# 捆绑包与显示模式

**捆绑包（Bundle）**：

能够组织和优化CSS以及JavaScript文件，是由视图和布局引发浏览器向服务器请求的文件。

**显示模式（Display Mode）**：

针对不同的设备采用不同的视图。

## 理解默认脚本库

在创建除Empty以外的任一MVC项目时，Visual Studio都会在Scripts文件夹中添加一组JavaScript库，最主要并常用的有：

* jquery-1.8.2.js：jQuery库，可使得在浏览器中操作HTML元素变得简单而容易，与HTML标准部分的内建API相比，其优势尤其明显。
* jquery-ui-1.8.24.js：jQuery UI库，通过HTML元素创建富用户控件，为Web应用程序创建美观的UI，该库建立在jQuery库之上。
* jquery.mobile-1.1.0.js：jQuery Mobile库，为移动设备创建富用户控件。jQuery Mobile建立在jQuery之上，且只会添加到使用Mobile模板选项创建的项目中。
* jquery.validate.js：jQuery Validation库，执行HTML表单元素的输入验证。
* knockout-2.2.0.js：Knockout库，将“模型-视图-视图模型”模式运用于Web程序的客户端部分，作用是将Web程序中客户端的数据从显示给用户的元素中分离出来。通常被称为MVC的浏览器。
* modernizr-2.6.2.js：Modernizr库，用于检测浏览器对HTML5及CSS3的支持情况，能够在支持情况下使用最新功能，而在不支持时可以优雅降级。

**以下是Visual Studio及MVC专用库：**

* jquery-1.8.2.intellisense.js：在视图中编写jQuery代码时，为Visual Studio提供智能感应的功能。
* jquery.unobtrusive-ajax.js：提供MVC框架渐近式Ajax特性，依赖于jQuery。
* jquery.validate-vsdoc.js：在编写使用jQuery验证库的代码时，为Visual Studio提供智能感应的功能。
* jquery.validate.unobtrusive.js：提供MVC框架渐近式验证特性，依赖于jQuery。

对于Visual Studio及MVC专用的库，不需要我们做任何事情，Visual Studio会自动使用它们。

这里列出的都是常规的版本，同时出现的还会有压缩版——一般在发布的时候使用，可以节省很多空间，并减少网络带宽，节约网络资源。

## 准备示例

项目：ClientFeatures

项目模板：Basic（基本）

模型类：Appointment.cs

using System;

using System.ComponentModel.DataAnnotations;

namespace ClientFeatures.Models

{

public class Appointment

{

[Required]

public string ClientName { get; set; }

[DataType(DataType.Date)]

public DateTime Date { get; set; }

public bool TermsAccepted { get; set; }

}

}

控制器：Home

using ClientFeatures.Models;

using System;

using System.Web.Mvc;

namespace ClientFeatures.Controllers

{

public class HomeController : Controller

{

public ViewResult MakeBooking()

{

return View(new Appointment

{

ClientName = "Adam",

Date = DateTime.Now.AddDays(2),

TermsAccepted = true

});

}

public JsonResult MakeBooking(Appointment appt)

{

// 在实际项目中，这里是存储新 Appointment 的语句

return Json(appt, JsonRequestBehavior.AllowGet);

}

}

}

视图：MakeBooking.cshtml

@model ClientFeatures.Models.Appointment

@{

ViewBag.Title = "Make A Booking";

AjaxOptions ajaxOpts = new AjaxOptions

{

OnSuccess = "processResponse"

};

}

<h2>Book an Appointment</h2>

<link rel="stylesheet" href="~/Content/CustomStyles.css" />

<script src="~/Scripts/jquery-1.8.2.min.js"></script>

<script src="~/Scripts/jquery.validate.js"></script>

<script src="~/Scripts/jquery.validate.unobtrusive.js"></script>

<script src="~/Scripts/jquery.unobtrusive-ajax.js"></script>

<script type="text/javascript">

function processResponse(appt) {

$('#successClientName').text(app.ClientName);

$('#successDate').text(processDate(app.Date));

switchViews();

}

function processDate(dateString) {

return new Date(parseInt(dateString.substr(6, dateString.length - 8))).toDateString();

}

function switchViews() {

var hidden = $('.hidden');

var visible = $('.visible');

hidden.removeClass('.hidden').addClass('.visible');

visible.removeClass('.visible').addClass('.hidden');

}

$(document).ready(function () {

$('#backButton').click(function (e) {

switchViews();

})

});

