文档13 购物页面前端

## 前端页面大致设计

### 首页

展示所有图书，一排展示四本，默认8本一页。



### 按图书类别/图书出版社查看

展示所有图书，一排展示四本，默认8本一页。



### 用户中心

访问页面的，获取客户端用户是否登录，如果未登录，显示“请登录”连接，登录成功后，跳转回首页。

### 我的收货地址

提供收货地址展示，新增，修改，删除，设置默认地址等。

### 我的订单

### 我的购物车

## 前端数据展示

### 非分页数据展示方式

在前端页面制作过程中，如果要展示后台数据不需要分页展示，实现起来相对简单。

1. 首先通过业务逻辑方法以DataTable方式返回需要展示的数据。
2. 在Action中调用业务逻辑方法，并将其数据保存到ViewData变量中。
3. 在前端页面中从ViewData中取数DataTable数据，通过Razor表达式的循环语句实现数据的输出。

例如“按图书类别查看”时，左侧需要展示所有的图书类别，实现方法如下：

1. 在客户端的业务类“bks\_BookType”中提供返回图书类别的业务方法。

//查询图书类别

public static DataTable getTable(db.dbEntities dc)

{

efHelper ef = new efHelper(ref dc);

string sql = " SELECT rowID,bookTypeCode,bookTypeName FROM dbo.bks\_BookType ";

DataTable table = ef.ExecuteDataTable (sql);

return table;

}

1. 在图书类别展示的控制器内增加“showNav”Action，内部增加如下代码

//栏目导航分布视图

[OutputCache(Duration =1000)]

[ChildActionOnly]

public ActionResult bookTypeNav()

{

**ViewData["bks\_BookTypeTable"]** = db.bll.client.bks\_BookType.getTable(dc);

return PartialView();

}

1. 为“showNav”视图中添加如下代码：

<ul>

<li><a href="@Url.Action("Index", "bookType")">所有图书</a></li>

@{

System.Data.DataTable table = (System.Data.DataTable)**ViewData["bks\_BookTypeTable"]**;

}

@foreach (System.Data.DataRow row in table.Rows)

{

<li>

<a href="@Url.Action("Index","bookType",new { type = row["bookTypeCode"] })">

@row["bookTypeName"].ToString()

</a>

</li>

}

</ul>

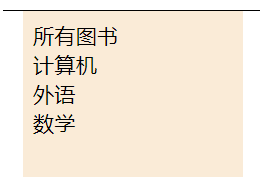
1. 将子Action嵌入到页面的左侧导航栏中。

<div class="left">

@Html.Action ("showNav","bookType")

</div>

1. 查询运行效果



### 分页数据展示方式

在前端页面制作过程中，如果要展示后台数据需要分页展示，则采用的和后端一样的开发方案，这里已图书的展示为例，具体方法如下：

1. 前端的母版页Head标记中需要引入和后端母版页Head标记中一样的样式和JS文件。



1. 在“dbLibrary\client\view”中创建“bks\_Book.cs”视图模型，代码如下：



1. 在“按图书类别展示”的控制其中添加如下的Action。



1. “Index”的前端视图代码如下：

@model db.client.view.bks\_Book

@section header{

<style type="text/css">

.left {

width: 220px;

float: left;

background-color: antiquewhite;

font-size: 16pt;

padding-left: 10px;

margin-left: 20px;

padding-top: 10px;

min-height: 730px;

box-sizing: border-box;

}

.right {

width: 740px;

float: right;

padding-top: 10px;

font-size: 12pt;

box-sizing: border-box;

}

.right li {

float: left;

width: 220px;

height: 300px;

margin-left: 20px;

border: 1pt dotted black;

margin-bottom: 15px;

box-sizing: border-box;

}

</style>

<script type="text/javascript">

//刷新当前页面的数据

function refleshData(exeCountSql) {

$("#container").pager("reflesh", exeCountSql);

}

//初始化分页

$(function () {

$("#container").pager();

});

</script>

}

<div class="left">

@Html.Action("showNav","bookType")

</div>

<div class="right">

<div class="container" id="container" data-width="@Model.PageWidth" data-url="/onlineSell/Default/Index">

<div class="search" style="padding-left:20px; padding-right:20px;">

<span>图书名称：@Html.TextBoxFor(a => a.bookName)</span>

<div class="toolbar">

<a class="opSearch">查询</a>

</div>

</div>

<div class="pager"></div>

<div class="showData noTable">

@{Html.RenderPartial("\_ShowData");}

</div>

</div>

</div>

1. 目录内“\_ShowData”代码如下：

@\*数据展示的公用视图\*@

@model rui.pagerBase

@{ string ctrl = ViewContext.RouteData.Values["controller"].ToString(); }

<input type="hidden" class="pagerinfo"

data-rowcount="@Model.RowCount"

data-pagesize="@Model.PageSize"

data-pageindex="@Model.PageIndex" data-pagecount="@Model.PageCount"

data-orderfield="@Model.OrderField" data-orderway="@Model.OrderWay" />

<ul style="margin-right:5px;">

@for (int i = 0; i < Model.dataTable.Rows.Count; i++)

