# 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;

}

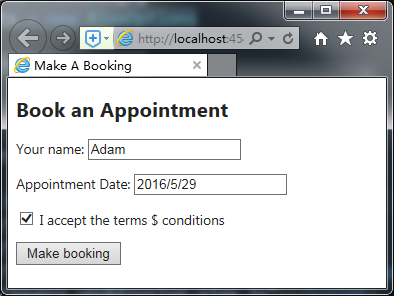
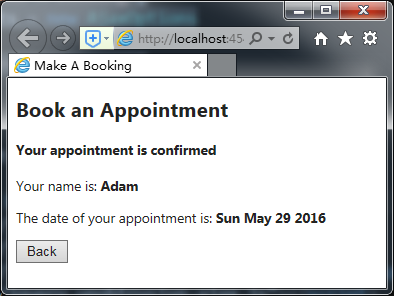
div.hidden {

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：

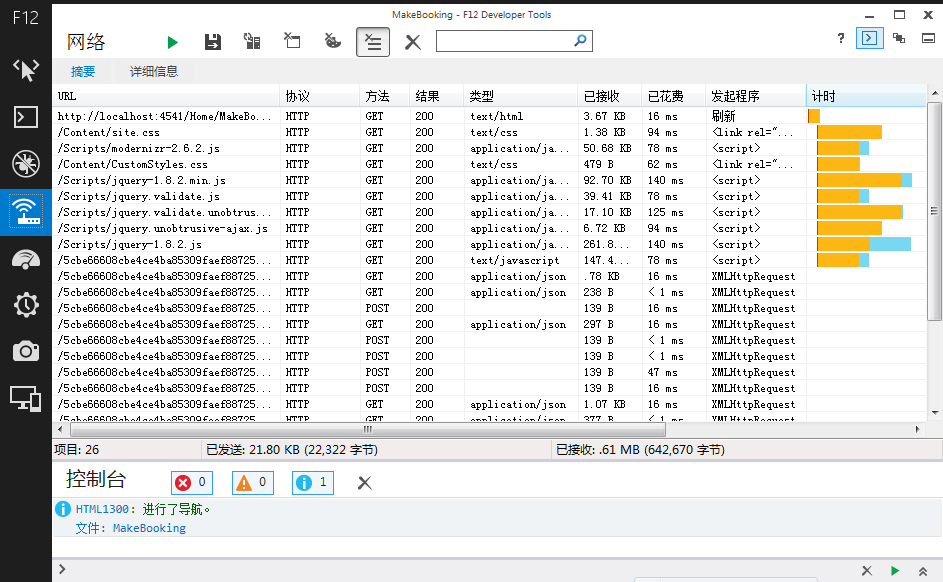
## 管理脚本与样式表

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：

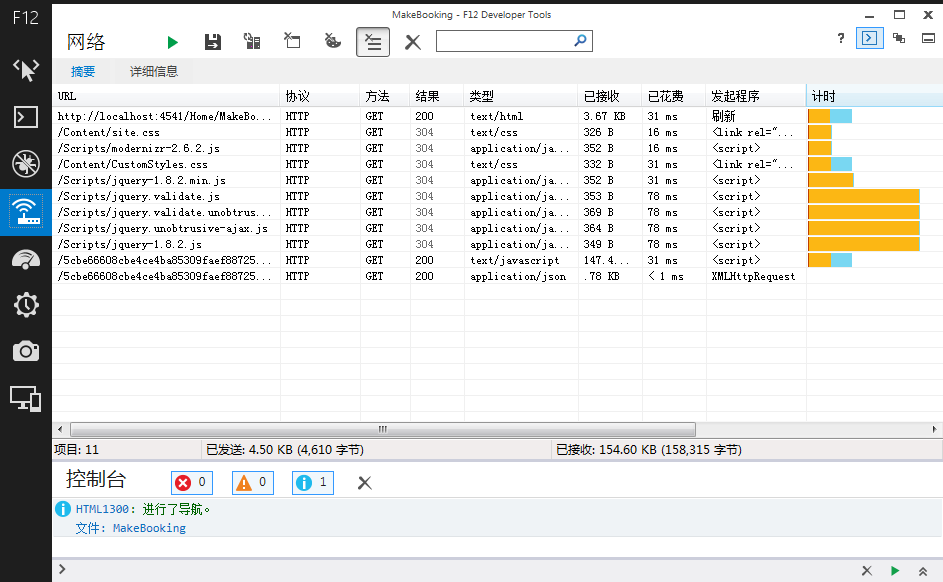


下面列出的是从上图看到的一些关键数据：

* 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

使用捆绑包之前，首先要做的是创建视图。当然也可以没有这一步，但这可以让MVC框架能够为应用程序执行最大限度的优化。

创建一个新的文件夹，路径及名称为/Scripts/Home，在该文件夹中添加一个新的脚本MakeBooking.js。这是需要遵守的约定，以便按控制器来组织各个页面的JavaScript文件（即按控制器名（不含“Controller”部分）及动作方法名组织JavaScript脚本的约定）：

function processResponse(appt) {

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

$('#successDate').text(processDate(appt.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();

})

});

上面代码只是将原来视图中的那部分脚本转移到了独立的js文件中。接着就是要修改视图文件了。修改原则是希望浏览器只请求其所需要的文件，适当地保留需要负责副本的请求，所以可以删除已经创建捆绑包的那些link和script元素，保留唯一的一个指向新建的那个专门的js文件MakeBooking.js的那个script元素，具体如下：

@model ClientFeatures.Models.Appointment

@{

ViewBag.Title = "Make A Booking";

AjaxOptions ajaxOpts = new AjaxOptions

{

OnSuccess = "processResponse"

};

}

<h4>Book an Appointment</h4>

**<script src="~/Scripts/Home/MakeBooking.js" type="text/javascript"></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>

可以在视图文件中引用捆绑包，但最好是创建能够多个视图共享的捆绑包——即在布局中运用捆绑包：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width" />

<title>@ViewBag.Title</title>

