**[优化网站设计（十七）：延迟或按需加载内容](http://www.cnblogs.com/chenxizhang/archive/2013/05/16/3081941.html)**

**前言**

网站设计的优化是一个很大的话题,有一些通用的原则,也有针对不同开发平台的一些建议。这方面的研究一直没有停止过，我在不同的场合也分享过这样的话题。

作为通用的原则，雅虎的工程师团队曾经给出过35个最佳实践。这个列表请参考**Best Practices for Speeding Up Your Web Site** <http://developer.yahoo.com/performance/rules.html>，同时，他们还发布了一个相应的测试工具Yslow <http://developer.yahoo.com/yslow/>

我强烈推荐所有的网站开发人员都应该学习这些最佳实践，并结合自己的实际项目情况进行应用。 接下来的一段时间，我将结合ASP.NET这个开发平台，针对这些原则，通过一个系列文章的形式，做些讲解和演绎，以帮助大家更好地理解这些原则，并且更好地使用他们。

**准备工作**

为了跟随我进行后续的学习，你需要准备如下的开发环境和工具

1. Google Chrome 或者firefox ，并且安装 Yslow这个扩展组件.请注意，这个组件是雅虎提供的，但目前没有针对IE的版本。
   1. <https://chrome.google.com/webstore/detail/yslow/ninejjcohidippngpapiilnmkgllmakh>
   2. <https://addons.mozilla.org/en-US/firefox/addon/yslow/>
   3. 你应该对这些浏览器的开发人员工具有所了解，你可以通过按下F12键调出这个工具。
2. Visaul Studio 2010 SP1 或更高版本，推荐使用Visual Studio 2012
   1. <http://www.microsoft.com/visualstudio/eng/downloads>
3. 你需要对ASP.NET的开发基本流程和核心技术有相当的了解，本系列文章很难对基础知识做普及。

**本文要讨论的话题**

这一篇我和大家讨论的是第十六条原则：[Postload Components](http://developer.yahoo.com/performance/rules.html#postload) （**延迟**或**按需**加载内容）

页面加载过程中，除了页面本身的内容之外，可能需要加载很多额外的资源，例如我们常说的：

1. 脚本
2. 样式表
3. 图片

我在之前的文章中，已经有针对脚本和样式表做了一些优化的建议，请参考

1. [优化网站设计（二）：使用CDN](http://www.cnblogs.com/chenxizhang/archive/2013/04/30/3051686.html)
2. [优化网站设计（五）：在顶部放置样式定义](http://www.cnblogs.com/chenxizhang/archive/2013/05/01/3053216.html)
3. [优化网站设计（六）：在文档底部放置脚本定义或引用](http://www.cnblogs.com/chenxizhang/archive/2013/05/01/3053299.html)
4. [优化网站设计（八）：将脚本文件和样式表作为外部文件引用](http://www.cnblogs.com/chenxizhang/archive/2013/05/01/3053562.html)
5. [优化网站设计（十）：最小化JAVASCRIPT和CSS](http://www.cnblogs.com/chenxizhang/archive/2013/05/02/3054141.html)
6. [优化网站设计（十二）：删除重复脚本](http://www.cnblogs.com/chenxizhang/archive/2013/05/05/3061004.html)

这一条原则的核心是：**延迟或按需加载**。首先来针对我们比较最经常用到的脚本为例进行讲解。

**针对脚本的按需加载**

我们可以想象一下，一个真正的网站项目中，会有各种各样的脚本文件，其中还包含很多基础的框架（例如jquery,knockoutjs 等），这些脚本文件可能都或多或少需要在页面中引用。问题在于，如果页面一多起来，或者复杂起来，我们可能不太能准确地知道某个页面是否真的需要某个脚本。（难道不是这样吗？），一个蹩脚的解决方案是，那么就在母版页中，一次性将所有可能用到的框架脚本都引用进来吧。你是这样做的吗？

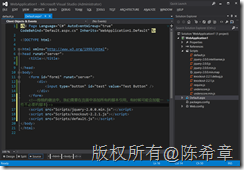
如果你真的这样做，那么，可能可以一时地解决问题。但实际上存在一个问题，在某些页面上，可能只用到一个脚本库，但为了你的方便，以后都需要全部加载所有的脚本库了。

随着项目的进一步开发，脚本之间的依赖会进一步复杂，要维护这些脚本确实是一个大问题。

在当年雅虎的团队写下这条原则的时候，他们提到了一个他们自己开发的组件来实现按需加载脚本，这个组件叫做GET，是包含在YUI这套工具包中的。<http://yuilibrary.com/yui/docs/get/>  ，它的意思就是可以动态地，按需加载脚本和样式表。

