本章内容：

1. 平稳退化：确保网页在没有JavaScript 的情况下也能正常工作；
2. 分离JavaScript：把网页的结构和内容与 JavaScript 脚本的动作行为分开；
3. 向后兼容性：确保老版本的浏览器不会因为你的JavaScript 脚本而死掉；
4. 性能考虑：确定脚本执行的性能最优。

平稳退化：

在访问者的浏览器不支持JavaScript 的情况下仍能顺利地浏览网站，就是说，虽然某些功能无法使用，但是最基本的操作仍能顺利完成。

JavaScript 使用 window 对象的 open() 方法来创建新的浏览窗口：

*window.open(url, name, features)*

三个参数都是可选的。

1. url：想在新窗口里打开的网页的URL地址。如果省略这个参数，屏幕上将打开一个空白的浏览器窗口；
2. name：新窗口的名字。可以在代码里通过这个名字与新窗口进行通信；
3. features：这是一个以逗号分割的字符串，其内容是新窗口的各种属性。这些属性包括：新窗口的尺寸（宽度和高度）以及新窗口被启用或者禁用的各种浏览功能（工具条、菜单条、初始显示位置， 等等）。对于这个参数应该掌握以下原则：新窗口的浏览功能要少而精。

示例：

***function popUp(winURL){***

***window.open(winURL, “popup”, “width=320,height=480”);***

***}***

“JavaScript:” 伪协议：

语法：

***<a href=”javascript:popUp(‘http://www.example.com/’);”>Example</a>***

内嵌的事件处理函数：

***<a href=”#” onclick=”popUp(‘http://www.example.com/’);return false;”>Example</a>***

“#” 符号仅供文档内部使用的标记符号。在某些浏览器里，# 链接指向当前文档的开头。把 href 属性的值设置为 # 只是为了创建一个空链接。实际工作全部由 onclick 属性负责完成。

这个技巧与 ”javascript:” 伪协议都不能平稳退化。如果用户已经金庸了浏览器的JavaScript功能，这样的链接将毫无用处。

具体到popUp() 函数，为其中的JavaScript 代码预留出退路很简单：在链接里把 href 属性设置为真实存在的URL 地址，让它成为一个有效的链接：

***<a href=”http://www.example.com/” onclick=”popUp(‘http://www.example.com/’);return false;”>Example</a>***

***<a href=”http://www.example.com/” onclick=”popUp(this.getAttribute(‘href’));return false;”>Example</a>***

***<a href=”http://www.example.com/” onclick=”popUp(this.href);return false;”>Example</a>***

href 属性现在有合法的值，即使JavaScript 被禁用，这个链接也是可用的。虽然这个链接在功能上打了折扣，但是没有彻底失效。

向CSS 学习

结构与样式的分离：

可以在外部JavaScript 文件中把一个事件添加到HTML 文档中的某个元素上：

***element.event = action...***

例：

***element.onclick = function(){***

***......***

***}***

向后兼容：

比较古老的浏览器很可能无法理解DOM 提供的方法和属性。

对象检测：

解决这个问题最简单的方法是，检测浏览器对JavaScript 的支持程度。

只要把某个方法打包在一个if 语句里，就可以根据这条if 语句的条件表达式的求值结果是true（这个方法存在）还是false（这个方法不存在）来决定应该采用怎样的行动。这种检测称为对象检测（object detection）。

***if (method){***

***statements***

***}***

一定要删掉方法名后的圆括号：

***function myFunction(){***

***if (document.getElementById){***

***statements using getERlementById***

***}***

***}***

***或***

***if (!getElementById || !getElementsByTagName) return false;***

浏览器嗅探技术：

“浏览器嗅探”指通过提取浏览器供应商提供的信息来解决向后兼容问题，但是这种技术风险非常大。

原因如下：

1. 浏览器有时会“撒谎”；
2. 为了适用于多种不同的浏览器，浏览器嗅探脚本会变得越来越复杂；
3. 许多浏览器嗅探脚本在进行这类测试时要求浏览器的版本号必须得到精确匹配。因此每当市场上出现新版本，就要修改这些脚本。

性能考虑：

尽量少访问DOM 和尽量减少标记：

访问DOM 的方式对脚本性能会产生非常大的影响：

***if (document.getElementsByTagName(“a”).length > 0){***

***var links = document.getElementsByTagName(“a”);***

***for (var i = 0; i < links.length; i++){***

***// 对每个链接做点处理***

***}***

***}***

这段代码会先取得所有<a>检查个数是否大于0，如果大于0，会再次取得所有<a>。更好的方法是：

***var links = document.getElementsByTagName(“a”);***

***if (links.length > 0){***

***for (var i = 0; i < links.length; i++){***

***// 对每个链接做点处理***

***}***

***}***

在多个函数都会取得一组类似元素的情况下，可以考虑重构代码，把搜索结果保存在一个全局变量里，或者把一组元素直接以参数形式传递给函数。

另一个需要注意的地方，就是要尽量减少文档中的标记数量。过多不必要的元素只会增加DOM树的规模，进而增加便利DOM树以查找特定元素的时间。

合并和放置脚本：

推荐将多个js 脚本文件合并为一个脚本文件，减少加载页面时发送的请求数量。

把所有<script> 标签都放到文档的末尾，</body> 标记之前，就可以让页面变得更快。即使这样，在加载脚本时，window 对象的 load 事件依然可以执行对文档进行的各种操作。

压缩脚本：

指的是把脚本文件中不必要的字节，如空格和注释，统统删除，从而达到“压缩”文件的目的。有很多工具可以完成这件事。

多数情况下，你应该有两个版本，一个是工作副本，可以修改代码并添加注释；另一个是精简副本，用于放在站点上。通常为了与非精简版本区分开，最好在精简副本的文件名中加上 min 字样：

***<script src=”scripts/scriptName.min.js”></script>***

推荐代码压缩工具：

* Douglas Crockford 的 JSMin(http://www.crockford.com/javascript/jsmin.html);
* 雅虎的 YUI Compressor (http://developer.yahoo.com/yui/compressor);
* 谷歌的 Closure Compiler (http://closure-compiler.appspot.com/home).