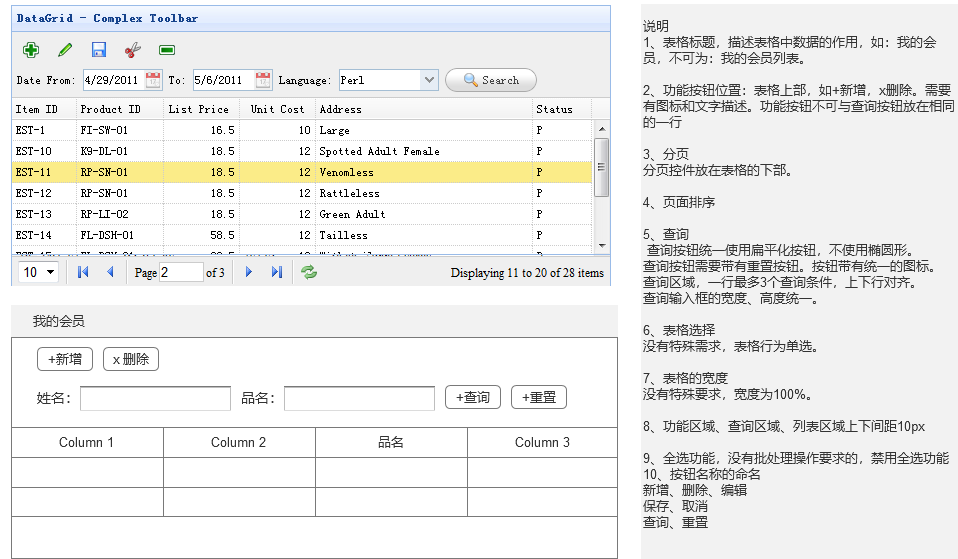
一、列表页面的排版与布局



1.引用的css:

<link href="../App\_Themes/xp/Style.css" rel="stylesheet" />

1. 列表页面布局html示例：

操作区：放置“新增”“删除”等操作按钮

查询区：放置查询条件和“查询”“重置”按钮，每行最多三个查询条件，另外重置按钮图标需要统一(//)统一采用：

列表区：内容长度不确定的列，如客户名称、报关品名等，统一采用左对齐。

设置nowrap:false，保证内容过长时，可以自动换行显示。



<body class="easyui-layout">

<div id="topBar">

<div id="tool">

<a id="btnAdd" href="javascript:void(0);" class="easyui-linkbutton" data-options="iconCls:'icon-add'" onclick="Add()">新增</a>

</div>

<div id="search">

<ul>

<li>

<span class="lbl">国家地区代码:</span>

<input class="easyui-textbox search" id="Code" />

<span class="lbl">中文名称: </span>

<input class="easyui-textbox search" id="Name" />

<span class="lbl">英文名称: </span>

<input class="easyui-textbox search" id="EnglishName" />

<a id="btnSearch" href="javascript:void(0);" class="easyui-linkbutton" data-options="iconCls:'icon-search'" onclick="Search()">查询</a>

<a id="btnReset" href="javascript:void(0);" class="easyui-linkbutton" data-options="iconCls:'icon-redo'" onclick="Reset()">重置</a>

</li>

</ul>

</div>

</div>

<div id="data" data-options="region:'center',border:false">

<table id="datagrid" title="国家地区列表" class="easyui-datagrid" style="width: 100%; height: 100%" toolbar="#topBar"

singleselect="true" fitcolumns="true">

<thead>

<tr>

<th field="Code" data-options="align:'center'" style="width: 150px">国家地区代码</th>

<th field="Name" data-options="align:'center'" style="width: 100px">名称</th>

<th field="EnglishName" data-options="align:'center'" style="width: 200px">英文名称</th>

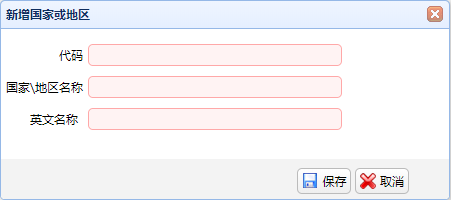
</tr>

</thead>

</table>

</div>

1. 新增\编辑页面排版与布局



详情窗口：不要“取消”,只保留关闭按钮

<body class="easyui-layout">

<div id="content">

<form id="form1" runat="server">

<table id="editTable">

<tr>

<td class="lbl">代码</td>

<td>

<input class="easyui-textbox input" id="Code" name="Code"

data-options="required:true,validType:'length[1,4]',tipPosition:'bottom',missingMessage:'请输入代码'" />

</td>

</tr>

<tr>

<td class="lbl">国家\地区名称</td>

<td>

<input class="easyui-textbox input" id="Name" name="Name"

data-options="required:true,validType:'length[1,50]',tipPosition:'bottom',missingMessage:'请输入国家或地区名'" />

</td>

</tr>

<tr>

<td class="lbl" style="padding-right: 10px">英文名称</td>

<td>

<input class="easyui-textbox input" id="EnglishName" name="EnglishName"

data-options="required:true,validType:'length[1,50]',tipPosition:'bottom',missingMessage:'请输入英文名'" />

</td>

</tr>

</table>

</form>

</div>

<div id="dlg-buttons" data-options="region:'south',border:false">

<a id="btnSave" class="easyui-linkbutton" data-options="iconCls:'icon-save'" onclick="Save()">保存</a>

<a class="easyui-linkbutton" data-options="iconCls:'icon-cancel'" onclick="Close()">取消</a>

</div>

</body>

1. 新增\编辑页面表单验证

1.表单提交前的验证：

if (!$("#form1").form('validate')) {

return;

}

控件设置：注意可输入最大、最小长度、数据类型的验证，提示信息。

data-options="required:true,validType:'length[1,50]',tipPosition:'bottom',missingMessage:'请输入币种名称'" />

data-options里面的基础验证不通过，弹出提示信息，不需要再弹出提示窗口；

复杂的业务验证，根据实际情况，可以弹窗提示。

2.提示信息显示位置：

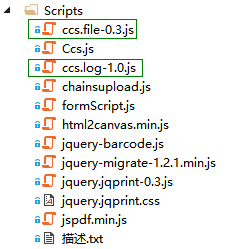
提示信息要在窗口里面完整显示，如果左边或右边显示不开，则将提示信息显示在输入框的下方



1. JS脚本命名规范

引入的js类库或自行封装的js插件统一放到Scripts目录下

命名规范为”ccs.” + 功能 + “-” + 版本号，如”ccs.log-1.0.js”（或”wl.” + 功能 + “-” + 版本号，如”wl.log-1.0.js”）

