Web 框架界面代码分析

版本：1.0

编制人：伍华聪

序号	修改人	修改日期	修改后版本	修改说明
1	伍华聪	2014-12-5	V1.0	正式版

目　录

1. 引言

1.1. 背景

前几年，我一直在做 Web 项目，使用 ASP.NET 的 WebForm 方式开发了几个较为大型的行业管理系统，经过不断的完善和技术积累，也逐渐形成了一个标准的信息管理系统的 Web 开发框架，这个是早期技术的 Web 开发框架，包含了权限管理模块、菜单管理模块、字典管理以及行业管理业务模块等，同时开发了一些 Web 的相关控件，包括 Web 分页控件、Web 查询控件、Web 文件上传等控件，同时也已经结合代码生成工具 Database2Sharp 进行大部分的代码生成，包括 Web 前端的一些界面。

在原先的 Web 框架里面，主要是采用 FrameSet 的原始方式进行布局，很多内容依靠 Javascript 类库进行操作，小部分采用了 EasyUI 的一些特性，总体来说，是比较传统的一种框架模式，这个框架里面我已经集成了用户角色等权限方面的管理和控制、菜单管理、字典管理、业务流程审批管理等模块，因此对开发常规的行业应用有着比较快的开发效率，不过缺点也比较明显，就是在多浏览器支持方面，没有做的很好，框架里面采用的布局、样式及技术等方面不够统一，不够新颖，但即使这样，这套框架也顺利用来开发了几套很大规模的行业应用系统了。

随着 Web 前端技术的发展，MVC 的技术也已经渐趋成熟，JQuery 等应用技术已经是遍地开花，应用非常广泛，有很多组件模块可以复用，界面的美观以及易用方面都有很大的提高。为了引入更新、更强大的 Web 技术，我对 Web 开发框架整体进行了改造，使用基于 **MVC4 + EasyUI + Enterprise Library** 等相关的技术对原有的 Web 开发框架进行了升级，并引入了功能强大且流行很广的 EasyUI 界面组件，以及 Uploadify 文件上传组件、LODOP 打印组件、CKEditor 富文本编辑控件、Tags-Input 标签录入控件、HighCharts 图表展示控件、Word/Excel 文档导出组件等，对《Web 开发框架》整体进行加强和完善，该《Web 开发框架》是我们经过多年的项目积累，吸收众多框架产品的前沿思路，以及客户的宝贵意见，反复提炼优化而成的。

1.2. 编写目的

本文档主要介绍如何在《Web 开发框架》的基础上，通过分析各个模块相关的代码、各种常见组件的使用、特殊应用场景的处理，熟悉掌握框架里面相关代码和组件的使用，以便能够更全面的、更真实了解《Web 开发框架》的相关特点和如何使用框架的相关模块代码。

1.3. 参考资料

序号	名称	版本/日期	来源
1	《Web 开发框架-架构设计说明书.doc》		内部
2	《Winform 开发框架-架构设计说明书.doc》		内部
3	《循序渐进开发 Web 项目操作说明书.doc》		内部

1.4. 术语缩写及约定

1 在本文件中出现的“Web开发框架”一词，除非特别说明，均指基于**MVC + EasyUI + Enterprise Library**等相关的技术研制的最新Web开发框架。

2 在本文安装.NET框架中，除非特别说明，均指 **.NET 4.5** 框架。

3 本文的开发环境为Visual Studio 2013，基于 MVC5 的Web开发技术。

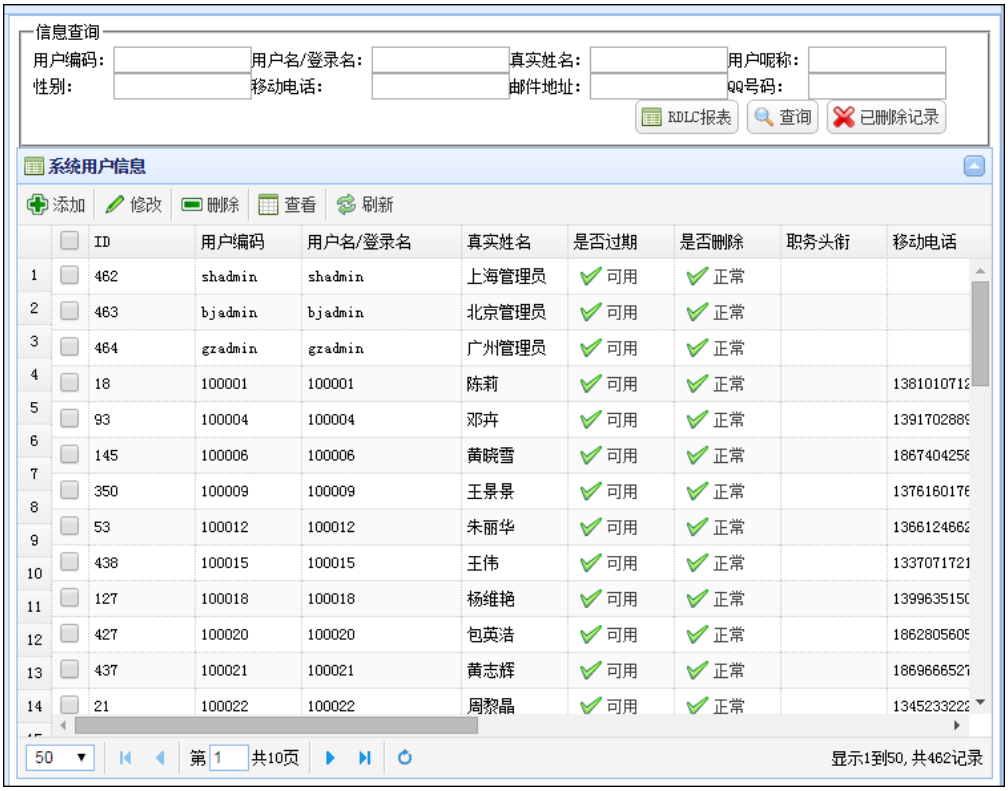
2. Web 框架界面代码的分析

2.1. 分页查询

在很多场合，分页是必须的，一个是可以节省响应时间，二是提高数据检索效率，使得我们能够给系统界面提供更好的用户体验。

在 Winform 开发框架里面，我通用把分页部分独立做为一个分页控件，那样给用户提供更好的一致的界面，而且也能封装一些常规的菜单处理。不过在 MVC 界面处理方面，这个有所不同，这里只需要通过对 EasyUI 的 Datagrid 控件进行一定的设置处理，就可以很好支持分页处理；另外 Web 框架的控制器基类也进行了相应的封装，以配合 EasyUI 的 Datagrid

控件的条件分页处理。



2.1.1. 控制器后台的分页支持

在 MVC 的控制器基类里面，封装了增删改查等大多数通用的处理函数，里面就包括了分页的控件处理，下面是这个类的总体介绍。

```
namespace WHC.MVCWebMis.Controllers
{
    /// <summary>
    /// 本控制器基类专门为访问数据业务对象而设的基类
    /// </summary>
    /// <typeparam name="B">业务对象类型</typeparam>
    /// <typeparam name="T">实体类类型</typeparam>
    36 个引用
    public class BusinessController<B, T> : BaseController
        where B : class
        where T : WHC.Framework.ControlUtil.BaseEntity, new()
    {
        构造函数及常用

        对象添加、修改、查询接口

        返回集合的接口

        基础接口

        其他接口

        用户机构角色相关操作
    }
}
```

下面是控制器分页的主要逻辑代码

```
/// <summary>
/// 根据条件查询数据库,并返回对象集合(用于分页数据显示)
/// </summary>
/// <returns>指定对象的集合</returns>
5 个引用
public virtual ActionResult FindWithPager()
{
    //检查用户是否有权限，否则抛出MyDenyAccessException异常
    base.CheckAuthorized(AuthorizeKey.ListKey);

    string where = GetPagerCondition();
    PagerInfo pagerInfo = GetPagerInfo();
    List<T> list = baseBLL.FindWithPager(where, pagerInfo);

    //Json格式的要求{total:22,rows:{}}
    //构造成Json的格式传递
    var result = new { total = pagerInfo.RecordCount, rows = list };
    return ToJsonContentDate(result);
}
```

获取分页条件和分页信息

根据条件获取某页数据

根据数据和分页信息返回JSON数据

上面的代码里面，控制器根据 EasyUI 的 Datagrid 传回来的参数，构造好分页条件和分页信息，代码如下所示。

```
/// <summary>
/// 获取分页操作的查询条件
/// </summary>
/// <returns></returns>
13 个引用
protected virtual string GetPagerCondition()
{
    string where = "";

    //增加一个CustomedCondition条件，根据客户这个条件进行查询
    string CustomedCondition = Request["CustomedCondition"] ?? "";
    if (!string.IsNullOrWhiteSpace(CustomedCondition))
    {
        where = CustomedCondition;//直接使用条件
    }
    else
    {
        根据数据库字段列，对所有可能的参数进行获值，然后构建查询条件

        MyRegion

        where = condition.BuildConditionSql().Replace("Where", "");
    }

    return where;
}
```

判断是自定义条件还是常规条件

对常规条件进行解析处理

而其中获取分页参数（也就是当前是几页，每页显示几行）。

```
/// <summary>
/// 根据Request参数获取分页对象数据
/// </summary>
/// <returns></returns>
6 个引用
protected virtual PagerInfo GetPagerInfo()
{
    int pageIndex = Request["page"] == null ? 1 : int.Parse(Request["page"]);
    int pageSize = Request["rows"] == null ? 10 : int.Parse(Request["rows"]);

    PagerInfo pagerInfo = new PagerInfo();
    pagerInfo.CurrenetPageIndex = pageIndex;
    pagerInfo.PageSize = pageSize;

    return pagerInfo;
}
```

获取page和rows两个参数值

2.1.2. 界面的分页代码

在上面的分页处理函数里面，FindWithPager 函数提供了对条件的分页处理，这个函数通过条件获取指定页码的数据，然后返回一个 JSON 格式的数据给视图界面处理，视图上声明了一个 DataGrid 控件，并通过脚本进行绑定即可。

```
<table id="grid" title="用户操作"
```

```
data-options="iconCls:'icon-view'" fit="true"></table>
```

```
//实现对DataGird控件的绑定操作
function InitGrid(queryData) {
    $('#grid').datagrid({    //定位到Table标签，Table标签的ID是grid
        url: '/Province/FindWithPager',    //指向后台的Action来获取当前用户的信息的Json格式的数据
        title: '全国省份',
        iconCls: 'icon-view',
        height: 650,
        width: function () { return document.body.clientWidth * 0.9 },//自动宽度
        nowrap: true,
        autoRowHeight: true,
        striped: true,
        collapsible: true,
        pagination: true,
        pageSize: 50,
        pageList: [50, 100, 200],
        rownumbers: true,
        //sortName: 'ID',    //根据某个字段给easyUI排序
        sortOrder: 'asc',
        remoteSort: false,
        idField: 'ID',
        queryParams: queryData,  //异步查询的参数
        columns: [[
            { field: 'ck', checkbox: true },    //选择
              { title: '省份名称', field: 'ProvinceName', width: 200, sortable: true },
          ]],
```

表格分页属性设置

上面的函数是初始化分页控件，我们可以在页面完成加载后，进行分页数据的显示操作，如下代码就是进行数据的显示。

```
$(function () {
    initEditor();      //初始化CKEditor
    var queryData = {  };//可添加一些预设条件
    InitGrid(queryData);//初始化Datagrid表格数据
    InitDictItem();    //初始化字典信息

    BindSearchEvent();//绑定查询按钮事件
});
```

而如果我们输入条件，触发查询按钮的时候，我们就是需要给 DataGrid 控件传递一些参数，并执行数据的重新初始化就可以实现数据更新显示了。

```
//绑定搜索按钮的的点击事件
function BindSearchEvent() {
    //按条件进行查询数据，首先我们得到数据的值
    $("#btnSearch").click(function () {
        //得到用户输入的参数
        //取值有几种方式：$("#id").combobox('getValue'), $("#id").datebox('getValue')
        //字段增加WHC_前缀字符，避免传递如URL这样的Request关键字冲突
        var queryData = {
            WHC_ProvinceName: $("#txtProvinceName").val()
          }
        //将值传递给DataGrid
        InitGrid(queryData);

        //传递给导出操作
        exportCondition = queryData;

        return false;
    });
}
```

2.2. 插入操作

在一个业务系统里面，除了分页，增删改查是必须要有的常规操作，这些操作对各个业务对象来说，基本上逻辑都一样的，只是他们传递的数据不太一样而已，因此它们往往也是放在基类进行定义，可能根据不同的需要重载一下函数就可以了。

在 Web 框架里面，我们不仅在业务层基类对这些常规的操作进行了封装，而且在控制器基类的层面，也对这些常规的操作进行了统一封装处理，基本上我们在控制器子类不需要额外的代码就能处理这些增删改查的操作了，实现一个业务的增删改查是非常方便快捷的。

2.2.1. 控制器后台的操作支持

在控制器后台，对插入操作进行了统一的封装，返回一个结果对象 CommonResult 的 JSON 数据，这样如果操作有错误，那么后台可以传递错误信息到界面上，实现友好的提示，这样比简单返回的成功、失败的布尔值有更好的用户体验效果。同时我们在其中预留一个 `OnBeforeUpdate` 函数供子类进行重写，这样可以在插入前统一修改对象里面的数据。

这个 `Insert` 函数里面的参数 `T`，是一个泛型的变量，在子类调用的时候，这个 `T` 会被替换为具体的实体对象类型，从而得到强类型的接口函数。这样，在视图到控制器的数据转换中，系统会自动把 `Form` 里面的集合对象（键值内容）转换为具体的实体类属性信息，并对应上这个插入的函数。

```
/// <summary>
/// 在更新数据前对数据的修改操作
/// </summary>
/// <param name="info"></param>
/// <returns></returns>
11 个引用
protected virtual void OnBeforeUpdate(T info)
{
    //留给子类对参数对象进行修改
}

/// <summary>
/// 插入指定对象到数据库中
/// </summary>
/// <param name="info">指定的对象</param>
/// <returns>执行操作是否成功。</returns>
8 个引用
public virtual ActionResult Insert(T info)
{
    //检查用户是否有权限，否则抛出MyDenyAccessException异常
    base.CheckAuthorized(AuthorizeKey.InsertKey);

    CommonResult result = new CommonResult();
    if (info != null)
    {
        try
        {
            OnBeforeInsert(info);
            result.Success = baseBLL.Insert(info);
        }
        catch(Exception ex)
        {
            LogTextHelper.Error(ex);//错误记录
            result.ErrorMessage = ex.Message;
        }
    }
    return ToJsonContent(result);
}
```

可统一修改信息

通用返回结果

操作错误记录并存放在结果对象

2.2.2. 界面层的代码

在 index.cshtml 视图代码里面，里面有一个对话框的 DIV 是用承载增加或编辑的界面内容的，如下所示。

```
<!--------------------------添加/修改信息的弹出层---------------------------->
<div id="DivAdd" class="easyui-dialog" style="width:680px;height:250px;padding:10px 20px"
     closed="true" resizable="true" modal="true" data-options="iconCls: 'icon-add',buttons: '#dlg-buttons'">
    <form id="ffAdd" method="post" novalidate="novalidate">
        <div id="tabAdd" class="easyui-tabs">
            <div title="基本信息" data-options="iconCls:'icon-view'" style="padding:5px 5px">
                <table>...</table>
            </div>
        </div>
        <div style="text-align:right; padding-top:10px">
            <input type="hidden" id="ID" name="ID" />
            <a href="javascript:void(0)" class="easyui-linkbutton" id="btnAddOK" iconcls="icon-ok" onclick="SaveEntity()">确定</a>
            <a href="javascript:void(0)" class="easyui-linkbutton" iconcls="icon-cancel" onclick="javascript:$('#DivAdd').dialog('close')">关闭</a>
        </div>
    </form>
</div>
```

在具体的对话框里面展示界面控件

保存和关闭对话框代码

```
//绑定添加按钮的事件
function SaveEntity() {
    //判断表单的信息是否通过验证
    var validate = $("#ffAdd").form('validate');
    if (validate == false) {
        return false;
    }

    var postData = $("#ffAdd").serializeArray();
    $.post(url, postData, function (json) {
        var data = $.parseJSON(json);
        if (data.Success) {
            //添加成功  1.关闭弹出层，2.刷新DataGird
            showTips("保存成功");
            $("#DivAdd").dialog("close");
            $("#grid").datagrid("reload");
            $("#ffAdd").form("clear");
        }
        else {
            showError("保存失败:" + data.ErrorMessage, 3000);
        }
    }).error(function () {
        $.messager.alert("提示", "您未被授权使用该功能，请联系管理员进行处理。", 'warning');
    });
}
```

验证输入内容是否通过

使用POST方式提交数据

如有操作错误信息，则显示

上面代码里，有一行代码：

```
var postData = $("#ffAdd").serializeArray();
```

就是把整个 Form 里面的元素的值，转换为一个键值集合，然后 POST 提交到指定的控制器 URL 里面了。对于插入记录，它的控制器 URL 类似如下所示，其中 City 是具体的对象名称：

```
url = '/City/Insert';
```

2.3. 删除操作

2.3.1. 控制器后台的操作支持

删除操作也是在控制器基类里面进行处理的，删除操作分为几种，常规的做法是获取界面的删除 ID 集合，然后逐一删除，如下代码所示。

```
/// <summary>
/// 删除多个ID的记录
/// </summary>
/// <param name="ids">多个id组合，逗号分开（1,2,3,4,5）</param>
/// <returns></returns>
2 个引用
public virtual ActionResult DeleteByIds(string ids)
{
    //检查用户是否有权限，否则抛出MyDenyAccessException异常
    base.CheckAuthorized(AuthorizeKey.DeleteKey);

    CommonResult result = new CommonResult();
    try
    {
        if (!string.IsNullOrEmpty(ids))
        {
            List<string> idArray = ids.ToDelimitedList<string>(",");
            foreach (string strId in idArray)
            {
                if (!string.IsNullOrEmpty(strId))
                {
                    baseBLL.Delete(strId);
                }
            }
            result.Success = true;
        }
    }
    catch (Exception ex)
    {
        LogTextHelper.Error(ex);//错误记录
        result.ErrorMessage = ex.Message;
    }
    return ToJsonContent(result);
}
```

还有两种是根据 ID 或者条件进行记录的删除，如下面界面代码所示。

根据 ID 进行记录删除的控制器方法如下所示。

```
/// <summary>
/// 根据指定对象的ID,从数据库中删除指定对象
/// </summary>
/// <param name="id">指定对象的ID</param>
/// <returns>执行成功返回<c>true</c>，否则为<c>false</c>。</returns>
0 个引用
public virtual ActionResult Delete(string id)
{
    //检查用户是否有权限，否则抛出MyDenyAccessException异常
    base.CheckAuthorized(AuthorizeKey.DeleteKey);

    CommonResult result = new CommonResult();
    try
    {
        if (!string.IsNullOrEmpty(id))
        {
            result.Success = baseBLL.Delete(id);
        }
    }
    catch (Exception ex)
    {
        LogTextHelper.Error(ex);//错误记录
        result.ErrorMessage = ex.Message;
    }
    return ToJsonContent(result);
}
```

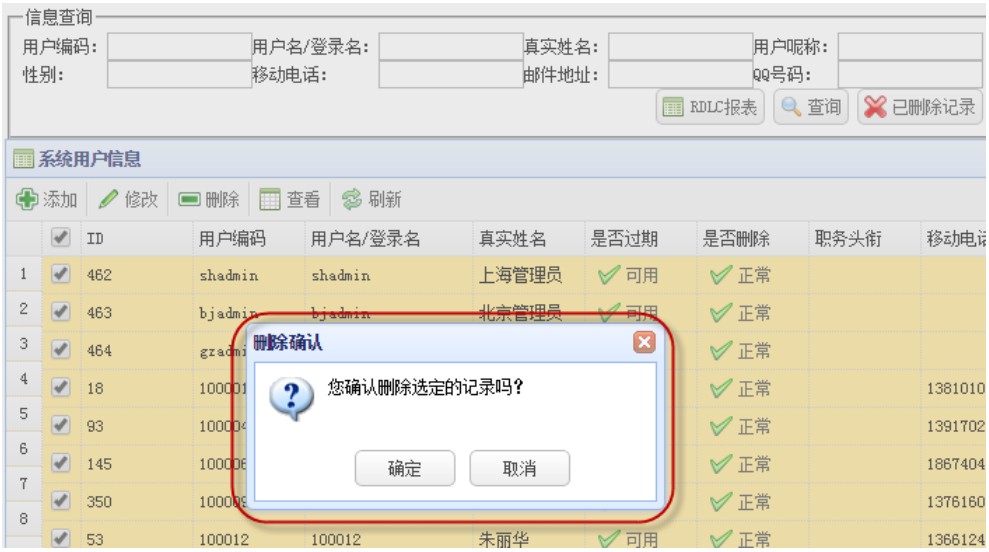
或者根据条件语句进行记录的删除的控制器方法如下所示。

```
/// <summary>
/// 根据指定条件,从数据库中删除指定对象
/// </summary>
/// <param name="condition">删除记录的条件语句</param>
/// <returns>执行成功返回<c>true</c>，否则为<c>false</c>。</returns>
0 个引用
public virtual ActionResult DeleteByCondition(string condition)
{
    //检查用户是否有权限，否则抛出MyDenyAccessException异常
    base.CheckAuthorized(AuthorizeKey.DeleteKey);

    CommonResult result = new CommonResult();
    try
    {
        if (!string.IsNullOrEmpty(condition))
        {
            result.Success = baseBLL.DeleteByCondition(condition);
        }
    }
    catch (Exception ex)
    {
        LogTextHelper.Error(ex);//错误记录
        result.ErrorMessage = ex.Message;
    }
    return ToJsonContent(result);
}
```

