**[学习ASP.NET MVC(七)——我的第一个ASP.NET MVC 查询页面](http://www.cnblogs.com/chillsrc/p/3918277.html)**

       在本篇文章中，我将添加一个新的查询页面（SearchIndex），可以按书籍的种类或名称来进行查询。这个新页面的网址是http://localhost:36878/Book/ SearchIndex。该页面中用一个下拉框来显示种类，用一文本框让用户输入书籍的名称。当用户在点击“查询”按钮之后，页面会被刷新，显示用户的查询结果。控制器会根据用户提交查询参数，由动作方法（Action Motehd）去解析用户提交的参数值，并使用这些值来查询数据库。

**第一步，创建SearchIndex查询页面**  
            首先，在BookController中添加一个SearchIndex动作方法。该方法将返回一个包含HTML表单的视图。代码如下：

[复制代码](javascript:void(0);)

public ActionResult SearchIndex(string searchString)

{

var books = from m in db.Books

select m;

if (!String.IsNullOrEmpty(searchString))

{

books = books.Where(s => s.Name.Contains(searchString));

}

return View(books);

}

[复制代码](javascript:void(0);)

        上面SearchIndex方法代码中的第一行如下面，其作用是创建一个LINQ查询从数据库中筛选相应的书籍数据：

var books = from m in db.Books

select m;

       这一句表示查询已经建立，但是尚未对数据存储区中的数据进行筛选。   
       如果“searchString”参数变量中有一个字符串，则这个“books”查询语句将以“searchString”参数变量的值做为查询条件进行筛选相应的书籍数据，如下面的代码：

if (!String.IsNullOrEmpty(searchString))

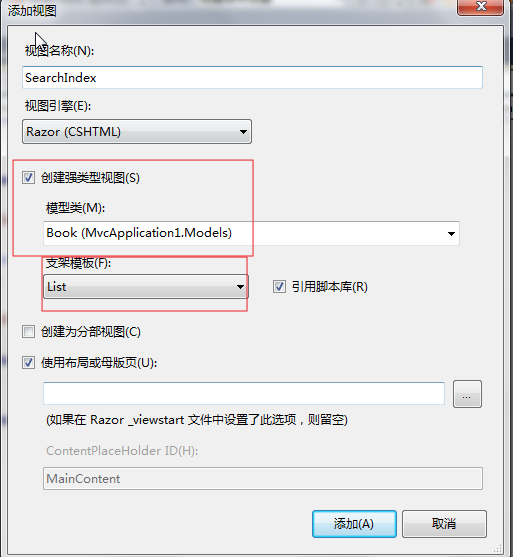
{

books = books.Where(s => s.Name.Contains(searchString));

}

       上面代码中的s =>s.Title代码是一个Lambda表达式。 Lambda 表达式用在基于方法的 LINQ 查询中,作为诸如 Where 和 Where 等标准查询运算符方法的参数。  例如上面的代码中使用的方法。LINQ查询时未执行它们的定义，或者当他们通过调用一个方法进行修改，或在排序时进行修改。相反，查询执行的延迟，这意味着一个表达式的计算被延迟，直到其变量实际上的迭代结束或者可以调用的方法(ToList方法)被调用。在SearchIndex这个页面中，查询是在SearchIndex视图下执行。

       接下来，我们来制作一个SearchIndex视图，以将这个查询界面呈现给用户看。在Visual Studio中的BookController文件中右键单击里面的SearchIndex方法，在弹出菜单中单击“添加视图”菜单。在添加视图对话框中，指定你要一个Book对象传递给视图模板作为它的模型类。在支架模板列表中，选择列表，然后单击“添加”。如下图。



        当单击“添加”按钮，将创建一个Views\Book\ SearchIndex.cshtml视图模板。因为你选择了列表中的支架模板列表中，Visual Studio会自动生成（搭建）的视图中的一些默认的标记。基架创建一个HTML表单。它检查Book类和创建的代码来呈现为类的每个属性<label>元素。下面的清单显示生成的创建视图：

[复制代码](javascript:void(0);)

