.XMLDOM")`”。
- 如果以上两种方法都不支持, 则提示出错信息。

### 9.3 使用 JavaScript 解析 JSON 数据

JSON 是存储和交换文本信息的语法，它的优势是比 XML 更小、更快、更容易解析。因为 JSON 语法是 JavaScript 语法的子集，因此使用 JavaScript 解析 JSON 数据会更容易。JSON 解析效果如图 9-4 结果所示。

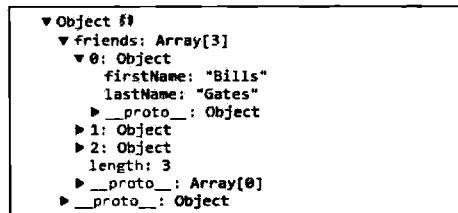


图 9-4 JSON 解析结果（控制台打印）

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 /*=====JavaScript 解析 JSON 数据 start=====*/
04 var jsonStr = '{ "friends": [+
05 { "firstName": "Bills", "lastName": "Gates" }, +
06 { "firstName": "George", "lastName": "Bush" }, +
07 { "firstName": "Thomas", "lastName": "Carter" }] }';
08 }
09 //JavaScript 解析 JSON 数据
10 //
11
12 trim = function(chars){ //去除字符串左右两边的空格
13 return (chars || "").replace(/\u202f|\u00a0|+\u202f|\u00a0+$/g, "");
14 },
15
16 parseJSON = function(jsonData){ //解析函数
17
18 if (typeof jsonData === 'object') { //判断是否为对象
19 return jsonData; //直接返回对象
20 }
21 //如果存在原生的 JSON 解析 API，则使用原生的解析 API
22 if (window.JSON && window.JSON.parse) {
23 return window.JSON.parse(jsonData); //解析 JSON 字符
24 }
25
26 if (typeof jsonData === "string"){
27 jsonData = trim(jsonData); //简单的过滤字符，保证前后没有空格
28
29 if (jsonData){

```

```

30 //利用 Function 的特性，构造 JSON 对象
31 return (new Function("return " + jsonData))();
32 }
33 }
34 }
35 console.log(parseJSON(jsonStr));
36 }
37 </script>

```

#### 【代码说明】

- 第 1 步，检测传输的参数是否为对象，如果是对象则直接返回。
- 第 2 步，JavaScript 代码第 29~32 行检测 jsonData 是否为字符串，如果是字符串则简单的过滤，保证没有空字符，然后利用函数的特殊性质，直接 new 一个字符对象。

## 9.4 跨浏览器的 Ajax

Ajax 与服务器交换数据，不会重新加载整个页面，而是创建动态的网页。在 2005 年，Google 通过 Google Suggest 让 Ajax 流行起来，这个服务会将用户在谷歌搜索框中输入的字符传送给服务器，然后接收服务器返回的搜索提示供用户采纳。本例演示这个请求和返回的过程，效果如图 9-5 所示。

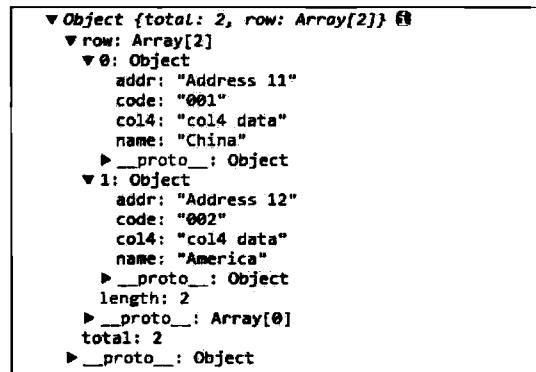


图 9-5 ajax.php 返回的数据

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03
04 /*
05 JavaScript 解析 JSON 数据
06 */
07
08 function trim(chars){
09 return (chars || "").replace(/^\s|\u00A0)+|(\s|\u00A0)+$/g, "");

```

```

10 }
11
12 function parseJSON(jsonData){ //解析函数
13 if (typeof jsonData === 'object') { //判断是否为对象
14 return jsonData; //直接返回对象
15 }
16 //如果存在原生的 JSON 解析 API，则使用原生的解析 API
17 if (window.JSON && window.JSON.parse) {
18 return window.JSON.parse(jsonData); //解析 JSON 字符
19 }
20
21 if (typeof jsonData === "string") {
22 jsonData = this.trim(jsonData); //简单的过滤字符，保证前后没有空格
23 if (jsonData) { //如果不是空字符串
24 //利用 Function 的特性，构造 JSON 对象
25 return (new Function("return " + jsonData))();
26 }
27 }
28 }
29
30 /*
31 * 夸浏览器的 Ajax (JavaScript 实现无刷新 Ajax)
32 */
33
34 function ajax(options){
35
36 if(!options || !options.url){ //检测是否存在请求的 URL
37 return false; //Ajax 调用失败
38 }
39
40 /*
41 * 数据初始化
42 */
43
44 options.data = options.data || ""; //待传送的值
45 options.method = (options.method || "GET").toUpperCase(); //类型默认是 GET
46 options.async = options.async || true; //异步 (true) 或同步 (false)
47
48 /*
49 * 响应类型，如果请求的是 XML 文件，则默认类型为 XML，否则默认为 JSON，
50 * 目前支持 3 种： JSON、XML、TEXT
51 */
52 options.responseType = options.responseType || (/xml/.test(options.url) ? "xml" :
53 "json");
54 options.successCall = options.successCall || false; //成功回调