</script>

<div id="formDiv" class="visible">

@using (Ajax.BeginForm(ajaxOpts))

{

@Html.ValidationSummary(true)

<p>@Html.ValidationMessageFor(m => m.ClientName)</p>

<p>Your name: @Html.EditorFor(m => m.ClientName)</p>

<p>@Html.ValidationMessageFor(m => m.Date)</p>

<p>Appointment Date: @Html.EditorFor(m => m.Date)</p>

<p>@Html.ValidationMessageFor(m => m.TermsAccepted)</p>

<p>@Html.EditorFor(m => m.TermsAccepted) I accept the terms $ conditions</p>

<input type="submit" value="Make booking" />

}

</div>

<div id="successDiv" class="hidden">

<h4>Your appointment is confirmed</h4>

<p>Your name is: <b id="successClientName"></b></p>

<p>The date of your appointment is: <b id="successDate"></b></p>

<button id="backButton">Back</button>

</div>

上面视图中，两个div元素，一个会在视图第一次渲染时显示给用户，它含有一个已启用Ajax的表单。当表单被递交并在接受到服务器对Ajax请求作出的相应时，应采取的应对方式是隐藏第一个div中的表单，同时显示另一个div元素，显示预约所确定的细节。

视图中link元素中引用的/Content文件夹中的CustomStyles .css文件为自定义添加的CSS文件，内容如下：

div.hidden {

display: none;

}

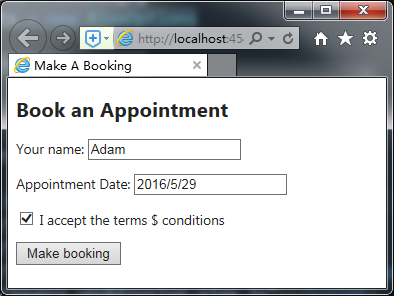
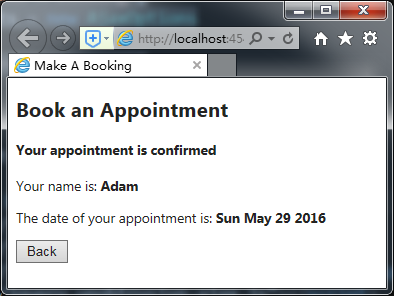
div.hidden {

display: block;

}

此处希望创建一个用于复杂视图的典型场景，但又不需要创建一个复杂的应用程序，这就是为什么添加的CSS文件只有两盒样式，也是为什么对一个十分简单的视图使用一连串的jQuery库的原因。其关键思想是有许多文件要进行管理。当在编写一个实际程序时，所受到的考验恰恰是需要在视图中处理许多脚本和样式文件。

现在启动程序看就可以看到效果了：

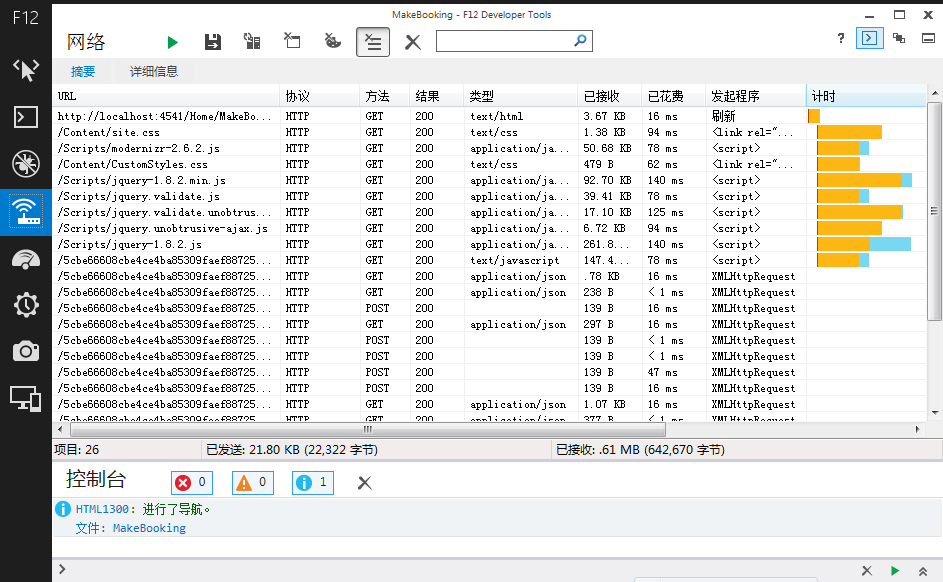
## 管理脚本与样式表

上面示例中视图代码中混用了Scripts文件夹中的库、Content文件夹中的CSS样式表、本地的Script元素，还有HTML和Razor标记。但是，按照示例那样写还是存在着一些隐含的问题，我们可以将脚本和样式表进行管理以改善。

