技术管理系列之

《Jurassic.Frameworks 3框架开发文档》

版本号：Version 3.0

**侏罗纪软件股份有限公司**

**二零一七年 五月**

版本修订记录

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 重要说明 | 人员 | 时间 | 状态 |
| V1.0 | 初始版本，A | 王家新 | 2015.5.20 | 初稿 |
| V1.1 | A | 王家新 | 2015.10.22 | A |
| V1.2 | A | 王家新 | 2015.11.13 | A |
| V1.3 | A | 王家新 | 2015.12.22 | A |
| V1.4 | A | 王家新 | 2016.1.25 | A |
| V2.2 | A | 王家新 | 2016.9.12 | A |
| V3.0 | A | 王家新 | 2017.4.21 | 修订中 |

A - 添加 M – 修改 D – 删除

状态： 草稿，初稿，修订中，审批中，已审批，已发布

目录

[1. WebFrame框架 5](#_Toc462584980)

[1.1. 概述 5](#_Toc462584981)

[1.1.1. 开发目的 5](#_Toc462584982)

[1.2. 框架简介 6](#_Toc462584983)

[1.2.1. 框架结构图 6](#_Toc462584984)

[1.3. 开发及运行环境 6](#_Toc462584985)

[1.4. 项目建立步骤 6](#_Toc462584986)

[1.4.1. 解决方案建立步骤 6](#_Toc462584987)

[1.4.2. 从WebFrame 1.0到WebFrame1.1的升级注意事项： 7](#_Toc462584988)

[1.4.3. 从WebFrame 1.1到WebFrame1.2的升级注意事项： 10](#_Toc462584989)

[1.4.4. 从WebFrame 1.2到WebFrame1.3的升级注意事项： 11](#_Toc462584990)

[1.4.5. 升级到WebFrame2.0： 11](#_Toc462584991)

[1.5. 数据库安装： 12](#_Toc462584992)

[1.6. 公共模块的配置和运行 12](#_Toc462584993)

[1.7. 服务端Controller代码编写方法 13](#_Toc462584994)

[1.8. 前端JS代码编写方法 16](#_Toc462584995)

[1.8.1. $.regButton(options)扩展 16](#_Toc462584996)

[1.8.2. $.newGET(url, data, callback，dataType) 扩展—无缓存的GET请求 19](#_Toc462584997)

[1.8.3. $.newPOST(url, data, callback,dataType) 扩展—POST请求 20](#_Toc462584998)

[1.8.4. $.getSync (url, data, dataType) 扩展 --- 同步GET请求直接返回结果 20](#_Toc462584999)

[1.8.5. $.postSync (url, data, dataType) 扩展 --- 同步POST请求直接返回结果 20](#_Toc462585000)

[1.8.6. $.lang(key) js端的多语言API 21](#_Toc462585001)

[1.8.7. 出错处理 21](#_Toc462585002)

[1.9. View视图的编写方法 21](#_Toc462585003)

[1.10. Global.asax的编写方法 23](#_Toc462585004)

[1.11. 搜索框的编程 23](#_Toc462585005)

[1.11.1. 如何控制搜索框是否显示？ 24](#_Toc462585006)

[1.12. 框架可选组件 Jurassic.Web\*\*\*\* 25](#_Toc462585007)

[1.12.1. Jurasssic.WebUpload 文件上传组件 25](#_Toc462585008)

[1.12.2. Jurasssic.WebReport Web报表组件 27](#_Toc462585009)

[1.12.3. Jurasssic.WebHtmlEditor 在线HTML编辑器组件 29](#_Toc462585010)

[1.12.4. Jurasssic.WebRepeater 分页数据展示组件 30](#_Toc462585011)

[1.12.5. Jurasssic.WebQuery 高级查询与数据采集组件 32](#_Toc462585012)

[1.13. 关于Model, 推荐的做法 46](#_Toc462585013)

[1.14. 常见问题及解答 (FAQ) 48](#_Toc462585014)

[1.15. 附录 49](#_Toc462585015)

[1.15.1. 在编写本框架的过程中，参考了以下网址： 49](#_Toc462585016)

[1.15.2. MVC的学习网站 49](#_Toc462585017)

[1.15.3. Ninject 官网及开源代码 50](#_Toc462585018)

[1.15.4. Mini-UI官网 50](#_Toc462585019)

[2. WinFrame框架 50](#_Toc462585020)