```

```

55 options.failureCall = options.failureCall || false; //失败回调
56
57
58 var xmlhttp;
59
60 if (window.XMLHttpRequest)// code for IE7+, Firefox, Chrome, Opera, Safari
61 {
62 xmlhttp=new XMLHttpRequest();
63 }
64 Else //IE 6、IE 5
65 {
66 xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
67 }
68 xmlhttp.onreadystatechange=function()
69 {
70 if (xmlhttp.readyState == 4 && xmlhttp.status == 200) //成功回调
71 {
72 if(options.successCall){
73 options.successCall(getResponseData(xmlhttp,
74 options.responseType));
75 }
76 }
77
78 if(xmlhttp.readyState == 4 && xmlhttp.status != 200){ //失败回调
79 if(options.failureCall){
80 //参数 1—xml 响应对象，参数 2—状态码
81 options.failureCall(xmlhttp, xmlhttp.status);
82 }
83 }
84 }
85
86
87 xmlhttp.open(options.method, options.url + (options.method == "GET" ? "?" +
88 options.data : ""), options.async);
89 if(options.method != "GET" && options.data){
90 xmlhttp.send(options.data);
91 }else{
92 xmlhttp.send();
93 }
94 return true; //Ajax 调用成功
95 }
96
97 function getResponseData(xmlhttp, type){ //解析响应的 Ajax 数据
98 var resData = xmlhttp.responseText; //获得字符串形式的响应数据
99 //默认值为 json，现在 JSON 格式的流行度优于其他格式，将此放在第一位

```

```

100 if(type === "json"){
101 return parseJSON(resData);
102 }
103 if(type === "xml"){
104 return xmlhttp.responseXML; //获得 XML 形式的响应数据
105 }
106 if(type === "text"){
107 return resData;
108 }
109 }
110 /*-----跨浏览器的 Ajax-----*/
111 ajax({
112 "url": ".JAPI/ajax.php",
113 "successCall":function(msg){ //成功回调
114 console.log(msg);
115 },
116 "failureCall":function(xmlRes, resCode){//失败回调
117 }
118 },
119);
120 });
121 });
122 </script>

```

本例 PHP 代码如下：

```

01 <?php
02 $jarr=array("total"=>2,'row'=>array(
03 array('code'=>'001','name'=>'China','addr'=>'Address 11','col4'=>'col4 data'),
04 array('code'=>'002','name'=>'America','addr'=>'Address 12','col4'=>'col4
05 'data')
06)
07);
08 echo json_encode($jarr); //将数组进行 json 编码

```

#### 【代码说明】

- 第 1 步，所有现代的浏览器都支持 XMLHttpRequest 对象，不过 IE 5~IE 6 使用内置的 API ActiveXObject。因此首先尝试创建 XMLHttpRequest 对象，如果存于此对象则直接使用此对象，否则创建 ActiveXObject 对象。
- 第 2 步，根据传递的参数来采用不同的数据请求方式 GET 或 POST。
- 第 3 步，调用前几节的 API 解析返回服务器的数据，包括 JSON、XML、TEXT 3 种格式。
- 第 4 步，跨浏览器有跨域的问题，通过后台 php 脚本读取跨域的数据，然后返回给 JavaScript 处理。

## 9.5 使用 Ajax 轻松加载文件

Ajax 不但可以实现一些 API 的数据请求，还可以加载一些文件中的数据，这样就可以将一些固定的数据存放在文件中，减少接到数据请求就必须查找数据库的网络负担和服务器负担。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03
04 /*
05 JavaScript 解析 JSON 数据
06 */
07
08 function trim(chars){ //去除字符串左右两边的空格
09 return (chars || "").replace(/^(\\s|\\u00A0)+(\\s|\\u00A0)+$/g, "");
10 }
11
12 function parseJSON(jsonData){ //解析函数
13 if (typeof jsonData === 'object') { //判断是否为对象
14 return jsonData; //直接返回对象
15 }
16 //如果存在原生的 JSON 解析 API，则使用原生的解析 API
17 if (window.JSON && window.JSON.parse) {
18 return window.JSON.parse(jsonData); //解析 JSON 字符
19 }
20
21 if (typeof jsonData === "string") {
22 jsonData = this.trim(jsonData); //简单的过滤字符，保证前后没有空格
23 if (jsonData) { //如果不是空字符
24 //利用 Function 的特性，构造 JSON 对象
25 return (new Function("return " + jsonData))();
26 }
27 }
28 }
29
30 /*
31 *跨浏览器的 Ajax (JavaScript 实现无刷新 Ajax)
32 */
33
34 function ajax(options){
35 if(!options || !options.url){ //检测是否存在请求的 URL
36 return false; //Ajax 调用失败
37 }

```

```

38
39 /*
40 * 数据初始化
41 */
42
43 options.data = options.data || ""; //待传送的值
44 //传递类型， 默认是 GET
45 options.method = (options.method || "GET").toUpperCase();
46 options.async = options.async || true; //异步 (true) 或同步 (false)
47
48 /*
49 *响应类型,如果请求的是 XML 文件，则默认类型为 XML，否则默认为 JSON,
50 *目前支持 3 种： TEXT、XML、JSON
51 */
52 options.responseType = options.responseType || (/xml/.test(options.url) ? "xml" :
53 "json");
54 options.successCall = options.successCall || false;//成功回调
55 options.failureCall = options.failureCall || false; //失败回调
56
57
58 var xmlhttp;
59
60 if (window.XMLHttpRequest)// code for IE7+, Firefox, Chrome, Opera, Safari
61 {
62 xmlhttp=new XMLHttpRequest();
63 }
64 else//IE6, IE5
65 {
66 xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
67 }
68 xmlhttp.onreadystatechange=function()
69 {
70 if (xmlhttp.readyState == 4 && xmlhttp.status == 200) //成功回调
71 {
72 if(options.successCall){
73 options.successCall(getResponseData(xmlhttp,
74 options.responseType));
75 }
76 }
77
78 if(xmlhttp.readyState == 4 && xmlhttp.status != 200){ //失败回调
79 if(options.failureCall){
80 //参数 1---xml 响应对象，参数 2---状态码
81 options.failureCall(xmlhttp, xmlhttp.status);
82 }

```