**@Styles.Render("~/Content/css")**

**@Scripts.Render("~/bundles/modernizr")**

</head>

<body>

@RenderBody()

**@Scripts.Render("~/bundles/jquery")**

@RenderSection("scripts", required: false)

</body>

</html>

捆绑包是分别使用@Styles.Render和@Scripts.Render辅助器方法添加的。为了达到预期目标，我们需要编辑\_Layout.cshtml文件，以使用新定义的捆绑包，删除不想使用的引用：

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width" />

<title>@ViewBag.Title</title>

**@Styles.Render("~/Content/css")**

</head>

<body>

@RenderBody()

**@Scripts.Render("~/bundles/clientfeaturesscriptes")**

@RenderSection("scripts", required: false)

</body>

</html>

好了，现在可以看看最终生成的HTML，下面是由用于~/Content/css捆绑包的Styles.Render方法生成的结果：

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

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

而这些是Scripts.Render方法生成的：

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

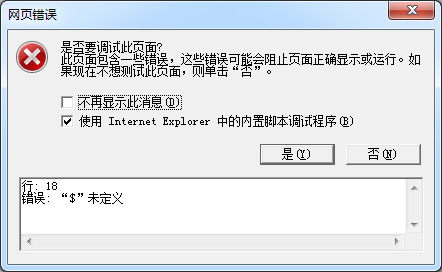
<script src="/Scripts/jquery.unobtrusive-ajax.js"></script>

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

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

### 使用Scripts小节

到现在为止，该示例还不能很好的工作，原因是包含在专业的js文件MakeBooking.js中的脚本代码需要依靠jQuery为按钮建立事件处理程序。也就是，jQuery文件要在MakeBooking.js之前加载，但在布局中能够看出实际的加载顺序正好相反，导致视图中的script元素出现在布局中的script元素之前，导致按钮不能正常工作，甚至有的浏览器还会报出JavaScript错误，比如我用的IE11就报出了如下图的错误：



解决上面的问题，可以有两种方式：

1. 将Scripts.Render调用移入视图的head元素；
2. 利用\_Layout.cshtml文件中定义的“可选脚本小节（Optional Scripts Section）”——使用这种方式其实就是定义了一个占位符，当视图中有可选脚本小节时，便将视图中的实际代码放在这儿。

