一两个月前在淘宝内网里看到一个优化Javascript代码的竞赛，发现有不少的人对Javascript的执行和装载的基础并不懂，所以，从那天起我就想写一篇文章，但一直耽搁了。自上篇《[浏览器渲染原理简介](http://coolshell.cn/articles/9666.html)》，正好也可以承前启后。

首先，我想说一下Javascript的装载和执行。通常来说，浏览器对于Javascript的运行有两大特性：**1）载入后马上执行，2）执行时会阻塞页面后续的内容（包括页面的渲染、其它资源的下载）**。于是，如果有多个js文件被引入，那么对于浏览器来说，这些js文件被被串行地载入，并依次执行。

因为javascript可能会来操作HTML文档的DOM树，所以，浏览器一般都不会像并行下载css文件并行下载js文件，因为这是js文件的特殊性造成的。所以，如果你的javascript想操作后面的DOM元素，基本上来说，浏览器都会报错说对象找不到。因为Javascript执行时，后面的HTML被阻塞住了，DOM树时还没有后面的DOM结点。所以程序也就报错了。

**传统的方式**

所以，当你写在代码中写下如下的代码：

<script type="text/javascript"

src="http://coolshell.cn/asyncjs/alert.js"></script>

基本上来说，head里的 <script>标签会阻塞后续资源的载入以及整个页面的生成。我专门做了一个示例你可以看看：[**示例一**](http://coolshell.cn/asyncjs/async_test01.html)。 注意：我的alert.js中只有一句话：alert(“hello world”) ，这更容易让你看到javascript是怎么阻塞后面的东西的。

所以，你知道为什么有很多网站把javascript放在网页的最后面了，要么就是动用了window.onload或是docmuemt ready之类的事件。

另外，因为绝大多数的Javascript代码并不需要等页面，所以，我们异步载入的功能。那么我们怎么异步载入呢？

**document.write方式**

于是，你可能以为document.write()这种方式能够解决不阻塞的方式。你当然会觉得，document.write了的<script>标签后就可以执行后面的东西去了，这没错。对于在同一个script标签里的Javascript的代码来说，是这样的，但是对于整个页面来说，这个还是会阻塞。 下面是一段测试代码：

<script type="text/javascript" language="javascript">

function loadjs(script\_filename) {

document.write('<' + 'script language="javascript" type="text/javascript"');

document.write(' src="' + script\_filename + '">');

document.write('<'+'/script'+'>');

alert("loadjs() exit...");

}

var script = 'http://coolshell.cn/asyncjs/alert.js';

loadjs(script);

alert("loadjs() finished!");

</script>

<script type="text/javascript" language="javascript">

alert("another block");

</script>

你觉得alert的顺序是什么？你可以在不同的浏览器里试一试。这里的想关的测试页面：[**示例二**](http://coolshell.cn/asyncjs/async_test02.html)。

**script的defer和async属性**

IE自从IE6就支持defer标签，如：

<script defer type="text/javascript" src="./alert.js" >

</script>

对于IE来说，这个标签会让IE并行下载js文件，并且把其执行hold到了整个DOM装载完毕（DOMContentLoaded），多个defer的<script>在执行时也会按照其出现的顺序来运行。最重要的是<script>被加上defer后，其不会阻塞后续DOM的的渲染。但是因为这个defer只是IE专用，所以一般用得比较少。

而我们标准的的HTML5也加入了一个异步载入javascript的属性：async，无论你对它赋什么样的值，只要它出现，它就开始异步加载js文件。但是， async的异步加载会有一个比较严重的问题，那就是它忠实地践行着“载入后马上执行”这条军规，所以，虽然它并不阻塞页面的渲染，但是你也无法控制他执行的次序和时机。你可以[看看这个示例去感受一下](http://coolshell.cn/asyncjs/async_test01.async.html)。

支持 async标签的浏览器是：Firefox3.6+，Chrome 8.0+，Safari 5.0+，IE 10+，Opera还不支持（[来自这里](http://caniuse.com/#feat=script-async)）所以这个方法也不是太好。因为并不是所有的浏览器你都能行。

**动态创建DOM方式**

这种方式可能是用得最多的了。

function loadjs(script\_filename) {

var script = document.createElement('script');

script.setAttribute('type', 'text/javascript');

script.setAttribute('src', script\_filename);

script.setAttribute('id', 'coolshell\_script\_id');

script\_id = document.getElementById('coolshell\_script\_id');

if(script\_id){

document.getElementsByTagName('head')[0].removeChild(script\_id);