[2.1. 概述 50](#_Toc462585021)

[2.1.1. 开发目的 50](#_Toc462585022)

[2.2. 框架简介 51](#_Toc462585023)

[2.2.1. 框架结构图 51](#_Toc462585024)

[2.3. 开发及运行环境 52](#_Toc462585025)

[2.4. 各版本升级注意事项 52](#_Toc462585026)

[2.4.1. 从1.1或1.0升级到1.3 52](#_Toc462585027)

[2.4.2. 升级到WinFrame 2.0: 54](#_Toc462585028)

[2.5. 项目建立步骤 54](#_Toc462585029)

[2.5.1. 解决方案建立步骤 54](#_Toc462585030)

[2.5.2. 数据库安装： 54](#_Toc462585031)

[2.6. 公共模块的配置和运行 54](#_Toc462585032)

[2.7. 客户端模块编写方法 55](#_Toc462585033)

[2.7.1. 一般性窗体 55](#_Toc462585034)

[2.7.2. 可停靠面板 57](#_Toc462585035)

[2.7.3. 开始菜单中的自定义控件 58](#_Toc462585036)

[2.8. 本框架的配置文件使用。 61](#_Toc462585037)

[2.9. WCF服务的编写方法 63](#_Toc462585038)

[2.10. 如何实现客户端无缝调用WCF服务(或本地服务) 66](#_Toc462585039)

[2.11. 自动更新组件AutoUpdateModule使用方法 69](#_Toc462585040)

[2.11.1. 实现原理 69](#_Toc462585041)

[2.11.2. 服务端编译、配置和安装 70](#_Toc462585042)

[2.11.3. 客户端调用和配置 70](#_Toc462585043)

[2.11.4. 使用和维护 71](#_Toc462585044)

[2.11.5. 扩展接口 71](#_Toc462585045)

[2.12. 常见问题及解答 (FAQ) 71](#_Toc462585046)

[2.13. 附录 72](#_Toc462585047)

[2.13.1. 在编写本框架的过程中，参考了以下网址： 72](#_Toc462585048)

[2.13.2. SCSF的学习网站 72](#_Toc462585049)

[3. Jurassic.Frameworks的VS模板插件 73](#_Toc462585050)

[3.1. 概述 73](#_Toc462585051)

[3.1.1. 开发目的 73](#_Toc462585052)

[3.2. 运行环境 73](#_Toc462585053)

[3.3. 安装及使用 74](#_Toc462585054)

# WebFrame框架

## 概述

Jurassic.WebFrame框架（以下简称框架）是基于ASP.net MVC4和mini-UI的前端js组件构建，使用Ninject作为IOC容器，用于一般性管理系统的轻量级的程序框架。

### 开发目的

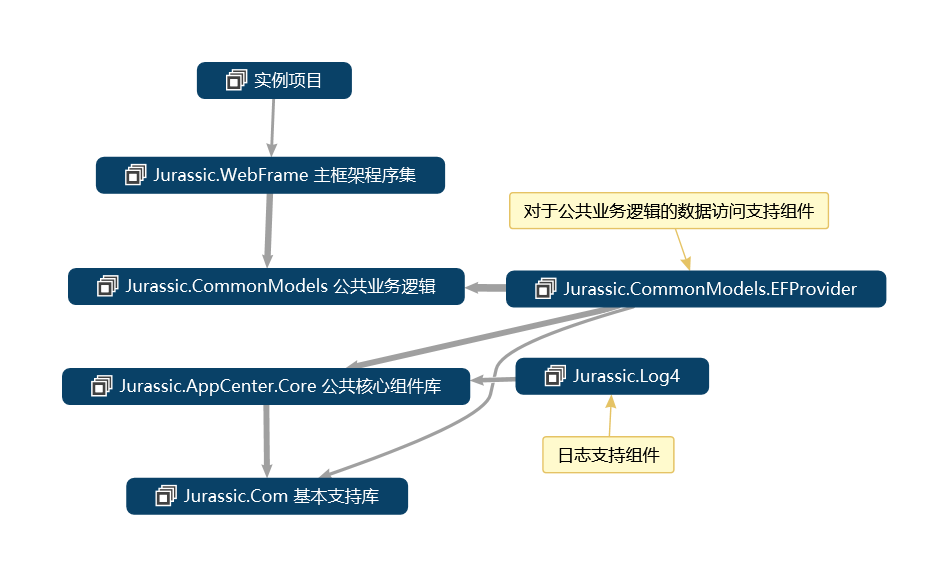
开发此框架的目的是为对未来公司各个基于Web平台的项目统一技术框架，并在此框架基础上建立统一的开发规范，减少各项目之间各自技术选型和框架不统一的情况。