以上这些删除操作的控制器方法，都是返回一个通用的结果对象 `CommonResult` 的 JSON 数据，这种操作是我们常规操作数据库的返回记录，可以承载是否成功，如果有错误则传递错误等信息，非常方便。

2.3.2. 界面层的代码

在列表界面里面，提供复选框选择要操作的记录，然后操作删除按钮，就会提示确认是否删除，这个操作界面如下所示。



它的操作按钮是通过 DataGrid 的 ToolBar 属性进行添加的。

```
toolbar: [{
    id: 'btnAdd',
    text: '添加',
    iconCls: 'icon-add',
    handler: function () {
        ShowAddDialog();//实现添加记录的页面
    }
}, '-', {
    id: 'btnEdit',
    text: '修改',
    iconCls: 'icon-edit',
    handler: function () {
        ShowEditOrViewDialog();//实现修改记录的方法
    }
}, '-', {
    id: 'btnDelete',
    text: '删除',
    iconCls: 'icon-remove',
    handler: function () {
        Delete();//实现直接删除数据的方法
    }
}, '-', {
```

而删除操作的脚本代码如下所示。

```
//实现删除数据的方法
function Delete() {
    //得到用户选择的数据的ID
    var rows = $("#grid").datagrid("getSelections");
    if (rows.length >= 1) {
        //遍历出用户选择的数据的信息，这就是用户用户选择删除的用户ID的信息
        var ids = "";   //1,2,3,4,5
        for (var i = 0; i < rows.length; i++) {
            ids += rows[i].ID + ",";
        }
        //最后去掉最后的那一个,
        ids = ids.substring(0, ids.length - 1);
        var postData = { Ids: ids };

        //然后确认发送异步请求的信息到后台删除数据
        $.messager.confirm("删除确认", "您确认删除选定的记录吗？", function (action) {
            if (action) {
                $.ajax({
                    type: 'POST',
                    url: '/City/DeletebyIds',
                    dataType: 'json',
                    data: postData,
                    success: function (data) {
                        if (data.Success) {
                            showTips("删除选定的记录成功");

                            $("#grid").datagrid("reload");
                            //当删除完成之后，第二次删除的时候还记得上次的信息，这样是不可以的，所以我们需要清除第一次的信息
                            rows.length = "";//第一种方法
                            $("#grid").datagrid("clearSelections");//第二种方法
                        }
                        else {
                            showError("操作失败：" + data.ErrorMessage, 3000);
                        }
                    }
                });
            }
        });
    }
    else {
        $.messager.alert("提示", "请选择你要删除的数据");
    }
```

1 获取删除的ID集合

2 先确认是否进行删除

3 提交到具体的控制方法

4 删除成功后进行更新

5 失败则显示错误

2.4. 获取对象

我们需要在界面上显示业务对象的内容，就需要获取业务对象的相关实体类信息，这样就可以通过脚本绑定到界面控件上。

获取对象，包括获取对象的实体类和实体类集合等操作。

2.4.1. 控制器后台的操作支持

后台控制器基类提供了很多获取对象的操作方法，最常用的是根据 ID 获取对象实体类类信息，这个实体类包含了对应业务表的字段信息。

```
/// <summary>
/// 查询数据库,检查是否存在指定ID的对象
/// </summary>
/// <param name="id">对象的ID值</param>
/// <returns>存在则返回指定的对象,否则返回Null</returns>
0 个引用
public virtual ActionResult FindByID(string id)
{
    //检查用户是否有权限，否则抛出MyDenyAccessException异常
    base.CheckAuthorized(AuthorizeKey.ViewKey);

    ActionResult result = Content("");
    T info = baseBLL.FindByID(id);
    if (info != null)
    {
        result = ToJsonContentDate(info);
    }

    return result;
}
```

获取对象的集合，并返回 JSON 数据的操作如下所示。

```
/// <summary>
/// 根据条件查询数据库,并返回对象集合
/// </summary>
/// <param name="condition">查询的条件</param>
/// <returns>指定对象的集合</returns>
0 个引用
public virtual ActionResult Find(string condition)
{
    //检查用户是否有权限，否则抛出MyDenyAccessException异常
    base.CheckAuthorized(AuthorizeKey.ListKey);

    ActionResult result = Content("");
    if (!string.IsNullOrEmpty(condition))
    {
        List<T> list = baseBLL.Find(condition);
        result = ToJsonContentDate(list);
    }
    return result;
}
```

获取对象集合

2.4.2. 界面层的代码

对于控件绑定对象的属性，我们先要通过 ID 获取到对应的对象（JSON 数据对象），然后把对象的值赋值给不同的控件，界面代码如下所示。

```
<!-------------------------添加/修改信息的弹出层---------------------------->
<div id="DivAdd" class="easyui-dialog" style="width:680px;height:250px;padding:10px 20px"
    closed="true" resizable="true" modal="true" data-options="iconCls: 'icon-add',buttons: '#dlg-buttons'">
    <form id="ffAdd" method="post" novalidate="novalidate">
        <div id="tabAdd" class="easyui-tabs">
            <div title="基本信息" data-options="iconCls:'icon-view'" style="padding:5px 5px">
                <table>
                    <tr>
                        <td>
                            <table id="tblAdd1" class="view">
                                <tr>
                                    <th>
                                        <label for="CityName">城市名称：</label>
                                    </th>
                                    <td>
                                        <input class="easyui-validatebox" type="text" id="CityName" name="CityName" />
                                    </td>

                                    <th>
                                        <label for="ZipCode">邮编：</label>
                                    </th>
                                    <td>
                                        <input class="easyui-validatebox" type="text" id="ZipCode" name="ZipCode" />
                                    </td>
                                </tr>
                                <tr>
                                    <th>
                                        <label for="ProvinceID">省份ID：</label>
                                    </th>
                                    <td colspan="3">
                                        <input class="easyui-combotree" type="text" id="ProvinceID" name="ProvinceID" style="width:200px;" />
                                    </td>
                                </tr>
                            </table>
                        </td>
                    </tr>
                </table>
            </div>
        </div>
        <div style="text-align:right; padding-top:10px">
            <input type="hidden" id="ID" name="ID" />
            <a href="javascript:void(0)" class="easyui-linkbutton" id="btnAddOK" iconcls="icon-ok" onclick="SaveEntity()">确定</a>
            <a href="javascript:void(0)" class="easyui-linkbutton" iconcls="icon-cancel" onclick="javascript:$('#DivAdd').dialog('close')">关闭</a>
        </div>
    </form>
</div>
```

获取对象并绑定属性到界面上的操作脚本如下所示。

```
//绑定编辑详细信息的方法
function BindEditInfo(ID) {
    //使用同步方式，使得联动的控件正常显示
    $.ajaxSettings.async = false;

    //首先用户发送一个异步请求去后台实现方法
    $.getJSON("/City/FindByID?id=" + ID, function (info) {
        //赋值有几种方式：.datebox('setValue', info.Birthday);.combobox('setValue', info.Status);
        //.val(info.Name);.combotree('setValue', info.PID);.numberbox('setValue', info.Number);
        $("#ID").val(info.ID);
        $("#CityName").val(info.CityName);
        $("#ZipCode").val(info.ZipCode);
        $("#ProvinceID").combotree('setValue', info.ProvinceID);

        isAddOrEdit = 'edit';//新增对话框标识
    });
}
```

几种不同的赋值方式，供参考

2.5. 更新操作

更新操作和插入操作类似，都是获取 Form 表单的属性值列表，然后转换为 Json 数据，提交到具体的控制器更新操作中。在视图里面，通过下面的脚本代码：

```
var postData = $("#ffAdd").serializeArray();
```

把整个 Form 里面的元素的值，转换为一个键值集合，然后 POST 提交到指定的控制

器 URL 里面了。

2.5.1. 控制器后台的操作支持

```
/// <summary>
/// 更新对象属性到数据库中
/// </summary>
/// <param name="info">指定的对象</param>
/// <param name="id">主键ID的值</param>
/// <returns>执行成功返回<c>true</c>，否则为<c>false</c>。</returns>
2 个引用
public virtual ActionResult Update(string id, FormCollection formValues)
{
    //检查用户是否有权限，否则抛出MyDenyAccessException异常
    base.CheckAuthorized(AuthorizeKey.UpdateKey);

    T obj = baseBLL.FindByID(id);
    if (obj != null)
    {
        //遍历提交过来的数据（可能是实体类的部分属性更新）
        foreach (string key in formValues.Keys)
        {
            string value = formValues[key];
            System.Reflection.PropertyInfo propertyInfo = obj.GetType().GetProperty(key);
            if (propertyInfo != null)
            {
                try
                {
                    // obj对象有key的属性，把对应的属性值赋值给它(从字符串转换为合适的类型)
                    //如果转换失败，会抛出InvalidCastException异常
                    propertyInfo.SetValue(obj, Convert.ChangeType(value, propertyInfo.PropertyType), null);
                }
                catch { }
            }
        }
    }

    CommonResult result = new CommonResult();
    try
    {
        result.Success = Update(id, obj);
    }
    catch (Exception ex)
    {
        LogTextHelper.Error(ex);//错误记录
        result.ErrorMessage = ex.Message;
    }

    return ToJsonContent(result);
}
```

对FormCollection进行映射到实体类上，设置属性值。

更新对象数据，并返回结果和错误信息

在 Update(id, obj)的操作里面，再次对更新操作进行了一个封装，主要是考虑需要重写的操作方便，同样更新操作也有一个可供子类实现特殊信息修改的 OnBeforeUpdate 方法。

```
/// <summary>
/// 更新对象属性到数据库中(方便重写的时候操作)
/// </summary>
/// <param name="id">主键ID的值</param>
/// <param name="info">指定的对象</param>
/// <returns>执行成功返回<c>true</c>，否则为<c>false</c>。</returns>
11 个引用
protected virtual bool Update(string id, T info)
{
    OnBeforeUpdate(info);
    return baseBLL.Update(info, id);
}
```

2.5.2. 界面层的代码

```
//修改或查看明细信息（绑定显示数据）
function ShowEditOrViewDialog(view) {
    //首先取出来用户选择的数据的ID
    var rows = $("#grid").datagrid("getSelections");
    //首先取出来值判断用户只能选择一个
    if (rows.length == 0) {
        $.messager.alert("提示", "请选择一条记录", "error");
        return;
    }
    else if (rows.length > 1) {
        $.messager.alert("提示", "每次只能修改/查看一条记录，你已经选择了<font color='red'  size='6'>" + rows.length + "</font>条", "error");
        return;
    }

    ID = $("#grid").datagrid("getSelections")[0].ID;  //获取到了用用户选
    if (view == null) {
        //处理修改的信息
        $("#DivAdd").dialog('open').dialog('setTitle', '修改信息');
        url = '/City/Update?ID=' + ID;
        //绑定修改详细信息的方法
        BindEditInfo(ID);
    }
    else {
        //处理查看详细
        $("#DivView").dialog('open').dialog('setTitle', '查看详细信息');
        //绑定查看详细信息方法
        BindViewInfo(ID);
    }
}
```

弹出修改对话框，并绑定对象的属性值

更新操作和插入逻辑类似，只是更新操作，它的 url 已经变化为：

```
url = '/City/Update?ID=' + ID;
```

更新操作的保存数据脚本如下所示：

```
//绑定添加按钮的事件
function SaveEntity() {
    //判断表单的信息是否通过验证
    var validate = $("#ffAdd").form('validate');
    if (validate == false) {
        return false;
    }

    var postData = $("#ffAdd").serializeArray();
    $.post(url, postData, function (json) {
        var data = $.parseJSON(json);
        if (data.Success) {
            //添加成功  1.关闭弹出层，2.刷新DataGird
            showTips("保存成功");
            $("#DivAdd").dialog("close");
            $("#grid").datagrid("reload");
            $("#ffAdd").form("clear");
        }
        else {
            showError("保存失败:" + data.ErrorMessage, 3000);
        }
    }).error(function () {
        $.messager.alert("提示", "您未被授权使用该功能，请联系管理员进行处理。", 'warning');
    });
}
```

验证输入内容是否通过

使用POST方式提交数据

如有操作错误信息，则显示

2.6. 字典数据绑定

在很多场合，不管是 Winform 还是 Web 的项目，都是要有字典模块的，很多下拉列表需要绑定字典的值，而且我们也希望在统一的字典管理界面对数据进行维护。而正是由于指定数据绑定的通用性，我在 ComponentUtil.js 脚本里面定义了一个通用的数据字典绑定函数 BindDictItem，如下所示。这样应用了这个脚本的所有视图界面代码，都可以通过这个方法快速绑定字典类型的值到具体的控件上了。

```
//绑定字典内容到指定的控件
function BindDictItem(control, dictTypeName) {
    $('#' + control).combobox({
        url: '/DictData/GetDictJson?dictTypeName=' + encodeURI(dictTypeName),
        valueField: 'Value',
        textField: 'Text'
    });
}
```

2.6.1. 控制器后台的操作支持

```
/// <summary>
/// 根据字典类型获取对应的字典数据，方便UI控件的绑定
/// </summary>
/// <param name="dictTypeName">字典类型名称</param>
/// <returns></returns>
0 个引用
public ActionResult GetDictJson(string dictTypeName)
{
    List<CListItem> treeList = new List<CListItem>();
    CListItem pNode = new CListItem("", "");
    treeList.Insert(0, pNode);

    Dictionary<string, string> dict = BLLFactory<DictData>.Instance.GetDictByDictType(dictTypeName);
    foreach (string key in dict.Keys)
    {
        treeList.Add(new CListItem(key, dict[key]));
    }
    return ToJsonContent(treeList);
}

/// <summary>
/// 根据字典类型获取对应的字典数据，方便UI控件的绑定
/// </summary>
/// <param name="dictTypeName">字典类型名称</param>
/// <returns></returns>
0 个引用
public ActionResult GetDictTreeJson(string dictTypeName)
{
    List<EasyTreeData> treeList = new List<EasyTreeData>();
    EasyTreeData pNode = new EasyTreeData("", "请选择");
    treeList.Insert(0, pNode);

    Dictionary<string, string> dict = BLLFactory<DictData>.Instance.GetDictByDictType(dictTypeName);
    foreach (string key in dict.Keys)
    {
        treeList.Add(new EasyTreeData(key, dict[key]));
    }
    return ToJsonContent(treeList);
}
```

普通的下拉列表绑定方法

树形下拉列表绑定方法

2.6.2. 界面层的代码

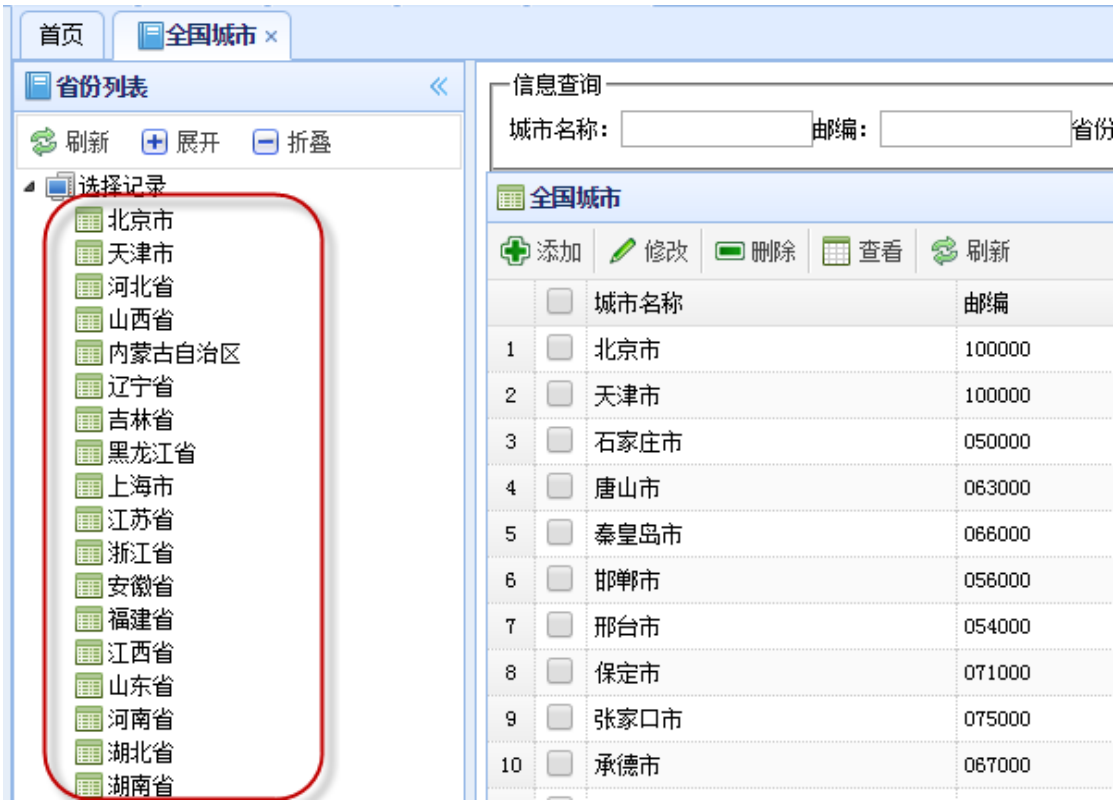
```
<tr>
     <th>
        <label for="Rank">职务：</label>
    </th>
    <td>
        <select class="easyui-combobox" id="Rank" name="Rank" style="width:120px;" ></select> <!-
    </td>
      <th>
        <label for="Titles">职称：</label>
    </th>
    <td>
        <select class="easyui-combobox" id="Titles" name="Titles" style="width:120px;" ></select>
    </td>
</tr>
```

```
//初始化字典信息
function InitDictItem() {
    BindDictItem("Titles", "职称");
    BindDictItem("Rank", "职务");
    BindDictItem("Importance", "重要级别");
    BindDictItem("Recognition", "对公司认可程度");
    BindDictItem("InterestDemand", "客户利益诉求");
    BindDictItem("CareFocus", "客户关心重点");
    BindDictItem("ResponseDemand", "负责需求");
    BindDictItem("RelationShip", "与公司关系");
    BindDictItem("Nationality", "民族");
    BindDictItem("Political", "政治面貌");
    BindDictItem("JobType", "职业类型");
    BindDictItem("Eduction", "学历");
    BindDictItem("Animal", "属相");
    BindDictItem("Constellation", "星座");
    BindDictItem("MarriageStatus", "婚姻状况");
    BindDictItem("HealthCondition", "健康状况");
    BindDictItem("BodyType", "体型");
}
```

2.7. 树信息绑定

树形控件的绑定也是在 Web 开发中常见的操作，树形控件显示数据比较直观，因此经常性的用在一些有层次化的数据列表展示里面。

EasyUI 的树形控件的绑定，主要是通过 Json 数据的传递实现绑定的，通过指定一个 `EasyTreeData` 对象的 JSON 集合，即可赋值给对应的树控件了。



2.7.1. 控制器后台的操作支持

```
/// <summary>
/// 获取所有的省份
/// </summary>
/// <returns></returns>
0 个引用
public ActionResult GetAllProvince()
{
    List<EasyTreeData> treeList = new List<EasyTreeData>();
    EasyTreeData pNode = new EasyTreeData("", "选择记录", "icon-computer");
    treeList.Add(pNode);

    List<ProvinceInfo> provinceList = BLLFactory<Province>.Instance.GetAll();
    foreach (ProvinceInfo info in provinceList)
    {
        EasyTreeData item = new EasyTreeData(info.ID, info.ProvinceName, "icon-view");
        pNode.children.Add(item);
    }
    return ToJsonContent(treeList);
}
```

返回一个EasyTreeData对象集合的JSON

或类似的其他操作方法。

```
/// <summary>
/// 根据用户获取对应人员层次的树Json
/// </summary>
/// <param name="deptId">用户所在部门</param>
/// <returns></returns>
0 个引用
public ActionResult GetUserTreeJson(int deptId)
{
    List<EasyTreeData> treeList = new List<EasyTreeData>();
    treeList.Insert(0, new EasyTreeData(-1, "无"));

    List<UserInfo> list = BLLFactory<User>.Instance.FindByDept(deptId);
    foreach (UserInfo info in list)
    {
        treeList.Add(new EasyTreeData(info.ID, info.FullName, "icon-user"));
    }

    return ToJsonContent(treeList);
}
```