下面使用的是第二种方式做的修改，请参考：

@model ClientFeatures.Models.Appointment

@{

ViewBag.Title = "Make A Booking";

AjaxOptions ajaxOpts = new AjaxOptions

{

OnSuccess = "processResponse"

};

}

<h4>Book an Appointment</h4>

**@section scripts{**

**<script src="~/Scripts/Home/MakeBooking.js" type="text/javascript"></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>

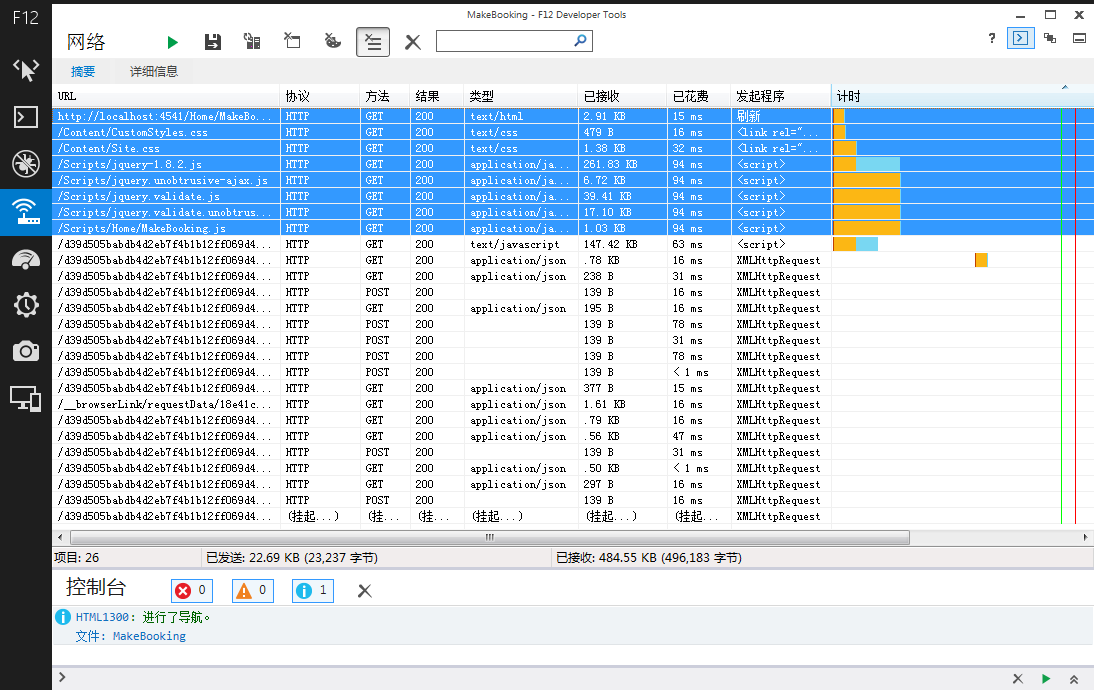
这样一来，script小节会出现在布局中对Scripts.Render的调用之后，视图专用的脚本也就会在jQuery之后才加载，按钮元素也就能正常工作，且不会再报类似之前那种错误了。

这里还是要提醒一下，在使用捆绑包时，这是经常会出现的错误，有必要对其加以重视，这也是为什么专门把这一问题介绍一下的原因。

注：关于可选脚本小节，可参考“视图”一章内关于“对Razor视图添加动态内容”中的“使用分段”小节的介绍，理解分段的概念及其工作原理。

### 修改后的资料分析

现在清理一下缓存，并导航到/Home/MakeBooking，看看这一调整的效果：



下面是主要信息的摘要：

* 浏览器对/Home/MakeBooking形成了8个请求（主要请求项）
* 2个用于CSS文件的请求
* 5个用于JavaScript文件的请求
* 从浏览器发送给服务的内容有23237字节
* 从服务器发给浏览器的有496183字节

