# 一、概述

很多地方需要将查询的结果导出为文件，比如Excel文件；现在Oversea的Silverlight框架的DataGrid有提供了一个导出当前页的功能，并且已经实现了；但对于导出全部（比如查询结果有分页的情况，但需要一次把所有页的结果导出），就只有一个按钮和对应的点击事件，但没有真正的导出的实现；而本文档就将介绍和说明，在ECCentral项目中，如何来实现这个导出全部的功能实现。

# 二、基本使用说明

## 1. 先看一个使用例子

在Portal端有提供了一个RestClient的扩展方法ExportFile来导出数据文件，先看一个代码片段例子：我们会在Facade里写一个方法，方法里包含如下代码：

|  |
| --- |
| // 1. 设置导出数据来源的NoBizQuery Restful Service，注意必须是NoBizQuery的，即只能返回QueryResult或QueryResultList  // 可以是和UI上的查询公用同一个NoBizQuery Restful Service  string relativeUrl = "/RMAService/RMATracking/Query";  // 2. 设置调用NoBizQuery Restful Service所需要的入参（也就是查询条件）  RMATrackingQueryFilter queryFilter = queryVM.ConvertVM<RMATrackingQueryVM, RMATrackingQueryFilter>();  queryFilter.CompanyCode = CPApplication.Current.CompanyCode;  queryFilter.PagingInfo = new PagingInfo  {  PageSize = ConstValue.MaxRowCountLimit,  PageIndex = 0,  SortBy = string.Empty  };  // 3. 设置需要导出到Excel里的列，以及列的相关属性（如列标题、列宽度、格式字符串、样式等），  // 例子里的CustomerSysNo、CustomerID等就是NoBizQuery Restful Service查询出的DataTable里的Column的ColumnName，  // 如果这里设置的ColumnName在Restful Service查询出的DataTable里不存在，那么执行ExportFile方法后会报出相关异常，  // **注意，这里只是一个例子，真实代码里，列名都不是写死的字符串，应该都是从资源文件里取出的，这样支持多语言文本**  ColumnSet set = new ColumnSet()  .Add("CustomerSysNo", "顾客系统编号")  .Add("CustomerID", "顾客账号")  .Add("CustomerName", "顾客姓名")  .Add("Email", "顾客邮箱")  .Add("RegisterTime", "注册时间", "yyyy年MM月dd HH:mm:ss");  // 4. 传入前面三步设置的变量，执行导出数据的动作，该方法执行完后就会弹出下载保存的提升框，让用户下载保存数据文件  restClient.ExportFile(relativeUrl, queryFilter, new ColumnSet[] { set }); |

最后在UI上的View的cs文件里，在“导出全部”动作的处理事件里调用上面的Facade方法即可；上面的注释已经有大概说明了这段代码的逻辑和作用；下面会主要介绍一下ColumnSet的相关方法和ExportFile方法的相关重载；

## 2. 用来设置导出列的属性的类ColumnSet

ColumnSet主要用来保存对导出列及其相关属性的设置用的，将会被导出方法所使用（会被传送到Service端，用来控制最终的所导出的文件里包含了那些数据列，以及每一列如何来生成）；  
为一个导出的数据文件设置导出列的方式有两种：

(1). 调用Add方法，手工向ColumnSet对象里添加要导出的数据列信息；

|  |
| --- |
| ColumnSet set = new ColumnSet()  .Add("CustomerSysNo", ResRMATracking.Grid\_CustomerSysNo)  .Add("CustomerID", ResRMATracking.Grid\_CustomerID)  .Add("CustomerName", ResRMATracking.Grid\_CustomerName)  .Add("Email", ResRMATracking.Grid\_Email)  .Add("RegisterTime", ResRMATracking.Grid\_Register, ResRMATracking.Excel\_Format\_Register); |

Add方法总共有9个重载，主要包含如下参数：

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 必传 | 参数说明 |
| fieldName | string | 是 | 指定Service端查询出结果集的列名 |
| title | string | 是 | 该列在导出文件中的列标题 |
| valueFormat | string | 否 | 用来格式化数据的字符串，**注意**：对于查询结果待导出数据集中类型为DateTime、数值类型的数据，该字符串是用作{真实数据}.ToString(string format)的参数，而对于其他类型的数据，该字符串时用作string.Format(string format, object v)的第一个参数，真实数据会作为第二个参数来生成最终的导出到文件里的显示文本；注意两种format格式字符串的写法是不一样的； |
| width | int | 否 | 设置导出文件中，该列的宽度 |
| horizontalAlignment | 枚举 | 否 | 设置导出文件中，该列数据单元格中水平对齐的方式 |
| verticalAlignment | 枚举 | 否 | 设置导出文件中，该列数据单元格中垂直对齐的方式 |
| hasBorder | bool | 否 | 设置导出文件中，该列数据单元格是否显示边框 |