2.7.2. 界面层的代码

```
<div data-options="region:'west',split:true,title:'省份列表',iconCls:'icon-book'" style="width: 250px; padding: 1px;">
    <div style="padding: 1px; border: 1px solid #ddd;">
        <a href="#" class="easyui-linkbutton" data-options="plain:true,iconCls:'icon-reload'" id="A4" onclick="reloadTree()">刷新</a>
        <a id="expandAllBtn" href="#" class="easyui-linkbutton" data-options="plain:true,iconCls:'icon-expand'" onclick="return false;">展开</a>
        <a id="collapseAllBtn" href="#" class="easyui-linkbutton" data-options="plain:true,iconCls:'icon-collapse'" onclick="return false;">折叠</a>
    </div>
    <div>
        <ul id="treeDemo"></ul>
    </div>
</div>
```

```
//重新加载树形结构（异步）
function reloadTree() {
    $("#loading").show();

    $('#treeDemo').tree({
        url: '/Province/GetAllProvince',
        onClick: function (node) {
            loadData(node.id); //树单击节点操作
        }
    });

    InitDictItem();//同时刷新字典

    $("#loading").fadeOut(500);
}
```

树控件的绑定


```
//加载指定的对象数据
var clickId = "";
function loadData(id) {
    $("#loading").show();

    var queryParams = $('#grid').datagrid('options').queryParams;
    //由于ProvinceID数据库类型为数值型，避免生成>=的查询条件，使用CustomedCondition替代
    queryParams.CustomedCondition = "ProvinceID = " + id;

    $("#grid").datagrid("reload");
    $('#grid').datagrid('uncheckAll');

    clickId = id;

    $("#loading").fadeOut(500);
}
//初始化对象
$(document).ready(function () {
    reloadTree();

    $("#expandAllBtn").bind("click", function () {
        expandAll("treeDemo");
    });
    $("#collapseAllBtn").bind("click", function () {
        collapseAll("treeDemo");
    });

    $("#loading").center();//loading的图片显示居中
});
```

2.8. 文件上传 Uploadify 的使用

在很多场合，我们需要通过 Web 方式从客户端上传一些文件到服务器端，而在这方面，Uploadify 这个控件做的非常不错。

Uploadify 是 JQuery 一个著名的上传插件，利用 Flash 技术，Uploadify 越过浏览器的限制，控制了整个上传的处理过程，实现了客户端无刷新的文件上传，这样就实现了在客户端的上传进度控制，所以，你首先要确定浏览器中已经安装了 Adobe 的 Flash 插件。Uploadify 当前有两个版本，基于 Flash 是免费的，还有基于 HTML5 的收费版，我们这里使用比较广泛的免费版。

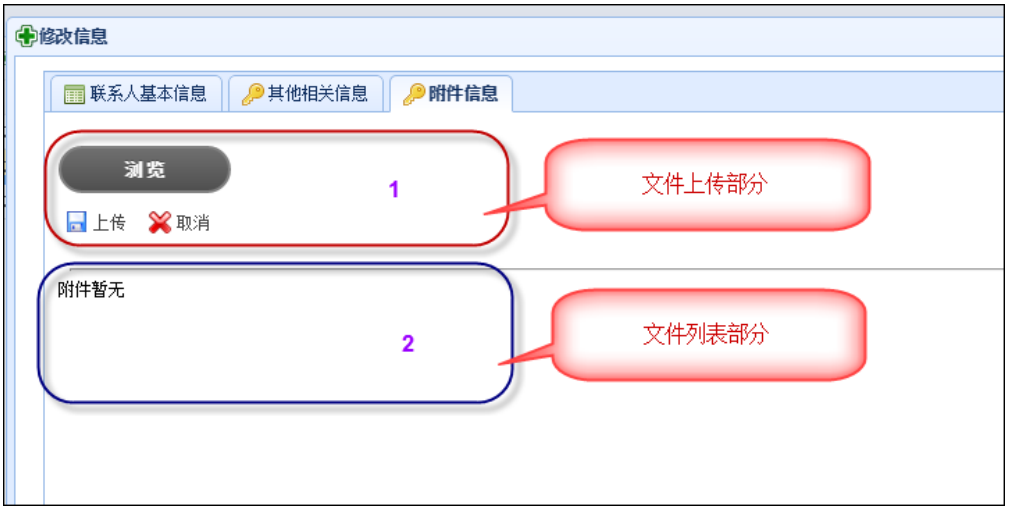
这个组件需要 Jquery 库的支持，一般情况下，需要添加 Jquery 的 js 库，如下所示：

```
<script type="text/javascript" src="~/Scripts/jquery-2.0.3.min.js"></script>
```

不过由于我的 Web 开发框架是基于 EasyUI 的，一般在网页的开始就会引用相关的类库，已经包含了 Jquery 的类库了，所以上面的不需要独立引用。一般情况下，我们只需要添加 Javascript 类库（jquery.uploadify.js），另外加上他的样式文件（uploadify.css）即可，不过我们在 MVC 里面，已经利用了 BundleConfig 引入了对应的 JS 和 CSS 文件，不用

再操心了。

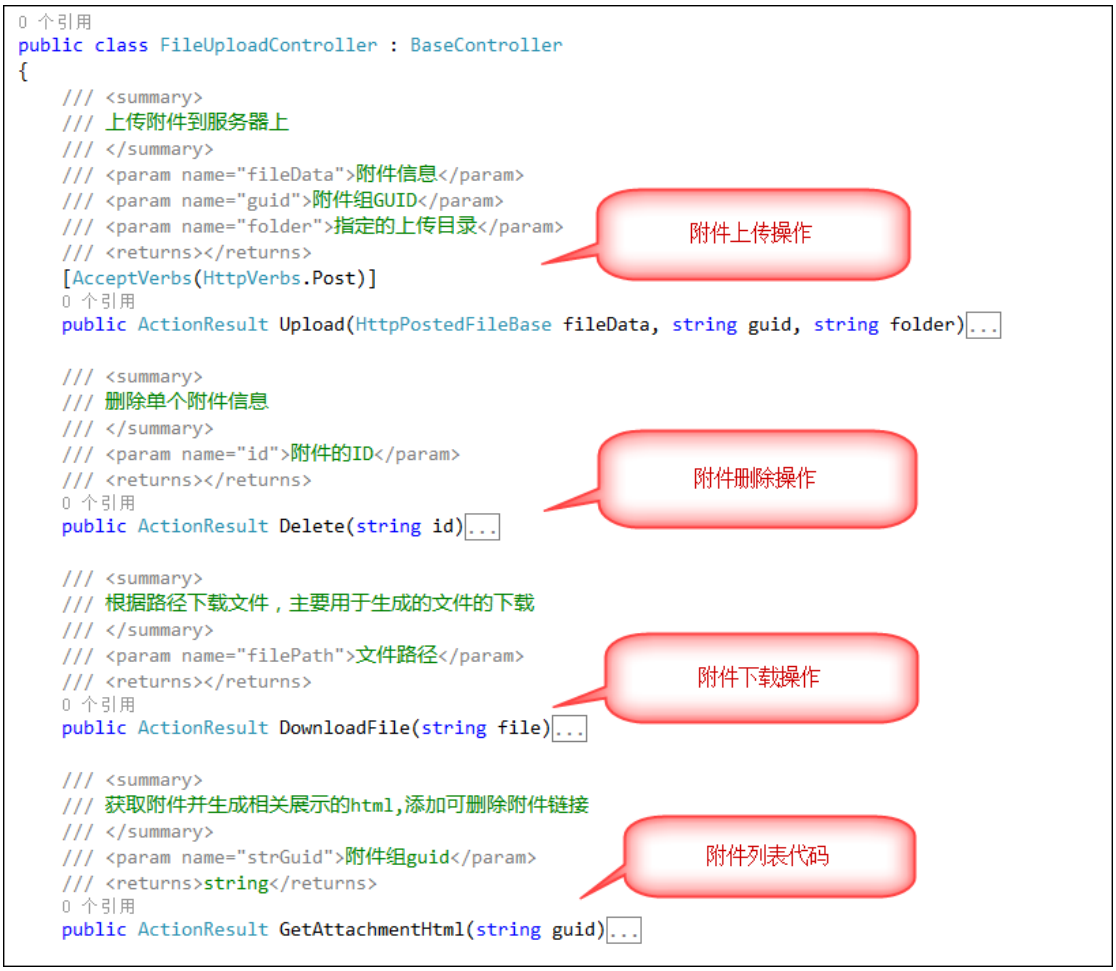
为了实现文件的上传，一般需要在 HTML 代码中放置两个控件，一个是用来上传的控件，一个是用来显示已上传列表的控件，还有就是添加上传和取消上传的按钮操作，如下所示。



如果是编辑状态下，那么列表就是实际已经上传的文件了。

2.8.1. 控制器后台的操作支持

所有的附件上传操作，都是由控制器 FileUploadController 进行处理，把它存储在对应的 TB_FileUpload 表里面，这个和 Winform 框架的附件存储方式保持一致，这样所有的附件都是存储在这个表里面。这个附件管理的控制器 FileUploadController 主要管理附件的上传、删除、下载、列表界面生成、查看附件代码等操作功能。



控制器中，对附件上传的操作代码逻辑，主要就是解析传入的参数和里面的 HttpPostedFileBase 文件集合， 然后存储到后台的数据库即可。

```
FileUploadInfo info = new FileUploadInfo();
info.FileData = ReadFileBytes(fileData);
if (info.FileData != null)
{
    info.FileSize = info.FileData.Length;
}
info.Category = folder;
info.FileName = fileName;
info.FileExtend = fileExtension;
info.AttachmentGUID = guid;
info.AddTime = DateTime.Now;
info.Editor = CurrentUser.Name;//登录人
//info.Owner_ID = OwerId;//所属主表记录ID

result = BLLFactory<FileUpload>.Instance.Upload(info);
```

读取文件信息并存储到数据库

2.8.2. 界面层的代码

界面代码如下所示。

```
<div title="附件信息" style="padding:5px;min-height:500px" data-options="iconCls:'icon-key'">
    <div style="padding:5px;">
        <input class="easyui-validatebox" type="hidden" id="AttachGUID" name="AttachGUID" />
        <br />
        <input id="file_upload" name="file_upload" type="file" multiple="multiple">
        <a href="javascript:void(0)" class="easyui-linkbutton" id="btnUpload" data-options="plain:true,iconCls:'icon-save'"
            onclick="javascript: $('#file_upload').uploadify('upload', '*')">上传</a>
        <a href="javascript:void(0)" class="easyui-linkbutton" id="btnCancelUpload" data-options="plain:true,iconCls:'icon-cancel'"
            onclick="javascript: $('#file_upload').uploadify('cancel', '*')">取消</a>

        <div id="fileQueue" class="fileQueue"></div>
        <br /><hr style="width:98%"/>
        <div id="div_files"></div>
        <br />
    </div>
</div>
```

对界面进行初始化的脚本函数代码如下所示。

```
<script type="text/javascript">
    $(function () {
        //添加界面的附件管理
        $('#file_upload').uploadify({
            'swf': '/Content/JQueryTools/uploadify/uploadify.swf',  //FLash文件路径
            'buttonText': '浏  览',                                  //按钮文本
            'uploader': '/FileUpload/Upload',                       //处理ASHX页面
            'queueID': 'fileQueue',                          //队列的ID
            'queueSizeLimit': 10,                            //队列最多可上传文件数量，默认为999
            'auto': false,                                   //选择文件后是否自动上传，默认为true
            'multi': false,                                   //是否为多选，默认为true
            'removeCompleted': true,                         //是否完成后移除序列，默认为true
            'fileSizeLimit': '10MB',                         //单个文件大小，0为无限制，可接受KB,MB,GB等单位的字符串值
            'fileTypeDesc': 'Image Files',                   //文件描述
            'fileTypeExts': '*.gif; *.jpg; *.png; *.bmp;*.tif;*.doc;*.docx;*.xls;*.xlsx;*.ppt;*.pptx;*.zip;*.rar',  //上传的文件后缀过滤器
            'onQueueComplete': function (event, data) {    //所有队列完成后事件
                ShowUpFiles($("#AttachGUID").val(), "div_files");
                $.messager.alert("提示", "上传完毕！");
            },
            'onUploadStart': function (file) {
                $("#file_upload").uploadify("settings", 'formData', { 'folder': '多媒体文件', 'guid': $("#AttachGUID").val() }); //动态传参数
            },
            'onUploadError': function (event, queueId, fileObj, errorObj) {
                //alert(errorObj.type + " : " + errorObj.info);
            }
        });
    });

    var attachguid = "";//用来记录附件组的ID，方便更新
    function deleteAttach(id) {
        DeleteAndRefreshAttach(id, attachguid, "div_files");
    }

</script>
```

Uploadify控件的使用和属性说明

在上面的脚本中，均有注释，一看就可以明白相关的属性了，不明白的也可以到官方网

站去查找了解。值得注意的就是：

```
'uploader': '/FileUpload/Upload'
```

这行就是提交文件给 MVC 的 Action 进行处理，我们在控制器 FileUpload 的 Upload 处理即可。

另外，在附件上传完毕后，我们需要对所在的界面进行更新，以便显示已上传的列表，那么我们需要增加下面的函数处理即可。

```
'onQueueComplete': function (event, data) {
```

最后说明非常值得注意的地方，就是文件上传的时候，我们需要动态获取界面上的一些元素的值，作为参数传递，那么我们就需要在 onUploadStart 函数中进行如下处理。

```
$("#file_upload").uploadify("settings", 'formData', { 'folder': '政策法规',
'guid': $("#Attachment_GUID").val() }); //动态传参数
```

为了操作方面，我们一般把一些脚本逻辑封装在常用的 JS 文件 ComponentUtil.js 里面，这样我们一个可以实现通用目的，二也有准确的智能提示，非常方便。

上面操作在文件上传完成后，马上会调用 `ShowUpFiles` 函数更新附件列表的信息，绑定到控件层上，这样我们就能及时看到文件的更新列表了。

```
//绑定附件列表
function ShowUpFiles(guid, show_div) {
    $.ajax({
        type: 'GET',
        url: '/FileUpload/GetAttachmentHtml?guid=' + guid,
        //async: false, //同步
        //dataType: 'json',
        success: function (json) {
            $("#" + show_div + "").html(json);
        },
        error: function (xhr, status, error) {
            $.messager.alert("提示", "操作失败" + xhr.responseText);
        }
    });
}
```

2.9. 附件管理和图片预览、Office 文档预览

在上面的附件上传到后台后，一般情况下，需要删除、下载、查看详细的文档内容等操作，对于一般的文件我们可能提供下载就可以了，对于图片，我们可以实现在线的图片预览功能。

2.9.1. 控制器后台的操作支持

删除文件的操作很简单，只需要根据附件的 ID（GUID 值），删除对应的文件，控制器返回一个通用的操作结果对象即可完成。

```
/// <summary>
/// 删除单个附件信息
/// </summary>
/// <param name="id">附件的ID</param>
/// <returns></returns>
0 个引用
public ActionResult Delete(string id)
{
    CommonResult result = new CommonResult();
    if (!string.IsNullOrEmpty(id))
    {
        try
        {
            result.Success = BLLFactory<FileUpload>.Instance.Delete(id);
        }
        catch(Exception ex)
        {
            LogTextHelper.Error(ex);
            result.ErrorMessage = ex.Message;
        }
    }
    return ToJsonContent(result);
}
```

而文件的预览或者下载操作，是绑定到界面控件上的代码构建出一个 HTML 代码片段实现。

```
/// <summary>
/// 获取附件并生成相关展示的html,添加可删除附件链接
/// </summary>
/// <param name="strGuid">附件组guid</param>
/// <returns>string</returns>
0 个引用
public ActionResult GetAttachmentHtml(string guid)
{
    string html = @"<li>附件暂无</li>";

    if (string.IsNullOrEmpty(guid))
        return Content(html);

    int seq = 1;
    StringBuilder sb = new StringBuilder();

    List<FileUploadInfo> fileList = BLLFactory<FileUpload>.Instance.GetByAttachGUID(guid);
    if (fileList != null && fileList.Count > 0)
    {
        构建附件展示的HTML代码

        return Content(result);
    }
    else
    {
        return Content(html);
    }
}
```

对于 Office 文档的在线预览，我们提供了两种方式，一种是在服务器本地利用组件生成对应的 HTML 文件，然后预览；一种是通过微软的在线文档预览实现。第一种响应速度快，不过文档有时候可能有些失真；第二种是需要服务发布到互联网，并且文档地址可以访问，提供比较完美的展现，缺点是比较慢。

```
/// <summary>
/// 根据附件ID，获取对应查看的视图URL。
/// 一般规则如果是图片文件，返回视图URL地址'/FileUpload/ViewAttach'；
/// 如果是Office文件（word、PPT、Excel）等，可以通过微软的在线查看地址进行查看：'http://view.officeapps.live.com/op/view.aspx?src='
/// 也可以进行本地生成HTML文件查看。如果是其他文件，可以直接下载地址。
/// </summary>
/// <param name="id">附件的ID</param>
/// <returns></returns>
0 个引用
public ActionResult GetAttachViewUrl(string id)
{
    string viewUrl = "";
    FileUploadInfo info = BLLFactory<FileUpload>.Instance.FindByID(id);
    if (info != null)
    {
        string ext = info.FileExtend.Trim('.').ToLower();
        string filePath = GetFilePath(info);

        bool officeInternetView = false;//是否使用互联网在线预览
        string hostName = HttpUtility.UrlPathEncode("http://www.iqidi.com/");//可以配置一下，如果有必要

        if (ext == "xls" || ext == "xlsx" || ext == "doc" || ext == "docx" || ext == "ppt" || ext == "pptx")
        {
            if (officeInternetView)
            {
                //返回一个微软在线浏览Office的地址，需要加上互联网域名或者公网IP地址
                viewUrl = string.Format("http://view.officeapps.live.com/op/view.aspx?src={0}{1}", hostName, filePath);
            }
            else
            {
                动态第一次生成文件
                viewUrl = webPath;
            }
        }
        else
        {
            viewUrl = filePath;
        }
    }
    return Content(viewUrl);
}
```

Office在线预览

Office本地第一次生成，然后预览

2.9.2. 界面层的代码

删除文件的 JS 代码如下所示。

```
//删除指定的附件后，对附件组进行更新
// id 删除附件id, attachguid 附件组ID, show_div 显示附件的Div
function DeleteAndRefreshAttach(id, attachguid, show_div) {
    $.messager.confirm("删除确认", "您确定要删除该附件吗？", function (action) {
        if (action) {
            $.ajax({
                type: 'POST',
                url: '/FileUpload/Delete?id=' + id,
                async: false,
                success: function (msg) {
                    ShowUpFiles(attachguid, show_div);//更新列表
                },
                error: function (xhr, status, error) {
                    $.messager.alert("提示", "操作失败"); //xhr.responseText
                }
            });
        }
    });
}
```

文件预览或者下载的 JS 处理代码如下所示。

```
//在新窗口中查看附件
function ShowAttach(id, ext) {
    var showWindow = true;//标识是否使用窗口查看。office文档+图片文档窗口查看，其他的直接下载
    var viewUrl = '/FileUpload/ViewAttach';
    var returnUrl;
    //var hostname = window.location.host;
    //var hostname = 'http://www.iqidi.com'

    var postData = { id: id };
    $.ajaxSettings.async = false;
    $.get("/FileUpload/GetAttachViewUrl", postData, function (url) {
        if (ext == 'xls' || ext == 'xlsx' || ext == 'doc' || ext == 'docx' || ext == 'ppt' || ext == 'pptx') {
            viewUrl = url;
        }
        else if (ext == "png" || ext == "jpg" || ext == "jpeg" || ext == "gif" || ext == "bmp" || ext == "tif") {
            viewUrl = "/" + url;
        }
        else {
            viewUrl = "/" + url;
            showWindow = false;
        }

        returnUrl = url;
    });
```