对比之前的642670字节，大约减少了1/3。

如果将程序从调试模式切换到部署配置时，将会更明显，具体做法是将Web.config文件中compilation元素上的debug属性设置为false即可：

<system.web>

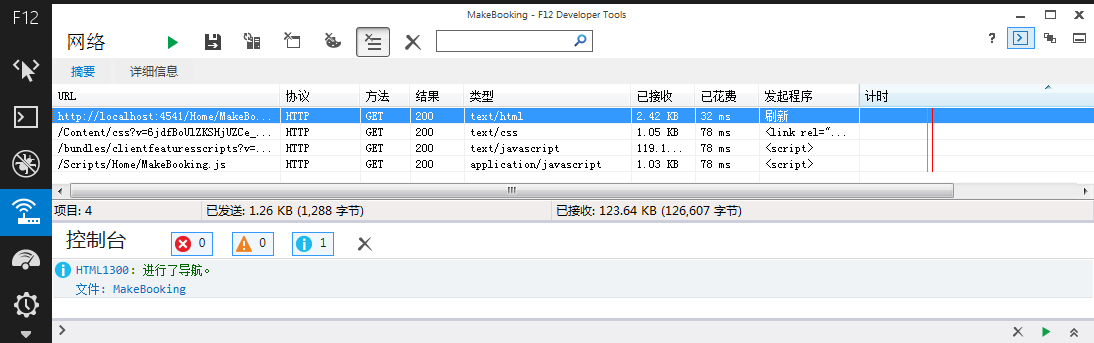
…

<compilation debug="**false**" targetFramework="4.5" />

…

</system.web>

此时再次重复上述步骤，将得到如下结果：



以下是摘要：

* 形成的请求为4个
* 1个请求用于CSS
* 2个请求用于JavaScript
* 浏览器发送给服务的内容有1288字节
* 服务器发送给浏览器的内容有126607字节

这次对CSS和JavaScript文件的请求变少了，原因是MVC框架在部署模式中，将一系列样式表和JavaScript文件联系在一起，并做了最小化，以使一个捆绑包中的内容能够通过一个单一的请求进行加载。如果查看程序渲染的HTML，就能明白是怎么实现的了：

Styles.Render方法生成的结果：

<link href="/Content/css?v=6jdfBoUlZKSHjUZCe\_rkkh4S8jotNCGFD09DYm7kBWE1" rel="stylesheet"/>

Scripts方法生成的结果：

<script src="/bundles/clientfeaturesscripts?v=KyclumLmAXQGM1-wDTwVUS31lpYigmXXR8HfERBGk\_I1"></script>

这些长长的URL以一个单一的数据库的形式，请求了一个捆绑包的内容。MVC框架对CSS数据所采取的最小化与JavaScript文件是不同的，这是为什么将样式表和脚本分别放在不同捆绑包中的原因。

虽然还可以继续优化，但对于我们这样的一个简单的程序，这已经足够了，比如如果将MakeBooking.js文件也添加到捆绑包中，还可以再消除一个请求，但这已经意义不大了。一开始那种视图专用的内容与布局的代码混合在一起的做法是提倡的，如果在更复杂的系统中，也会会创建第二个捆绑包，让它包含更多的自定义代码，但这里只是一个简单的示例程序，优化的准则是要清楚到什么程度为止。——目前，对应这样的一个示例程序已经达到目标了。

## 面向移动设备

移动设备和桌面设备有着很大的区别，最容易引起问题的是触摸交互和受限制的屏幕尺寸。

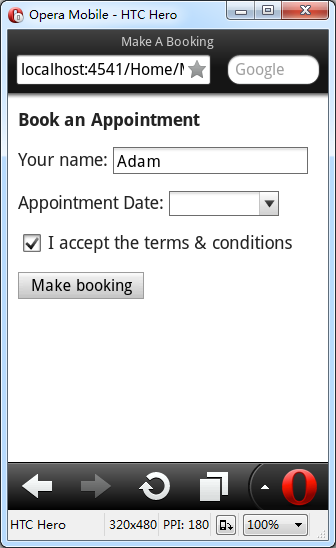
更多的移动开发方面的知识不是我们这里关注的，在此，只需要做些了解即可。

### 观察应用程序

开发一个移动设备的程序最好是从零开始建立一个新项目。但这里采取的办法是在原示例项目（ClientFeatures）基础中添加支持。

首先，需要一个能够模拟移动端展示的程序的工具，书中推荐的是Opera Mobile Emulator（Opera移动设备模拟器），该模拟器是免费的，可以从www.opera.com/developer/tools/mobile上下载。当然也可以使用Windows Phone、Android以及Blackberry等，但这些往往都很慢，用起来也很痛苦，因为它们都不是仅模拟浏览器，而是模拟了整个移动操作系统。