@model IEnumerable<MvcApplication1.Models.Book>

@{

ViewBag.Title = "书籍查询";

}

<h2>书籍查询</h2>

<p>

@Html.ActionLink("Create New", "Create")

</p>

<table>

<tr>

<th>

@Html.DisplayNameFor(model => model.Category)

</th>

<th>

@Html.DisplayNameFor(model => model.Name)

</th>

<th>

@Html.DisplayNameFor(model => model.Numberofcopies)

</th>

<th>

@Html.DisplayNameFor(model => model.AuthorID)

</th>

<th>

@Html.DisplayNameFor(model => model.Price)

</th>

<th>

@Html.DisplayNameFor(model => model.PublishDate)

</th>

<th></th>

</tr>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Category)

</td>

<td>

@Html.DisplayFor(modelItem => item.Name)

</td>

<td>

@Html.DisplayFor(modelItem => item.Numberofcopies)

</td>

<td>

@Html.DisplayFor(modelItem => item.AuthorID)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.PublishDate)

</td>

<td>

@Html.ActionLink("Edit", "Edit", new { id=item.BookID }) |

@Html.ActionLink("Details", "Details", new { id=item.BookID }) |

@Html.ActionLink("Delete", "Delete", new { id=item.BookID })

</td>

</tr>

}

</table>

[复制代码](javascript:void(0);)

      按F5， 运行该应用程序，并使用浏览器导航到/Book/ SearchIndex。在URL后面会追加一个查询字符串，如“?searchString=sql”。提交之后，符合条件的书籍数据将显示。如下图。



 **第三，修改SearchIndex查询方法**  
        我们也可以修改SearchIndex方法的参数，传入一个命名参数“ID”，这个“ID”参数将匹配在Global.asax文件中的缺省路由设置中的{id}占位符。路由设置如下：

{controller}/{action}/{id}

原来SearchIndex方法如下所示:

[复制代码](javascript:void(0);)

public ActionResult SearchIndex(string searchString)

{

var books = from m in db.Books

select m;

if (!String.IsNullOrEmpty(searchString))

{

books = books.Where(s => s.Name.Contains(searchString));

}

return View(books);

}

[复制代码](javascript:void(0);)

       修改后的SearchIndex方法将如下所示：

[复制代码](javascript:void(0);)

public ActionResult SearchIndex(string id)

{

string searchString = id;

var books = from m in db.Books

select m;

if (!String.IsNullOrEmpty(searchString))

{

books = books.Where(s => s.Name.Contains(searchString));

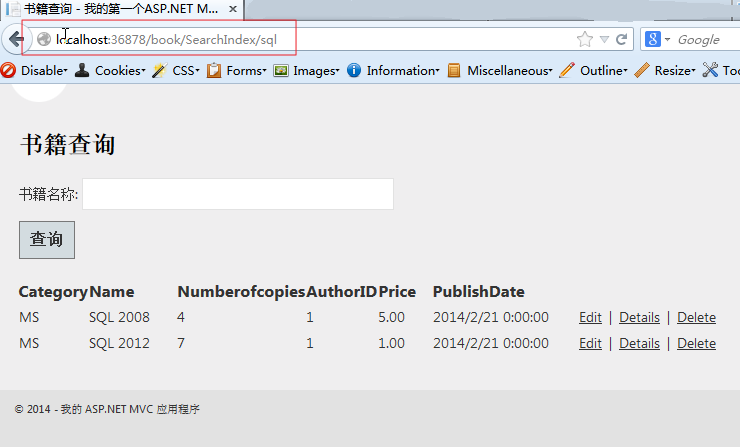
}

return View(books);

}

[复制代码](javascript:void(0);)

         现在，您可以把查询条件“书籍名称”作为路由数据（URL中的一段字符），而不是作为查询字符串值。如下图。



**第三，添加“书籍名称”查询条件**  
     通过上面的学习，我们实现了数据查询功能。但是我们不能希望普通用户也能像我们一样，会在URL中添加查询条件。当需要进行查询时，自己去修改URL中的查询条件内容来进行书籍查询。所以，需要添加一个查询页面，方便普通用户输入相应的查询条件，根据查询条件去查询书籍。