我对YUI的研究并不太多，而实际上这几年，Javascript这方面的技术突飞猛进，涌现了更多的创新性的设计。例如我今天要讲的[requirejs](http://requirejs.org/)。

我通过一个简单的例子给大家来讲解吧

[](http://images.cnitblog.com/blog/9072/201305/16152706-6c8e69e5041e4c4aa59e78db4da0c038.png)

这里有一个简单的网站，首页叫Default.aspx。根据我们的设计，这个页面需要加载jquery，以及可能的其他一些库，然后执行自己的一些逻辑。所以，我们会有如下的脚本引用

<!--传统的做法中，我们需要在页面中添加所有的脚本引用，有时候可能会加载一些不必要的脚本-->

<script src="Scripts/jquery-2.0.0.min.js"></script>

<script src="Scripts/knockout-2.2.1.js"></script>

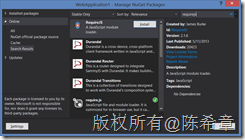
<script src="Scripts/default.js"></script>

这样做有什么问题吗？当然不是。只不过如我们之前所谈到的那样，这种预先加载所有脚本的方式，可能造成资源的浪费，而且这么多脚本引用在页面中，很容易引起混淆。为了更好地说明这一点，我给大家演示一个真实的场景：

1. 我们希望页面在加载的时候，只下载jquery这个库
2. 只有当用户点击了页面上面的那个按钮，我们才去下载knockout这个库

瞧！这就是所谓的按需加载。那么来看看我们将如何使用requirejs实现这个需求吧？

首先，你可以通过Nuget Package Manager 获取requirejs这个库。

[](http://images.cnitblog.com/blog/9072/201305/16152708-9eb75b88c0584824a739712ebd61b7af.png)

然后，在页面中，你只需要像下面这样定义脚本引用。（以后，你的页面中也只需要有这样一个引用）

<script src="Scripts/require.js" data-main="scripts/default"></script>

这里的data-main指的是主脚本。require.js会首先下载的一个脚本。你确实可以写成下面这样

<script src="Scripts/require.js" data-main="scripts/default.js"></script>

但是，正如你所见，.js是可以省略掉的。

接下来在default.js中，应该如何写脚本呢？下面是一个简单的例子

/// <reference path="require.js" />

/// <reference path="jquery-2.0.0.js" />

/// <reference path="knockout-2.2.1.js" />

//对requirejs进行一些基本配置

requirejs.config({

paths: {

jquery: "jquery-2.0.0.min" //指定一个路径别名

, knockout: "knockout-2.2.1"

}

});

//声明下面的代码是需要jquery这个库的

require(['jquery'], function () {

$(function () {

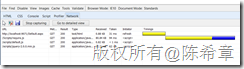
alert("Hello,jquery!");

});

});

我们看到，第一部分是对requirejs的基本配置，我们定义了两个别名。然后在第二部分，我们声明了下面的代码是需要jquery这个库的。

将页面运行起来之后，在浏览器中我们可以监控得到脚本下载的行为如下

[](http://images.cnitblog.com/blog/9072/201305/16152710-b744cf26bf704dc2a713b4b66d6d10a6.png)

如我们设想的那样，先加载了require.js，然后加载了default.js， 然后才是加载了jquery.js

有点意思，不是吗？虽然最后的结果也是加载了jquery，但这个加载方式与直接在页面中定义引用有着本质的区别，这是按需加载的。如果你对此还不太赞同，那么看了下面的例子，我相信你一定会同意的。

我们需要在default.js这个文件中，为页面上的那个按钮订阅点击事件，而且我们希望，只有当用户真的点击过了按钮，才会下载另外一个脚本（knockout），看看如何实现这个需求吧？

/// <reference path="require.js" />

/// <reference path="jquery-2.0.0.js" />

/// <reference path="knockout-2.2.1.js" />

//对requirejs进行一些基本配置

requirejs.config({

paths: {

jquery: "jquery-2.0.0.min" //指定一个路径别名

, knockout: "knockout-2.2.1"

}

});

//声明下面的代码是需要jquery这个库的

require(['jquery'], function ($) {

$(function () {

//只有用户点击了某个按钮，才动态加载knockoutjs

$("#test").click(

function () {

require(['knockout'], function (ko) {

alert(ko.version);

});

}

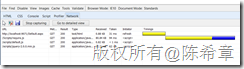
);

});

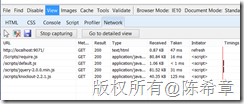
});

同样的，我们可以通过浏览器监控工具了解脚本下载的流程：

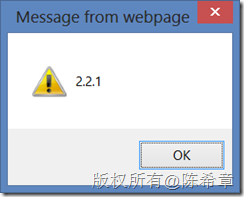
页面加载的时候，仍然只有三个脚本下载了

[](http://images.cnitblog.com/blog/9072/201305/16152713-7a5e051861ec49dd8ca8b2ce75826b11.png)