{

string rowID = Model.dataTable.Rows[i]["rowID"].ToString();

string keyCode = Model.dataTable.Rows[i][Model.keyField].ToString();

<li>

<div>图书名称：@Model.dataTable.Rows[i]["bookName"]</div>

</li>

}

</ul>

1. 运行查看效果。



在li中完成图书完成信息的布局设计，要求展示图书图片，图书名称，价格，提供加入购物车按钮等。

## 购物车的实现

### 购物车的实现方案

购物车主要用来存放客户的选购商品信息，当用户结账时，可以查看购物车中包含的商品，并可以对购物车内的商品进行调整，最后完成下单操作。

购物车内的数据如何保存，都保存那些数据项目？

**方案1：**存放在数据库表中，此种方案要求用户采购的时候必须首先登陆才可以，此方案能够长久的保存用户加入购物车内的商品，当用户下单成功时，将数据库内保存的信息清除掉，淘宝采用的该方法。

考虑一下数据库表结构如何设计？

**方案2：**存放在状态对象Session中，此种方案，无需用户登录就可以添加商品，但是浏览器关闭后，购物车内的信息可能会丢失，百度采用的该方案。

**两种方案可以结合使用：**

当用户未登录的时候添加购物车，则采用方案2进行存储，当点击“下单”的时候，要求用户必须登录，登录后把方案2中存储的数据写入到数据库中转换为方案1。下单的时候，从数据库中获取购物车的信息，并完成下单。

### 方案1的实现思路

1）在数据库内构建存放购物车表

2）当单击“添加购物车”时，首先判断用户是否登录，如果未登录，跳到登录页让用户登录，如果登录过，则把商品信息加入到购物车表中（以前有，则累加数量，没有则新增）。

3）当单击“我的购物车”时，能够获取此用户已添加购物车的数据，并显示在购物车中，并提供删除，数量调整等相关功能。

4）当单击“下单”时，选择收货地址，完成下单逻辑。

### 方案2的实现思路

1) session\_Start事件中，创建Dictionary<string,int> shopping = new Dictionary<string,int>()对象,Key存放商品编码，Value存放采购商品数量，并将其存放到Session[“shopping”]中。

2) “添加购物车”时，从Session[“shopping”]中取出Dictionary<string,string>,判断是否存在添加的商品，如果存在，则累加数量，如果不存在，则添加进去，数量设置为1，再把Dictionary<string,string>存回Session中。

3) “我的购物车”时，从Session[“shopping”]中取出Dictionary<string,string>,利用其中保存的商品列表和商品数量生成购物车展示，并提供数量和删除操作。

4）当单击“下单”时，选择收货地址，完成下单逻辑。

## 下单逻辑

1）首先判断库存数据是否够，不够则提醒用户无法下单。

2）在数据库生成订单表数据，

3）在数据库中生成订单明细表数据，

**4）减去对应商品的库存，按照入库早晚进行处理。**

5）删除购物车信息

2,3,4,5的操作要求满足事务控制。

下单逻辑自己实现

|  |  |  |
| --- | --- | --- |
| 入库时间（升序） | 入库数量 | 已售数量 |
| 6-5 | 10 | 6 |
| 6-6 | 10 | 10 |

客户买：10本；

处理完后的结果：

|  |  |  |
| --- | --- | --- |
| 入库时间（升序） | 入库数量 | 已售数量 |
| 6-5 | 10 | 10 |
| 6-6 | 10 | 6 |

首先判断最早入库的

If((入库数量—已售数量）>=采购数量)

已售数量+=采购数量

采购数量=0

else

采购数量-=（入库数量—已售数量）

已售数量=入库数量

取下一笔的入库数量和已售数量，继续处理如上的逻辑

## 开发大致流程

1. 在业务类根目录新增client目录，内部创建view和bll子目录分别用来保存前端的视图模型和业务逻辑类。
2. 在网站根目录中创建client目录，用来保存与前端有关的图片和样式。
3. 在“dbLibrary\client\bll”目录中创建“loginHelper”辅助类，用来完成用户的登录和登录判断。
4. 在“dbLibrary\client\bll”目录中创建“bks\_Customer”类，用来完成用户的注册，用户信息变更等业务功能。
5. 在“dbLibary\client\bll\”目录中创建“bks\_ShoppingTrolley”，用来完成购物车数量变更，购物车商品移除等业务方法。
6. 在“dbLibary\client\bll\”目录中创建“bks\_OrderInfo”，用来完成下单，订单删除等业务方法。
7. 在“dbLibary\client\bll\”目录中创建“bks\_OrderDetail”，用来完成用户评价的提交。
8. 在“dbLibary\client\bll\”目录中创建“bks\_CustomerAddress”，用来完成用户地址的维护功能。
9. 在“dbLibary\client\view”目录中创建“bks\_Book”模型类，代码参考后台图书的。
10. 只需要提供三个搜索条件，图书类别是按照类别查看的时候用，出版社是按照出版社查看的时候用，图书名称是用户搜索的时候用。
11. 创建“onlineSell”区域，定制区域内的母版页。
12. 定制