我在学习的时候下载的是“Opera Mobile Classic Emulator 12.1 for Windows”版本。安装完成后启动，并将模拟器的设置使用HTC Hera的配置。导航到/Home/MakeBooking将会看到结果如下：



### 使用移动专用的布局和视图

由于程序如此之简单，以至于没有什么丑陋的地方需要修改，只是它并没有在触摸屏上以易于操作的方式显示。后面，简单介绍如何创建移动专用版本的视图和布局，以方便对其内容的调整来适应目标设备。其实，这个过程很简单，需要做的只是创建其名称带有.Mobile的视图和布局即可：

视图：MakeBooking.Mobile.cshtml（注意，在扩展名前面加了.Mobile）

@model ClientFeatures.Models.Appointment

@{

ViewBag.Title = "Make A Booking";

AjaxOptions ajaxOpts = new AjaxOptions

{

OnSuccess = "processResponse"

};

}

**<h4>This is an MOBILE View</h4>**

@section scripts{

<script src="~/Scripts/Home/MakeBooking.js" type="text/javascript"></script>

}

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

@using (Ajax.BeginForm(ajaxOpts))

{

@Html.ValidationSummary(true)

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

**<p>Name: </p><p>@Html.EditorFor(m => m.ClientName)</p>**

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

**<p>Date: </p><p>@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>

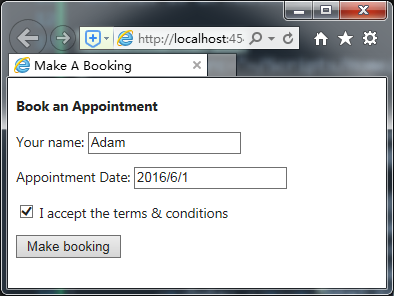
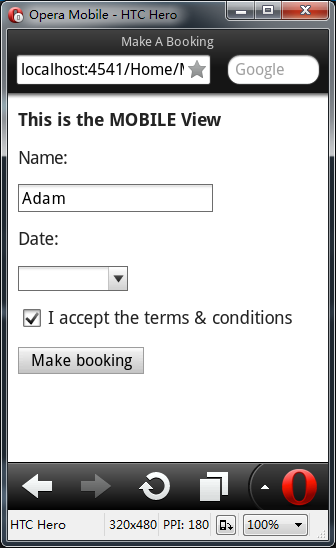
<p>The date of your appointment is: <b id="successDate"></b></p>

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

</div>

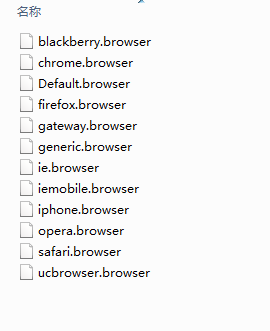
注意，为了能够使input元素的标签能够单独显示，上面对其做了一点改动。另外也对h4标题元素的内容做了调整，以使其更加醒目。

MVC框架能够智能识别并运用移动视图。比如如果使用桌面浏览器，将使用/Views/Home/MakeBooking.cshtml视图；如果是移动浏览器模拟器，则将看到由/Views/Home/MakeBooking.Mobile.cshtml视图渲染的HTML，如图：

MVC检测浏览器的方式是依赖于.NET框架包含的一组文件，这组文件是在类似C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config\Browsers目录的位置的，对于不同的系统会有些不同。该目录中有一系列文件，其中的每一个都与一个移动浏览器相关联。当MVC框架接收到移动模拟器的请求时，会使用“用户代理”字符串，这种用户代理字符串是所有浏览器都会发生的，于是便能确定该请求发自移动设备，也就是为什么会自动使用MakeBooking.Mobile.cshtml视图了。