根据不同文件后缀，构建不同的处理URL

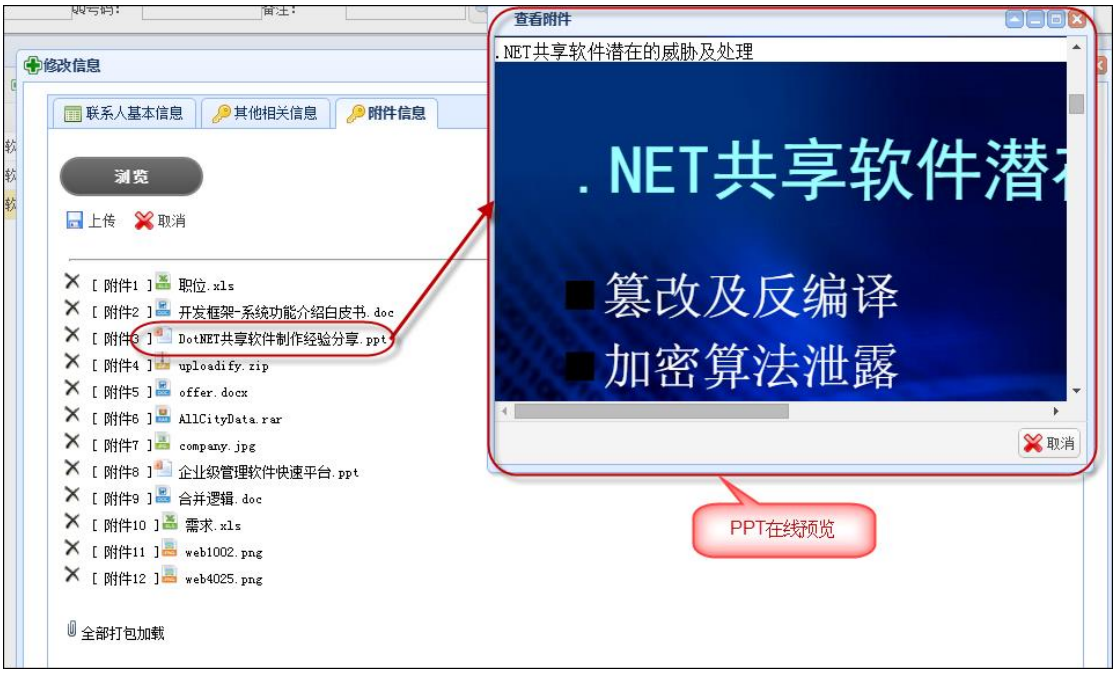
如果是Office文件或者图片文件，提供在线窗口预览

其中使用/FileUpload/GetAttachViewUrl 进行处理，获取真正的展示内容的 URL，主要是基于考虑，需要对 Office 文件进行 HTML 生成（第一次生成，后续打开即可）。

如果是图片文件，这里会把图片的地址作为一个 URL，在独立窗口进行打开；如果是其他附件，那么会自动下载到客户端。

```
    if (showWindow) {
        $.showWindow({
            title: '查看附件',
            useiframe: true,
            width: 900,
            height: 700,
            content: 'url:' + viewUrl,
            data: { url: returnUrl, ext: ext },
            buttons: [{
                text: '取消',
                iconCls: 'icon-cancel',
                handler: function (win) {
                    win.close();
                }
            }],
            onLoad: function (win, content) {
                //window打开时调用，初始化form内容
                if (content) {
                    //content.doInit(win);
                }
            }
        });
    }
    else {
        //附件直接下载，不用打开窗体
        window.open(viewUrl);
    }
```

下面是对 PPT 的预览效果。



2.9.3. 利用微软的平台进行 Office 文档的在线查看

在博客园很多文章里面，曾经有一些介绍 Office 文档预览查看操作的，有些通过转为 PDF 进行查看，有些通过把它转换为 Flash 进行查看，但是过程都是曲线救国，真正能够简洁方便的实现 Office 文档的预览的还是比较少，这里的 Office 文档包括了 Word、Excel、PPT 文档。本文介绍两种方式，一种方式是通过在线预览的方式，利用微软的平台进行 Office 文档的在线查看；一种是把 Office 文档生成 HTML 文件后进行查看。

一直以来，都希望找一个合适的 Office 文档查看的控件或者解决方案，这样客户在使用系统的时候，可以直接查看预览一些文档，而不需要安装 Office，或者下载到本地在查看。一个偶然的机会，在网上搜到微软自己提供了一个在线服务，可以实现把 Office 文档转换为在线的内容进行查看，大家可以通过下面的链接地址大致看看：http://blogs.office.com/2013/04/10/office-web-viewer-view-office-documents-in-a-browser/ 。

使用浏览器直接查看 Office 文档的链接地址格式如下所示：

http://view.officeapps.live.com/op/view.aspx?src=http%3a%2f%2fvideo.ch9.ms%2fbuild%2f2011%2fslides%2fTOOL-532T_Sutter.pptx

这个链接分为了两部分，一部分是 http://view.officeapps.live.com/op/view.aspx?src=，后

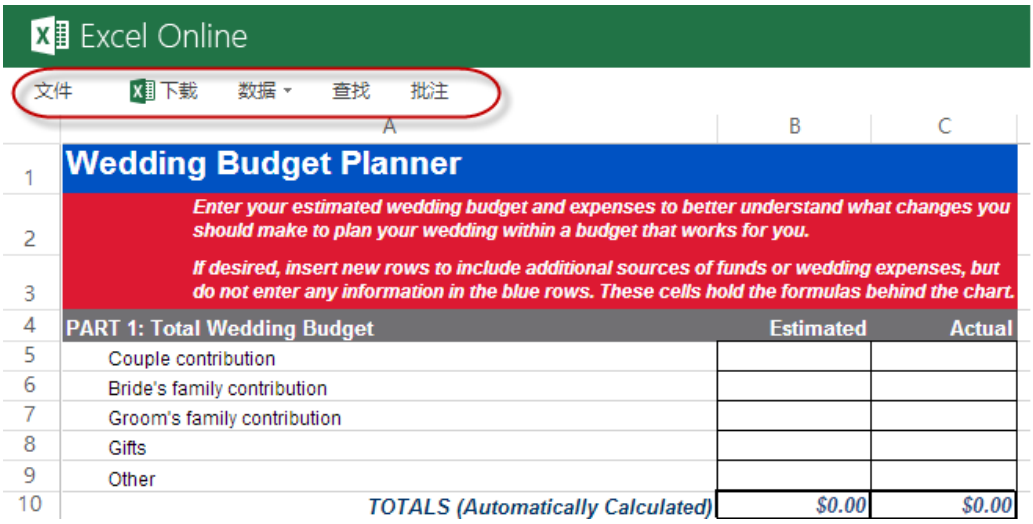
面那个是具体的文档地址，用 URLEncode 进行处理的链接地址。

非常酷的效果，使用这个服务唯一的要求就是你的网站是部署在互联网上的，这样服务才可以调用你的文档地址然后进行展示。

这个是使用微软的预览 Office 服务，当然你可以部署自己的 Office 预览服务，也就是需要安装 Office Web Apps 服务（系统要求为 Windows Server 2012）

一般情况下，Office Web Apps 要与其他应用配合使用，如下图所示（看起来还是很复杂的，这些东西好像也要独立安装在不同的机器）：

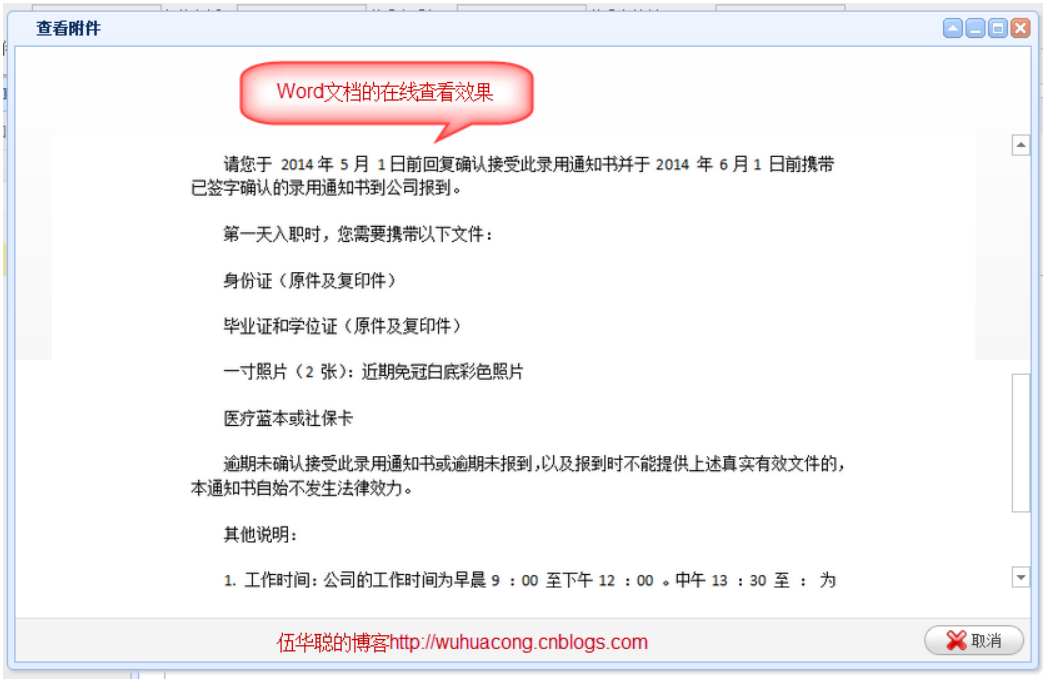
好在微软给我们提供了在线的 Office 文档预览服务，其实好像 Google Doc Viewer 也是可以的，但是已经不可使用了。这样，我们就可以利用公开的接口地址实现 Office 文档的在线预览了。以常用的 Excel 文档为例，它可以提供了完美的在线预览效果。



我们还注意到上面有一排菜单，可以展开进行相关功能的操作，尤其是文档的下载和打印很吸引。

这样我们就可以在我们的文档预览操作页面上进行集成了，下面是 Web 开发框架集成的一个例子，对 Office 文档进行查看。

Word 文档也是可以顺利进行预览，但是就没有打印和下载的功能了，不过预览的效果还是很不错的。



2.9.4. 把 Office 文档生成 HTML 文件后进行查看

上面说了，使用在线的 Office 预览服务来查看 Office 文档，如果我们的管理系统是在局域网内跑的应用，那么我们就是用不了了，那么我们如果需要使用这种在线预览 Office 文档的服务，应该如何操作呢?

虽然自行搭建 Office Web Apps 服务应用可以解决这个问题，但是一般来说，搭建这样

的平台环境，太过繁琐和昂贵了，有没有更好的方式实现 Office 文档的查看呢？

有的，下面我来介绍一下，如何使用 Aspose 组件把 Office 文档生成 HTML，然后进行查看的做法。

对应不同的 Office 文档，Aspose 提供了不同的组件，如 Aspose.Word、Aspose.Cells、Aspose.Slides 等不同的组件用来处理 Word/Excel/PPT 等几种文档。

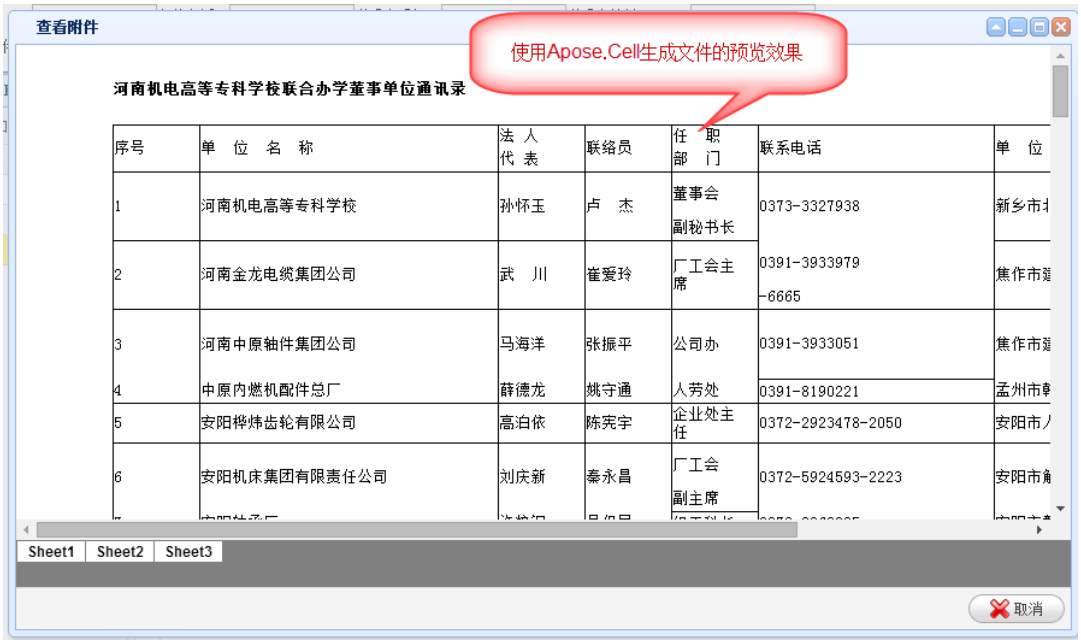
我们知道，Aspose 组件在处理 Office 文档方面非常的强大，但是也是收费软件，所以需要可以购买支持，但是我们这里纯粹讨论它的功能效果。

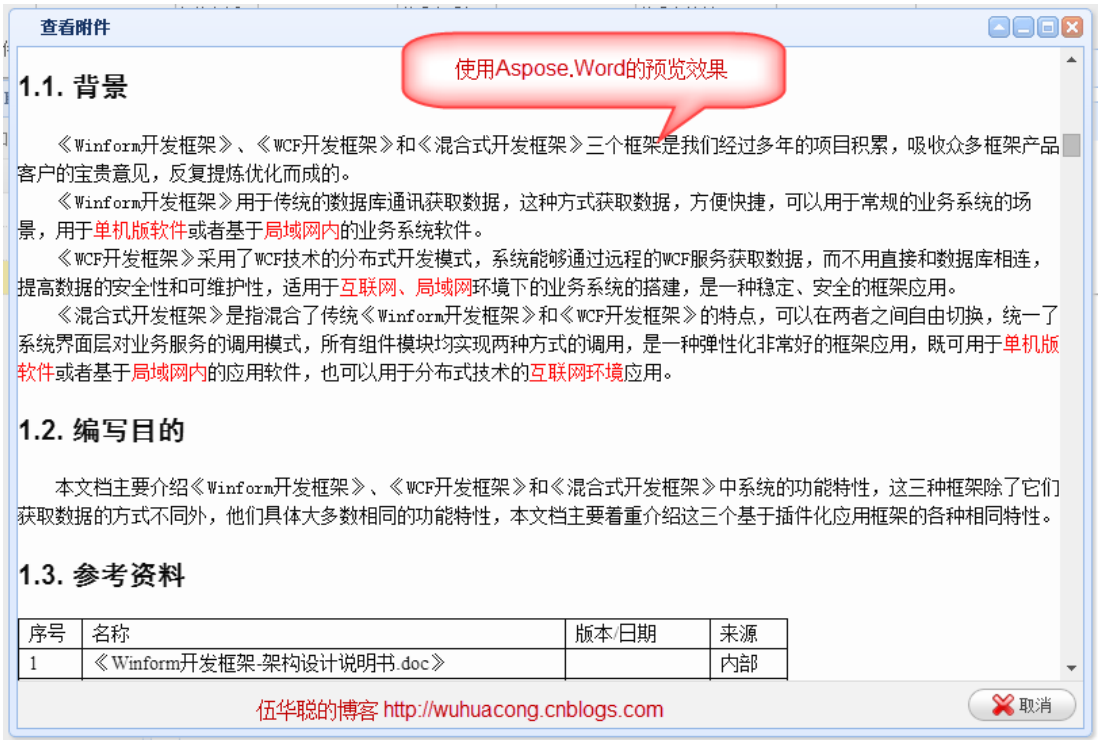
对不同的文件类型，我们调用不同的组件进行 HTML 的转换就可以了，核心部分代码如下所示。

```
#region 动态第一次生成文件
//检查本地Office文件是否存在，如不存在，先生成文件，然后返回路径供查看
string webPath = string.Format("/GenerateFiles/Office/{0}.htm", info.ID);
string generateFilePath = Server.MapPath(webPath);
if (!FileUtil.FileIsExist(generateFilePath))
{
    string templateFile = BLLFactory<FileUpload>.Instance.GetFilePath(info);
    templateFile = Path.Combine(System.AppDomain.CurrentDomain.BaseDirectory, templateFile.Replace("\\", "/"));

    if (ext == "doc" || ext == "docx")
    {
        Aspose.Words.Document doc = new Aspose.Words.Document(templateFile);
        doc.Save(generateFilePath, Aspose.Words.SaveFormat.Html);
    }
    else if (ext == "xls" || ext == "xlsx")
    {
        Workbook workbook = new Workbook(templateFile);
        workbook.Save(generateFilePath, SaveFormat.Html);
    }
    else if (ext == "ppt" || ext == "pptx")
    {
        templateFile = templateFile.Replace("/", "\\");
        PresentationEx pres = new PresentationEx(templateFile);
        pres.Save(generateFilePath, Aspose.Slides.Export.SaveFormat.Html);
    }
}
#endregion
```

这样，我们第一次使用的时候，判断目录里面是否有对应的 HTML 文件了，如果没有，就使用上面的代码生成就可以了，查看的时候，就返回对应的路径给客户端进行查看文件就可以了。

下面是几个文档的效果截图，供参考。


查看附件
使用Apose.Cell生成文件的预览效果
河南机电高等专科学校联合办学董事单位通讯录
序号
单 位 名 称
法 人 代 表
联络员
任 职 部 门
联系电话
单 位
1
河南机电高等专科学校
孙怀玉
卢 杰
董事会 副秘书长
0373-3327938
2
河南金龙电缆集团公司
武 川
崔爱玲
厂工会主席
0391-3933979 -6665
3
河南中原轴件集团公司
马海洋
张振平
公司办
0391-3933051
4
中原内燃机配件总厂
薛德龙
姚守通
人劳处
0391-8190221
5
安阳桦炜齿轮有限公司
高泊依
陈宪宇
企业处主任
0372-2923478-2050
6
安阳机床集团有限责任公司
刘庆新
秦永昌
厂工会 副主席
0372-5924593-2223
Sheet1
Sheet2
Sheet3
取消


查看附件
使用Aspose.Word的预览效果
1.1. 背景
《Winform开发框架》、《WCF开发框架》和《混合式开发框架》三个框架是我们经过多年的项目积累，吸收众多框架产品客户的宝贵意见，反复提炼优化而成的。
《Winform开发框架》用于传统的数据库通讯获取数据，这种方式获取数据，方便快捷，可以用于常规的业务系统的场景，用于单机版软件或者基于局域网内的业务系统软件。
《WCF开发框架》采用了WCF技术的分布式开发模式，系统能够通过远程的WCF服务获取数据，而不用直接和数据库相连，提高数据的安全性和可维护性，适用于互联网、局域网环境下的业务系统的搭建，是一种稳定、安全的框架应用。
《混合式开发框架》是指混合了传统《Winform开发框架》和《WCF开发框架》的特点，可以在两者之间自由切换，统一了系统界面层对业务服务的调用模式，所有组件模块均实现两种方式的调用，是一种弹性化非常好的框架应用，既可用于单机版软件或者基于局域网内的应用软件，也可以用于分布式技术的互联网环境应用。
1.2. 编写目的
本文档主要介绍《Winform开发框架》、《WCF开发框架》和《混合式开发框架》中系统的功能特性，这三种框架除了它们获取数据的方式不同外，他们具体大多数相同的功能特性，本文档主要着重介绍这三个基于插件化应用框架的各种相同特性。
1.3. 参考资料
序号
名称
版本/日期
来源
1
《Winform开发框架-架构设计说明书.doc》
内部
伍华聪的博客 http://wuhuacong.cnblogs.com
取消

2.10. Excel 数据导入导出

在我们常见的一些业务模块里面，往往 Excel 数据导入，是一个非常常见的功能，因此在 Web 框架里面提供对 Excel 数据的导出和导出操作，也是很必要的一个功能点。

我们知道，Web 上对 Excel 的处理和 Winform 的有所差异，如果是在 Web 上处理，我们需要把 Excel 文档上传到服务器上，然后读取文件进行显示，所以第一步是实现文件的上传操作，关于文件上传控件，具体可以参考上面小节的内容介绍。

Excel 数据的导入导出功能，在 Web 上的主界面如下所示。

导入界面如下所示。

在利用代码生成工具生成代码的时候，会同时上传一个 Import.cshtml 的视图，这个是用来做 Excel 数据的导入操作的。另外，在导入导出的控制器方法里面，绝大多数的代码逻辑已经准确生成，一般只需要进行一定的微调即可，这样可以减少很多编码的时间。

2.10.1. 控制器后台的操作支持

为了有效处理数据的导入，我们需要严格保证导入的数据是和模板的字段是匹配的，否则处理容易出错，也没有任何意义。为了实现这个目的，框架里面提供方法对字段进行检查，主要是确保 Excel 里面包含了完整的字段即可。

```
//导入或导出的字段列表
string columnString = "客户名称,编号,姓名,身份证号码,出生日期,性别,办公电话,家庭电话,传真,

/// <summary>
/// 检查Excel文件的字段是否包含了必须的字段
/// </summary>
/// <param name="guid">附件的GUID</param>
/// <returns></returns>
0 个引用
public ActionResult CheckExcelColumns(string guid)
{
    CommonResult result = new CommonResult();

    try
    {
        DataTable dt = ConvertExcelFileToTable(guid);
        if (dt != null)
        {
            //检查列表是否包含必须的字段
            result.Success = DataTableHelper.ContainAllColumns(dt, columnString);
        }
    }
    catch (Exception ex)
    {
        LogTextHelper.Error(ex);
        result.ErrorMessage = ex.Message;
    }

    return ToJsonContent(result);
}
```