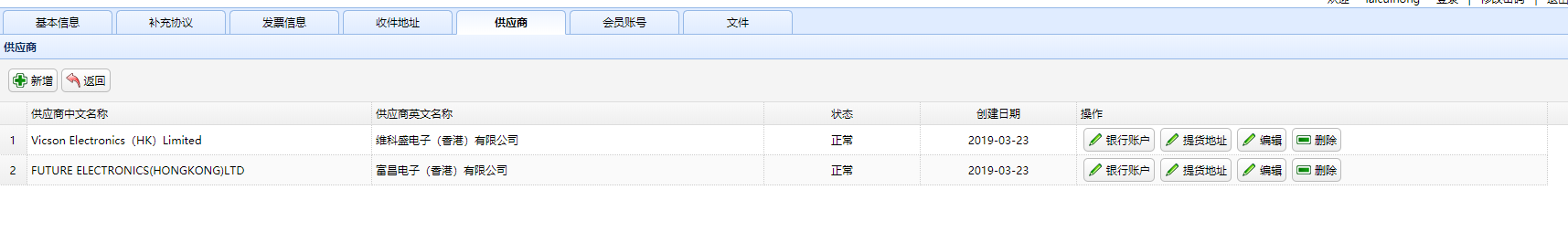


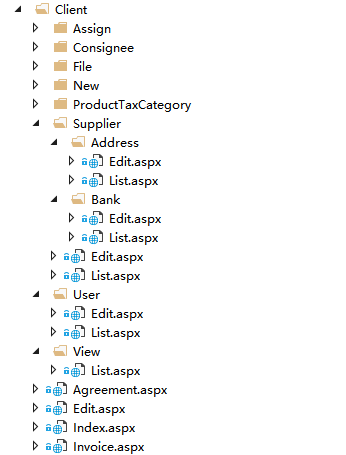
1. 代码目录结构与命名规范

1.树形目录结构：

根据具体的业务功能或使用者的角色，建立对应的文件夹，添加列表界面、编辑界面或详情界面等。

比如会员这边，包含供应商、收件地址，则分别建立文件夹“Supplier”、“Consignee”，供应商又包含提货地址、银行账户，则在“Supplier”文件夹下再创建“Address”和“Bank”文件夹，在对应的文件夹下根据实际业务需求添加列表/编辑界面



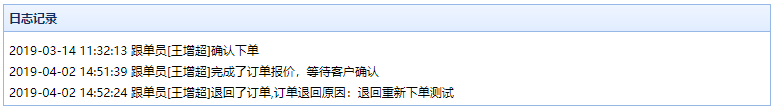


2.命名规范：

列表界面为“List.aspx”，编辑界面为“Edit.aspx”。

如供应商银行账号的列表界面，只需要在Client/Supplier/Bank文件夹下添加“List.aspx”即可，不能命名为“ClientSupplierBankList.aspx”，因为页面已经在对应目录下了，文件名不要再重复层级结构名称。