}

document.getElementsByTagName('head')[0].appendChild(script);

}

var script = 'http://coolshell.cn/asyncjs/alert.js';

loadjs(script);

这个方式几乎成了标准的异步载入js文件的方式，这个方式的演示请参看：[**示例三**](http://coolshell.cn/asyncjs/async_test03.html)。这方式还被玩出了JSONP的东东，也就是我可以为script的src指定某个后台的脚本（如PHP），而这个PHP返回一个javascript函数，其参数是一个json的字符串，返回来调用我们的预先定义好的javascript的函数。你可以看一下这个示例：[t.js](http://coolshell.cn/t.js) （这个示例是我之前在微博征集的[一个异步ajax调用的小例子](http://coolshell.cn/t.html)）

**按需异步载入js**

上面那个DOM方式的例子解决了异步载入Javascript的问题，但是没有解决我们想让他按我们指定的时机运行的问题。所以，我们只需要把上面那个DOM方式绑到某个事件上来就可以了。

比如：

**绑在window.load事件上**——[**示例四**](http://coolshell.cn/asyncjs/async_test04.html)

你一定要比较一下示例四和示例三在执行上有什么不同，我在这两个示例中都专门用了个代码高亮的javascript，看看那个代码高亮的的脚本的执行和我的alert.js的执行的情况，你就知道不同了）

window.load = loadjs("http://coolshell.cn/asyncjs/alert.js")

**绑在特定的事件上**——[**示例五**](http://coolshell.cn/asyncjs/async_test05.html)

<p style="cursor: pointer" onclick="LoadJS()">Click to load alert.js </p>

这个示例很简单了。当你点击某个DOM元素，才会真正载入我们的alert.js。

**更多**

但是，绑定在某个特定事件上这个事似乎又过了一点，因为只有在点击的时候才会去真正的下载js，这又会太慢了了。好了，到这里，要抛出我们的终极问题——**我们想要异步地把js文件下载到用户的本地，但是不执行，仅当在我们想要执行的时候去执行**。

要是我们有下面这样的方式就好了：

var script = document.createElement("script");

script.noexecute = true;

script.src = "alert.js";

document.body.appendChild(script);

//后面我们可以这么干

script.execute();

可惜的是，这只是一个美丽的梦想，今天我们的Javascript还比较原始，这个“JS梦”还没有实现呢。

所以，我们的程序员只能使用hack的方式来搞。

有的程序员使用了非标准的script的type来cache javascript。如：

<script type=cache/script src="./alert.js"></script>

因为”cache/script”，这个东西根本就不能被浏览器解析，所以浏览器也就不能把alert.js当javascript去执行，但是他又要去下载js文件，所以就可以搞定了。可惜的是，webkit严格符从了HTML的标准——对于这种不认识的东西，直接删除，什么也不干。于是，我们的梦又破了。

所以，我们需要再hack一下，就像N多年前玩preload图片那样，我们可以动用object标签（也可以动用iframe标签），于是我们有下面这样的代码：

function cachejs(script\_filename){

var cache = document.createElement('object');

cache.data = script\_filename;

cache.id = "coolshell\_script\_cache\_id";

cache.width = 0;

cache.height = 0;

document.body.appendChild(cache);

}

然后，我们在的最后调用一下这个函数。请参看一下相关的示例：[**示例六**](http://coolshell.cn/asyncjs/async_test06.html)

在Chrome下按 Ctrl+Shit+I，切换到network页，你就可以看到下载了alert.js但是没有执行，然后我们再用示例五的方式，因为浏览器端有缓存了，不会再从服务器上下载alert.js了。所以，就能保证执行速度了。

关于这种preload这种东西你应该不会陌生了。你还可以使用Ajax的方式，如：

var xhr = new XMLHttpRequest();

xhr.open('GET', 'new.js');

xhr.send('');

到这里我就不再多说了，也不给示例了，大家可以自己试试去。

最后再提两个js，一个是[ControlJS](http://stevesouders.com/controljs/)，一个叫[HeadJS](http://headjs.com/)，专门用来做异步load javascript文件的。

好了，这是所有的内容了，希望大家看过后能对Javascript的载入和执行，以及相关的技术有个了解。**同时，也希望各前端高手不吝赐教！**