```

83 }
84 }
85 }
86
87 xmlhttp.open(options.method, options.url + (options.method == "GET" ? "?" +
88 options.data : ""), options.async);
89 if(options.method != "GET" && options.data){
90 xmlhttp.send(options.data);
91 }else{
92 xmlhttp.send();
93 }
94 return true; //Ajax 调用成功
95 }
96
97 function getResponseData(xmlhttp, type){ //解析响应的 Ajax 数据
98 var resData = xmlhttp.responseText; //获得字符串形式的响应数据
99 /*因为默认值为 json，而且现在 JSON 格式的流行度优于其他格式，
100 因此将此放在第一位*/
101 if(type === "json"){
102 return parseJSON(resData);
103 }
104
105 if(type === "xml"){
106 return xmlhttp.responseXML; //获得 XML 形式的响应数据
107 }
108
109 if(type === "text"){
110 return resData;
111 }
112 }
113 /*=====使用 Ajax 轻松加载文件略=====*/
114
115 ajax({//加载 xml 文件
116 "url": "../ajax.xml",
117 "successCall":function(msg){ //成功回调
118 console.log(msg);
119 },
120 "failureCall":function(xmlRes, resCode){//失败回调
121
122 }
123 })
124
125 ajax({ //加载 JSON 文件
126 "url": "../ajax.json",
127 "successCall":function(msg){ //成功回调

```

```

128 console.log(msg);
129 },
130 "failureCall":function(xmlRes, resCode){//失败回调
131 }
132 })
133 });
134 };
135 </script>

```

本例 ajax.xml 文件的 XML 数据如下：

```

01 <?xml version="1.0" encoding="ISO-8859-1"?>
02 <CATALOG>
03 <WEB>
04 <TITLE>QingJs</TITLE>
05 <ADDRESS>http://www.qingjs.com</ADDRESS>
06 </WEB>
07 </CATALOG>

```

本例 ajax.JSON 文件的 JSON 数据如下：

```

01 {
02 "language":{
03 "json1":"Chinese",
04 "json2":"English"
05 }
06 }

```

#### 【代码说明】

- 使用 Ajax 可以加载一些文件，如 XML、JSON 文件等。
- JavaScript 代码第 115~133 行就是利用了 Ajax 的特性，加载了 XML、JSON 文件内的数据，方便程序使用。

## 9.6 跨浏览器读取 XML

跨浏览器读取 XML，主要涉及一个跨域的问题。跨域可以采取直接或迂回的策略，本例主要是采取迂回策略，效果如图 9-6 所示。

```

▼ #document
 ▼ <CATALOG>
 ▼ <WEB>
 <TITLE>hello:QingJs</TITLE>
 <ADDRESS>http://www.qingjs.com</ADDRESS>
 </WEB>
 </CATALOG>

```

图 9-6 跨域 XML 返回的数据

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){

```

```

03 /*
04 JavaScript 解析 JSON 数据
05 */
06
07 function trim(chars){ //去除字符串左右两边的空格
08 return (chars || "").replace(/^\s|\u00A0)+|(\s|\u00A0)+$/g, "");
09 }
10
11 function parseJSON(jsonData){ //解析函数
12
13 if (typeof jsonData === 'object') { //判断是否为对象
14 return jsonData; //直接返回对象
15 }
16 //如果存在原生的 JSON 解析 API，则使用原生的解析 API
17 if (window.JSON && window.JSON.parse) {
18 return window.JSON.parse(jsonData);//解析 JSON 字符
19 }
20
21 if (typeof jsonData === "string"){
22 jsonData = this.trim(jsonData); //简单的过滤字符，保证前后没有空格
23
24 if (jsonData) { //如果不是空字符
25 //利用 Function 的特性，构造 JSON 对象
26 return (new Function("return " + jsonData))();
27 }
28 }
29 }
30
31 /*
32 *跨浏览器的 Ajax (JavaScript 实现无刷新 Ajax)
33 */
34
35 function ajax(options){
36 if(!options || !options.url){ //检测是否存在请求的 URL
37 return false; //Ajax 调用失败
38 }
39
40 /*
41 * 数据初始化
42 */
43
44 options.data = options.data || ""; //待传送的值
45 //传送类型，默认是 GET
46 options.method = (options.method || "GET").toUpperCase();
47 options.async = options.async || true; //异步 (true) 或同步 (false)