**脚本及样式表加载的资料分析**

在对一个项目进行优化之前，最好是先做一些测量。对于本例使用的是IE11的“F12工具”进行测量。

启动程序，导航至/Home/MakeBooking，然后按F12键。之后点击“网络”选项卡（），点击“网络流量捕获”按钮（）。重载浏览器内容（刷新页面）将会得到如下图这样的结果：

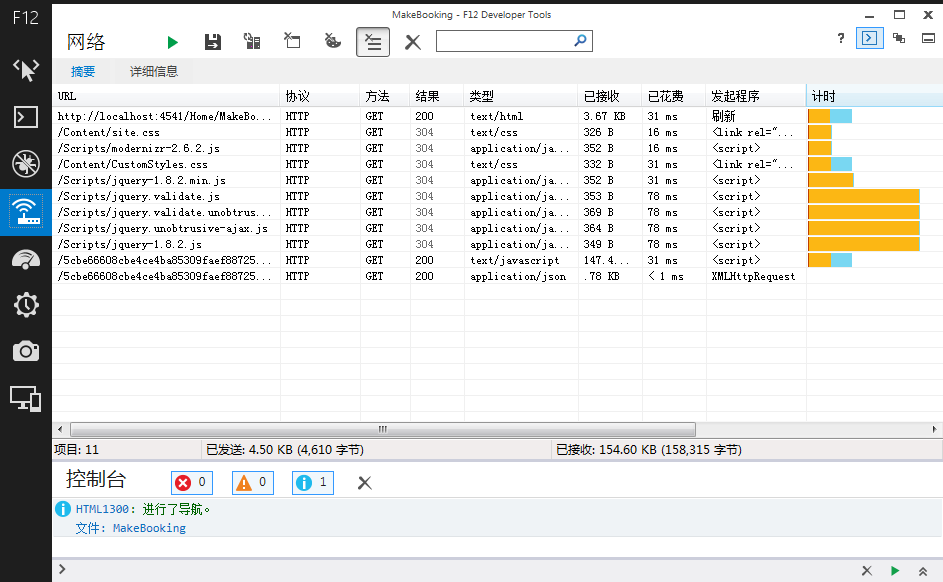


下面列出的是从上图看到的一些关键数据：

* 浏览器对Home/MakeBooking形成了9个请求
* 2个请求用于CSS文件
* 6个请求用于JavaScript文件
* 浏览器发送给服务的有22322字节
* 服务器发送给浏览器的有642670字节

这些都是最坏情况下的数据，因为在加载之前已经清理了缓存。如果在现实中，浏览器的缓存未清理，则浏览器会通过之前的请求缓冲，对此会有所改善。

如果不清理缓存，再次加载，则会是下面这种效果：



* 浏览器对Home/MakeBooking形成了9个请求
* 2个请求用于CSS文件
* 6个请求用于JavaScript文件
* 浏览器发送给服务的有4610字节
* 服务器发送给浏览器的有158315字节

如果注意观察，将会发现视图下载的JavaScript文件列表中已经重新创建了两个常见的问题。第一个是混用了最小化的和常规的JavaScript文件。这个问题不大，但对开发期间的调试会造成一定的影响，所以，最好不用混用。

第二个是同时下载了jQuery库的最小化和常规版本。发送这种情况是因为布局也会加载一些JavaScript和CSS文件，而用户缺乏相应的手段强制浏览器不用下载它已经拥有的代码。

后面的内容，将介绍一下如何有控制地获取脚本好CSS文件。更广泛意义上，也会展示如何减少浏览器需要发送给服务器的请求数，以及需要下载的数据量。

## 使用脚本和样式捆绑包

将JavaScript和CSS文件组织成**捆绑包**（Bundle），使其能够作为一个单一的单元进行处理。捆绑包是在/App\_Start/BundleConfig.cs文件中定义的。下面是由Visual Studio默认创建的：

using System.Web;

using System.Web.Optimization;

namespace ClientFeatures

{

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new ScriptBundle("~/bundles/jquery").Include(

"~/Scripts/jquery-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryui").Include(