把Excel文件转换为DataTable并验证是否具有指定的列名称。

而文件检查通过后，我们会在服务器解析对应的 Excel 文件，把它 Excel 里面的数据转换为 DataTable，最后初始化为实体类列表，并返回给调用页面就可以了。

```
/// <summary>
/// 获取服务器上的Excel文件，并把它转换为实体列表返回给客户端
/// </summary>
/// <param name="guid">附件的GUID</param>
/// <returns></returns>
0 个引用
public ActionResult GetExcelData(string guid)
{
    if (string.IsNullOrEmpty(guid))
    {
        return null;
    }

    List<ContactInfo> list = new List<ContactInfo>();

    DataTable table = ConvertExcelFileToTable(guid);
    if (table != null)
    {
        数据转换
    }

    其他转义操作

    var result = new { total = list.Count, rows = list };
    return ToJsonContentDate(result);
}
```

转换为是实体类集合

最后就是把 DataGrid 里面的数据保存到服务器对应的业务表里面了，在控制器里面保存业务对象的集合操作代码如下所示。

```
/// <summary>
/// 保存客户端上传的相关数据列表
/// </summary>
/// <param name="list">数据列表</param>
/// <returns></returns>
0 个引用
public ActionResult SaveExcelData(List<ContactInfo> list)
{
    CommonResult result = new CommonResult();
    if (list != null && list.Count > 0)
    {
        #region 采用事务进行数据提交

        DbTransaction trans = BLLFactory<Contact>.Instance.CreateTransaction();
        if (trans != null)
        {
            try
            {
                //int seq = 1;
                foreach (ContactInfo detail in list)
                {
                    //detail.Seq = seq++;//增加1
                    detail.CreateTime = DateTime.Now;
                    detail.Creator = CurrentUser.ID.ToString();
                    detail.Editor = CurrentUser.ID.ToString();
                    detail.EditTime = DateTime.Now;

                    BLLFactory<Contact>.Instance.Insert(detail, trans);
                }
                trans.Commit();
                result.Success = true;
            }
            catch (Exception ex)
            {
                LogTextHelper.Error(ex);
                result.ErrorMessage = ex.Message;
                trans.Rollback();
            }
        }
        #endregion
    }
    else
    {
        result.ErrorMessage = "导入信息不能为空";
    }

    return ToJsonContent(result);
}
```


而导出 Excel 数据的控制器方法，也主要就是生成一个 Excel 文件，供客户端下载即可。具体的代码逻辑如下所示。

```
/// <summary>
/// 根据查询条件导出列表数据
/// </summary>
/// <returns></returns>
0 个引用
public ActionResult Export()
{
    根据参数获取List列表

    把列表转换为DataTable

    把DataTable转换为Excel并输出

    //返回生成后的文件路径，让客户端根据地址下载
    return Content(filePath);
}
```

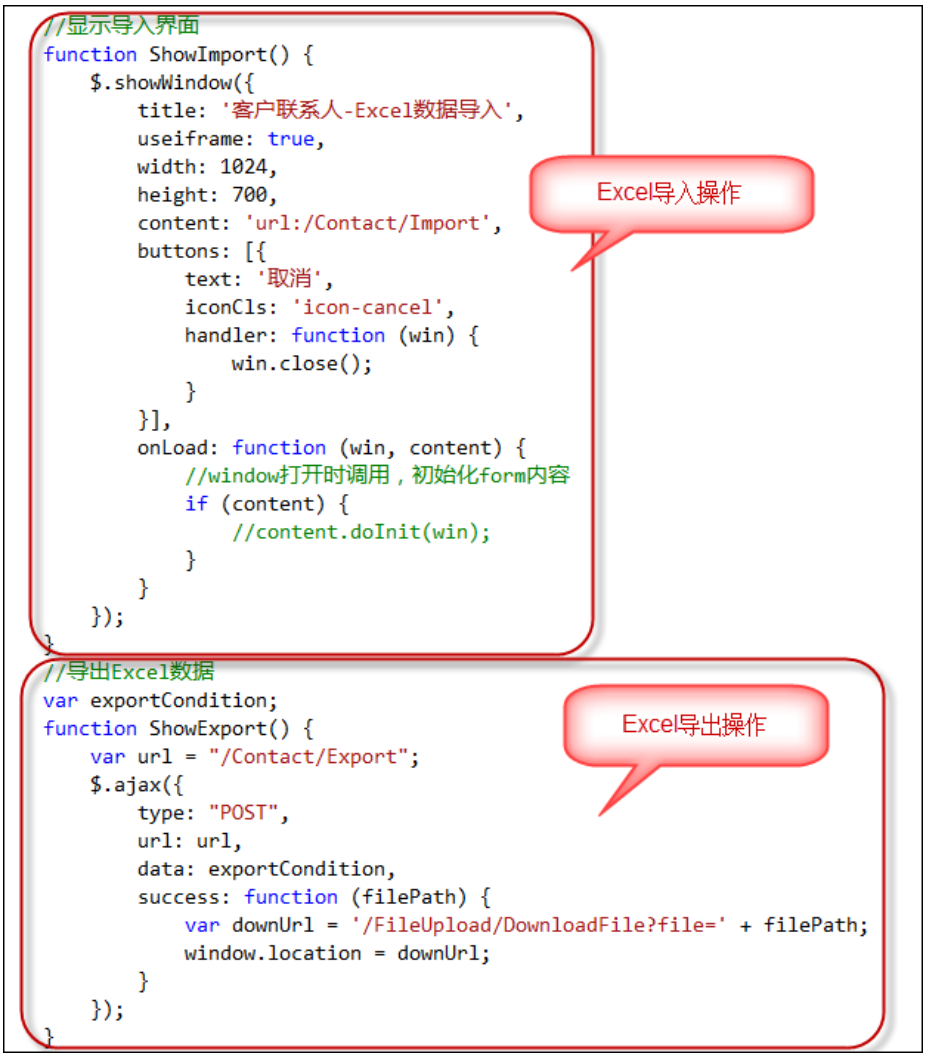
以上的代码，绝大部分都是通过代码生成工具，根据数据库的对应信息，进行动态生成的，一般情况下，只需要进行适当的微调即可，非常方便。

2.10.2. 界面层的代码

为了实现 Web 上的数据导入导出操作，我们需要增加两个按钮，一个是导入按钮，一个是导出按钮。

```
<td colspan="2" style="text-align:right">
    <a href="#" class="easyui-linkbutton" data-options="iconCls:'icon-search'" id="btnSearch">查询</a>
    <a href="javascript:void(0)" class="easyui-linkbutton" id="btnImport" iconcls="icon-excel" onclick="ShowImport()">导入</a>
    <a href="javascript:void(0)" class="easyui-linkbutton" id="btnExport" iconcls="icon-excel" onclick="ShowExport()">导出</a>
</td>
```

上面代码里面，对应的 JS 代码如下所示。


//显示导入界面
function ShowImport() {
$.showWindow({
title: '客户联系人-Excel数据导入',
useiframe: true,
width: 1024,
height: 700,
content: 'url:/Contact/Import',
buttons: [{
text: '取消',
iconCls: 'icon-cancel',
handler: function (win) {
win.close();
}
}],
onLoad: function (win, content) {
//window打开时调用，初始化form内容
if (content) {
//content.doInit(win);
}
}
});
}
Excel导入操作
//导出Excel数据
var exportCondition;
function ShowExport() {
var url = "/Contact/Export";
$.ajax({
type: "POST",
url: url,
data: exportCondition,
success: function (filePath) {
var downUrl = '/FileUpload/DownloadFile?file=' + filePath;
window.location = downUrl;
}
});
}
Excel导出操作

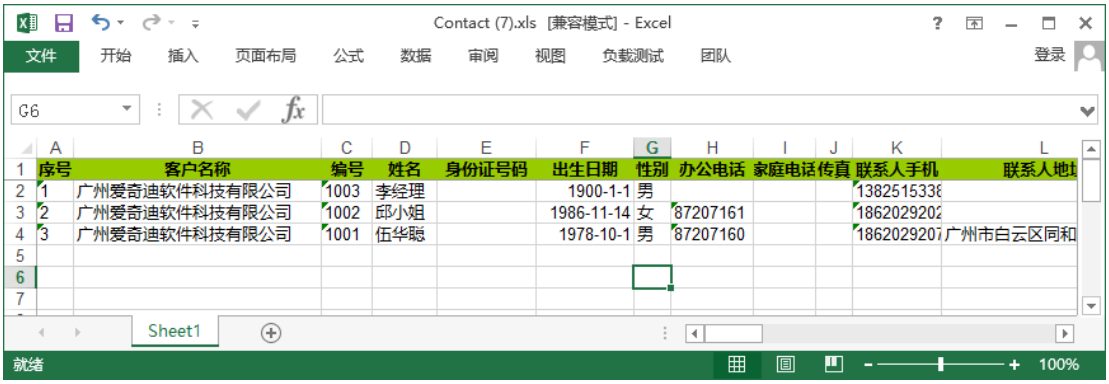
```
Import.cshtml

    //绑定添加按钮的事件
    function SaveEntity() {
        //判断表单的信息是否通过验证
        var validate = $("#ffAdd").form('validate');
        if (validate == false) {
            return false;
        }

        //构造集合对象
        var list = [];
        var rows = $("#grid").datagrid("getRows");
        for (var i = 0; i < rows.length; i++) {
            list.push({
                'Customer_ID': rows[i].Customer_ID, 'HandNo': rows[i].HandNo, 'Name': rows[i].Name, 'IDCarNo': rows[i]
                'OfficePhone': rows[i].OfficePhone, 'HomePhone': rows[i].HomePhone, 'Fax': rows[i].Fax, 'Mobile': rows
                'Email': rows[i].Email, 'QQ': rows[i].QQ, 'Note': rows[i].Note, 'Seq': rows[i].Seq, 'AttachGUID': rows
                'City': rows[i].City, 'District': rows[i].District, 'Hometown': rows[i].Hometown, 'HomeAddress': rows[
                'Eduction': rows[i].Eduction, 'GraduateSchool': rows[i].GraduateSchool, 'Political': rows[i].Political
                'Rank': rows[i].Rank, 'Department': rows[i].Department, 'Hobby': rows[i].Hobby, 'Animal': rows[i].Anim
                'MarriageStatus': rows[i].MarriageStatus, 'HealthCondition': rows[i].HealthCondition, 'Importance': ro
                'RelationShip': rows[i].RelationShip, 'ResponseDemand': rows[i].ResponseDemand, 'CareFocus': rows[i].C
                'BodyType': rows[i].BodyType, 'Smoking': rows[i].Smoking, 'Drink': rows[i].Drink, 'Height': rows[i].He
            });
        }

        var postData = { 'list': list };//可以增加其他参数，如{ 'list': list, 'Rucanghao': $("#Rucanghao").val() };
        postData = JSON.stringify(postData);

        $.ajax({
            url: '/Contact/SaveExcelData',
            type: 'post',
            dataType: 'json',
            contentType: 'application/json;charset=utf-8',
            traditional: true,
            success: function (data) {
                if (data.Success) {
                    //保存成功  1.关闭弹出层，2.刷新DataGird
                    $.messager.alert("提示", "保存成功");
                    $("#DivAdd").dialog("close");
                    //$("#grid").datagrid("reload");
                    $('#grid').datagrid('loadData', { total: 0, rows: [] });
                    $("#ffAdd").form("clear");
                }
                else {
                    $.messager.alert("提示", "保存失败:" + data.ErrorMessage);
                }
            },
            data: postData
        });
```

构建列表集合

集合转换为JSON格式

在服务端的控制器上保存数据

导出的 Excel 界面效果如下所示。



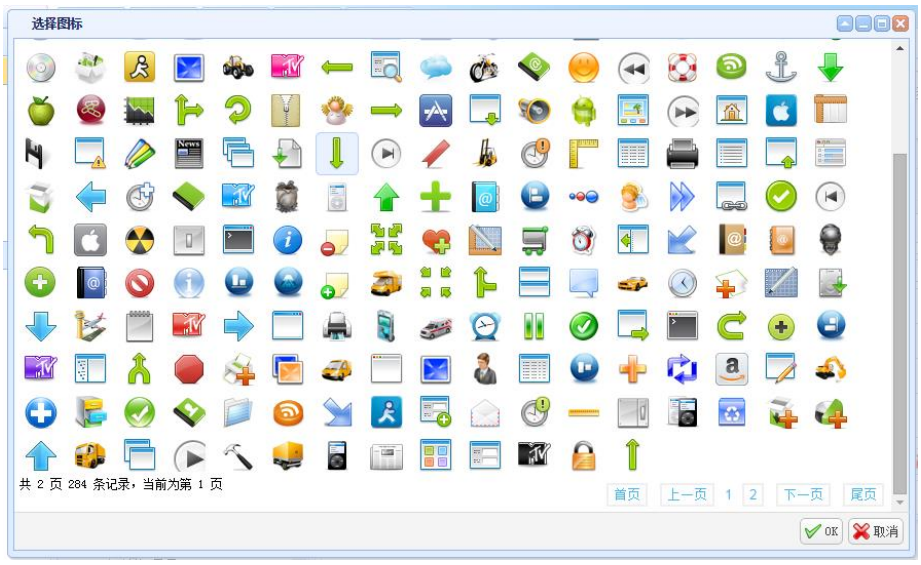
2.11. 图标样式生成和选择

在很多 Web 系统中，一般都可能提供一些图标的选择，方便配置按钮，菜单等界面元素的图标，从而是 Web 系统界面看起来更加美观和协调。但是在系统中一般内置的图标样式相对比较有限，而且硬编码写到样式表里面，这样给我们扩展使用有很多的不方便。基于这个原因，我想如果能够独立一个模块，自动根据图标生成图标 CSS 样式文件，并存储相应的记录到数据库里面，方便我们查询显示，那样我们使用起来就很方便了，最后有了这些数据，只需要做一个通用的图标选择界面，并可以在很多地方重用了。

Web 框架提供了一个通用的图标生成、图标选择管理模块，非常适合于动态设置菜单、按钮的图标操作，系统提供了一个对菜单图标样式动态配置的案例。

图标的展示通过 MVC 的分页处理实现，分页采用了开源组件 MvcPager，非常方便管理和查看。图标管理界面如下所示。

选择图标，弹出的图标选择对话框界面如下所示。



2.11.1. NVelocity 组件知识

为了方便根据读取的图标文件列表，生成对应的图标样式文件，我们可以利用 NVelocity 组件，基于模板进行 CSS 样式文件的生成。关于 NVelocity 的使用，可以参考我多篇关于它的介绍，这个组件非常强大，我自己的代码生成工具也是基于它编写了很多模板进行代码生成，具体可以参考一下《使用 NVelocity 生成内容的几种方式》这篇文章。

有了这些准备，我们可以定义一个模板的文件用来生成样式文件了，我们先看最终的样式文件效果。

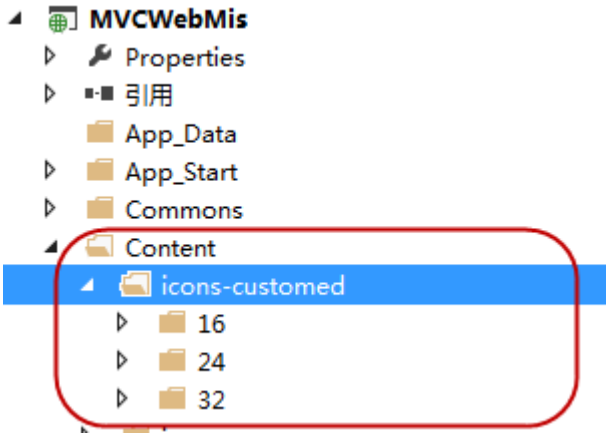
```
.icon-table{background:url('table.png') no-repeat center center;}
.icon-telephone{ background:url('telephone.png') no-repeat center center;}
.icon-user{background:url('user.png') no-repeat center center;}
.icon-view{background:url('view.png') no-repeat center center;}
.icon-word{background:url('word.png') no-repeat center center;}
```

根据以上组织效果，我们可以定义一个模板内容如下所示。

```
#foreach($item in ${FileNameList})
.${item.Text}{
    background:url('${item.Value}') no-repeat center center;
}
#end
```

其中 FileNameList 变量是一个基于名称和值的集合对象，我们遍历它进行生成就可以了。

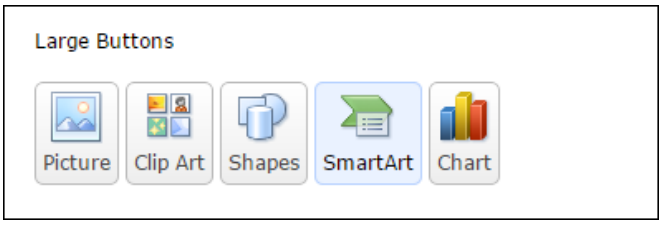
有了模板，我们还需要组织好对应的文件目录，一般来说，Web 的图标可以使用 16,24,32 这些标准大小的图标，适应不同场合的需要。因此我们创建几个不同的目录，并放入对应的模板文件和图标文件。



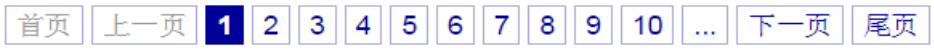
2.11.2. 图标的分页展示

为了有效查看我们生成在数据库的图标列表，我们需要一个合理的界面表现方式，用来显示图标信息。传统的使用 datagrid 的方式比较呆板，也不是很方便，所以我们需要自定义分页处理进行展现，基于重用一些优秀组件的原则，我侧重于使用一些现成的组件模块，MVC 分页方面，考虑使用杨涛的 MVC 分页控件（http://www.webdiyer.com/mvcpager/），这个功能看起来很不错。

图标的展现方式，我希望通过 easyui 的这个例子进行展现一组图表的效果。



然后系统通过把它们进行分页处理，选择一些好的分页样式表现方式



:

2.11.3. 控制器后台的操作支持

```
/// <summary>
/// 生成图标文件
/// </summary>
/// <param name="iconSize">图表尺寸，可选16,32等</param>
/// <returns></returns>
0 个引用
public ActionResult GenerateIconCSS(int iconSize = 16)
{
    CommonResult result = new CommonResult();

    string realPath = Server.MapPath("~/Content/icons-customed/" + iconSize);
    if (Directory.Exists(realPath))
    {
        List<CListItem> list = GetImageToList(realPath, iconSize);

        try
        {
            //使用相对路径进行构造处理
            string template = string.Format("Content/icons-customed/{0}/icon.css.vm", iconSize);
            NVelocityHelper helper = new NVelocityHelper(template);
            helper.AddKeyValue("FileNameList", list);

            helper.FileNameOfOutput = "icon";
            helper.FileExtension = ".css";
            helper.DirectoryOfOutput = realPath;//指定实际路径

            string outputFilePath = helper.ExecuteFile();
            if (!string.IsNullOrEmpty(outputFilePath))
            {
                result.Success = true;

                //写入数据库
                bool write = BLLFactory<Icon>.Instance.BatchAddIcon(list, iconSize);
            }
        }
        catch (Exception ex)
        {
            LogTextHelper.Error(ex);
            result.ErrorMessage = ex.Message;
        }
    }
    else
    {
        result.ErrorMessage = "没有找到图片文件";
    }

    return ToJsonContent(result);
}
```

获取指定目录下的图片文件名列表

使用模板生成样式文件

写入图标信息到数据库

上面是对图标生成的相关操作，为了展示现有的图标信息，我们还需要根据条件从分页数据库里面获取对应的图标对象信息，方便我们展示在界面上。

```
/// <summary>
/// 根据条件获取基于PagedList的对象集合，并返回给分页视图使用
/// </summary>
/// <param name="id">分页页码</param>
/// <param name="iconsize">图标尺寸</param>
/// <returns></returns>
3 个引用
private PagedList<IconInfo> GetPageList(int? id, int? iconsize = 16)
{
    int size = iconsize ?? 16;
    int pageIndex = id ?? 1;
    int pageSize = 200;

    PagerInfo pagerInfo = new PagerInfo();
    pagerInfo.CurrenetPageIndex = pageIndex;
    pagerInfo.PageSize = pageSize;

    string where = string.Format("iconsize = {0}", size);
    List<IconInfo> list = BLLFactory<Icon>.Instance.FindWithPager(where, pagerInfo);
    PagedList<IconInfo> pageList = pageList = new PagedList<IconInfo>(list, pageIndex, pageSize, pagerInfo.RecordCount);
    return pageList;
}

/// <summary>
/// 根据条件获取分页数据集合，并绑定到视图里面
/// </summary>
/// <param name="id">分页页码</param>
/// <param name="iconsize">图标尺寸</param>
/// <returns></returns>
0 个引用
public ActionResult AjaxSearchPost(int? id = 1, int? iconsize = 16)
{
    PagedList<IconInfo> pageList = GetPageList(id, iconsize);
    return View("AjaxSearchPost", pageList);
}
```