       一、改变SearchIndex方法的签名，并通过路由绑定到ID参数上，让SearchIndex方法接受一个名为“searchString”字符串作为参数。代码如下：

[复制代码](javascript:void(0);)

public ActionResult SearchIndex(string searchString)

{

var books = from m in db.Books

select m;

if (!String.IsNullOrEmpty(searchString))

{

books = books.Where(s => s.Name.Contains(searchString));

}

return View(books);

}

[复制代码](javascript:void(0);)

        二、打开Views\Book\ SearchIndex.cshtml文件，在@Html.ActionLink("Create New", "Create")的位置处，添加一个文本框与一个查询按钮。内容如下：

[复制代码](javascript:void(0);)

@using (Html.BeginForm())

{

<p> 书籍名称: @Html.TextBox("SearchString")<br />

<input type="submit" value="查询" /></p>

}

[复制代码](javascript:void(0);)

        下面的代码显示了Views\Book\ SearchIndex.cshtml文件的一部分与增加的筛选标记。

[复制代码](javascript:void(0);)

@Model IEnumerable<MvcApplication1.Models.Book>

@{

ViewBag.Title = "书籍查询";

}

<h2>书籍查询</h2>

@using (Html.BeginForm())

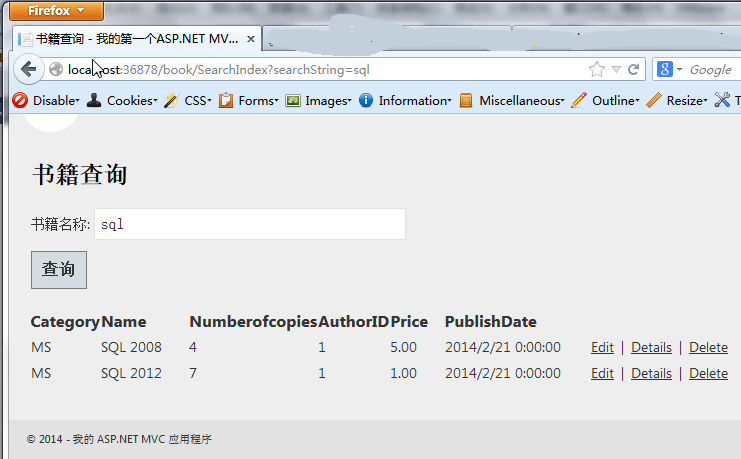
{

<p> 书籍名称: @Html.TextBox("SearchString")<br />

<input type="submit" value="查询" /></p>

}

[复制代码](javascript:void(0);)

        这个Html.BeginForm辅助方法创建了一个<form>标签。当用户通过单击“查询”按钮提交表单时，Html.BeginForm辅助方法将会把这个Form中的内容提交给控制器，同时刷新页面。  
       三、按F5，运行该应用程序，并在“书籍名称”文本框中输入“sql”，然后点击“查询”按钮，进行书籍查询。如下图。

       还有不需要重载SearchIndex的HttpPost的方法。因为该方法不改变应用程序的状态，只是是用来查询数据。   
       你可以象下面一样把[HttpPost]加在SearchIndex方法的上面。在添加了[HttpPost]之后，浏览器将首先调用匹配HttpPost的SearchIndex方法。HttpPost的SearchIndex方法运行结果如下图。

[复制代码](javascript:void(0);)

[HttpPost]

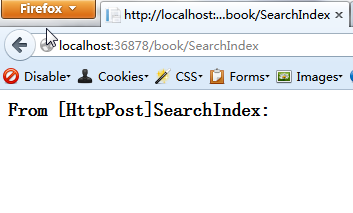
public string SearchIndex(FormCollection fc, string searchString)

{

return "<h3> From [HttpPost]SearchIndex: " + searchString + "</h3>";