主要以下几点:

* 统一Web平台的前后台框架；
* 在此框架基础上建立统一开发规范；
* 减小各项目在基础框架和组件实现上的差别；
* 新进员工快速培训，缩短适应期；
* 框架层和应用层分离，各自独立发展；
* 为各业务组件的抽提和复用提供公共的平台和技术支持；
* 框架和各组件本身作为公司技术资产，对提升生产力发挥持久的作用。

## 框架简介

### 框架结构图



框架主体

公用组件

具体项目

## 开发及运行环境

开发平台是vs2013, 无需安装其他第三方组件，依赖项均已nuget上下载并且已经集成到源代码管理中。

在部署到生产环境时，只需发布一个本地版本并拷贝到服务器相应目录即可。Web.config中的连接字符串根据服务器实际情况自行修改。服务器要求装有.net framework 4.0.

## 项目建立步骤

### 解决方案建立步骤

在SVN上有一模板文件夹在：

<http://61.183.159.222:8443/svn/项目管理规范/12-项目模板/WebTemplate>

1. 建立和“项目管理规范”同级的项目文件夹 比如 “XX系统”
2. 将WebTemplate文件夹整体COPY到 “XX系统/系统开发”文件夹下面，再重新命名相关文件。
3. 将WebTemplate文件夹重命名为实际解决方案名称。
4. 打开解决方案文件夹，将Jurassic.WebTempalte.sln重命名为 解决方案名称.sln
5. 重新命名解决方案中的项目文件夹Jurassic.WebTemplate为实际项目名称。
6. 打开Global.asax，调整其中的名称和命名空间。
7. 直接可编译运行。
8. 用户名密码是pmis 888888
9. 也可浏览数据库的UserProfie表，其中的用户名。如果需要重设密码，则可以先在数据库中对应的用户表UserProfile中填上对应的邮箱，再在系统登录界面输入用户名点击“忘记密码”即可以通过邮箱来找回密码。

另外，在2015-10-22发布 的WebFrame1.1版本中，推出了一个WebTemplate\_Lite的轻量级模板。该模板的特点是只引用了编译好的框架程序集，而不是源码项目。这样可以简化实际项目结构，加快项目的编译和运行速度。依据WebTemplate\_Lite建立实际解决方案的步骤和前述步骤基本相同。

所有框架的DLL，现在统一编译到了AppCenter/packages/Jurassic.Frameworks中，分为Debug和Release版本，并且会随框架的源码版本同步更新。 开发人员可以根据实际情况引用其中编译好的程序集。或按照以往的模式，继续引用源码版本。

### 从WebFrame 1.0到WebFrame1.1的升级注意事项：

WebFrame1.0到WebFrame1.1是一个二进制级兼容的项目。它们是连续性的升级。

它的主要改进在于整体打包机制:

在基础框架WebFrame.dll中打包了所有内置的管理视图和通用的登录，框架主页、布局页视图和相应的脚本资源。包括js文件、图片资源和css样式表。因此今后基于WebFrame1.1的升级不再需要拷贝除dll以外的其他物理文件到项目中。

具体的工作原理如下：

1. 在系统启动时，根据App\_Data下面的ResViews.txt文件记录的文件名（如果有的话），删除其中指定的所有文件。
2. 根据WebFrame.dll中编译打包的资源（视图、脚本文件等），将所有打包资源写回到指定文件夹，并记录在App\_Data/ResViews.txt中。
3. 项目进入正常启动流程。由于所有打包资源已经写回它该在的位置，所以系统会正常运行。
4. 下次系统启动，重复步骤1。

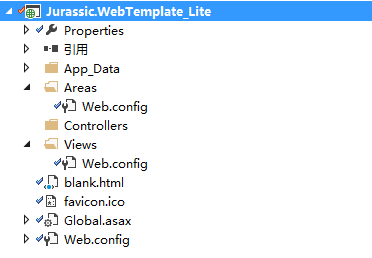
对性能的影响：

目前WebFrame中打包了335个资源文件。根据对初始化过程中的监控，花在上述步骤上的总时间约为1.2s。应该是可以接受的。特别是这个过程只是在应用程序初始化阶段才会进行一次，因此对系统整体性能影响还是非常小的。今后，会进一步调整算法，引入版本控制，把这个时间优化到毫秒级。

