红宝书笔记

2012年-2019年是js蓬勃发展的七年

第1章 什么是javascript

历史回顾

网景+sun：javascript

microsoft：JScript

1997年，js1.1作为提案被提交给欧洲计算机制造商协会（Ecma），TC39委员会打造出了ECMAScript脚本语言标准

此后，各家浏览器均已ECMAScript作为自己javascript实现的依据。

ECMAScript

ECMAScript并不局限于Web浏览器，这门语言没有输入和输出之类的方法，它是作为一个基准来定义的，以便在它之上再构建更加稳健的脚本语言。

web浏览器只是ECMAScript实现可能存在的一种宿主环境host environment。宿主环境提供ECMAScript 的基准实现和与环境自身交互必需的扩展。扩展（比如 DOM）使用 ECMAScript 核心类型和语法，提供特定于环境的额外功能。其他宿主环境还有服务器端 JavaScript 平台 Node.js 和即将被淘汰 的 Adobe Flash。

DOM

DOM（document object model）是一个应用编程接口API，用于在html中使用扩展的xml，DOM将整个页面抽象成一组分层节点。HTML 或 XML 页面的每个组成部分都是一种 节点，包含不同的数据。

（XML：可扩展标记语言，被设计用来传输和存储数据，不用于表现和展示数据， html则用来表现数据）

其它DOM

除了 DOM Core 和 DOM HTML 接口，有些其他语言也发布了自己的 DOM 标准。下面列出的语言 是基于 XML 的，每一种都增加了该语言独有的 DOM 方法和接口：

 可伸缩矢量图（SVG，Scalable Vector Graphics）

 数学标记语言（MathML，Mathematical Markup Language）

 同步多媒体集成语言（SMIL，Synchronized Multimedia Integration Language）

BOM

浏览器对象模型API，用于支持访问和操作浏览器的窗口，BOM主要针对浏览器窗口和子窗口，人们通常会把任何特定于浏览器的扩展都归在BOM的范畴内。比如，下面就是这样的一些扩展：

 弹出新浏览器窗口的能力；

 移动、缩放和关闭浏览器窗口的能力；

 navigator 对象，提供关于浏览器的详尽信息；

 location 对象，提供浏览器加载页面的详尽信息；

 screen 对象，提供关于用户屏幕分辨率的详尽信息；

 performance 对象，提供浏览器内存占用、导航行为和时间统计的详尽信息；

 对 cookie 的支持；

 其他自定义对象，如 XMLHttpRequest 和 IE 的 ActiveXObject。

第2章 HTML中的JavaScript

将javascript引入网页，首先要解决的它与html的关系问题，通过反复试错和讨论，最终达成了向网页中引入通用脚本能力的共识。

<script>标签

将js插入html中的主要方法就是使用<script>标签。

该标签里面的属性：type（表示代码中脚本语言的内容类型，这个值始终都是"text/javascript"）

使用<script>的方式有两种，通过它直接在网页中嵌入javascript代码（包含在script中的代码会被从上到下解释，不能出现字符串/script，因为会被理解成为结束标志，如需使用，加上转义字符\，在script标签中的代码被计算完成之前，页面的其余内容不会被加载，也不会被显示），以及通过它在网页中包含外部js文件（必须使用src属性，这个属性的值是一个URL，它指向包含javascript代码的文件，比如：<scirpt src='example.js'></script>，与解释行内js一样，在解释外部js文件时，页面也会阻塞，阻塞时间也包含下载文件的时间）。

script标签的强大在于它可以包含来自外部域的js文件，它的src属性可以是一个完整的URL，而且这个URL指向的资源可以跟包含它的html页面不在同一个域中。

```
<script src='http://www.somewhere.com/afile.js'><script>
```

浏览器在解析这个资源时，会向 src 属性指定的路径发送一个 GET 请求，以取得相应资源，假定 是一个 JavaScript 文件。这个初始的请求不受浏览器同源策略限制，但返回并被执行的 JavaScript 则受限 制。当然，这个请求仍然受父页面 HTTP/HTTPS 协议的限制。

标签位置

通常将所有的js引用放在body元素的页面内容后面，这样一来，页面会在处理js代码之前完全渲染页面，用户会感觉页面加载更快了，因为浏览器显示空白页面的时间更短了。

推迟执行脚本

script标签有一个defer的属性，这个属性表示脚本在执行的时候不会改变页面的结构。也就是说脚本会延迟到整个页面都解析完毕后再运行。因此，再srcipt上设置defer属性，相当于告诉浏览器立即下载，但是延迟执行。

<script defer src='example.js'></script>

但是defer属性只对外部脚本文件有效，且第一个推迟执行的脚本会在第二个推迟的脚本之前执行。

异步执行脚本

async属性，与defer属性类似，他们之间的区别就是async的脚本并不保证能够按照他们出现的次序执行。

```
<!DOCTYPE html>
<html>
 <head>
 <title>Example HTML Page</title>
 <script async src="example1.js"></script>
 <script async src="example2.js"></script>
 </head>
 <body>
 <!-- 这里是页面内容 -->
 </body>
</html>
```

第二个脚本可能先于第一个脚本执行，因此，两个脚本之间最好不要有依赖关系。async属性它的作用主要是告诉浏览器，不必等脚本下载和执行完后再加载页面，同样也不必等到该异步脚本下载和执行后再加载其它脚本。

加载动态脚本

除了script标签之外，可以通过向DOM中动态添加script元素加载指定的脚本，只要创建一个script元素并将其添加到DOM即可。

```
let script = document.createElement('script');
script.src='gibberish.js';
// 动态加载脚本是以异步方式加载的，相当于添加了async属性
// 但是不是所有浏览器都支持async属性，因此，如果要统一动态脚本的加载行为
// 需要将其明确设置为同步加载
script.async = false;
document.head.appendChild(script);
```

行内代码与外部文件

最佳实践是尽可能将js代码放在外部文件中，理由如下：

1、可维护性

js代码分散到很多的html页面，会导致维护困难，因此用一个目录保存所有的js文件，则更加容易维护

2、缓存

浏览器回根据特定的设置缓存所有的外部链接的js文件，这意味着如果两个页面都用到同一个文件，则该文件只需要下载一次。

3、适应未来

文档模式

IE5.5发明了文档模式的概念，可以使用doctype切换文档模式，最初的文档模式有两种：混杂模式quirks mode和标准模式standards mode

准标准模式

<noscript>元素

针对早期浏览器不支持js的问题，需要一个页面优雅降级的处理方案。<noscript>被用于给不支持js的浏览器提供替代内容。<noscript>元素可以包含任何可以出现在body中的html元素，script除外，在下列两种情况下，浏览器将显示包含在noscript中的内容

1、浏览器不支持脚本

2、浏览器对脚本的支持被关闭

任何一个条件被满足，包含在noscript中的内容就会被渲染，否则，浏览器不会渲染noscript中的任何内容。

```
<!DOCTYPE html>
<html>
 <head>
 <title>Example HTML Page</title>
 <script defer="defer" src="example1.js"></script>
 <script defer="defer" src="example2.js"></script>
 </head>
 <body>
 <noscript>
 <p>This page requires a JavaScript-enabled browser.</p>
 </noscript>
 </body>
</html>
```

上述例子是在脚本不可用时让浏览器显示的一段话，如果浏览器支持脚本，则用户永远不会看到它。