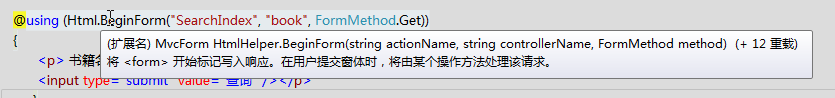
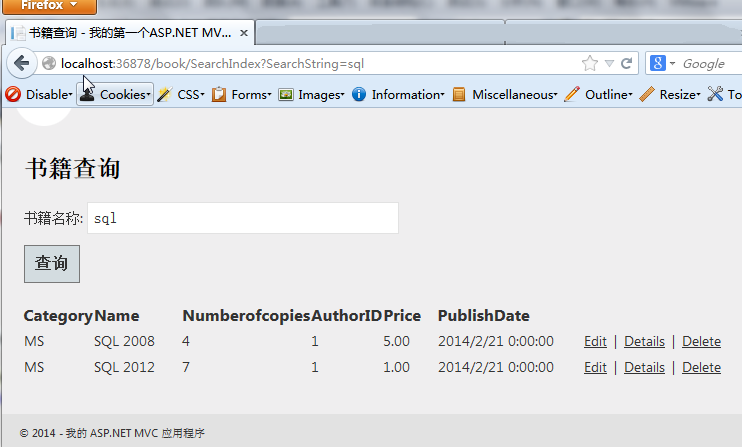
}

[复制代码](javascript:void(0);)



         然而，即使添加了SearchIndex的HttpPost版本的方法。假设一下如下场景，你想把某个查询结果发给你的朋友，你又想只发一个特定链接给朋友，让他们可以查询查看相同的一个查询结果。请注意，如果是相同的查询结果，一个为HTTP POST请求，另一个为相同的URL GET请求（例如本地主机：XXXXX /图书/ SearchIndex ） 。则HTTP POST请求在URL本身没有带查询参数的情况下，是无法看到查询结果的。如果只是一个URL，这就意味着你与朋友们永远不能看到相同的查询结果。  
        你应该使用以下的解决方案，使用BeginForm的重载方法，指定POST请求应该在URL中添加的相应的查询参数信息 ，它应该被路由到SearchIndex方法的HTTPGET版本。使用下面的代码替换现有参数的BeginForm方法将：如下图。

@using (Html.BeginForm("SearchIndex", "book", FormMethod.Get))

  
       现在，当你点击“查询”按钮时，URL中将包含查询条件的字符串。查询请求也将去请求HTTPGET的 SearchIndex操作方法，即使你有一个HttpPost的SearchIndex方法。如下图。

**第三，添加“书籍种类”查询方法**  
      第一步，删除代码中HttpPost版本的SearchIndex方法。

       第二步，添加新的查询功能。在页面中添加一个“书籍种类”的查询条件，让用户可以通过“书籍种类”查询到相关书籍。用下面的代码替换SearchIndex方法：

[复制代码](javascript:void(0);)

public ActionResult SearchIndex(string Category, string searchString)

{

var cateLst = new List<string>();

var cateQry = from d in db.Books

orderby d.Category

select d.Category;

cateLst.AddRange(cateQry.Distinct());

ViewBag.category = new SelectList(cateLst);

var books = from m in db.Books

select m;

if (!String.IsNullOrEmpty(searchString))

{

books = books.Where(s => s.Name.Contains(searchString));

}

if (string.IsNullOrEmpty(Category))

return View(books);

else

{

return View(books.Where(x => x.Category == Category));

}

}

[复制代码](javascript:void(0);)

      这个新的SearchIndex方法增加了一个新的查询条件，即Category。代码的前几行创建一个List对象从数据库中查询“书籍种类”，并将查询到的结果添加到List对象中。

       下面的代码是通过一个LINQ查询从数据库把所有书籍种类都查了出来。

var cateQry = from d in db.Books

orderby d.Category

select d.Category;

        以下代码的作用是使用泛型List集合的AddRange方法所有书籍类型添加到列表中。 （如果没有DISTINCT修饰符，会将相同的类型增加到列表中 - 例如，MS,SAP将会被我们添加两次）。

cateLst.AddRange(cateQry.Distinct());