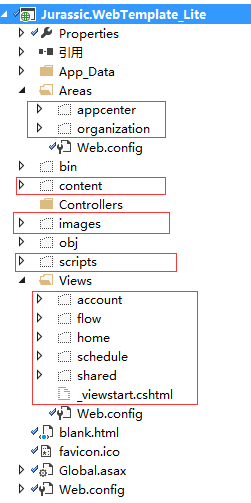
此外，WebFrame1.1还对以下细节进行了改进：

1. 调整了部分页面样式：登录页、主页的提示信息、二级页面样式，使其更美观。
2. 对mini-ui进行了升级，使其不再弹出版权过期的警告框。
3. 对通用搜索框进行了定义，现在可以很方便支持页面级数据的搜索。
4. 对内部角色管理功能进行了改进，现在可以显示角色下所有用户列表。

从1.0升级到1.1, 因为引入了新的资源打包机制，所以需要特别注意以前一些在1.0版本中以实体文件形式存在的资源，在1.1版本中不再作为项目的一部分，参考下图：



对比版本1.0, 让我们看看具体删除了哪些文件或文件夹：



在升级到1.1后，项目文件夹结构已经变得非常简单， 建议按照上图的文件夹结构，删除已有项目中上图文件夹结构中不存在的所有文件和文件夹。但有一种情况例外：就是项目中已经定制了某个页面（比如登录页），那就不要删除。否则系统会用默认打包的登录页替换掉它。如果不删除就不会自动替换。

总之， 一个总的原则就是：如果视图（或其他文件）是自己定制的，就保留，如果是继承系统默认的，就删除。

无论是根据WebTemplate项目或WebTemplate\_Lite项目创建的项目都照此办理。

**如果不删除这些文件，会有什么影响?**

可能页面样式会错位。

**这样做，有什么好处？**

这样做，就保证了今后框架的升级，只涉及到一个DLL的升级就是WebFrame.DLL。而不需要去拷贝其他关联的物理文件。版本的一致性得到充分保证。

**如果我将框架生成的视图改写，是否下次启动仍然会被覆盖？**

确实会。

解决这个问题的办法是将ResViews.Txt中有该文件地址的那一行删除掉。

比如定制自己的home/index.cshtml.

需要把app\_data/resview.txt中关于home/index.cshtml那一行删掉，否则下次定制的页就被打包的版本给覆盖了。

### 从WebFrame 1.1到WebFrame1.2的升级注意事项：

WebFrame1.2主要的改进在于以下几点：

1. 增加了多标签界面风格，可以设置打开页面以多选项卡显示。
2. 增加了系统配置，一些界面中显示的静态内容，诸如系统名称、公司名称等可以通过配置来实现。
3. 增加了资源管理（即文件上传组件）作为可选组件。
4. 修复了一些小的BUG。

从1.1升级到1.2, 根据实际项目需要，手工调整的有以下3点：

当然，即使什么也不做，也不会影响现有项目的运行。

1. **关于系统配置**(上面提到的第2点)。如果需要用到系统配置这些常量，那么就需要从框架继承过来的主页(Home/Index.cshtml)和登录页(Account/Login.cshtml)。这两个视图页包含了配置的编码。下面一行的代码就是一例：

ViewBag.Title = Html.Str("System\_Name", "北京侏罗纪软件股份有限公司应用管理系统");

其中Html.Str()是框架定义的扩展，其作用是返回指定Key对应的当前语言环境的字符串。第一个参数是Key, 第二个参数是找不到时的默认值。

如果上述两页面已经被实际系统所定制，并被排除在框架自动更新的视图之外，如果没有这方面的配置要求，可以先不管，如果需要，则需参照标准的模板中的视图代码实现自己相应视图的可配置的版本。  
如果上述两页面没有被实际系统定制，并随着框架的更新而自动更新，则需要先运行系统，然后在系统管理-语言管理界面,修改以下Key对应的值：



再点保存按钮即可。

1. **关于资源管理组件**，由于涉及到关系型数据库的数据存储，需要调整框架自带的数据库表结构如下：如果暂时用不到文件上传也可以跳过。

运行脚本：[1.0升1.1数据库升级脚本\_SqlServer.sql](file:///D:\Wang\Documents\svn\公用组件库\dotNet\AppCenter\Docs\1.0升1.1数据库升级脚本_SqlServer.sql)