通过Add方法加入到ColumnSet的列的显示是将跟随在当前已经加入的所有列的最后，所以列的显示顺序和通过Add方法加入的先后顺序保持一致的；如果Add了重复的fieldName，那么运行时会抛出相应的异常；

(2). 直接先根据页面上显示查询结果的DataGrid来生成导出列，然后如果需要的话可以再调用Insert或Set或Remove方法来人工再调整列信息；

|  |
| --- |
| ColumnSet set = new ColumnSet(this.DataGrid\_Query\_ResultList)  .Insert(5, "UpdateUser", ResRMATracking.Grid\_UpdateUser)  .Insert(0, "SysNo", ResRMATracking.Grid\_CustomerID)  .Remove("Email")  .Add("Rank", ResRMATracking.Grid\_Rank)  .SetWidth("CustomerName", 50)  .SetFormat("RegisterTime", ResRMATracking.Excel\_Format\_Register)  .Set("TotalAmout", ResRMATracking.Excel\_Format\_TotalAmout, 30); |

* Insert方法和Add方法类似，用来添加导出列，两者区别在于Add方法只能在当前导出列最后来追加列，而Insert方法则可以通过其方法的第一个参数index来控制所插入的列的显示位置（从0开始排）；
* Set方法是用来对已经加入ColumnSet的待导出列的信息进行修改的，根据fieldName来查找到相应列以做修改，如果名为fieldName的列在ColumnSet中不存在，那么会抛出相应的异常；
* Remove方法是根据fieldName在ColumnSet中找到对应的列，并将其从ColumnSet中删除，如果名为fieldName的列在ColumnSet中不存在，方法也会正常执行返回，并不会抛异常；

## 3. 真正负责导出的方法：RestClient的扩展方法ExportFile

该方法总共有4个重载：

|  |
| --- |
| public static class FileExporter  {  // Get方式  public static void ExportFile(this RestClient restClient, string relativeUrl, ColumnSet[] columns = null)  public static void ExportFile(this RestClient restClient, string relativeUrl, string exporterName, ColumnSet[] columns = null)  // Post方式  public static void ExportFile(this RestClient restClient, string relativeUrl, object condition, ColumnSet[] columns = null)  public static void ExportFile(this RestClient restClient, string relativeUrl, object condition, string exporterName, ColumnSet[] columns = null)  } |

前2个方法是一组，后2个方法是一组，基本用法很类似于RestClient的Query方法的两个重载；  
当负责读取生成导出数据的NoBizQuery Restful Service是采用的Get方式时，就使用前2个方法；而如果负责读取生成导出数据的NoBizQuery Restful Service是采用的Post方式时，就使用后两个方法，参数object condition就为Post的参数（一般都是NoBizQuery的查询Filter对象）；

|  |  |  |  |
| --- | --- | --- | --- |
| 参数名 | 类型 | 必传 | 参数说明 |
| restClient | RestClient | 是 | Facade中构建好的用来负责和Restful Service交互的RestClient对象 |
| relativeUrl | string | 是 | NoBizQuery Restful Service的相对Url |
| condition | object | 否 | 如果调用的没有该对象的重载方法，那就使用Get方式请求Restful Service，否则将使用Post方式请求Restful Service；该参数就表示需要Post到NoBizQuery Restful Service的数据； |
| columns | ColumnSet[] | 否 | 指定导出数据文件中数据列的设置；注意这里是一个ColumnSet的数组，因为我们的NoBizQuery Restful Service的返回数据有可能是多个DataTable（当返回QueryResultList的时候），所以每个DataTable都会有一个ColumnSet来配置其导出列，所以这里的数组将对应返回数据的DataTable的数组（数组index做对应）；如果一个DataTable没有对应的ColumnSet，那么会默认将查询结果的DataTable的所有Column都导出（以ColumnName为导出的列的Title）； |
| exporterName | string | 否 | 如果没有设置该参数，那么默认使用Excel的导出；另外我们可以在Service端编写一个自己的数据文件导出的实现类，实现接口ECCentral.Service.Utility.IFileExport，然后通过配置文件配置好，并在配置文件中设定一个唯一的name，也就对应到这里的这个参数，这样通过该参数我们可以决定Service端使用哪一个文件导出类，从而实现更灵活的扩展，可以导出其他文件格式并完全自定义导出的格式，比如word、pdf等，只要实现相应的类即可；下面第三大点扩展使用介绍里将会详细说明这部分； |