上述目录中的文件大体参考下图：



### 创建自定义显示模式

默认情况下，MVC框架只检测移动设备，并将其他方面的各项工作作为一个单一的类别加以处理。如果需对不同种类的设备采取具体的响应，则需要创建自己的显示模式。

这里选择使用Amazon的Kindle Fire配置资料，它是Opera Mobile Emulator中自带的一种配置。由于它是平板设备上的一款浏览器，因此，它发生的用户代理字符串与.NET Framwork期望的从Opera Mobile不同，因此默认将会使用标准的MakeBooking.cshtml视图。

下面通过对Global.asax文件进行修改，以使其能够为Opera Tablet（指平板电脑的Opera浏览器）创建一个新的显示模式：

using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Http;

using System.Web.Mvc;

using System.Web.Optimization;

using System.Web.Routing;

**using System.Web.WebPages;**

namespace ClientFeatures

{

public class MvcApplication : System.Web.HttpApplication

{

protected void Application\_Start()

{

**DisplayModeProvider.Instance.Modes.Insert(0,**

**new DefaultDisplayMode("OperaTablet")**

**{**

**ContextCondition = (context =>**

**context.Request.UserAgent.IndexOf("Opera Tablet", StringComparison.OrdinalIgnoreCase) >= 0)**

**});**

AreaRegistration.RegisterAllAreas();

WebApiConfig.Register(GlobalConfiguration.Configuration);

FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);

RouteConfig.RegisterRoutes(RouteTable.Routes);

BundleConfig.RegisterBundles(BundleTable.Bundles);

}

}

}

通过上面的修改，MVC框架将查找诸如/Views/Home/MakeBooking.OperaTablet.cshtml的视图。

提示：MVC框架会依次检查DisplayModeProvider.Instance.Modes属性所返回的集合中的每一种显示模式，并在发现其ContextCondition表达式返回true时，停止查找。有一个默认的备用显示模式能够匹配任何请求，并使用默认视图。需要确保显示模式能够在该备用模式之前得到查询，所以，上述代码中使用了Insert方法将对象放到了零索引的位置。

为了利用这一显示模式，此处添加了一个新的视图文件：MakeBooking.OperaTablet.cshtml：

@model ClientFeatures.Models.Appointment

@{

ViewBag.Title = "Make A Booking";

AjaxOptions ajaxOpts = new AjaxOptions

{

OnSuccess = "processResponse"

};

}

<h4>This is the OPERA TABLET View</h4>

@section scripts{

<script src="~/Scripts/Home/MakeBooking.js" type="text/javascript"></script>

}

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

@using (Ajax.BeginForm(ajaxOpts))

{

@Html.ValidationSummary(true)

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

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

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

<p>Date: </p><p>@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>

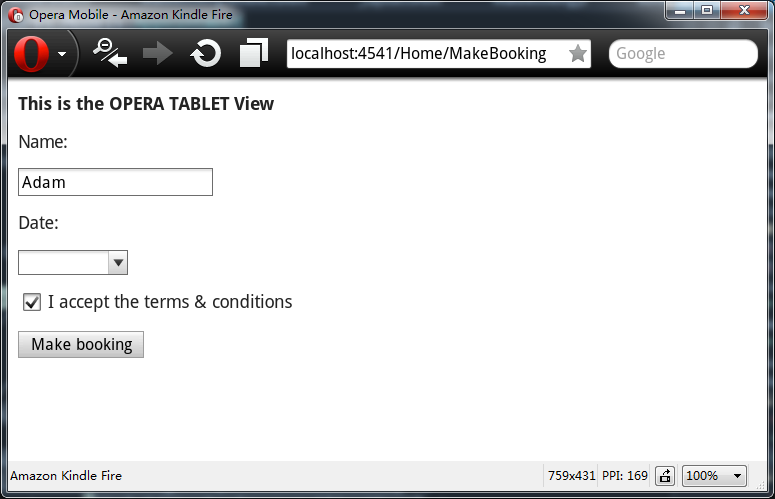
<p>Your name is: <b id="successClientName"></b></p>

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

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

</div>

下面是修改后的效果：



到这里，已经简单地介绍了如何创建自定义显示模式，虽然上图中的显示还是不很近人意，但也能明确地说明了其原理。

注意：在创建自定义显示模式时要小心。它很容易捕获不正确的客户端请求，或漏掉在某个特定类型客户端上运行不同版本浏览器之间的一些微妙的差异，建议用一系列设备进行彻底的测试。