```

```
48
49 /*
50 *响应类型,如果请求的是 XML 文件, 则默认类型为 XML, 否则默认为 JSON,
51 *目前支持 3 种: TEXT、XML、JSON
52 */
53 options.responseType = options.responseType || (/xml/.test(options.url) ? "xml" :
54 "json");
55 options.successCall = options.successCall || false; //成功回调
56 options.failureCall = options.failureCall || false; //失败回调
57
58 var xmlhttp;
59
60 if (window.XMLHttpRequest)// code for IE7+, Firefox, Chrome, Opera, Safari
61 {
62 xmlhttp=new XMLHttpRequest();
63 }
64 else//IE6, IE5
65 {
66 xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
67 }
68 xmlhttp.onreadystatechange=function()
69 {
70 if (xmlhttp.readyState == 4 && xmlhttp.status == 200) //成功回调
71 {
72 if(options.successCall){
73 options.successCall(getResponseData(xmlhttp,
74 options.responseType));
75 }
76 }
77
78 if(xmlhttp.readyState == 4 && xmlhttp.status != 200){ //失败回调
79 if(options.failureCall){
80 //参数 1--xml 响应对象, 参数 2--状态码
81 options.failureCall(xmlhttp, xmlhttp.status);
82 }
83 }
84 }
85
86 xmlhttp.open(options.method, options.url + (options.method == "GET" ? "?" +
87 options.data : ""), options.async);
88 if(options.method != "GET" && options.data){
89 xmlhttp.send(options.data);
90 }else{
91 xmlhttp.send();
92 }
```

```

93 return true; //Ajax 调用成功
94 }
95
96 function getResponseData(xmlhttp, type){ //解析响应的 Ajax 数据
97 var resData = xmlhttp.responseText; //获得字符串形式的响应数据
98 /*因为默认值为 json, 而且现在 JSON 格式的流行度优于其他格式,
99 因此将此放在第一位*/
100 if(type === "json"){
101 return parseJSON(resData);
102 }
103
104 if(type === "xml"){ //获得 XML 形式的响应数据
105 return xmlhttp.responseXML;
106 }
107
108 if(type === "text"){
109 return resData;
110 }
111 }
112 /*跨浏览器的 XML 读取代码—解决跨域问题*/
113 function getReqxml(urlXml, callFun){
114 ajax({ //加载 XML 文件
115 "url" : "../../API/reqxml.php",
116 "data" : "url=" + urlXml,
117
118 "successCall" : function(msg){ //成功回调
119 if(callFun){
120 callFun({
121 "status":"ok", //请求成功状态码
122 "data":msg //成功的数据
123 });
124 }
125 },
126
127 "failureCall" : function(xmlRes, resCode){ //失败回调
128 if(callFun){
129 callFun({
130 "status":"error", //请求失败状态码
131 "data":"请求失败" //失败的数据
132 });
133 }
134 }
135 })
136 }
137 /*=====跨浏览器读取 XML 代码=====*/

```

```

138 getReqxml("http://1.qingjs.sinaapp.com/ajax.xml", function(xmlData){
139 console.log(xmlData.data);
140 });
141 };
142 </script>

```

#### 【代码说明】

- 前面已经提到，跨浏览器读取文件主要是面临跨域的问题，本例利用后台脚本语言获取待读取的文件，将获取的文件数据响应给前端的 Ajax。
- 这个迂回策略，不但可以请求 XML，还可以请求 JSON 等形式的跨域。

## 9.7 HTML 5 版 JavaScript 实现的 MP3 播放器

自 HTML 5 兴起之日，W3C 便对外公布了大量的 API，方便开发者使用。其中，HTML 5 对富媒体提供了比较友好的支持，可以方便地制作音乐播放器或视频播放器。本例效果如图 9-7 所示。

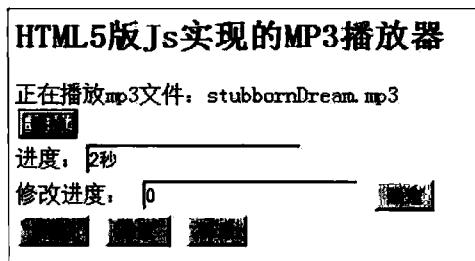


图 9-7 MP3 播放器

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(X
03 /*
04 * "HTML 5 版 JavaScript 实现的 MP3 播放器
05 */
06 var playerMp3 = function(option){
07 var playerMp3 = function(option) // MP3 播放器
08 /*
09 this.audio = option.audio;
10 this.audioAction = option.audioAction;
11 this.process = option.process;
12 this.setProcessO = option.setProcess;
13 this.setProcessOK = option.setProcessOK;
14 this.volup = option.volup;
15 this.voldown = option.voldown;
16 this.muted = option.muted;

```

```
17
18 var self = this;
19
20 this.audioAction.onclick = function(){ //播放控制
21 if(this.value == "播放"){
22 self.start();
23 this.value = "暂停";
24 }else{
25 self.pause();
26 this.value = "播放";
27 }
28 }
29
30 this.volup.onclick = function(){ //音量增大
31 self.setVolup();
32 }
33
34 this.voldown.onclick = function(){ //音量减小
35 self.setVoldown();
36 }
37
38 this.muted.onclick = function(){ //静音、发声
39 self.setMute();
40 }
41
42 setInterval(function(){ //获取播放进度
43 self.getProcess();
44 },1000)
45
46 this.setProcessOK.onclick = function(){ //确定修改进度
47 self.setProcess();
48 }
49
50 }
51
52 playerMp3.prototype.start = function(){ //开始播放
53 this.audio.play();
54 }
55
56
57 playerMp3.prototype.pause = function(){ //暂停播放
58 this.audio.pause();
59 }
60
61 playerMp3.prototype.getProcess = function(){ //获取播放进度
```

```

62 this.process.value = Math.floor(this.audio.currentTime) + "秒";
63 }
64
65 playerMp3.prototype.setProcess = function(){//设置播放进度
66 this.audio.currentTime = (this.setProcessO.value || 0);
67 }
68
69 playerMp3.prototype.setVolup = function() //音量+
70 var v = this.audio.volume + 0.1;
71
72 this.audio.volume = (v > 1 ? 1 : v);
73 }
74
75 playerMp3.prototype.setVoldown = function(){ //音量-
76 var v = this.audio.volume - 0.1;
77
78 this.audio.volume = (v < 0 ? 0 : v);
79 }
80
81 playerMp3.prototype.setMute = function(){ //静音/发声
82 this.audio.muted = !this.audio.muted;
83 this.audio.muted ? (this.muted.value = "发声") : (this.muted.value = "静音");
84 }
85
86
87 return new playerMp3(mp3); //实例化播放器
88 }
89 /*=====HTML 5 版 JavaScript 实现的 MP3 播放器=====*/
90 playerMp3({
91 "audio":document.getElementById("audio"), //音频对象
92 "audioAction":document.getElementById("audioAction"),//音频控制播放或暂停
93 "process":document.getElementById("process"), //播放进度
94 "setProcess":document.getElementById("setProcess"), //修改播放进度
95 "setProcessOK":document.getElementById("setProcessOK"),//确定修改进度
96 "volup":document.getElementById("volup"), //音量增大
97 "voldown":document.getElementById("voldown"), //音量减小
98 "muted":document.getElementById("muted") //静音
99 })
100 };
101 </script>

```

本例 HTML 代码如下：