1. 公共组件
2. 日志记录：



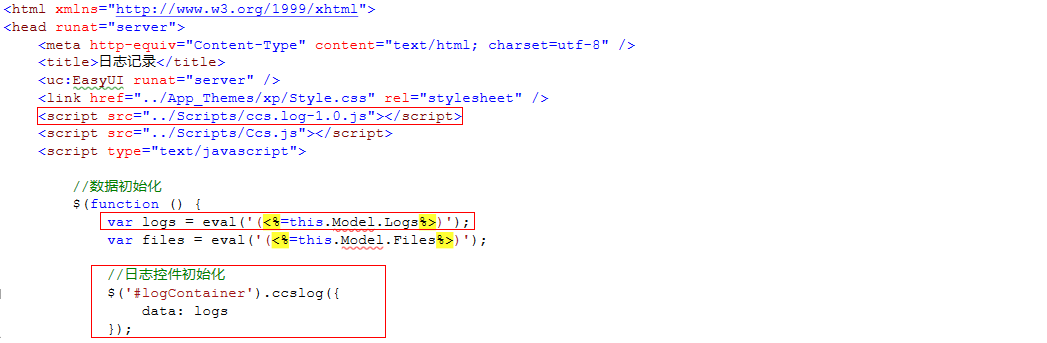
1>获取日志记录



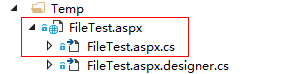
2>页面添加一个div



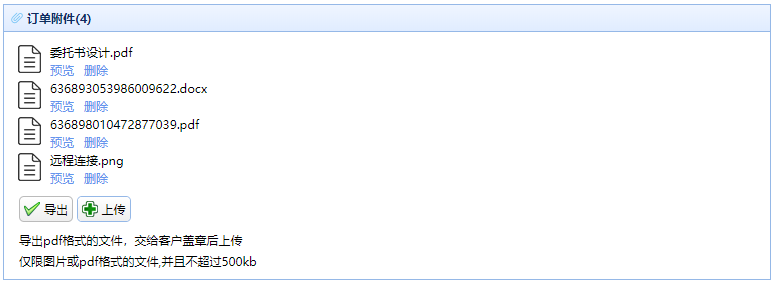
3>引入ccs.log-1.0.js，将日志记录绑定到控件



代码详见：

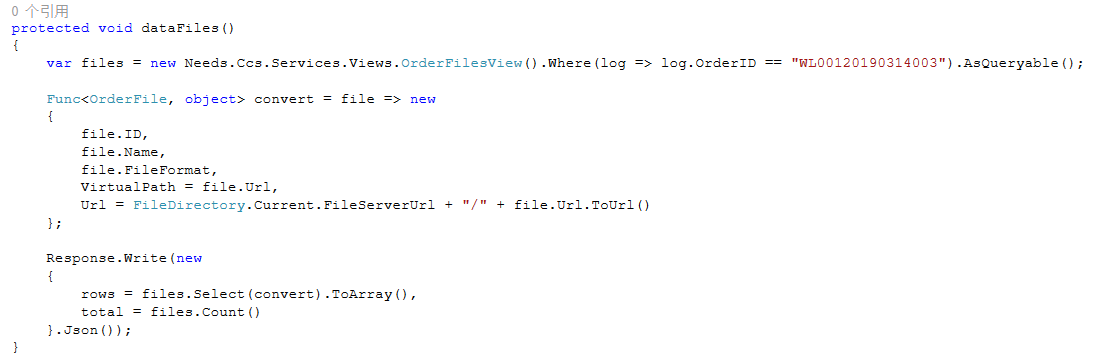


2.附件管理：

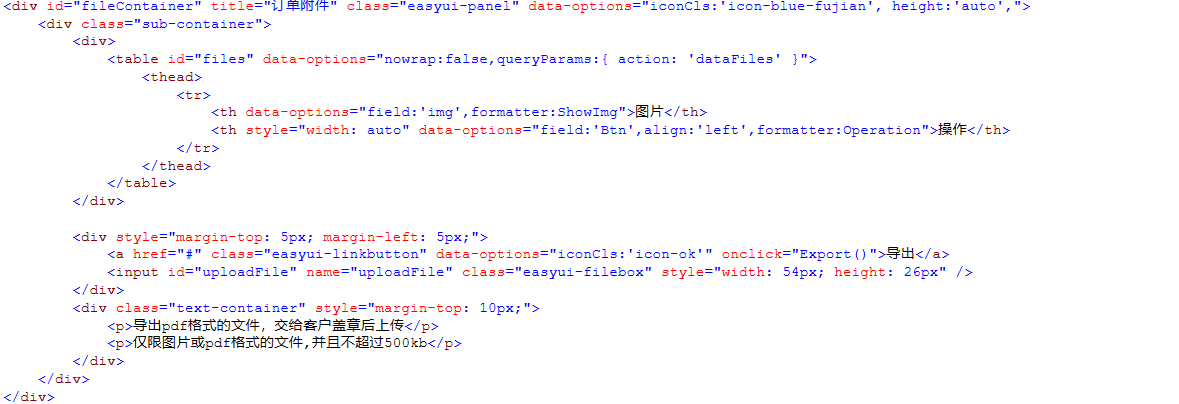


目前借助easyui datagrid实现，后续可能封装独立的控件

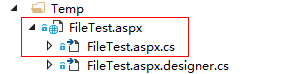
1>初始加载文件数据



2>页面布局

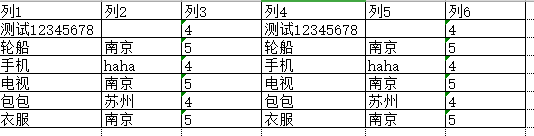


3>其它代码详见：

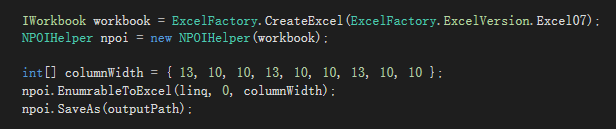


3.Excel组件（支持多个sheet）

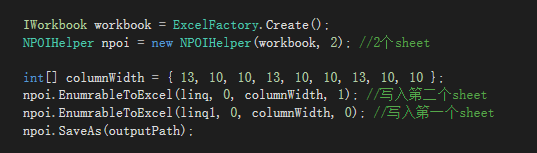
1>基础版excel导出效果

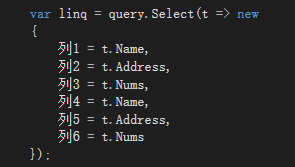


1. 业务端调用。

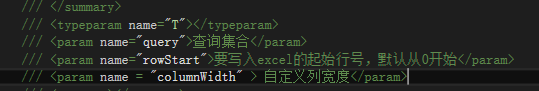


多个sheet

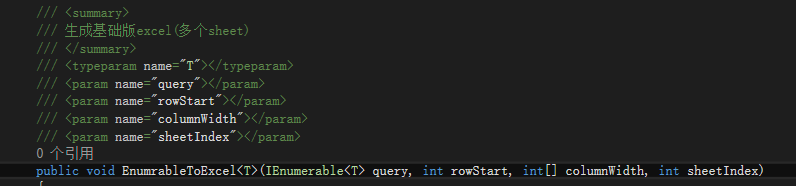


参数：

3>封装代码



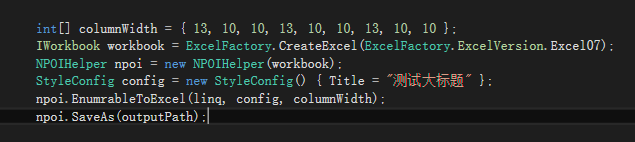




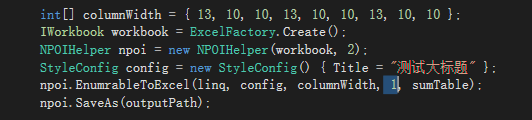
4>大标题样式Excel导出效果

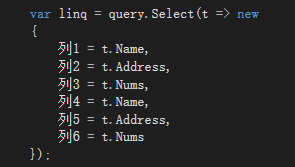


5>业务端调用



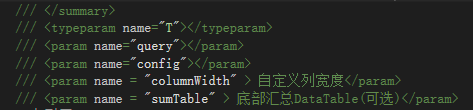
多个sheet

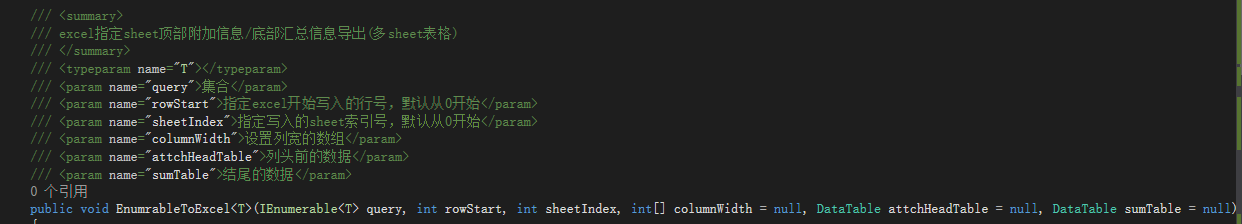


参数：

1. 封装代码



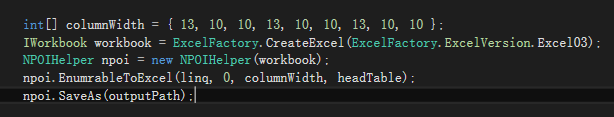


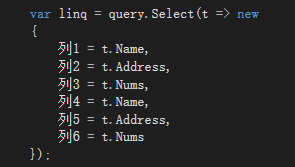


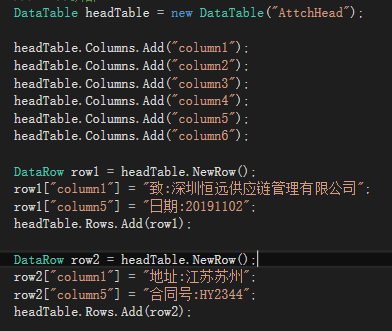
1. 顶部附加信息Excel导出效果



1. 业务端调用



参数：linq:



headTable: 顶部附加信息table设计：

内容出现在excel

第一行第一列，

第一行第五列，

第二行第一列，

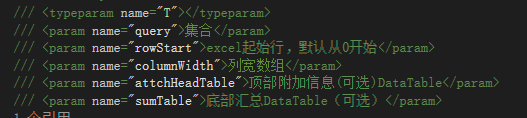
第二行第五列。

dataTable也要设计成两行五列。

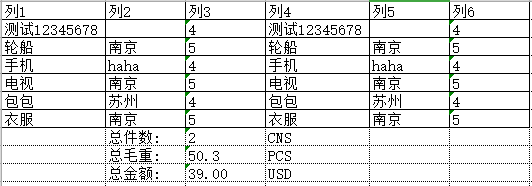
且第一列和第五列有值。

1. 封装代码

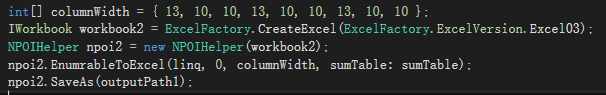


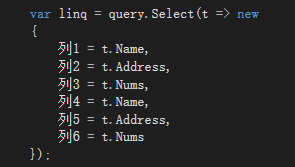


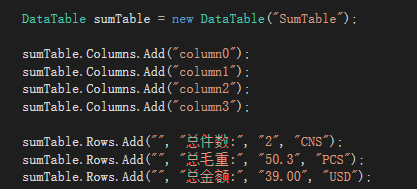
1. 底部汇总信息excel导出效果



1. 业务端调用



参数：linq



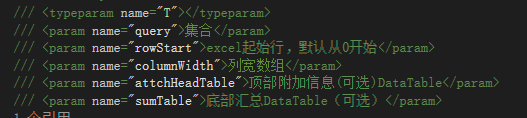
sumTable :如效果图显示。

底部汇总DataTable是三行四列

且第一列是空值。

1. 封装代码

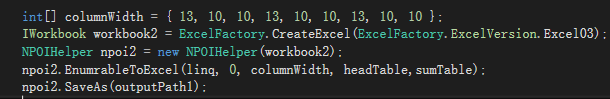




1. 顶部附加+底部汇总 excel导出效果



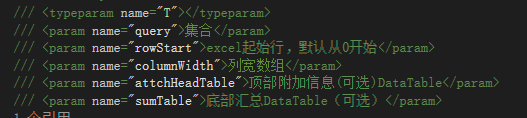
1. 业务端调用



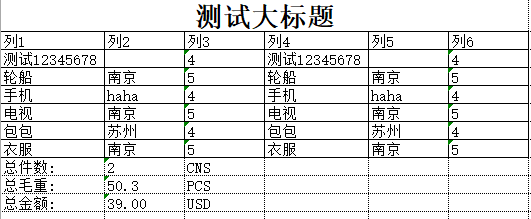
参数参考11> 和 8>

1. 封装代码

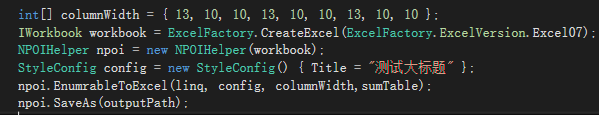




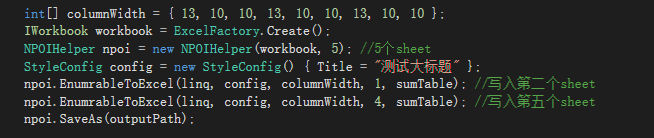
1. 大标题+底部汇总 excel导出效果。



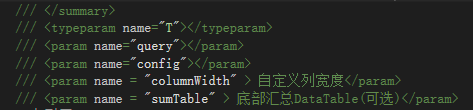
1. 业务端调用

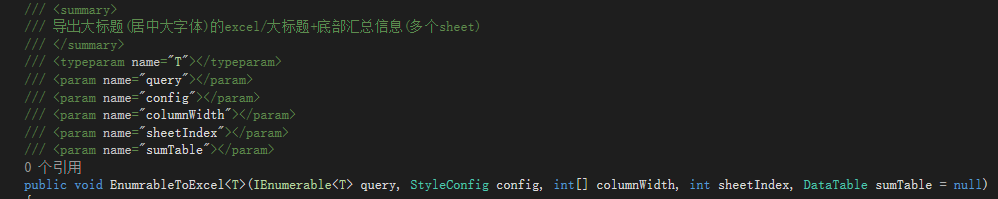
linq, sumTable参考11>

多个sheet：



1. 封装代码





1. 根据上传模板生成excel导出效果

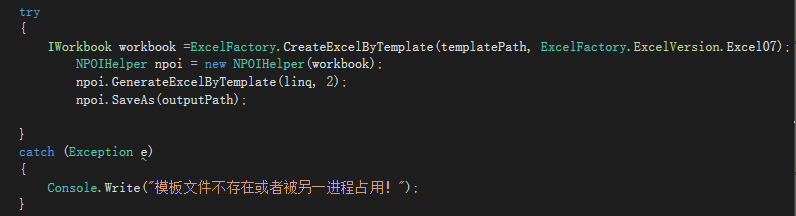
模板：比较复杂的样式



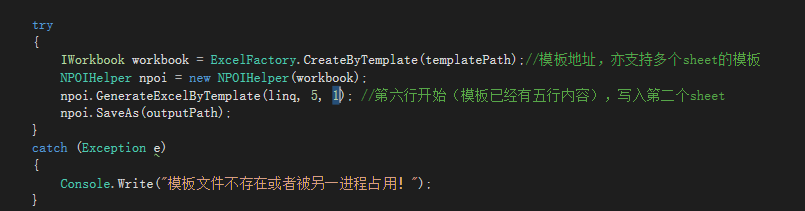
效果excel:

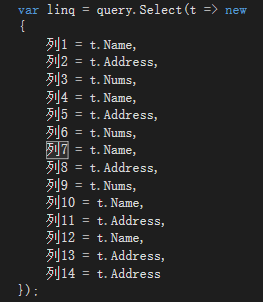


1. 业务端调用



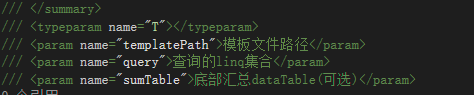
多个sheet模板。Sheet index默认从0开始



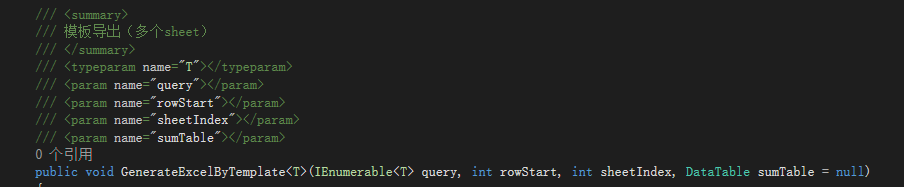
参数linq:

1. 封装代码





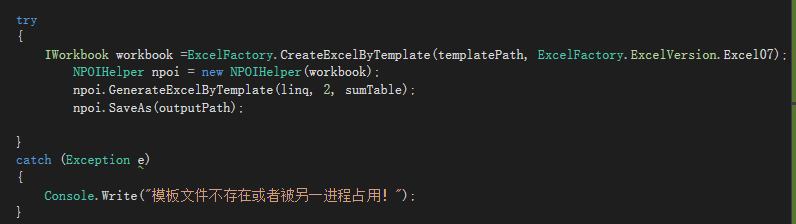
多个sheet代码封装。



1. 根据上传模板生成+底部汇总excel导出效果。



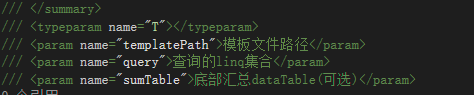
1. 业务端调用

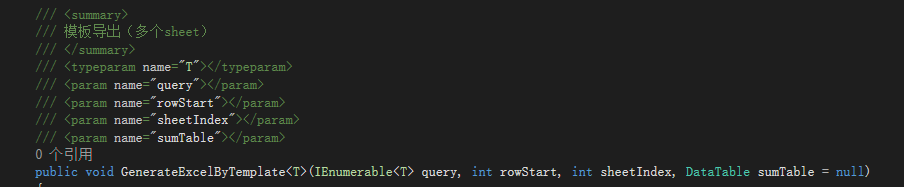


参数：linq 参考20>, sumTable 参考11>

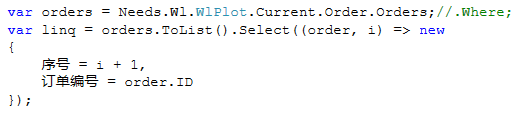
1. 封装代码



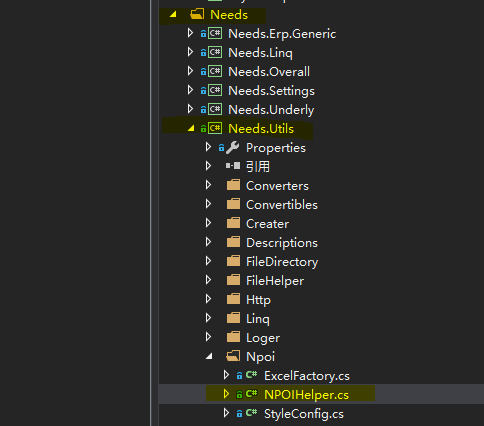




1. 导出excel需要序号的，使用下面代码，

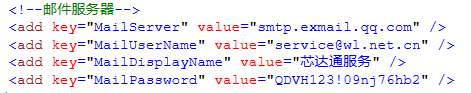


1. 详细代码：



4.邮件服务：