但是，如果点击了按钮，则会有第四个脚本下载

[](http://images.cnitblog.com/blog/9072/201305/16152714-af2a9a5a278745afac14940e411158de.png)

同时，从下面的对话框来看，也可以断定确实是执行了相应的脚本的。因为我们当前使用的knockout脚本的版本确实是2.2.1。

[](http://images.cnitblog.com/blog/9072/201305/16152715-c81eec4aa827422da05a072ee4a50e16.png)

这的确是一个很不错的机制。如果大家有兴趣，还可以继续深入研究，现在jquery为了更好地满足动态加载和按需加载的需要，甚至都提供了模块化的版本。请参考 <http://projects.jga.me/jquery-builder/>

**针对样式表文件的按需加载**

我相信按需加载脚本文件这样的设计，足够引起你的兴趣了。很自然地，你可能会有这样一个问题，能不能实现对样式表的按需加载呢？

听起来不错，而且应该也不难，但目前没有现成的实现。当然YUI中的GET是可以用的。

requirejs的官方有一个解释，有兴趣可以参考一下  <http://requirejs.org/docs/faq-advanced.html#css>

他们也提供了一个建议的脚本来按需加载样式表

function loadCss(url) {

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

link.type = "text/css";

link.rel = "stylesheet";

link.href = url;

document.getElementsByTagName("head")[0].appendChild(link);

}

你可以将这个脚本访问任何的地方，进行调用。例如我是下面这样做的

/// <reference path="require.js" />

/// <reference path="jquery-2.0.0.js" />

/// <reference path="knockout-2.2.1.js" />

//对requirejs进行一些基本配置

requirejs.config({

paths: {

jquery: "jquery-2.0.0.min" //指定一个路径别名

, knockout: "knockout-2.2.1"

}

});

//声明下面的代码是需要jquery这个库的

require(['jquery'], function ($) {

$(function () {

//只有用户点击了某个按钮，才动态加载knockoutjs

$("#test").click(

function () {

**loadCss('default.css');**

require(['knockout'], function (ko) {

alert(ko.version);

});

}

);

});

});

function loadCss(url) {

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

link.type = "text/css";

link.rel = "stylesheet";

link.href = url;

document.getElementsByTagName("head")[0].appendChild(link);

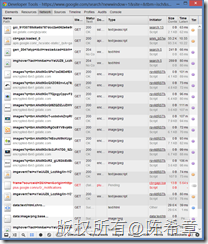
}

**针对图片的按需加载**

本文的最后一部分我们来谈谈图片的按需加载的问题。如果你的页面包含了大量的图片，掌握这个原则将非常有助于提高网页的加载速度。

大家可以设想一下图片搜索引擎的结果页面，例如 <https://www.google.com/search?newwindow=1&site=&tbm=isch&source=hp&biw=1468&bih=773&q=microsoft&oq=microsoft&gs_l=img.3...1779.4076.0.4399.9.7.0.0.0.0.0.0..0.0...0.0...1ac.1j4.12.img.aajYF7y8xW8>

我在images.google.com中搜索microsoft，毫无疑问，这会返回成千上万张图片。

[](http://images.cnitblog.com/blog/9072/201305/16152725-9fc69e31256d4c66a12bdfcc66d34ef8.png)[](http://images.cnitblog.com/blog/9072/201305/16152736-3feecc62a2dc4ae3a3ed8ee99166bbc6.png)

那么，这些图片是不是要一次性全部加载进来呢？显然不可能，你可能会说，应该做分页会不会好一些？分页通常是一个好技术，但在搜索引擎的页面中，分页不是一个很好的选择（作为用户并不见得愿意去点击页面导航按钮），目前主流的是图片搜索引擎的做法都是不采用分页，而是随着用户的滚动条滑动来按需获取图片。

这是一个相当重要的设计，但稍微思考一下，应该不是很简单的一个工作。幸运的是，雅虎的团队提供了一个很好的组件(ImageLoader)可以直接使用：<http://yuilibrary.com/yui/docs/imageloader/>

关于这个组件的用法，有几个在线的演示页面：

* [Basic Features of the ImageLoader Utility](http://yuilibrary.com/yui/docs/imageloader/basic-features.html)
* [Loading Images Below the Fold](http://yuilibrary.com/yui/docs/imageloader/below-fold.html)
* [Using ImageLoader with CSS Class Names](http://yuilibrary.com/yui/docs/imageloader/imageloader-class-names.html)

如果你习惯用jquery,那么可以参考一下下面这个说明

<http://www.appelsiini.net/projects/lazyload>