```

01 <div><h2>HTML 5 版 JavaScript 实现的 MP3 播放器</h2></div>
02 <style>
03 audio{

```

```

04 display: none;
05 }
06 </style>
07 <div>
08 <div>正在播放 mp3 文件: stubbornDream.mp3</div>
09 <div><input id="audioAction" type="button" value="播放"/></div>
10 <div>进度: <input id="process" value="0"></div>
11 <div>
12 修改进度:
13 <input id="setProcess" type="text" value="0">
14 <input id="setProcessOK" type="button" value="确定">
15 </div>
16 <div>
17 <input type="button" id="volup" value="音量+">>
18 <input type="button" id="voldown" value="音量-">>
19 <input type="button" id="muted" value="静音">>
20 </div>
21 </div>
22 <audio id="audio" src="../stubbornDream.mp3" controls="controls">
23 </audio>

```

### 【代码说明】

- HTML 5 增加了两个富媒体 API: video、audio。本例主要应用了 audio 的音频 API。
- 在页面中，增加音频属性 audio 标签，用于播放音频，由于不同的浏览器支持不同的编码格式，本例采用基于 Chrome 浏览器的 MP3 格式的音频编码案例。
- audio 不仅支持音频播放，还提供了一些强大的事件驱动 API，例如：addTextTrack() 可以向音频/视频添加新的文本轨道；canPlayType() 检测浏览器是否能播放指定的音频/视频类型，具体 API 可以参考 [http://www.w3school.com.cn/tags/html\\_ref\\_audio\\_video\\_dom.asp](http://www.w3school.com.cn/tags/html_ref_audio_video_dom.asp)。
- 本例主要是封装了 audio 的一些 API，利用 API 与 UI 的单击事件结合，响应不同的播放事件需求，如增大音量、减小音量、播放音频、暂停音频、快进到指定的进度等。

## 9.8 Ajax 实现动态导航

有一些导航是动态的，隔段时间会根据用户对菜单导航的偏好程度，产生不同的导航结果，Ajax 可以很好地实现这个效果。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(X
03 /*
04 JavaScript 解析 JSON 数据
05 */

```

```

06
07 function trim(chars){ //去除字符串左右两边的空格
08 return (chars || "").replace(/^(\\s|\\u00A0)+|(\\s|\\u00A0)+$/g, " ");
09 }
10
11 function parseJSON(jsonData){ //解析函数
12
13 if (typeof jsonData === 'object') { //判断是否为对象
14 return jsonData; //直接返回对象
15 }
16 //如果存在原生的 JSON 解析 API，则使用原生的解析 API
17 if (window.JSON && window.JSON.parse) {
18 return window.JSON.parse(jsonData); //解析 JSON 字符
19 }
20
21 if (typeof jsonData === "string") {
22 jsonData = trim(jsonData); //简单的过滤字符，保证前后没有空格
23 if (jsonData) //如果不是空字符
24 //利用 Function 的特性，构造 JSON 对象
25 return (new Function("return " + jsonData))();
26 }
27 }
28 }
29
30 /*
31 * 夺浏览器的 Ajax (JavaScript 实现无刷新 Ajax)
32 */
33
34 function ajax(options){
35 if(!options || !options.url){ //检测是否存在请求的 URL
36 return false; //Ajax 调用失败
37 }
38
39 /*
40 * 数据初始化
41 */
42
43 options.data = options.data || ""; //待传递的值
44 //传递类型，默认是 GET
45 options.method = (options.method || "GET").toUpperCase();
46 options.async = options.async || true; //异步 (true) 或同步 (false)
47
48 /*
49 * 响应类型,如果请求的是 XML 文件，则默认类型为 XML，否则默认为 JSON,
50 * 目前支持 3 种：TEXT、XML、JSON

```

```

51 */
52 options.responseType = options.responseType || (/xml/.test(options.url) ? "xml" :
53 "json");
54 options.successCall = options.successCall || false; //成功回调
55 options.failureCall = options.failureCall || false; //失败回调
56
57 var xmlhttp;
58 if (window.XMLHttpRequest)// code for IE7+, Firefox, Chrome, Opera, Safari
59 {
60 xmlhttp=new XMLHttpRequest();
61 }
62 else //IE6、IE5
63 {
64 xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");
65 }
66 xmlhttp.onreadystatechange=function()
67 {
68 if (xmlhttp.readyState == 4 && xmlhttp.status == 200) //成功回调
69 {
70 if(options.successCall){
71 options.successCall(getResponseData(xmlhttp,
72 options.responseType));
73 }
74 }
75
76 if(xmlhttp.readyState == 4 && xmlhttp.status != 200){ //失败回调
77 if(options.failureCall){
78 //参数 1—xml 响应对象，参数 2—状态码
79 options.failureCall(xmlhttp, xmlhttp.status);
80 }
81 }
82 }
83
84
85 xmlhttp.open(options.method, options.url + (options.method == "GET" ? "?" +
86 options.data : ""), options.async);
87 if(options.method != "GET" && options.data){
88 xmlhttp.send(options.data);
89 }else{
90 xmlhttp.send();
91 }
92 return true; //ajax 调用成功
93 }
94
95 function getResponseData(xmlhttp, type){ //解析响应的 ajax 数据