1>Web.config配置



2>发送邮件



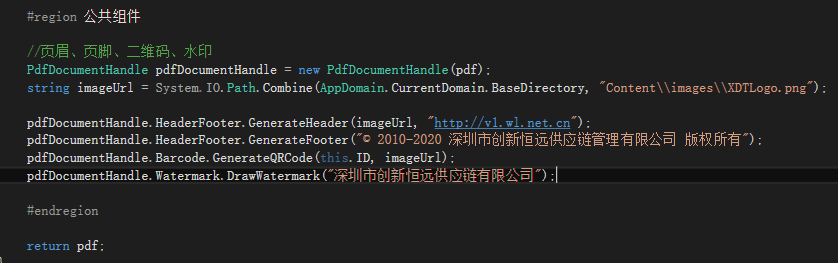


1. 短信服务：

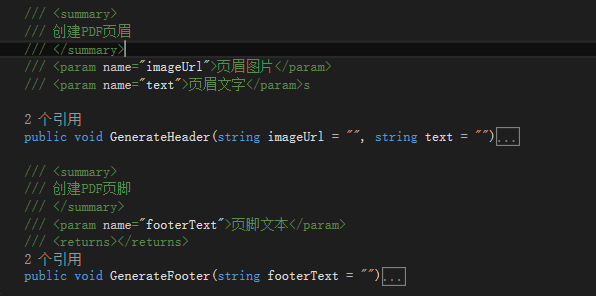
1>发送邮短信

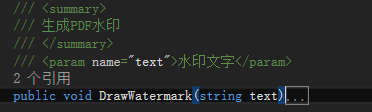


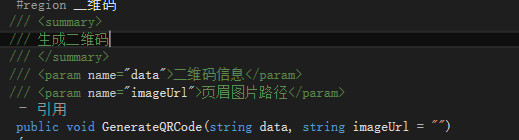
1. PDF组件
2. 业务端代码



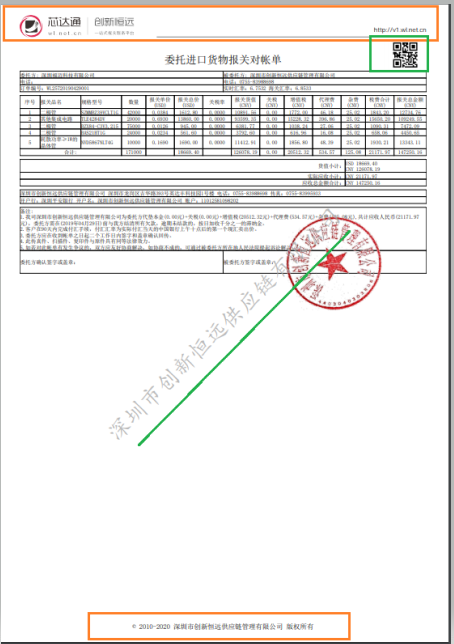
1. 封装代码





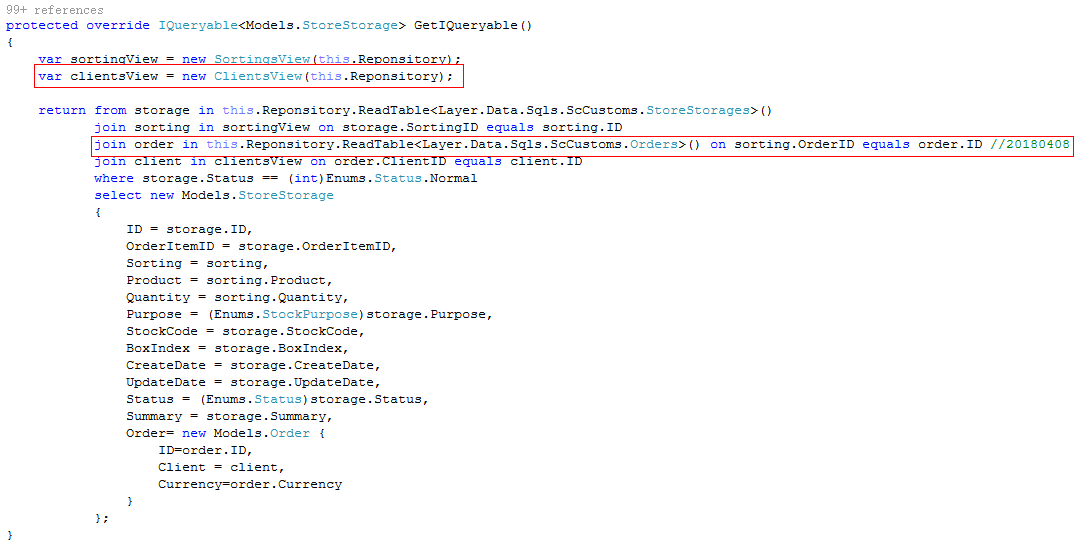


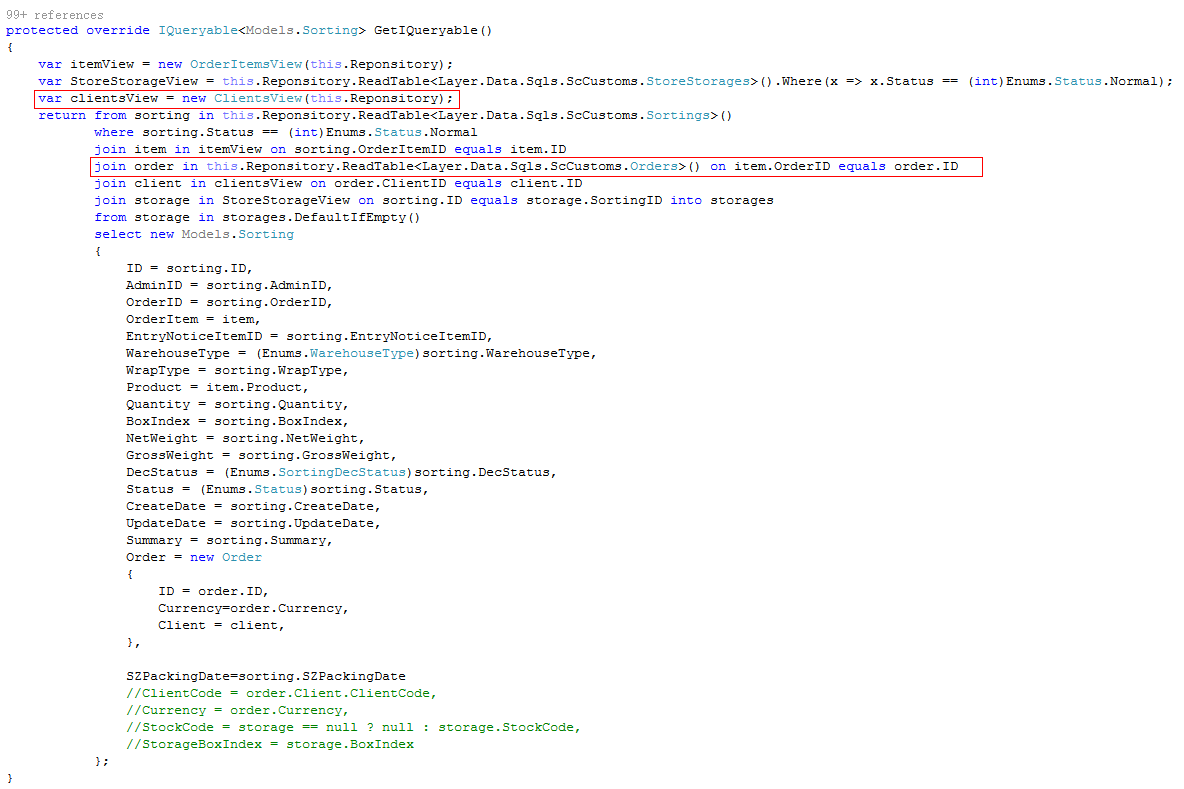
1. Pdf效果图（图片有压缩，和实际产生的稍有点差别。）



1. 开发规范
2. 视图(View)：
3. 目前项目中很多的视图查询存在重复查询的问题，这会导致查询性能的下降，甚至出现循环引用的问题。

比如StoreStoragesView查询了SortingsView、ClientsView和Order，而SotingsView也查询了ClientsView和Order，这就导致了重复查询。