"~/Scripts/jquery-ui-{version}.js"));

bundles.Add(new ScriptBundle("~/bundles/jqueryval").Include(

"~/Scripts/jquery.unobtrusive\*",

"~/Scripts/jquery.validate\*"));

bundles.Add(new ScriptBundle("~/bundles/modernizr").Include(

"~/Scripts/modernizr-\*"));

bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/site.css"));

bundles.Add(new StyleBundle("~/Content/themes/base/css").Include(

"~/Content/themes/base/jquery.ui.core.css",

"~/Content/themes/base/jquery.ui.resizable.css",

"~/Content/themes/base/jquery.ui.selectable.css",

"~/Content/themes/base/jquery.ui.accordion.css",

"~/Content/themes/base/jquery.ui.autocomplete.css",

"~/Content/themes/base/jquery.ui.button.css",

"~/Content/themes/base/jquery.ui.dialog.css",

"~/Content/themes/base/jquery.ui.slider.css",

"~/Content/themes/base/jquery.ui.tabs.css",

"~/Content/themes/base/jquery.ui.datepicker.css",

"~/Content/themes/base/jquery.ui.progressbar.css",

"~/Content/themes/base/jquery.ui.theme.css"));

}

}

}

其中静态方法RegisterBundles会在MVC程序第一次启动时，通过Global.asax中的Application\_Start方法调用。RegisterBundles方法以一个BundleCollection对象为参数，通过使用它的Add方法注册新的文件捆绑包。

也可以分别创建用于脚本和样式表的捆绑包，重要的是将这些文件类型分开，因为MVC框架对这些文件的优化是不同的。样式是由StyleBundle类表示，而脚本是由ScriptBundle类表示。

创建一个新的捆绑包，实际上就是在创建StyleBundle或ScriptBundle类的一个实例。它们都有一个构造函数参数，即引用捆绑包的路径。其作用是作为浏览器请求捆绑包内容的一个URL，因此，重要的是要为这些路径使用一个与应用程序所支持的路径无冲突的URL方案。最安全的做法是以~/bundles或~/Content作为起始路径。

一旦创建了上述的StyleBundle或ScriptBundle对象，就可以使用Include方法添加捆绑包所包含的样式或脚本文件的细节。有一些较好的做法，可以参考下面对BundleConfig类进行的修改：

using System.Web;

using System.Web.Optimization;

namespace ClientFeatures

{

public class BundleConfig

{

public static void RegisterBundles(BundleCollection bundles)

{

bundles.Add(new StyleBundle("~/Content/css").Include("~/Content/\*.css"));

bundles.Add(new ScriptBundle("~/bundles/clientfeaturesscripts").Include(

"~/Scripts/jquery-{version}.js",

"~/Scripts/jquery.validate.js",

"~/Scripts/jquery.validate.unobtrusive.js",

"~/Scripts/jquery.unobtrusive-ajax.js"));

}

}

}

上述修改中，首先使用~/Content/css路径对StyleBundle进行了修改，并将Include方法的参数改为~/Content/\*.css，以能够使该捆绑包包含程序中所有的CSS文件。文件后缀.css前面的星号（\*）是一个通配符，表示Content文件夹中的所有CSS文件，但这里忽略了文件的顺序，当然这在此处并不重要。——实际上，在浏览器中，CSS文件的加载顺序是不重要的，因此使用通配符的方式是很好的选择。但是，如果要使用CSS的样式优先规则，则需要分别列出这些文件，以保证顺序的正确。

ScriptBundle中的路径设置为~/bundles/clientfeaturesscripts，这个捆绑包中使用Include方法逐一指定了需要的脚本文件，并以逗号分隔，原因是此处只需要部分脚本文件，并关注脚本的加载和执行的顺序。

注意jQuery库文件的指定方式：~/Scripts/jquery-{version}.js，文件名中的{version}部分相对灵活，因为这样做，会匹配指定文件的任一版本，它会使用程序的配置，选择该文件的常规或最小化版本。

提示：使用常规版或最小版是由Web.config文件中的compilation元素决定的。当其debug属性被设置为true时，使用常规版；而当debug为false时，则使用最小化版。

这么写的好处是可以将所使用的库的更新为新版本，而不必重新定义捆绑包。缺点是{version}标志无法区分同一个文件夹中同一个库的不同版本。因此，必须确保Scripts文件夹中只有一个版本的库。

### 运用捆绑包

### 使用Scripts小节

### 修改后的资料分析

## 面向移动设备

### 观察应用程序

### 使用移动专用的布局和视图

### 创建自定义显示模式