      然后，该代码存储在ViewBag对象流派列表。

ViewBag.category = new SelectList(cateLst);

     下面的代码演示如何检查Category参数。如果它不是空的，该代码进一步限制了书的查询所选择的书限制到指定的种类。

[复制代码](javascript:void(0);)

if (string.IsNullOrEmpty(Category))

return View(books);

else

{

return View(books.Where(x => x.Category == Category));

}

[复制代码](javascript:void(0);)

**第四，在SearchIndex查询页面中添加“书籍种类”查询条件**  
          添加一个HTML样式的下拉列表框（DropDownList）到Views\Book\ SearchIndex.cshtml文件中，放在“书籍名称”文本框的前面。已完成的代码如下所示：

[复制代码](javascript:void(0);)

<p>

@using (Html.BeginForm("SearchIndex","book",FormMethod.Get)){

<p>书籍种类： @Html.DropDownList("category", "All")

书籍名称： @Html.TextBox("SearchString")

<input type="submit" value="查询" /></p>

}

</p>

[复制代码](javascript:void(0);)

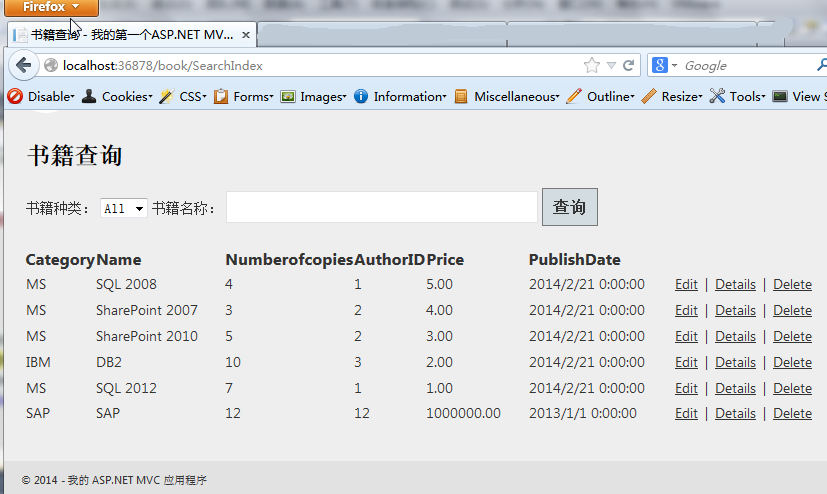
        按F5，运行这个应用程序，在浏览器中浏览/Book/ SearchIndex这个地址。就可以按“书籍种类”、“书籍名称”进行书籍信息查询。如下图1,图2。  
 

图1

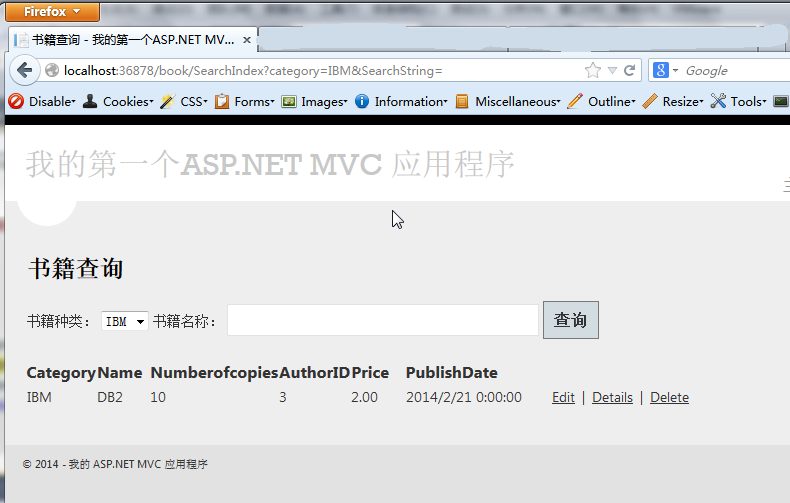


图2  
 在本篇中学习如何创建一个新的查询操作方法和视图，让用户通过书籍名称和书籍类型来进行查询。