通用的图标分页查询操作

根据条件，获取图标对象到具体的PageList集合里

根据条件分页获取数据

2.11.4. 界面层的代码

在列表展示界面里面，为了更加通用一点，我们需要设置一些图标大小的条件，供我们查询展示，这里使用的界面代码如下所示。

```
@using (Html.BeginForm("AjaxSearchPost", "Icon", new RouteValueDictionary { { "id", "" } }, FormMethod.Get))
{
    <span>尺寸：</span>
    <select class="easyui-combobox" id="IconSize" name="IconSize" style="width:100px">
        <option value="16">16x16</option>
        <option value="24">24x24</option>
        <option value="32">32x32</option>
    </select>
    <input type="submit" value="搜索(S)" accesskey="S" />
    <a href="#" class="easyui-linkbutton" data-options="iconCls:'icon-ok'" id="btnGenerateCSS">重新生成图表库</a>
}
```

图标大小条件

图标列表展示界面代码如下所示。

```
<div id="contents">
    @using Webdiyer.WebControls.Mvc;
    @model PagedList<WHC.MVCWebMis.Entity.IconInfo>
    @foreach (var item in Model)
    {
        <a href="javascript:void(0)" class="easyui-linkbutton" id="@item.ID"
           data-options="plain:true,iconCls:'@item.IconCls',size:'large'"> </a>
    }
</div>
<div>
    <div style="float:left;width:50%">
        共 @Model.TotalPageCount 页 @Model.TotalItemCount 条记录，当前为第 @Model.CurrentPageIndex 页
    </div>
    @Html.Pager(Model, new PagerOptions { PageIndexParameterName = "id" }, new { style = "float:right", id = "badoopager" })
</div>
```

使用MvcPager分页控件对模型数据进行展示

生成图标样式的 JS 代码如下所示。

```
//绑定按钮的的点击事件
function BindEvent() {
    $("#btnGenerateCSS").click(function () {
        $.messager.confirm("操作确认", "您确认重新生成图标记录吗？", function (action) {
            if (action) {
                //图表尺寸
                var iconSize = $("#IconSize").combobox('getValue');
                //alert(iconSize);
                var postData = "";

                $.ajax({
                    type: 'POST',
                    url: '/Icon/GenerateIconCSS?iconSize=' + iconSize,
                    dataType: 'json',
                    data: postData,
                    success: function (data) {
                        if (data.Success) {
                            showTips("操作成功");
                            location.reload();
                        }
                        else {
                            showError("操作失败：" + data.ErrorMessage, 3000);
                        }
                    }
                });
            }
        });
    });
}
```

上面介绍了图标的生成以及列表分页展示图标内容，但是有时候，我们还需要一个可以弹出单独界面选择图标样式的操作，如我们在配置菜单的地方，需要设置菜单的图标样式。

```
//弹出界面选择图标并返回
function SelectIcon(id, value) {
    $.showWindow({
        title: '选择图标',
        useiframe: true,
        width: 960,
        height: 640,
        content: 'url:/Icon/Select',
        data: { id: $(id), value: $(value) },
        buttons: [{
            text: 'OK',
            iconCls: 'icon-ok',
            handler: 'doOK'
        }, {
            text: '取消',
            iconCls: 'icon-cancel',
            handler: function (win) {
                win.close();
            }
        }],
        onLoad: function (win, content) {
            //window打开时调用，初始化form内容
            if (content) {
                content.doInit(win);
            }
        }
    });
}
```

弹出选择图标样式的对话框后，在其中的图标，都响应了单击事件，一旦单击图标，那么会获得当前的样式，设置到主界面上，并关闭弹出的对话框。

```
<div id="contents">
    @using Webdiyer.WebControls.Mvc;
    @model PagedList<WHC.MVCWebMis.Entity.IconInfo>
    @foreach (var item in Model)
    {
        <a href="javascript:void(0)" class="easyui-linkbutton" onclick="SelectItem(this, '@item.IconCls')" id="@item.ID"
           data-options="plain:true,iconCls:'@item.IconCls',size:'large',toggle:true"> </a>
    }
</div>
<div>
    <div style="float:left;width:50%">
        共 @Model.TotalPageCount 页 @Model.TotalItemCount 条记录，当前为第 @Model.CurrentPageIndex 页
    </div>
    @Html.Pager(Model, new PagerOptions { PageIndexParameterName = "id" }, new { style = "float:right", id = "badoopager" })
</div>
```

单击选中后执行处理

对应的单击响应脚本函数如下所示。

```
//选择某个图标
function SelectItem(my, iconCls) {
    //如果选中，则取消选中
    $('a.easyui-linkbutton').each(function () {
        var opts = $(this).linkbutton('options');
        if (opts.selected) {
            $(this).linkbutton('unselect');
        }
    });

    //设置目标样式，并关闭弹出对话框
    //my.linkbutton('select');
    if (winObject) {
        var id = winObject.getData('id');
        id.attr('class', iconCls);

        var value = winObject.getData('value');
        value.val(iconCls);
        winObject.close();
    }
}
```

2.12. 统计图表 Highcharts 使用

在我们做各种应用的时候，我们可能都会使用到图表统计，以前接触过一些不同的图表控件，但图表控件 Highcharts，其强大的功能和丰富的互动效果，令人难以忘怀。本篇主要介绍在 Web 开发中使用图表控件 Highcharts，以及对其进行统一汉化等操作，让我们的程序功能更加丰富，内容更加美观。

统计图表是很多应用程序需要拥有的功能，为了更好展示图表的使用操作，框架提供了多种样式的图表演示。

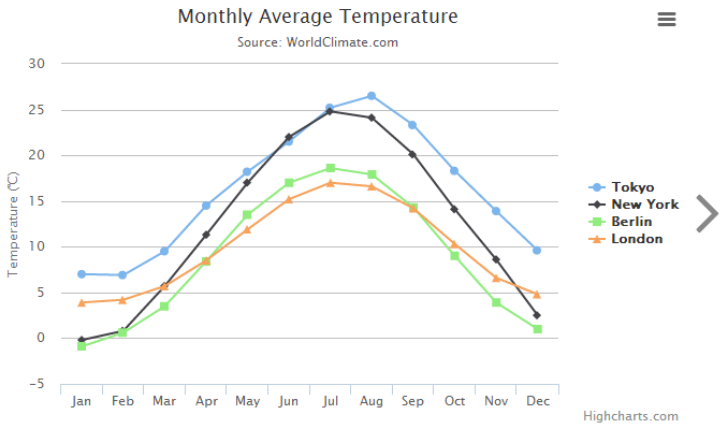
2.12.1. Highcharts 基础介绍

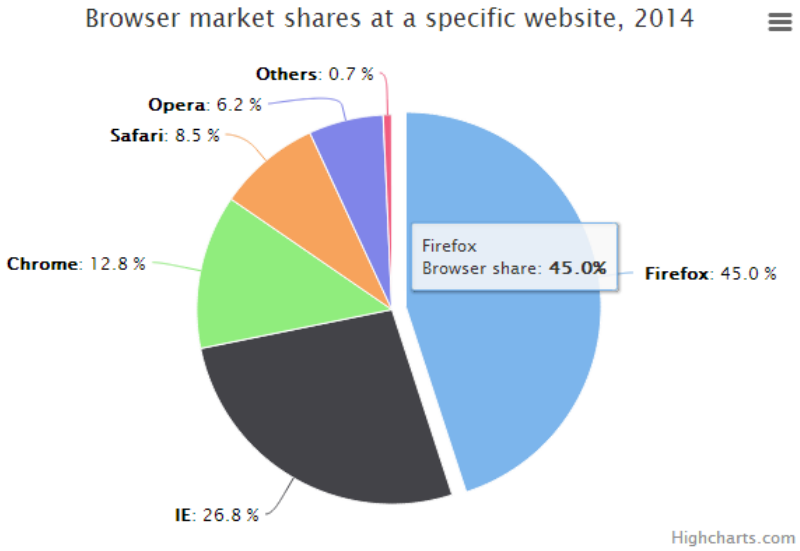
Highcharts 是一个非常流行，界面美观的纯 Javascript 图表库。它主要包括两个部分：Highcharts 和 Highstock。Highcharts 可以为您的网站或 Web 应用程序提供直观，互动式的图表。目前支持线，样条，面积，areaspline，柱形图，条形图，饼图和散点图类型。Highstock 可以为您方便地建立股票或一般的时间轴图表。它包括先进的导航选项，预设的日期范围，日期选择器，滚动和平移等等。

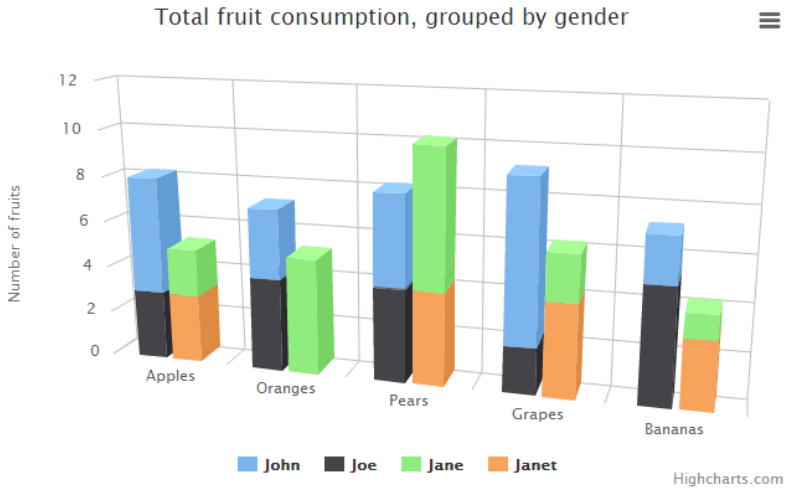
HighChartS 官网：http://www.highcharts.com/

HighCharts Demo:http://www.highcharts.com/demo/

Highcharts 支持曲线图、饼图、柱状图、仪表图、散点图等等几十种图形，界面展示效果非常丰富，3D 效果也很好看。列出几个供参考下吧







Highcharts 使用 jQuery 等 Javascript 框架来处理基本的 Javascript 任务。因此，在使用 Highcharts 之前，需要在页面头部引用这些脚本文件。如果你使用 jQuery 作为基本框架，那么你需要在页面头部同时引用 jQuery 和 Hightcharts 两个文件就可以了。

2.12.2. 控制器后台的操作支持

大多数情况下，我们都是通过动态构造数据的方式，提供数据给图表控件，实现图表的展示的，这种一般通过 JSON 数据进行传递，这里以集团公司人员和部门组成为例，构建用于图表展示的 JSON 数据。

集团人员分布情况数据生成如下：

```
/// <summary>
/// 统计各个分子公司的人数，返回Json字符串，供图表统计
/// </summary>
/// <returns></returns>
0 个引用
public ActionResult GetCompanyUserCountJson()
{
    Dictionary<string, int> dict = new Dictionary<string, int>();

    List<OUInfo> ouList = BLLFactory<OU>.Instance.GetTopGroup();
    foreach (OUInfo info in ouList)
    {
        List<OUInfo> companyList = BLLFactory<OU>.Instance.GetAllCompany(info.ID);
        foreach (OUInfo companyInfo in companyList)
        {
            string condition = string.Format("Company_ID={0} AND Deleted=0", companyInfo.ID);
            int count = BLLFactory<User>.Instance.GetRecordCount(condition);
            if (!dict.ContainsKey(companyInfo.Name))
            {
                dict.Add(companyInfo.Name, count);
            }
        }
    }

    return ToJsonContent(dict);
}
```

图表的数据是一个字典的JSON数据

部门分布情况的数据生成如下。

```
/// <summary>
/// 获取公司部门的数量，返回Json字符串，供图表统计
/// </summary>
/// <returns></returns>
0 个引用
public ActionResult GetCompanyDeptCountJson()
{
    Dictionary<string, int> dict = new Dictionary<string, int>();

    List<OUInfo> ouList = BLLFactory<OU>.Instance.GetTopGroup();
    foreach (OUInfo info in ouList)
    {
        List<OUInfo> companyList = BLLFactory<OU>.Instance.GetAllCompany(info.ID);
        foreach (OUInfo companyInfo in companyList)
        {
            string condition = string.Format("Company_ID={0} AND Deleted=0", companyInfo.ID);
            int count = BLLFactory<OU>.Instance.GetRecordCount(condition);
            if (!dict.ContainsKey(companyInfo.Name))
            {
                dict.Add(companyInfo.Name, count);
            }
        }
    }

    return ToJsonContent(dict);
}
```

图表的数据是一个字典的JSON数据

2.12.3. 界面层的代码

使用图表控件 Highcharts 的时候，需要确认 JS 应用正确，如下所示。

```
<head>
<title>用户管理</title>
<meta name="viewport" content="width=device-width" />
@using System.Web.Optimization;
@Scripts.Render("~/bundles/jquery")
@Styles.Render("~/Content/css")
@Scripts.Render("~/bundles/jquerytools")
@Styles.Render("~/Content/jquerytools")
@Scripts.Render("~/bundles/jquerytools/highcharts")
```

我们来分析下一个饼图的 Demo 代码，其中图表的 Series 数据，主要就是字典集合的 JSON 格式，因此我们可以通过动态从后台获取对应的数据，从而实现我们动态的图表了。

```
series: [{
    type: 'pie',
    name: 'Browser share',
    data: [
        ['Firefox',   45.0],
        ['IE',       26.8],
        {
            name: 'Chrome',
            y: 12.8,
            sliced: true,
            selected: true
        },
        ['Safari',    8.5],
        ['Opera',     6.2],
        ['Others',   0.7]
    ]
}]
```

如我一般倾向于使用 Jquery 的 Ajax 方式，调用后台获得数据，然后进行绑定的。首先我们先在脚本函数里面，初始化一个 chart 对象，并把其中涉 series 数据 data 设置为空就是了。

```
//初始化对象
$(function () {
    var chart1 = new Highcharts.Chart({
        chart: {
            renderTo: "container1",
            plotBackgroundColor: null,
            plotBorderWidth: null,
            plotShadow: false,
        },
        title: {
            text: '集团分子公司人员组成'
        },
        tooltip: {
            pointFormat: '{series.name}: <b>{point.y}</b>'
        },
        plotOptions: {
            pie: {
                allowPointSelect: true,
                cursor: 'pointer',
                dataLabels: {
                    enabled: true,
                    format: '<b>{point.name}</b>: {point.percentage:.1f} %',
                    style: {
                        color: (Highcharts.theme && Highcharts.theme.contrastTextColor) || 'black'
                    }

                },
                //showInLegend: true
            }
        },
        series: [{
            type: 'pie',
            name: '人员数量',
            data: []
        }]
    });
```

初始化一个饼图对象，数据后面通过动态方式赋值

先以空白集合初始化

第二步是通过 Ajax 调用后台连接获得数据，然后绑定到具体的属性上就可以了，具体代码如下所示。

```
//通过Ajax获取图表1数据
$.ajaxSettings.async = false;
var data1 = [];
$.getJSON("/User/GetCompanyUserCountJson", function (dict) {
    for (var key in dict) {
        if (dict.hasOwnProperty(key)) {
            data1.push([key, dict[key]]);
        }
    };
    chart1.series[0].setData(data1);
});
```

获取数据，并初始化图表对象

而图表的 HTML 代码则是如下所示，主要就是新增一个 div，id 为 container1，用来放置图表就是了。

```
<div class="box-title">
    <div style="float: left">
        <img src="~/Content/icons-customed/16/user.png" alt="" width="20" height="20" />
        图表1
    </div>
    <div style="float: right; padding-right: 10px;">更多</div>
</div>
<div class="box-content" style="height: 310px">
    <div id="container1" style="height: 300px;max-width:500px"></div>
</div>
```

2.13. 弹出扩展对话框

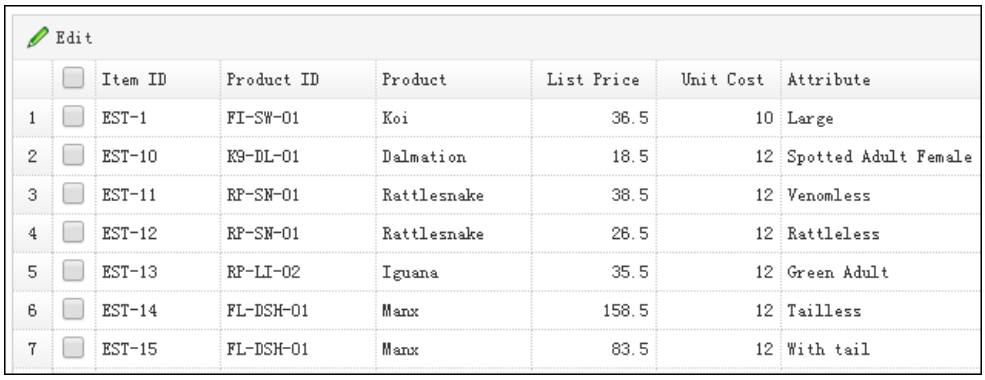
在 Web 框架里面，我们把各个层的内容，都统一放在一个 index.cshtml 的视图界面里面，包括主列表界面、新增编辑界面、查看详细信息界面等内容，以及一些选择性的对话框，都统一放在一个 index.cshtml 的文件里面。

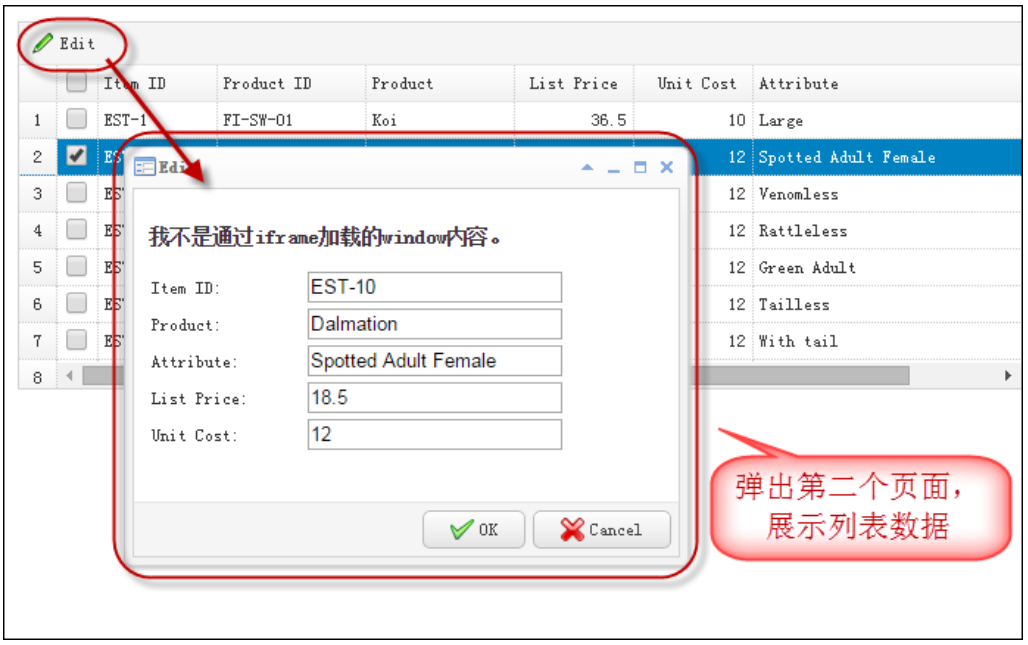
这些 Div 层都是基于 EasyUI 本身的对话框窗体，但是有时候，我们可能需要重用一些窗体界面，这些如果每次放到一个视图里面，就会导致视图代码比较重复的情况。而且普通的 EasyUI 对话框层，它的视图只能在布局内部拖动，不能拖出外面，导致只能比较小的高度。

为了解决这个问题，在 Web 框架的部分界面里面，我们使用了基于 EasyUI 的扩展类库，以便扩充弹出式对话框的功能，实现可以打开外面页面的功能，同时也能在整个 Web 空间上自由拖动。

2.13.1. 标准使用实例

标准实例里面，在第一个页面的 DataGrid 里面，通过传值方式，在对话框里面调用第二个页面展示记录数据。





其界面代码如下所示：

```
$(function(){
    $('#dg').datagrid({
        columns:[[
            {checkbox: true},
            {field: 'itemid', width: 80, title: 'Item ID', sortable: true},
            {field: 'productid', width: 100, title: 'Product ID'},
            {field: 'productname', width: 100, title: 'Product'},
            {field: 'listprice', width: 80, align: 'right', title: 'List Price', sortable: true},
            {field: 'unitcost', width: 80, align: 'right', title: 'Unit Cost'},
            {field: 'attr1', width: 250, title: 'Attribute'},
            {field: 'status', width: 60, align: 'center', title: 'Status', hidden: true}
        ]],
        singleSelect: true,
        rownumbers: true,
        url: '../window/datagrid_data1.json',
        height: 250,
        width: 700,
        toolbar:[{
            text: 'Edit',
            iconCls: 'icon-edit',
            handler: doEdit
        }]
    });
});
```