# 三、扩展使用

## 1. 整体概述

这里的扩展主要是指在Service端，可以扩展定义相应的数据导出实现逻辑，并通过配置就可以来完全控制数据导出，实现自定义的数据导出方式；Service端文件导出模块的主要结构如下：  


"负责数据导出的Service" 将负责接收Portal的调用，然后到NoBizQuery Restful Service获取相应的数据，然后组装成一个标准的DataTable；然后调用FileExporterFactory的CreateExporter方法，读取配置并根据Portal端传来的exporterName来创建出一个IFileExport接口的具体实现类的对象实例（默认情况下将创建ExcelFileExporter类型的对象实例），再调用该对象的接口方法CreateFile，把之前准备好的数据DataTable以及Portal端传来的导出列设置信息（即前面讲到的Portal端构造的ColumnSet）传进去，由这个IFileExport接口的具体实现类来负责生成最终要下载的文件的byte[]，接着就可以将文件保存到硬盘上WebHost所在目录下的相关目录里，然后将文件的url返回Portal端，以供Portal程序自动打开浏览器去下载；

所以这里的扩展点，主要就是对IFileExport接口的扩展实现，并通过FileExporter.config文件来配置注入新的实现类；

对于扩展实现，可以从IFileExport派生，完全重新来实现整个所要导出的文件的生成过程，这样可以控制导出任意格式的文件，而不仅仅是Excel文件；

另外如果是要导出Excel文件但当前的ExcelFileExporter可能某些地方不合要求，那么也可以从现有的ExcelFileExporter类来派生，然后override重写ExcelFileExporter中的部分protected virtual方法来实现定制化的特殊需求，而大部分的生成Excel文件的代码部分则可以重用了；

## 2. 从接口IFileExport扩展

接口IFileExport的定义如下：

|  |
| --- |
| public interface IFileExport  {  byte[] CreateFile(List<DataTable> data, List<List<ColumnData>> columnSetting, out string fileName);  } |

接口仅一方法，根据传入的数据和列设置信息，来生成文件的byte[]，以及文件名（该文件名将会作为保存到硬盘上的文件的文件名，也会是Portal端下载是看到的文件名）；  
第一个参数data代表了一组DataTable（前面有讲过了，我们的NoBizQuery是可以返回QueryResultList的）；  
第二个参数columnSetting代表了一组的列设置信息，这个是从Portal端构造后传过来的；

## 3. 从ExcelFileExporter类扩展

ExcelFileExporter类的扩展点如下：

|  |
| --- |
| public class ExcelFileExporter : IFileExport  {  // 1. 层次一  byte[] CreateFile(List<DataTable> data, List<List<ColumnData>> columnSetting, out string fileName)  // 2. 层次二  protected virtual string SetFileName(List<DataTable> data)  protected virtual int MaxDataTableRowCountLimit { get; }  protected virtual bool ThrowExceptionWhenDataTableRowCountExceedLimit { get; }  protected virtual Exception BuildBizException(string msg, int queryResultRowCount)  protected virtual void BuildSheet(Worksheet worksheet, int sheetIndex, DataTable data, List<ColumnData> columnSetting)  // 3. 层次三  protected virtual HorizontalAlignments SetDefaultHorizontalAlignmentsForType(Type type)  protected virtual void SetSheetColumnsWith(Worksheet worksheet, int sheetIndex, List<ColumnData> columnSetting)  protected virtual void SetXFForHeader(int sheetIndex, int excelColumnIndex, XF xf)  protected virtual void SetXFForDataCell(int sheetIndex, int excelRowIndex, int excelColumnIndex, XF xf)  protected virtual object FormatCellValue(object value, string format, int sheetIndex, int excelRowIndex, int excelColumnIndex)  } |

上面的方法可以分为三个层次，

(1). 第一个层次的方法为接口方法CreateFile，它里面会调用第二个层次的方法；重写该方法的话，基本上等于重新实现接口了，只是说可能有些protected方法可以拿来用，但建议没有特别需要的话，不用重写该方法；