```

```

96
97 var resData = xmlhttp.responseText; //获得字符串形式的响应数据
98 /*因为默认值为 json, 而且现在 JSON 格式的流行度优于其他格式,
99 因此将此放在第一位*/
100 if(type === "json"){
101 return parseJSON(resData);
102 }
103
104 if(type === "xml"){
105 return xmlhttp.responseXML; //获得 XML 形式的数据
106 }
107
108 if(type === "text"){
109 return resData;
110 }
111}
112/*
113 Ajax 实现导航
114 */
115function getAjaxNavigation (){
116 var createNav = function(data){ //创建菜单导航列表
117 var nav = document.getElementById("ajaxNavigation"), //获取对象
118 navHtml = '<h2>Ajax 实现导航</h2>', //菜单字符
119 l = data.length, //数据长度
120 i = 0, //开始值
121 d = null; //遍历的当前值
122
123 for(; i < l ; i++){
124 d = data[i];//当前数据
125 navHtml += '<div>' + d.name
126 +'</div>'; //累加菜单
127 }
128
129 nav.innerHTML = navHtml;
130 }
131}
132
133 ajax({ //加载 JSON 文件
134 "url": "../navigation.json",
135 "successCall":function(msg){ //成功回调
136 createNav(msg.menu);
137 },
138 "failureCall":function(xmlRes, resCode){ //失败回调
139
140 }
}

```

```

141 })
142 }
143 getAjaxNavigation();
144 };
145 </script>

```

本例 HTML 代码如下：

```

01 <h2>Ajax 实现导航</h2>
02 <div id='ajaxNavigation'></div>

```

本例 CSS 代码如下：

```

01 <style type="text/css">
02 #ajaxNavigation{
03 position: relative;
04 top:0px;
05 left: 0px;
06 background-color: #ccc;
07 width: 100%;
08 z-index: 1000;
09 }
10 #ajaxNavigation div{
11 position: relative;
12 float: left;
13 border: #cc2123 1px solid;
14 width: 19%;
15 height: 30px;
16 text-align: center;
17 line-height: 30px;
18 }
19 </style>

```

#### 【代码说明】

- 第 1 步，获取数据。JavaScript 代码第 133~141 行通过 Ajax 的方式请求“navigation.json”文件，获取数据。
- 第 2 步，解析数据。获取数据后，遍历数据，解析为按一定规律显示的 UI。
- 第 3 步，显示数据。将待显示的数据 UI，显示到 ID 为 ajaxNavigation 的 DOM 上。

## 9.9 类似百度的自动完成功能

早期最火的 Ajax 应用是谷歌的搜索自动完成功能，后来百度添加了类似功能，可以很好地辅助用户搜索想要输入的关键词。本例模拟自动完成的基本原理，不采用 Ajax 技术，效果如图 9-8 所示。

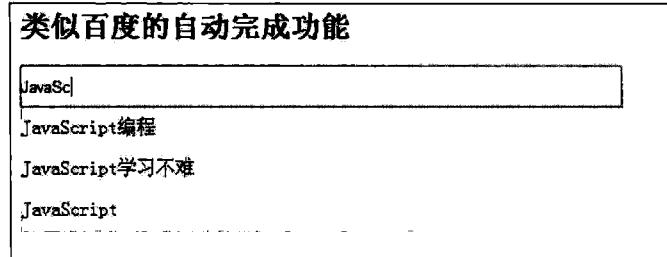


图 9-8 自动填充的数据

本例 JavaScript 代码如下：

```
01 <script type="text/javascript">
02 window.onload = function(){
03 var _lexicon = [] //默认联想词库
04 _lexicon.push("美女");
05 _lexicon.push("美眼美女");
06 _lexicon.push("Qingjs");
07 _lexicon.push("火影忍者");
08 _lexicon.push("新浪微博");
09 _lexicon.push("腾讯微博");
10 _lexicon.push("QQ 控件");
11 _lexicon.push("中国强");
12 _lexicon.push("12·阿卡丽");
13 _lexicon.push("888 网上");
14 _lexicon.push("88 发财");
15 _lexicon.push("JavaScript 编程");
16 _lexicon.push("JavaScript 教程");
17 _lexicon.push("JavaScript 学习不难");
18 _lexicon.push("HTML 5 很好");
19 _lexicon.push("HTML 5 学习");
20 _lexicon.push("英雄联盟")
21];
22 var thread = -1; //超时填充线程
23 function createAutoHtml(autoCompleteList, str){
24 var _html = "";
25 if(str.length > 0){
26 _lexicon = _lexicon,
27 regStr = null,
28 n = 0,
29 v = "";
30 }
31 regStr = new RegExp("(" + str + ")", "g"); //匹配合适的联想词
32 }
33 }
```

```

35 for(i in _lexicon){ //检索匹配词
36 if(n >= 3){
37 break; //限制最多3个联想词
38 }
39 v = _lexicon[i];
40
41 if(regStr.test(v)){
42 n++;
43 _html += "<div onclick='addContent(this)'>" + v + "</div>";
44 }
45 }
46
47 if(!_html){ //如果不存在联想词，隐藏联想层
48 autoCompleteList.style.display = "none";
49 return false;
50 }else{ //存在联想词，显示联想层
51 autoCompleteList.innerHTML = _html;
52 autoCompleteList.style.display = "block";
53 return true;
54 }
55 }
56 }
57
58 function getTypeElement(es, type){ //获取指定类型的节点
59 var esLen = es.length,
60 i = 0,
61 eArr = [],
62 esl = null;
63 for(; i < esLen ; i++){
64 esl = es[i];
65 if(esl.nodeName.replace("#", "").toLowerCase() == type){
66 eArr.push(esl);
67 }
68 }
69 return eArr;
70 }
71
72 function autoComplete(){
73 console.log(lexicon)
74
75 var autoComplete = document.getElementById("autoComplete"),
76 autoCompleteList = null,
77 str = "";
78
79 var createAuto = function(_this){