DataGrid初始化，并增加编辑按钮

其中 DataGrid 里面的编辑按钮事件脚本如下所示。

```
function doEdit(){
    var row = $('#dg').datagrid('getSelected');
    if(row){
        $.showWindow({
            title: 'Edit',
            content: 'url:window/_content3.html',
            data:{record: row, datagrid: $('#dg')},
            buttons:[{
                text: 'OK',
                iconCls: 'icon-ok',
                handler: 'doOK' //此方法在_content3.html中
            },{
                text: 'Cancel',
                iconCls: 'icon-cancel',
                handler: function(win){
                    win.close();
                }
            }],
            onLoad: function(win, body){
                //window打开时调用，初始化form内容
                var record = win.getData('record');
                body.$('#itemid').val(record.itemid);
                body.$('#productname').val(record.productname);
                body.$('#attr1').val(record.attr1);
                body.$('#listprice').val(record.listprice);
                body.$('#unitcost').val(record.unitcost);
            }
        });
    }else{
        $.messager.alert('警告','请选择要编辑的行。','warning');
    }
}
```

传递主页面的数据对象

打开第二个页面的时候，初始化界面元素的值

第二个界面的 JS 和页面代码如下所示。

javascript:

```javascript
function doOK(win){
    var target = win.getData('datagrid');
    var row = win.getData('record');
    var index = target.datagrid('getRowIndex', row);
    target.datagrid('updateRow',{
        index: index,
        row: {
            listprice: $('#listprice').val(),
            unitcost: $('#unitcost').val()
        }
    });
    win.close();
}

function doCancel(win){
    win.close();
}
```

确认和取消的事件处理

html:

```html
<div class="window-form">
    <h3>我不是通过iframe加载的window内容。</h3>
    <div>
        <label class="form-label">Item ID:</label>
        <input type="text" id="itemid" readonly>
    </div>
    <div>
        <label class="form-label">Product:</label>
        <input type="text" id="productname" readonly>
    </div>
    <div>
        <label class="form-label">Attribute:</label>
        <input type="text" id="attr1" readonly>
    </div>
    <div>
        <label class="form-label">List Price:</label>
        <input type="text" id="listprice">
    </div>
    <div>
        <label class="form-label">Unit Cost:</label>
        <input type="text" id="unitcost">
    </div>
</div>
```

从上面的例子，可以看到，主界面里面可以传值给第二个页面，而且可以在对话框初始化的时候，对第二个页面的内容进行初始化。第二个界面里面，可以响应确认和取消按钮事件，并获得主界面传递过来的参数对象，并进行调用处理。

2.13.2. Web 框架案例

在 Web 框架里面，也用到了一些这样的弹出对话框，如选择图标样式的对话框，它本身是一个可以重用的对话框界面，是一个独立的页面。当我们选择图标执行一个处理事件后，会给主界面的控件元素赋值，并关闭对话框。

另外就是每个自动生成的视图界面，都有一个导入 Excel 的界面，这个页面是一个独立

的视图界面（import.cshtml），因此这里也是属于使用扩展的弹出式对话框操作，不过这个界面操作并没有传入参数。显示导入数据的界面的 JS 代码如下所示。

```
//显示导入界面
function ShowImport() {
    $.showWindow({
        title: '客户联系人-Excel数据导入',
        useiframe: true,
        width: 1024,
        height: 700,
        content: 'url:/Contact/Import',
        buttons: [{
            text: '取消',
            iconCls: 'icon-cancel',
            handler: function (win) {
                win.close();
            }
        }],
        onLoad: function (win, content) {
            //window打开时调用，初始化form内容
            if (content) {
                //content.doInit(win);
            }
        }
    });
}
```

还有就是在联系人管理模块里，新建联系人的时候，选择所属客户，会弹出一个独立的客户列表界面供选择，这个也是独立页面的对话框。这个界面会传入两个对象参数。

```
//绑定选择客户按钮的事件
function BindSelectCustomerEvent() {
    $("#btnSelectCustomer").click(function () {
        $.showWindow({
            title: '选择客户',
            useiframe: true,
            width: 900,
            height: 600,
            content: 'url:/Customer/SelectCustomer',
            data: { id: $('#Customer_ID'), name: $('#Customer_Name') },
            buttons: [{
                text: '确认',
                iconCls: 'icon-ok',
                handler: 'doOK' //此方法在弹出页面中
            }, {
                text: '取消',
                iconCls: 'icon-cancel',
                handler: function (win) {
                    win.close();
                }
            }],
            onLoad: function (win, content) {
                //window打开时调用，初始化form内容
                if (content) {
                    content.doInit(win);
                }
            }
        });
    });
}
```

传递参数到目标界面

SelectCustomer.cshtml | index.cshtml

```
//初始化操作
function doInit(win) {
    //var record = win.getData('record');
    //$('#itemid').val(record.itemid);
}

//页面确认操作
function doOK(win) {
    var id = win.getData('id');
    var name = win.getData('name');

    var row = $('#grid').datagrid('getSelected');
    if (row) {
        id.val($("#grid").datagrid('getSelections')[0].ID);//客户ID
        name.val($("#grid").datagrid('getSelections')[0].Name);//客户名称

    } else {
        $.messager.alert('提示', '请选择一条记录。', 'warning');
    }

    win.close();
}
```

根据传入对象，处理主界面控件的内容更新。

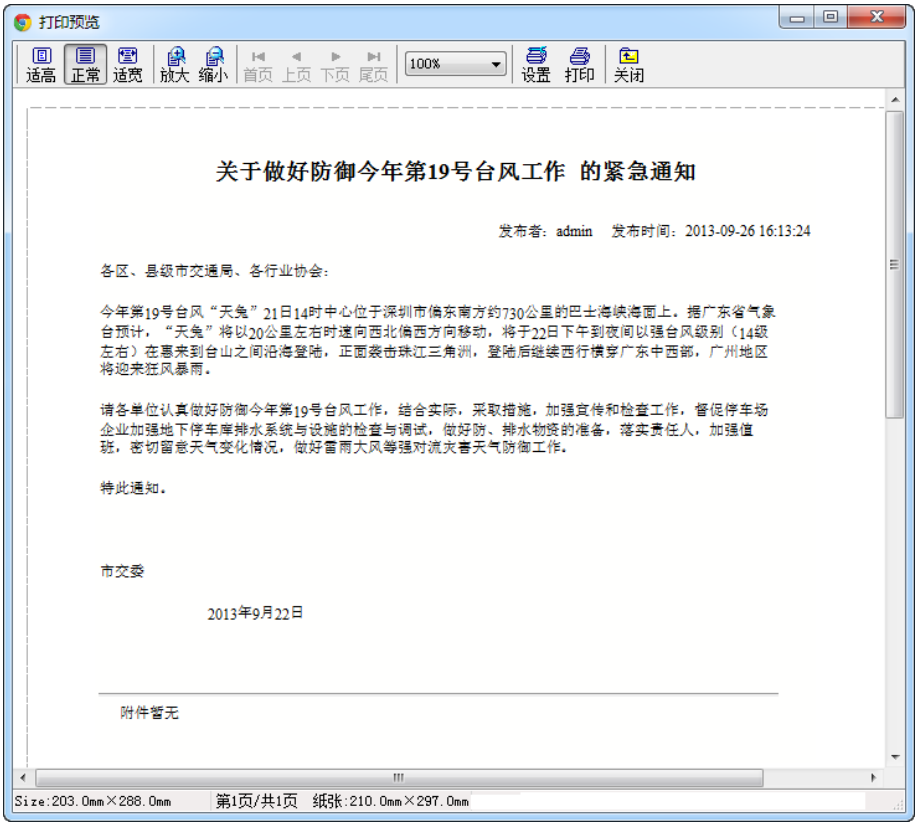
在弹出的第二个界面里面的页面确认操作中，对传入过来的对象进行重新赋值，达到返回对话框值的目的，通过关闭弹出对话框。

2.14. LODOP 打印模块的使用

2.14.1. LODOP 打印控件介绍

在 Web 框架里面，通知公告内容的打印预览界面如下所示，该模块集成了 LODOP 打印控件，因此预览效果非常美观，这个打印控件在很多场合表现出色，还可以用作证件的套打操作。

普通内容的打印界面如下所示：



证件套打的打印界面如下所示。

Lodop 是专业 WEB 控件,用它既可裁剪输出页面内容，又可用程序代码直接实现复杂打印。控件功能强大,却简单易用,所有调用如同 JavaScript 扩展语句,主要接口函数如下：

- PRINT_INIT(strPrintTaskName) 打印初始化
- SET_PRINT_PAGESIZE(intOrient,intPageWidth,intPageHeight,strPageName) 设定纸张大小
- ADD_PRINT_HTM(intTop,intLeft,intWidth,intHeight,strHtml) 增加超文本项
- ADD_PRINT_TEXT(intTop,intLeft,intWidth,intHeight,strContent) 增加纯文本项
- ADD_PRINT_TABLE(intTop,intLeft,intWidth,intHeight,strHtml) 增加表格项
- ADD_PRINT_SHAPE(intShapeType,intTop,intLeft,intWidth,intHeight,intLineStyle, intLineWidth,intColor)画图形
- SET_PRINT_STYLE(strStyleName, varStyleValue) 设置对象风格
- PREVIEW 打印预览
- PRINT 直接打印
- PRINT_SETUP 打印维护
- PRINT_DESIGN 打印设计

2.14.2. 使用 LODOP 打印控件

在使用 LODOP 前，我们需要在页面中引入“LODOP/CheckActivX.js”文件，这个文件是对 LODOP 的检查并初始化对象的，这样我们在页面就可以直接使用全局对象进行打印等操作了。默认情况下，我们生成的视图界面里面，已经在包含了这个引用了（在 BundleConfig 里面）。

要实现 Web 界面内容的打印，主要是要构造待打印的 HTML 代码即可，一般情况下，

我们可以把界面的某部分作为内容输出给打印控件，必要的时候，设置一下 HTML 代码的样式，这样会使得界面和我们浏览器看到的结果一致。

```
@*LODOPWeb打印控件操作*@
<script type="text/javascript">
    var LODOP; //声明为全局变量

    function Preview() {//正本打印
        CreateLicenseData();
        LODOP.SET_SHOW_MODE("PREVIEW_IN_BROWSE", 1);
        LODOP.PREVIEW();
    };
    function PrintA() {
        CreateLicenseData();
        LODOP.PRINTA();
    };
    function Setup() {//正本打印维护
        CreateLicenseData();
        LODOP.PRINT_SETUP();
    };
    function Design() {//正本打印设计
        CreateLicenseData();
        LODOP.PRINT_DESIGN();
    };

    function CreateLicenseData() {
        LODOP = getLodop();

        LODOP.PRINT_INIT("通知公告");
        var strBodyStyle = "<link type='text/css' rel='stylesheet' href='/Content/Themes/Default/style.css' />";
        strBodyStyle += "<style><!--table { border:1;background-color: #CBCBCC } td {background-color:#FFFFFE;border: 1; }";
        strBodyStyle += " th { background-color:#F1F1E3;padding-left:5px;border:1}--></style>";
        var strFormHtml = strBodyStyle + "<body>" + document.getElementById("printContent").innerHTML + "</body>";
        LODOP.ADD_PRINT_HTM(20, 40, 610, 900, strFormHtml);
        LODOP.PREVIEW();
    }
```

打印预览操作

直接打印操作

构造打印内容，添加样式使打印效果和浏览器一致

一般的套打，包含了几部分操作：打印预览、打印维护、打印设计。

打印预览和打印维护是面向终端用户的，打印维护是指内容不能修改删除、但位置可以调整，给不同的打印机不同的尺寸打印提供调整位置的可能性。

打印设计是面向开发人员的，开始需要通过这个功能来设计好套打的界面，就是根据套打证件的背景图片，大致摆放好各个内容的位置。

在套打的时候，注意需要通过下面代码来设置显示背景图片（打印的时候，是不打印背景的）。

```
    LODOP.ADD_PRINT_SETUP_BKIMG("<img  src=Report/证件背景.jpg'  />");
    LODOP.SET_SHOW_MODE  ("BKIMG_IN_PREVIEW",1); //打印预览时是否包含背景图
```

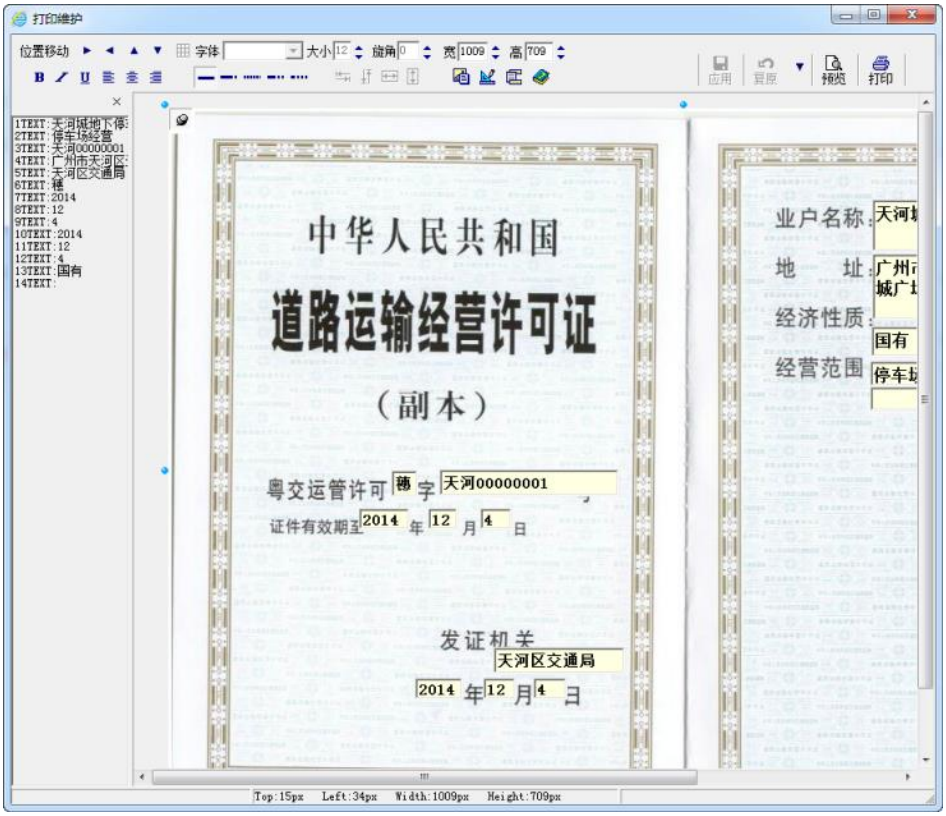
如下面的就是证件套打案例里面的实例代码。

```
function Preview() {//正本打印
    CreateLicenseData();
    LODOP.SET_PREVIEW_WINDOW(2, 0, 0, 800, 600, "");
    LODOP.SET_PRINT_PAGESIZE(2, 0, 0, "A3");
    LODOP.PREVIEW();
};
function Setup() {//正本打印维护
    CreateLicenseData();
    LODOP.PRINT_SETUP();
};
function Design() {//正本打印设计
    CreateLicenseData();
    LODOP.PRINT_DESIGN();
};

function CreateLicenseData() {
    LODOP = getLodop();
    LODOP.SET_PRINT_PAPER(0, 0, 1576, 1100, "道路运输经营许可证正本套打");
    LODOP.ADD_PRINT_TEXT(574, 514, 250, 52, "天河城地下停车场 "); //业户名称
    LODOP.SET_PRINT_STYLEA(1, "FontSize", 21);
    LODOP.SET_PRINT_STYLEA(1, "Bold", 1);
    LODOP.ADD_PRINT_TEXT(676, 514, 450, 52, "停车场经营"); //经营范围
    LODOP.SET_PRINT_STYLEA(2, "FontSize", 21);
    LODOP.SET_PRINT_STYLEA(2, "Bold", 1);
    LODOP.ADD_PRINT_TEXT(492, 1086, 203, 40, "天河00000001"); //许可证号
    LODOP.SET_PRINT_STYLEA(3, "FontSize", 16);
```

在背景图片上构建的打印内容（套打数据）

```
    LODOP.ADD_PRINT_TEXT(272,687,244,25, ""); //停车场类型
    LODOP.SET_PRINT_STYLEA(14, "FontSize", 12);
    LODOP.SET_PRINT_STYLEA(14, "Bold", 1);

    LODOP.ADD_PRINT_SETUP_BKIMG("<img src='Images/许可证副本.JPG' />");
    LODOP.SET_SHOW_MODE("BKIMG_IN_PREVIEW", 1);
}
```

上面的位置及文本内容的代码，是通过这个打印控件的设计模式进行调整和修改的，可以在其中增加内容，也可以调整具体的文本位置，从而实现套打内容的设计，最后把固定好的内容复制到页面上即可。



2.15. 富文本控件 CkEditor 和 Ckfinder 控件

Web 开发上有很多 HTML 的编辑控件，如 CKEditor、kindeditor 等等，很多都做的很好，本小节主要介绍在 MVC 界面里面，CKEditor 的配置和使用。

CKEditor 的前身是 FCKEditor，随着它的更新，上传图片的功能被分离出去了，现在如果需要实现上传图片，要么自己写代码或者采用其他上传控件（如 Uploadify)，还有一种方法是使用 CKFinder，这两者的合并使用，能给我们带来更多的方便。

在利用代码生成工具生成的 Web 框架的视图代码里面，已经包含了对这个控件的应用，以及提供了一个初始化的函数，可以在其中进行设置；或者参考行业动态、法律法规模块里面对 CKEditor 的使用代码。

下面介绍的 CkEditor 和 Ckfinder 控件的使用知识，主要是基于对这个控件进行详细的了解，以及重新处理新下载控件的设置参数等内容。

2.15.1. CKEditor 的使用

CKEditor 的下载地址是 http://ckeditor.com/download，里面可以根据需要进行样

式的定制，Web 框架主要介绍当前较新的版本 4.4.1 的继承使用。而 CKFinder 的下载地址是：http://ckfinder.com/download。

CKEditor 的使用比较简单，一般情况下根据官方的指引选择适当的样式下载就可以了，使用的时候，基本不需要怎么样修改配置文件。在 MVC 的视图页面里面，添加相关的引用文件就可以了。

```
<script src="~/Content/JQueryTools/ckeditor/ckeditor.js"></script>
<script src="~/Content/JQueryTools/ckeditor/adapters/jquery.js"></script>
```

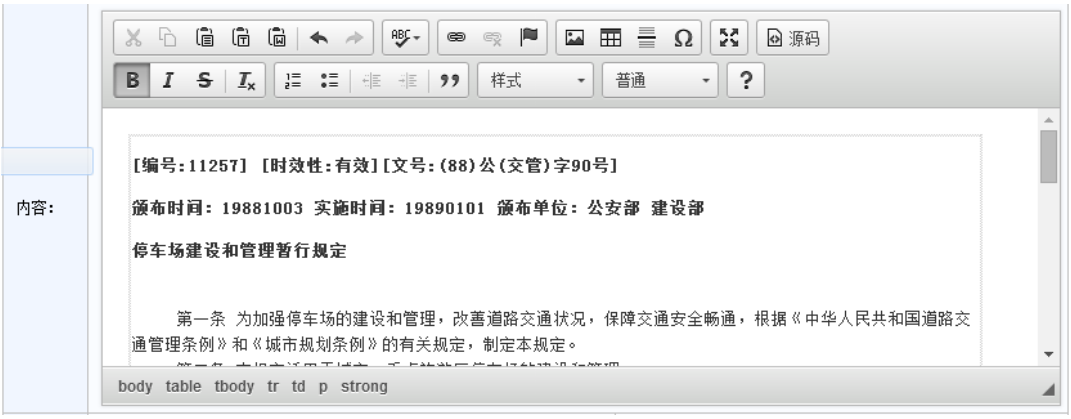
然后在界面添加需要编辑 HTML 内容的 textarea 控件，由于我的使用的是 EasyUI 的控件，因此设置了控件样式 class="easyui-validatebox"，也可以不管这里。

```
<tr>
    <th>
          <label for="Content">内容：</label>
     </th>
     <td>
         <textarea class="easyui-validatebox" id="Content" name="Content"
style="width:1024px"></textarea>
       </td>
 </tr>
```

我们一般使用的时候，需要一段 javascript 脚本进行初始化对应的控件，初始化代码如下所示。

```
<script>
    function initEditor() {
        $('#Content').ckeditor();//控件 1
        $('#Content1').ckeditor();//控件 2
    }
</script>
```