(2). 第二个层次的方法（或属性）有5个，都会被CreateFile方法所调用：

* **SetFileName方法**：用来设置导出文件的文件名的，入参为NoBizQuery的数据，也就是所可以根据查询出的数据来生成文件名，ExcelFileExporter该方法的实现是按照当前时间来生成文件名的，和要导出的数据无关；
* **MaxDataTableRowCountLimit属性**：用来设置一个Sheet允许导出的最大记录条数，ExcelFileExporter该属性的实现是从配置文件FileExporter.config中读取，如果配置文件中没有配置，则默认返回10000；
* **ThrowExceptionWhenDataTableRowCountExceedLimit属性**：返回一个bool值，表示当待导出的数据超过了MaxDataTableRowCountLimit属性时，是抛出异常中断导出操作，还是自动截断超出的部分数据继续导出文件；ExcelFileExporter该属性的实现是返回true，表示但当待导出的数据超过了MaxDataTableRowCountLimit属性时直接抛出异常，中断导出操作；
* **BuildBizException方法**：build一个自己的异常类型，当待导出的数据超过了MaxDataTableRowCountLimit属性并且ThrowExceptionWhenDataTableRowCountExceedLimit为true时，会抛出该方法所build出的异常对象；该方法参数msg是已经构造好了的错误提示消息，queryResultRowCount是当前的待导出记录数；ExcelFileExporter该方法的实现是用参数字符串msg来构造出一个BizException类型的异常对象；
* **BuildSheet方法**：这个是用来生成一个Excel Sheet的主要方法，该方法的第一个参数worksheet是使用的第三方的一个Excel组件MyXls里的类型，表示一个Excel Sheet的操作对象；第二个参数sheetIndex表示是在生成第几个Sheet（因为有可能多个DataTable要导出多个Sheet的Excel，是从0开始算）；第三个参数data是待导出的数据；第四个参数columnSetting是导出的列的设置；ExcelFileExporter该方法的实现是通过调用组合第三层次的方法来实现的；

(3). 第三个层次的方法有5个，都是被第二个层次的BuildSheet方法所调用：

* **SetDefaultHorizontalAlignmentsForType方法**：用来设置单元格里数据的水平对齐方式，会传入数据类型（因为不同数据类型可能水平对齐方式不一样，比如普通文本时左对齐，而数字可能就需要右对齐）；
* **SetSheetColumnsWith方法**：用来根据导出列的设置信息，来设置导出的Excel里的列的宽度；
* **SetXFForHeader方法**：为index为sheetIndex的Sheet的第excelColumnIndex列来设置样式；
* **SetXFForDataCell方法**：为index为sheetIndex的Sheet的第excelColumnIndex列第excelRowIndex行来设置样式；
* **FormatCellValue方法**：为index为sheetIndex的Sheet的第excelColumnIndex列第excelRowIndex行的数据格式化显示文本，其中第二个参数format就为Portal设置为该列的format格式字符串；

# 四、配置说明

Portal端无需任何配置；

Service端配置，分为Web.config和自定义配置文件2部分：

(1). 在Web.config的<appSettings>节点下，添加一个如下节点，value用来指明自定义配置文件的路径：

|  |
| --- |
| <!-- 设置数据文件导出服务的配置文件的路径， 支持绝对路径或相对于WebHost跟目录的路径-->  <add key="fileExporterConfigPath" value="Configuration/FileExporter.config"/> |

如果该节点没有设，或设为空或空白字符串，那么默认使用WebHost下的Configuration\FileExporter.config作为自定义配置文件的路径；

(2). 自定配置文件FileExporter.config的XML结构如下：

|  |
| --- |
| <?xml version="1.0"?>  <fileExporter default="ExportWord" expiry="00:05:00" maxRowCountLimit="10000">  <add name="ExportCustomer" type="ECCentral.Service.Customer.AppService.CustomerExcelFileExporter, ECCentral.Service.Customer.AppService" />  <add name="ExportWord" type="WordExporter.ExporterV1, WordExporter" />  </fileExporter> |

* 根节点fileExporter，有三个属性：

**default属性：**表示当Portal端在调用导出方法时没有指定exporterName的时候，Service端默认使用的导出实现类型，该属性值只能为下面某个add子节点的name属性的值；如果没有配置该属性或该属性为空，那么默认使用ECCentral.Service.Utility.ExcelFileExporter类型，将导出Excel文件；如果该属性配置的name不为空但又不在其下自己点中存在，那么还是会默认使用ECCentral.Service.Utility.ExcelFileExporter类型来导出Excel文件；

**expiry属性：**设置导出文件保存在服务器端后多长时间就可以删除了，其格式为“HH:mm:ss”；注意该属性只是设置了多长时间后文件可以删除，但不是说到了那个时间后文件就一定会删除；因为删除动作是由每次导出动作触发后才会伴随执行的（并没有背后线程来执行），也就是说如果一直没有导出动作触发，那么文件将一直保留在服务器硬盘上，即使早已过了设定的expiry时间；如果该属性没有设置，或者设置为空或者设置格式不正确，那么都会使用默认5分钟；

**maxRowCountLimit属性**：用来设置一个Sheet里最大允许导出的记录条数，默认为10000；

* fileExporter下面的add子节点就用来配置自定义的数据文件导出类型，name属性的值需要唯一，不能有重复，否则会报配置异常，Portal端的导出方法上的exportName参数就是对应到这里配置的name属性，用来告诉Service，应该使用哪个导出类来负责处理文件导出工作；