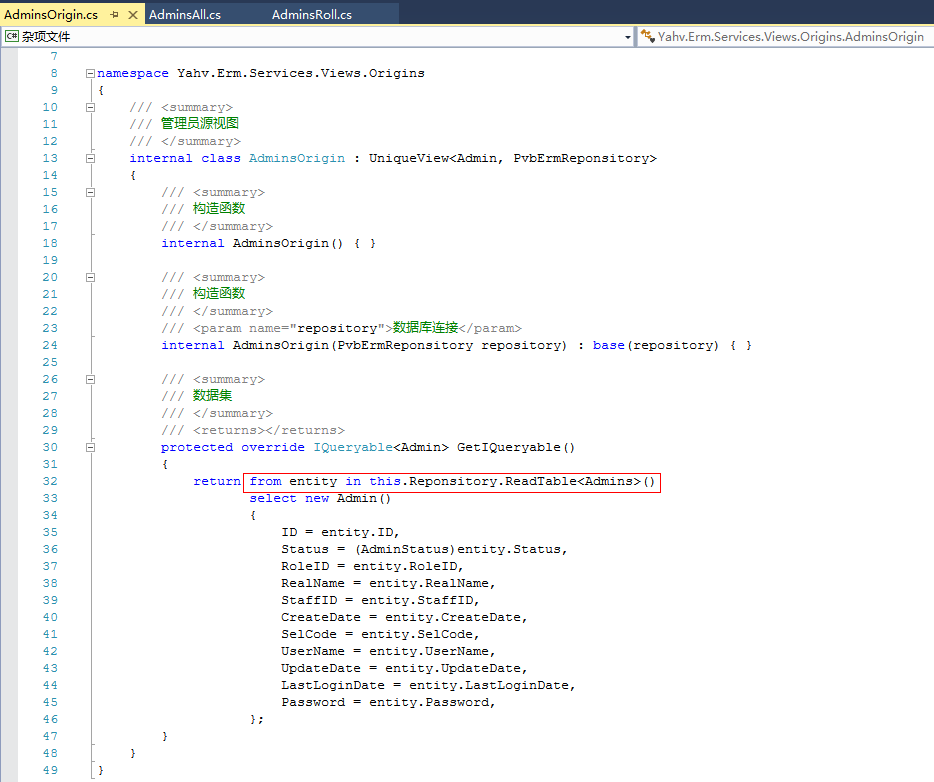
解决方案：

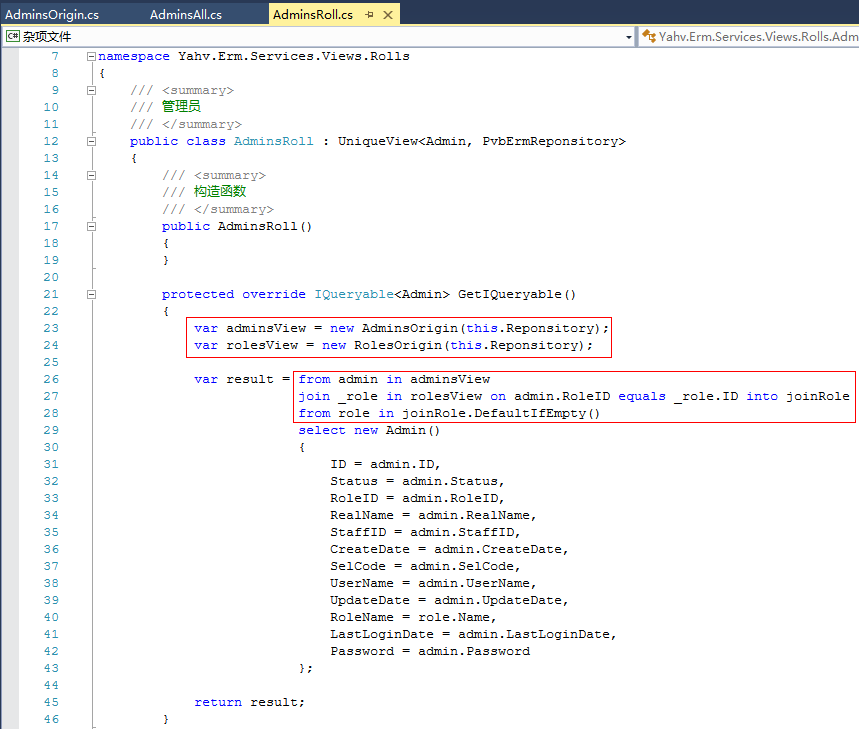
1. StoreStoragesView不直接查询SortingsView，而是改为读取Sortings表



1. 参考北京技术部目前的开发方式

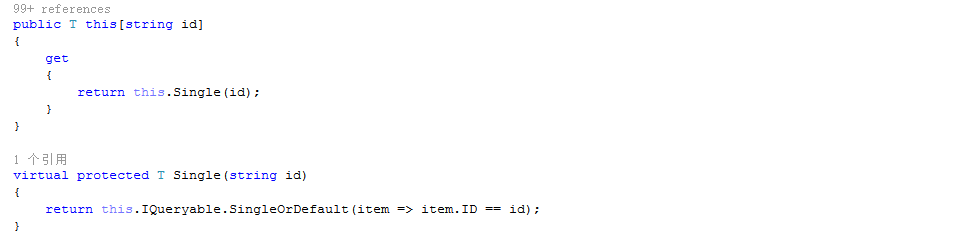
xxxOrigin是与数据库表一一对应的视图，不关联任何其他视图或表；xxxRoll是与前端业务展示所对应的视图，根据具体的业务需求关联对应的xxxOrigin视图





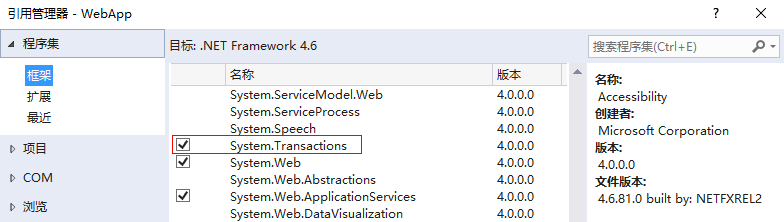
1. 数据库表为唯一主键的，对应视图应从UniqueView继承，数据库表为联合主键的，对应视图从QueryView继承。

UniqueView与QueryView的区别是UniqueView提供了索引器，而QueryView没有，因为QueryView对应数据库表采用了联合主键，一个ID是不能唯一确定一条记录的。



1. 事务：

项目中引用”System.Transactions”



1. 开启事务

业务代码写在using (TransactionScope ts = new TransactionScope()){}中，业务代码完成后调用ts.Complete();

using (TransactionScope ts = new TransactionScope())

{

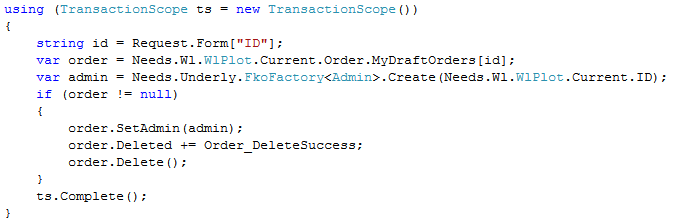
//业务逻辑代码

... ...

ts.Complete();