```

```

80 clearTimeout(thread); //清除定时
81 str = _this.value;
82 str = str.replace(/[|\`|——|!|@|@|, ||=|+|\|\||.|。|‘|’|%^|^&|(|)|(|)|……
83 |#|#|¥|$|?|、|【|】|“|”|“|“|?|“|、|“|{|}|:|；|“|“|{|}|《|》|“|“|+|<|>|~|~|`|`|×|“|“|
84 “|” |/g, ""); //过滤特殊字符，暂时不考虑检索
85
86 if(!str){
87 return;
88 }
89 thread = setTimeout(function(){ //如果有 1 秒钟的停留，则填充词库
90 if(str.length > 2 && !lexicon[str]){
91 lexicon[str] = str; //填充词库
92 }
93 }, 500);
94
95 autoCompleteList = document.getElementById("autoCompleteList");
96
97 /*
98 *构建联想词，不存在联想词则不执行事件绑定
99 */
100 if(!createAutoHtml(autoCompleteList, str)){
101 return;
102 }
103
104
105 /*
106 *绑定事件
107 */
108
109 var autoCompleteLists = getTypeElement(autoCompleteList.childNodes,
110 "div"),
111 i = 0,
112 l = autoCompleteLists.length;
113
114 for(; i < l ; i++){
115 autoCompleteLists[i].onclick = function(){
116 autoComplete.value = this.innerHTML;
117 autoCompleteList.style.display = "none";
118 }
119 }
120 }
121
122 autoComplete.onkeyup = function(){
123 createAuto(this);
124 }

```

```

125
126 autoComplete.onfocus = function(){
127 createAuto(this);
128 }
129 }
130 autoComplete();
131 };
132 </script>

```

本例 HTML 代码如下：

```

01 <div class="autoComplete">
02 <h2>类似百度的自动完成功能</h2>
03 <div><input type="text" id='autoComplete' /></div>
04 <div id='autoCompleteList'></div>
05 </div>

```

本例 CSS 代码如下：

```

01 <style type="text/css">
02 .autoComplete{
03 background-color: #F6F6F6;
04 width: 100%;
05 height: 200px;
06 }
07 #autoComplete{
08 width: 500px;
09 border: 0px;
10 height: 35px;
11 border: 1px solid #ccc;
12 }
13 #autoCompleteList{
14 display: none;
15 margin-top: 0px;
16 width: 500px;
17 border: 1px solid #ccc;
18 }
19 #autoCompleteList div{
20 width: 500px;
21 height: 35px;
22 line-height: 33px;
23 cursor: pointer;
24 }
25 </style>

```

#### 【代码说明】

- 本例基本构成：一个搜索框，一个自动填充框。当在搜索框中输入文字时，会触发交互事件，查询联想词库中是否有匹配的词，如果存在则显示出来。
- 在 JavaScript 代码第 89~93 行中，当用户输入符合规则的字符时停留超过“500”毫

秒，就会为词库 lexicon 填充数据，当再次搜索类似的文字时，就会显示刚才填充过的数据。

## 9.10 等级星投票效果

等级星投票效果在一些有点评功能的电子商务网站中有很大的需求。例如：影评网站中，用户就可以根据电影的等级星结果来查看某部影片的口碑，京东商城的用户可以通过星级投票的方式评价商品的好坏。本例效果如图 9-9 所示。

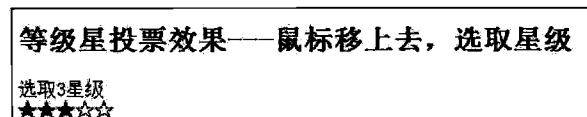


图 9-9 等级星投票效果

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 function getTypeElement(es, type){ //获取指定类型的节点
04 var esLen = es.length,
05 i = 0,
06 eArr = [],
07 esl = null;
08 for(; i < esLen ; i++){
09 esl = es[i];
10 if(esl.nodeName.replace("#", "").toLowerCase() == type){
11 eArr.push(esl);
12 }
13 }
14 return eArr;
15 };
16
17 function vote(e){ //等级星投票效果
18 var votes = getTypeElement(e.childNodes, "div"),
19 i = 0,
20 m = null,
21 k = 0,
22 n = 0,
23 allnum = 5,
24 voteed = document.getElementById("voteed");
25 for(; i < allnum ; i++){
26 m = votes[i];
27 m.setAttribute("data-num", i); //设置星星的级别
28 }
29 }
30 }
31 </script>

```

```

28 m.onmouseover = function(){ //鼠标移入
29 k = 0;
30 n = parseInt(this.getAttribute("data-num"), 10); //获取当前元素是第几个
31 voteed.innerHTML = "选取" + (n+1) + "星级";
32 for(; k < allnum; k++){
33 var v = votes[k];
34 v.innerHTML = parseInt(v.getAttribute("data-num"), 10) <= n ? "
35 ★" : "☆";
36 }
37 }
38 }
39 }
40 vote(document.getElementById("vote"));
41 };
42 </script>
```

本例 HTML 代码如下：

```

01 <h2>等级星投票效果---鼠标移上去，选取星级</h2>
02 <div id="voteed">选取：</div>
03 <div id="vote">
04 <div>☆</div>
05 <div>☆</div>
06 <div>☆</div>
07 <div>☆</div>
08 <div>☆</div>
09 </div>
```

本例 CSS 代码如下：