添加相应的脚本和控件初始化代码后，就可以在界面中呈现出需要的效果了。



而之后的控件使用，就和普通的使用没有两样了，如赋值语句如下所示。

```
$('#Content1').val(info.Content);//ckeditor 赋值
```

获取控件的值，也和普通给的一样

```
var content = $("#Content1").val();
```

2.15.2. CKFinder 的集成使用

虽然 CKFinder 已经独立出来，但是它也提供了完美的整合效果，我们把它下载后，在压缩包里面的 bin 目录里面找到相应的 ckFinder.dll，把它添加我们项目工程的引用里面去，我们才能正常使用文件上传功能。

然后修改 config.ascx 文件里面的一些设置，使得我们能够顺利使用。

第一步设置 CheckAuthentication 函数返回 True。

```
public override bool CheckAuthentication()
{
            return true;
}
```

第二步是设置 SetConfig 函数里面的 BaseURL，这个基础地址需要设置和我们项目的地址一致，否则上传文件有问题。

```
public override void SetConfig()
{
    // The base URL used to reach files in CKFinder through the browser.
    BaseUrl = "/Content/JQueryTools/ckfinder/userfiles/";
     ......................
```

第三步，整合 CKFinder 到 CKEditor，前面说了 CKEditor 默认是没有文件上传的功能操作的，需要添加 CKFinder 并进行配置才可以使用。这步骤需要在 CKEditor 里面的 config.js

文件里面修改下面的配置参数（红色部分就可以了）。

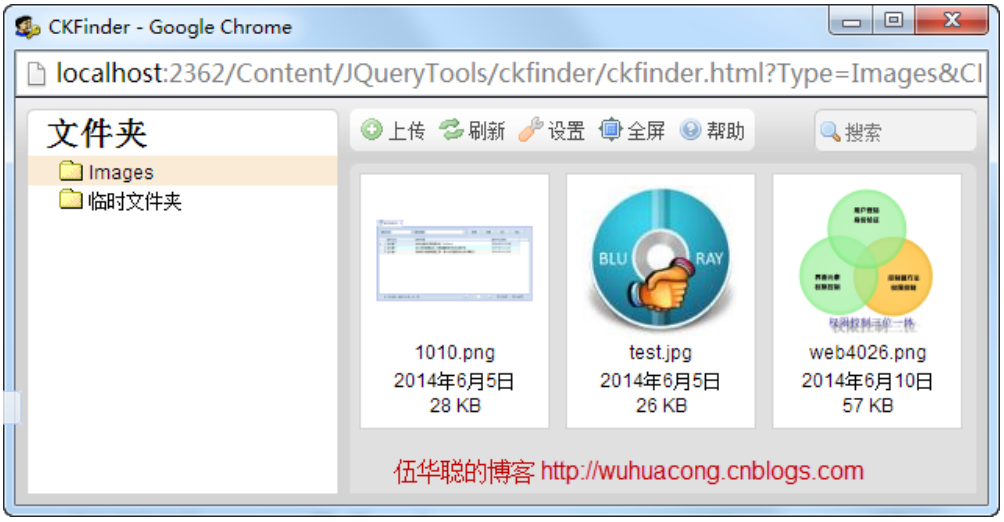
```
CKEDITOR.editorConfig = function( config ) {
    // Define changes to default configuration here.
    // For complete reference see:
    // http://docs.ckeditor.com/#!/api/CKEDITOR.config

.....................

    config.filebrowserBrowseUrl = '/Content/JQueryTools/ckfinder/ckfinder.html'; //上传文件时浏览服务文件夹
    config.filebrowserImageBrowseUrl = '/Content/JQueryTools/ckfinder/ckfinder.html?Type=Images'; //上传图片时浏览服务文件夹
    config.filebrowserFlashBrowseUrl = '/Content/JQueryTools/ckfinder/ckfinder.html?Type=Flash';  //上传 Flash 时浏览服务文件夹
    config.filebrowserUploadUrl = '/Content/JQueryTools/ckfinder/core/connector/aspx/connector.aspx?command=QuickUpload&type=Files'; //上传文件按钮(标签)
    config.filebrowserImageUploadUrl = '/Content/JQueryTools/ckfinder/core/connector/aspx/connector.aspx?command=QuickUpload&type=Images'; //上传图片按钮(标签)
    config.filebrowserFlashUploadUrl = '/Content/JQueryTools/ckfinder/connector/aspx/connector.aspx?command=QuickUpload&type=Flash'; //上传 Flash 按钮(标签)
};
```

2.15.3. 集成效果展示

通过以上代码进行整合，在插入图片的操作页面里面，会增加一个浏览服务器按钮，上传操作选项卡等功能，方便对文件的浏览和上传操作，具体效果如下所示。


图像属性
图像信息
链接
上传
URL
浏览服务器
目标窗口
<没有设置>
整合增加的按钮
伍华聪的博客 http://wuhuacong.cnblogs.com
确定
取消


图像属性
增加的上传文件操作
图像信息
链接
上传
上传到服务器
选择文件
web4026.png
上传到服务器
上传到服务器
伍华聪的博客 http://wuhuacong.cnblogs.com
确定
取消



以上就是我在我的 Web 框架里面整合的 HTML 编辑控件和 CKFinder 文件上传组件，这两个组合起来，能够非常方便构建图文并茂的文章内容。

2.15.4. MVC 的处理

这里需要注意的是由于 textarea 中有特殊字符，出于安全原因，默认情况 mvc 框架不允许提交的，应在相应方法上加上[ValidateInput(false)]属性。

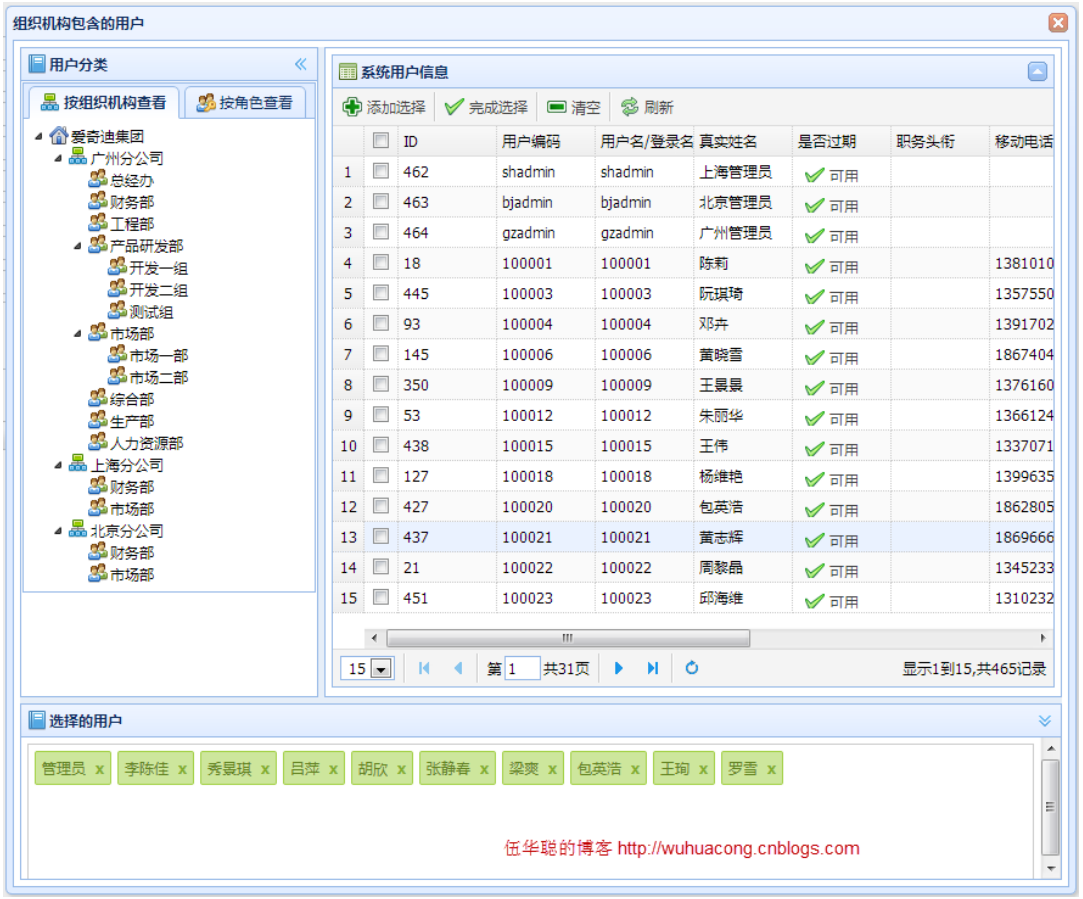
如我为了提交 HTML 内容，需要在控制器对象里面，重写了内容的写入和更新操作函数，如下所示。

```
[ValidateInput(false)]
public override ActionResult Insert(InformationInfo info)
{
    info.Editor = CurrentUser.Name;
    info.EditTime = DateTime.Now;

    return base.Insert(info);
}

[ValidateInput(false)]
public override ActionResult Update(string id, FormCollection formValues)
{
    return base.Update(id, formValues);
}
```

2.16. 标签控件 Tags-Input

我在利用 jQuery Tags Input 插件之前，一直想找一个合适的 Jquery 插件或者合适的做法，来实现我 Winform 框架里面权限系统的一个用户选择场景，就是能够记录用户的选择，并最终能够保存到数据库里面去。在 WInform 里面，我可以用自定义控件的方式，很好地实现了这个功能，但是在 Web 界面上，我尝试用 Jquery 试过了很多方法，没能实现这个效果，花了不少时间来寻找，终于找到这个不错的插件。

先来看看我的最终实现的 Web 界面效果，就是在权限管理系统里面，机构包含的用户编辑界面，或者是角色包含人员的编辑界面里面，提供一个地方来记录用户的选择，用户确认后，可以把记录的内容保存到数据库里面。



上图下面一个区域“**选择的用户**”里面就是我用到这个控件来展示用户选择的人员信息。其实这个 jQuery Tags Input 插件主要的用途，是用来记录用户输入的标签的，它可以在空白的地方接受输入的内容的，如下所示。

2.16.1. Tags Input 插件的使用

由于在 MVC 项目里面中集成使用，因此你需要整理好合适的路径，我的项目代码引用的路径如下所示。

```
<script src="~/Content/JQueryTools/Tags-Input/jquery.tagsinput.js"></script>
<link rel="stylesheet" type="text/css"
href="~/Content/JQueryTools/Tags-Input/jquery.tagsinput.css" />
```

不过由于在框架里面已经统一使用了 BundleConfig 来实现对控件 JS 和 CSS 文件的引入，实际上上面的代码不需要在视图 HTML 代码中出现。

简单的例子就是在需要的表单里创建一个包含 tags 列表的 input 输入框，你可以在 value 里设置默认或目前有的 tags，并用逗号隔开。

```
<input name="tags" id="tags" value="foo,bar,baz" />
```

这个插件可以屏蔽界面上的 Tag 标签输入，从而让脚本根据需要写入不同的标签。可以使用 addTag() and removeTag()函数增加和删除掉标签，代码如下：

```
$('#tags').addTag('foo');
$('#tags').removeTag('bar');
```

还可以用 imporTags()方法导进一组 tag 列表，需要注意的是这样会将 value 里设置 tag 替换掉。

```
$('#tags').importTags('foo,bar,baz');
```

如果传递参数为空，那么相当于清空列表了。

```
$('#tags').importTags('');
```

使用 tagExist()可以判断一个标签是否存在：

```
if ($('#tags').tagExist('foo')) { ... }
```

这个插件还可以接受自动提示的插入操作，如下所示。

```
$('#tags').tagsInput({
  autocomplete_url:'http://myserver.com/api/autocomplete'
});
```

如果想要在增加或移除掉标签的时候增加额外的功能或触发其它动作，你可以通过 onAddTag 和 onRemoveTag 这两个参数里指定回调函。这两个函数都返回了一个标签值作为参数.

```
$('#tags').tagsInput({
      width:'auto',
      onAddTag:function(tag){
          console.log('增加了'+tag)
    },
     onRemoveTag:function(tag){
        console.log('删除了'+tag)
     }});
```

前面讲了，可以屏蔽界面的 Tag 标签输入，而通过脚本插入标签，或者你想提供其它交互方式增加标签，可以增加一个值为 false 的 interactive 参数，这样就禁止了增加标签，而其它的功能和呈现都跟原来一样。

```
$('#tags').tagsInput({
    width:'auto',
    onRemoveTag:function(tag){
        console.log('移除标签：'+'"'+tag+'"')
    },
    interactive:false
});
```

2.16.2. 在项目中使用 Tags Input 插件

在项目需要使用的地方，添加一个 id=tags 的标签，如下所示。

```
<!--------------编辑组织机构包含用户的弹出层----------------->
<div id="DivEditUser" class="easyui-dialog" style="width:950px;height:620px;padding:5px 5px"
    <div id="cc" class="easyui-layout" fit="true" style="width:600px;height:350px;">
        <div data-options="region:'south',split:true" style="height:80px;">
            <input name="tags" id="tags" value="" />
        </div>
        <div data-options="region:'west',split:true" title="" style="width:320px;">
            <div id="ttUser" class="easyui-tabs" style="width:100%;height:auto;padding:5px;"
                <div title="按组织机构查看" data-options="iconCls:'icon-organ'" style="width:
                    <ul id="editTreeDept"></ul>
```

我们先初始化列表和 Tags 标签对象，并增加一个添加用户的封装和移除用户的封装操作，代码如下所示。

```
<script type="text/javascript">
    $(function () {
        $('#tags').tagsInput({
            width: 'auto',
            height: '100px',
            onRemoveTag: function (tag) {
                var i = addNameList.indexOf(tag);
                var id = addUserList[i];
                removeUser(id, tag);
            },
            interactive: false
        });
    });

    var addUserList = new Array();
    var addNameList = new Array();
    function addUser(id, name) {
        if ($.inArray(id, addUserList) == -1) {
            addUserList.push(id);
            addNameList.push(name);
            $('#tags').addTag(name);
        }
    }
    function removeUser(id, name) {
        if ($.inArray(id, addUserList) != -1) {
            addUserList.pop(id);
            addNameList.pop(name);
            $('#tags').removeTag(name);
        }
    }
</script>
```

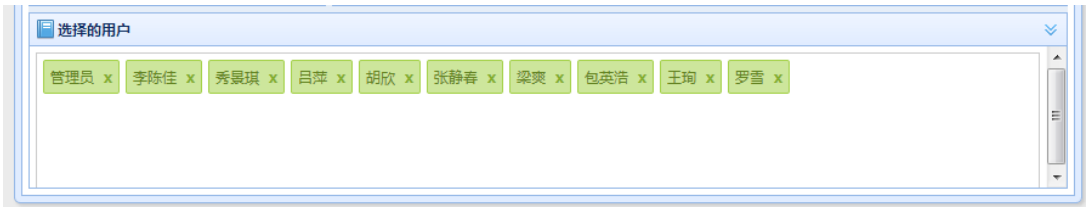
清除用户选择的 Tag 操作，代码也很简单了，前面介绍过了，熟练应用就是了。

```
//清空用户选择记录
function cleareChoise() {
    $('#tags').importTags('');
    addUserList = new Array();
    addNameList = new Array();
}
```

对于最重要的保存操作，就是把存储用户 ID 的列表，把他们传递给对应的 Ajax 调用就搞定了。

```
//完成选择并返回
function completeChoise() {
    var ouid = $('#txtID').val();
    if (ouid != "") {
        var url = '/OU/EditOuUsers?r=' + Math.random();
        saveAction(url, ouid, addUserList);
    }

    $("#DivEditUser").dialog('close');
    reloadRelation();//重新刷新
}

//保存机构用户操作
function saveAction(url, id, newList) {
    $.ajax({
        type: 'POST',
        url: url,
        async: false,
        data: { ouid: id, newList: newList.join(',') },
        success: function (result) {
            $.messager.alert("提示", "操作成功！ ");
            $('#DivEditUser').dialog('close');
            reloadRelation();
        },
        error: function (xhr, status, error) {
            $.messager.alert("提示", "操作失败"); //xhr.responseText
        }
    });
}
```

最后，我们就可以顺利看到真正的结果了。



2.17. RDLC 报表生成和预览

RDLC 是一个不错的报表，有着比较不错的设计模式和展现效果，在我的 Winform 开发里面，使用 RDLC 也是一个比较方便操作，如可以参考文章《DevExpress 的 XtraReport 和微软 RDLC 报表的使用和对比》或者《会员管理系统的设计和开发（2）-- RDLC 报表的设计及动态加载》进行了解。但是基于 MVC 方式，一样可以构建和展现基于 RDLC 的报表。

2.17.1. 控制器后台的操作支持

绑定 RDLC 报表，并赋值对应的数据源操作如下所示。

```
0 个引用
public ActionResult RDLCReport()
{
    return View("RDLCReport");
}

/// <summary>
/// 基于RDLC的报表数据操作
/// </summary>
/// <param name="format">图片格式</param>
/// <returns></returns>
0 个引用
public ActionResult UserRdlcReport(string format)
{
    LocalReport localReport = new LocalReport();
    localReport.ReportPath = Server.MapPath("~/Report/WHC.UserReport.rdlc");
    var dt = baseBLL.GetAll();

    ReportDataSource reportDataSource = new ReportDataSource("DataSet1", dt);
    localReport.DataSources.Add(reportDataSource);

    if(string.IsNullOrEmpty(format))
    {
        format = "Image";
    }

    string reportType = format;
    string deviceType = (format.ToLower() == "image") ? "jpeg" : format;
    string mimeType;
    string encoding;
    string fileNameExtension;
```

呈现的操作代码如下所示，默认我们以图片进行展现。

```
        Warning[] warnings;
        string[] streams;
        byte[] renderedBytes;

        renderedBytes = localReport.Render(
            reportType,
            deviceInfo,
            out mimeType,
            out encoding,
            out fileNameExtension,
            out streams,
            out warnings);

        return File(renderedBytes, (format.ToLower() == "image") ? "image/jpeg" : mimeType);
```

我们了解 RDLC 的话，应该知道，一般 RDLC 报表，它都是通过一个 DeviceInfo 的信息进行展现的，如下所示是一个标准的 DeviceInfo 对象。

```
    string deviceInfo =
      "<DeviceInfo>" +
      "  <OutputFormat>" + deviceType + "</OutputFormat>" +
      "  <PageWidth>8.5in</PageWidth>" +
      "  <PageHeight>11in</PageHeight>" +
      "  <MarginTop>0.5in</MarginTop>" +
      "  <MarginLeft>1in</MarginLeft>" +
      "  <MarginRight>1in</MarginRight>" +
      "  <MarginBottom>0.5in</MarginBottom>";
```

但是这样的内容，如果展现图片的话，就只会展示一页的内容，一般是 800 的高度这样子，但是我的报表里面可能有很多记录，如何能够让它全部展现出来呢?

方法是有的，不过不是很完美，就是需要计算大概的尺寸，然后修改 PageHeight 的数值，让它动态的删除最大的记录，达到全部内容都可以输出看到。

为了达到这个目的，我对图片格式输出的报表，对它的高度进行了一个简单的计算，然后换成它的标准高度，这样代码如下所示。

```
if(format.ToLower() == "image")
{
  double inchValue = (dt.Count / 37.0) * 11;
  deviceInfo += string.Format("  <PageHeight>{0}in</PageHeight>", inchValue);
 }
 else
 {
    deviceInfo += "  <PageHeight>11in</PageHeight>";
 }
```

2.17.2. 界面层的代码

```
<script type="text/javascript">
    $(function () {
        $("#myReport").attr("src", "/User/UserRdlcReport?format=Image");
        $("#autoUpdate").show();
    });
</script>
```

```
<body>
    <div style="padding:10px; border:1px solid black">
        <div>
            <a href="@Url.Action("UserRdlcReport", new { format = "Image" })" class="easyui-linkbutton" data-options="iconCls:'icon-view'">图片输出</a>
            <a href="@Url.Action("UserRdlcReport", new { format = "PDF" })" class="easyui-linkbutton" data-options="iconCls:'icon-view'"> PDF输出</a>
            <a href="@Url.Action("UserRdlcReport", new { format = "Excel" })" class="easyui-linkbutton" data-options="iconCls:'icon-view'">Excel输出</a>
            <a href="@Url.Action("UserRdlcReport", new { format = "Word" })" class="easyui-linkbutton" data-options="iconCls:'icon-view'">Word输出</a>
        </div>
    </div>
    <div id="autoUpdate" style="display: none; overflow-y: auto" class="SlideContainer">
        <table width="100%" height="100%">
            <tr>
                <td>
                    <table>
                        <tr>
                            <td></td>
                            <td></td>
                        </tr>
                    </table>
                </td>
            </tr>
            <tr><td><iframe id="myReport" width="100%" height="800"></iframe></td></tr>
        </table>
    </div>
</body>
</html>
```

几种报表输出方式

报表展现，放到Iframe里面