}

比如：



1. 异常处理

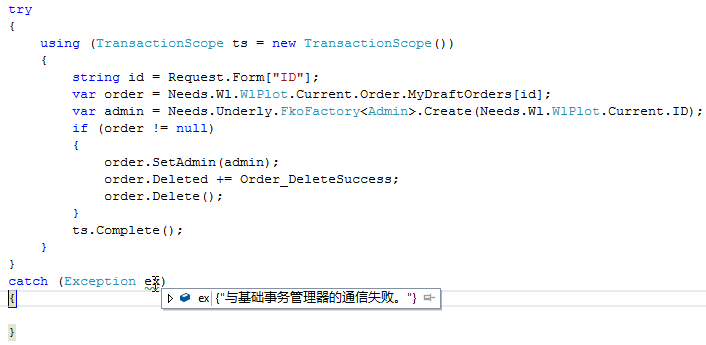
（1）”已禁用对分布式事务管理器（MSDTC）的网络访问。请使用组件服务管理工具启用DTC以便在MSDTC安全配置中及性能网络访问。”



解决方案：

<https://www.cnblogs.com/jianxuanbing/p/7242254.html>

（2）”与基础事务管理器的通信失败。”

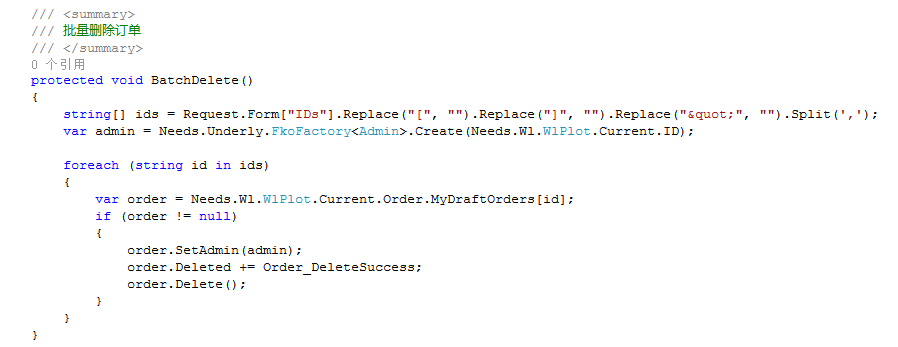


解决方案：

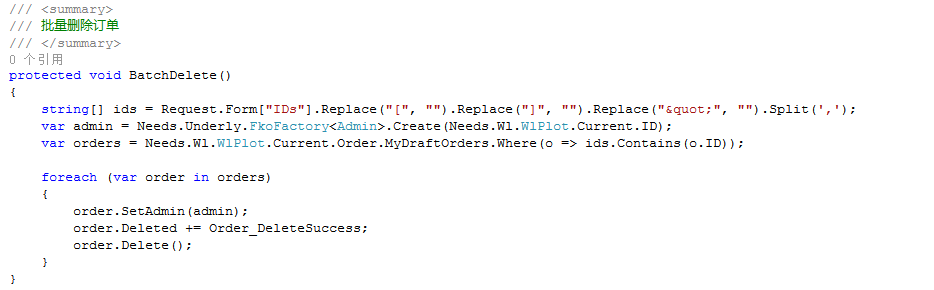
未解决，修要修改数据库服务器的配置

1. 批量操作：
2. 循环的正确写法：

错误写法：每次循环都根据订单id去查询对应订单



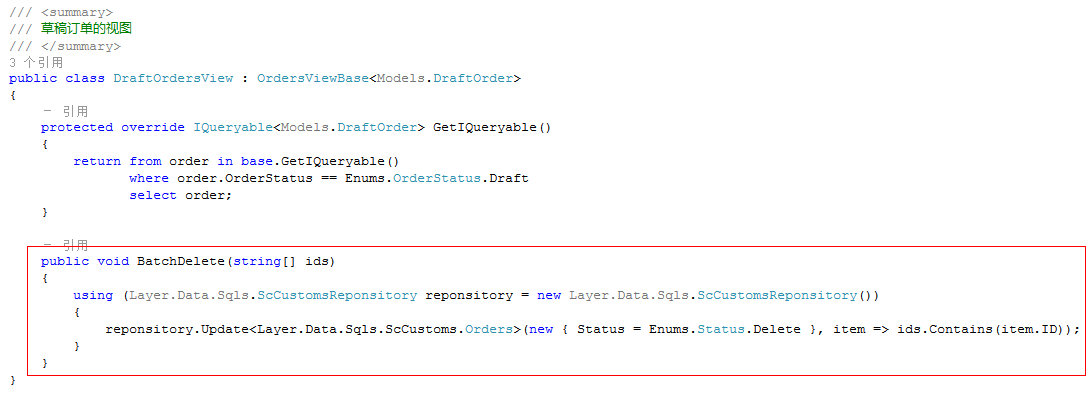
正确写法：先查出所有订单，再对订单进行遍历操作



1. 批量操作可以在对应的视图中写一个批量处理方法，提升性能。

比如批量删除订单，如果直接在前端的后台，查出Order对象，调用Delete()方法，每个Delete方法的内部处理都会新开数据库连接，导致性能低下。

可以在OrdersView里添加一个批处理方法BatchDelete(string[] ids)



1. 分页优化：
2. 目前的分页都是基于一个最终的复杂查询进行分页的

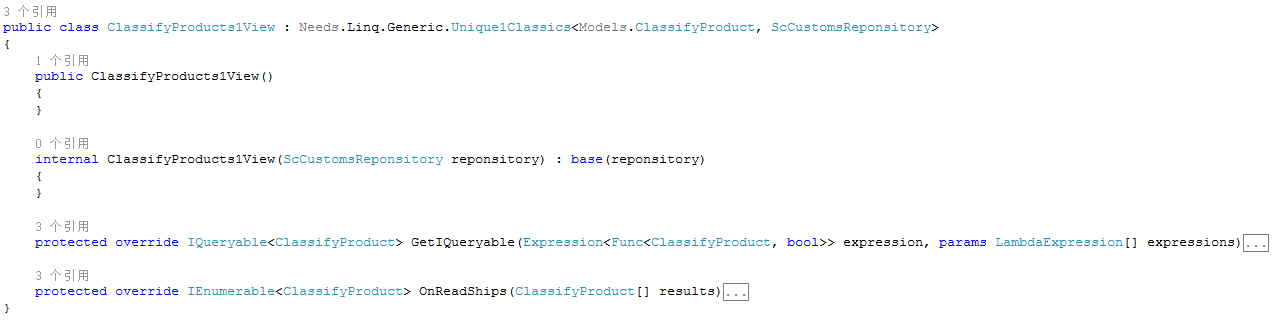






这样当产品上线一段时间，数据量变得很大时，会存在一定性能问题，最好先根据主表查询分页，再关联其他的相关表或视图





1. 常量配置：
2. 常量信息，比如公司信息等必须写在配置里，代码里统一引用，不能在代码里写死!否则后面需要更换相关信息非常麻烦