```

01 <style type="text/css">
02 #vote div{
03 position: relative;
04 float: left;
05 cursor: pointer;
06 }
07 </style>
```

#### 【代码说明】

- 默认将 5 颗空心星星显示在页面上。
- 在 JavaScript 代码第 32~36 行中，当鼠标在星星上移动时，自左向右开始计算，鼠标选中几颗星星，就显示几颗非空心星星。例如：选中第 3 颗星星，那么结果是 3 颗非空心星星，2 颗空心星星。

## 9.11 IE 导出表格

在日常的商业活动中，表格是比较常见的商业交流工具，因此 Web 应用也经常有将内

容导出到 Excel 的需求。在 IE 浏览器中，可以通过微软为 IE 开放的专用 API 实现将 Table 标签导出为表格的功能。

本例 JavaScript 代码如下：

```

01 <script type="text/javascript">
02 window.onload = function(){
03 /*将数据导出为 Excel*/
04 function exportExcel(tableid){
05 if (!!+[1,]) { //判断是否为 IE
06 alert("不是 IE 浏览器，不支持此方法！");
07 return;
08 }
09 var
10 curTbl = document.getElementById(tableid),
11 /*
12 * 如果 IE 浏览器报错：SCRIPT429: Automation 服务器不能创建对象
13 * 打开 Internet Explorer 单击“工具” | “选项”命令，
14 * 单击“安全”选项卡中的“自定义级别”选项卡，
15 * 将第 3 项“对没有标记为安全的 activex 控件进行初始化和脚本运行”
16 * 设置成“启用”即可
17 */
18
19 oXL = new ActiveXObject("Excel.Application"),
20 oWB = oXL.Workbooks.Add(), //创建 AX 对象 Excel
21 xlsheet = oWB.Worksheets(1), //获取 workbook 对象
22 sel = document.body.createTextRange(); //激活当前 sheet
23
24 sel.moveToElementText(curTbl); //把表格中的内容移到 TextRange 中
25 sel.select(); //全选 TextRange 中的内容
26 sel.execCommand("Copy"); //复制 TextRange 中的内容
27 xlsheet.Paste(); //粘贴到活动的 Excel 中
28 oXL.Visible = true; //设置 Excel 可见属性
29
30 try{
31 var fname = oXL.Application.GetSaveAsFilename("save.xls", "Excel
32 Spreadsheets (*.xls), *.xls");
33 }catch(error){
34 print("Nested catch caught " + error);
35 }finally{
36 if(fname){
37 oWB.SaveAs(fname);
38 oWB.Close(savechanges=false);
39
40 oXL.Quit();
41

```

```

42 oXL=null;
43 }else{
44 alert("导出失败");
45 }
46
47 window.setTimeout(function() //结束 Excel 进程，退出
48 CollectGarbage();
49 },1);
50
51 }
52 /*导出表格*/
53 document.getElementById("exportExcel").onclick = function(){
54 exportExcel("tableExcels");
55 }
56 }
57 };
58 </script>

```

本例 HTML 代码如下：

```

01 <h2>IE 导出 Excel</h2>
02 <table id="tableExcels" width="100%" border="1" cellspacing="0" cellpadding="0">
03 <tr>
04 <td colspan="5" align="center">WEB 页面导出为 EXCEL 文档的方法</td>
05 </tr>
06 <tr>
07 <td>前端语言</td>
08 <td>后端语言</td>
09 </tr>
10 <tr>
11 <td>JavaScript</td>
12 <td>PHP</td>
13 </tr>
14 <tr>
15 <td><div>CSS</div></td>
16 <td>JAVA</td>
17 </tr>
18 <tr>
19 <td>HTML</td>
20 <td>Ruby</td>
21 </tr>
22 </table>
23 <input type="button" id="exportExcel" value="导入到 EXCEL">

```

#### 【代码说明】

- IE 浏览器的 ActiveXObject 与 Windows 平台组合起来，提供了强大的访问服务功能，导出表格只是其中的一个小模块。

- 第1步，JavaScript代码第19~22行构建Excel.Application，创建Workbooks工作对象，为创建表格提供引用句柄。
- 第2步，创建并激活TextRange对象，将表格的内容复制一份进去。
- 第3步，将数据粘贴进表格Excel，另外有的IE浏览器会在安全方面限制此API，读者可以根据代码注释中的流程引导，修改设置就可以使用了。

IE设置如图9-10所示，代码运行结果如图9-11所示。

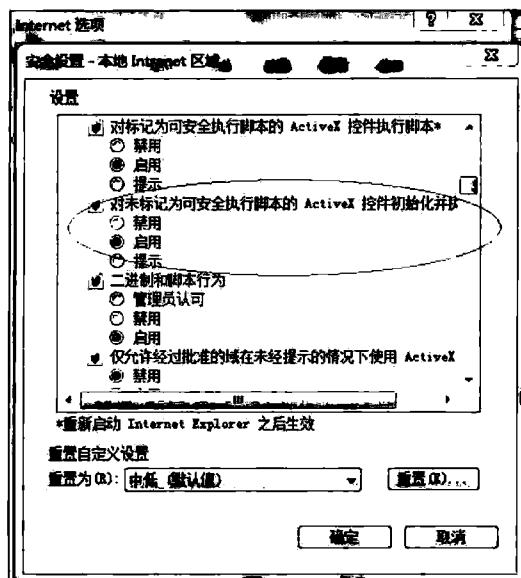


图9-10 IE浏览器设置

A1	B	C	D	E	F	G	H	I	J	K	L	M	N
1													
2													
3	JavaScript												
4	css												
5	html												
6													

图9-11 被导出的表格