请注意以上脚本文件的获取需要在svn中Update一下公用组件库目录。

3）**关于多选项卡,**如果需要启用此功能，可以在global.asax.cs中加上：

////如果需要系统默认以多标签形式显示页，请恢复以下代码

ninjectKernel.Bind<UserConfig>().ToSelf()

.WithPropertyValue("ShowTab", true);

### 从WebFrame 1.2到WebFrame1.3的升级注意事项：

**WebFrame1.3主要的改进：**

1. 增加了3种皮肤样式，分别是blacknewss, blue和win7。可以在用户配置页中动态预览皮肤样式并进行选择。
2. 增加了用户配置页，用户可以在配置页中查看登录历史，设定界面皮肤和决定是否多标签显示。点击首页登录头像即可进入用户配置页。
3. 资源管理（即文件上传组件）经过实际项目应用检验，进行了相关改进，提升了浏览器兼容性。

**从1.2到1.3版本升级方法：**

由于从1.2版本开始，所有项目资源都已整体打包，所以本次升级除了获取最新源码或编译后的DLL，并进行简单测试以外，不需要任何其他操作。

### 升级到WebFrame2.0：

**WebFrame 2.0 新增的内容：**

1）统一了框架的英文名称为Jurassic.Frameworks，并将以往两个说明文档合并为一个文档《 [Jurassic.Frameworks框架开发文档.docx](http://mail.jurassic.com.cn/webmail7.5/webmail.php?r=inboxSend/readMailBody/sourceFolder/INBOX.Sent/mailId/148/fileName//sid/53548440dc3f02812e2469477398622d)》。它的svn地址是：[http://221.232.134.250:8443/svn/公用组件库/dotNet/AppCenter/Docs](http://221.232.134.250:8443/svn/)。未来我们计划将这些技术文档搬到线上，类似于vs在线帮助文档的风格。  
2）研发了框架的vs插件jurassic.FrameworksVSIX.vsix，简化了根据框架创建项目以及在开发中创建对象的工作，分别提供了WinTemplate和WebTemplate两种模板，可用于windows forms窗体应用程序和web应用程序。欢迎大家试用，提出宝贵意见。vs插件的使用方法参见上述开发文档。  
  vs插件的地址在svn目录：<http://221.232.134.250:8443/svn/公用组件库/dotNet/AppCenter/packages/Jurassic.Frameworks>  
3）WebFrame增加了以下可选组件：  
      Jurassic.WebReport  -- 用于在MVC环境中无缝使用微软的RDLC报表  
      Jurassic.WebHtmlEditor -- 集成了KindEditor的在线HTML编辑器，KindEditor是一个成熟的开源编辑器。公司邮件系统就是使用的它。  
      Jurassic.WebRepeater -- 类似于asp.net Repeater控件的数据呈现方式，可以在MVC环境中分页展现数据，并使用了bootstrap布局组件。  
      关于以上这些组件的详细使用文档，可以参见文档中的说明。  
4）根据有些同事的反应WebFrame框架升级时文件自动覆盖有时会覆盖掉自己修改过的文件。这当然和具体操作有关。现在优化了逻辑。只要是比较文件日期，在上次运行过后编辑修改了的文件都不会被覆盖。举例来说，当你运行过一次使用WebFrame的系统以后，再去编辑比如Views/accout/login.cshtml这个文件并保存。那么下一次的覆盖操作将会自动跳过此文件。而不需要再去ResViews.txt文件中手动删除这个文件名。  
当然，如果对自己的定制修改不满意，你仍然可以删除掉这个文件。框架在运行时仍然会帮你恢复到原有的版本。  
5) 增强了框架统一内容管理的业务逻辑。有兴趣的同事可以看一下WebTemplate/NoteBookController的实现。未来这个内容管理组件还会继续加强。  
6）在WinForm框架的MDI多窗体界面中，新增了拖动窗体到外部形成独立窗体等一系列小功能，并修复了若干小BUG。

**升级方法**

从上一版本升级不需要任何额外操作，只需重新更新框架代码并重新编译即可。

### 升级到WebFrame 3.0：

WebFrame3.0是在原有框架编程模式和编程接口基本不变的情况下的底层第三方组件的升级。

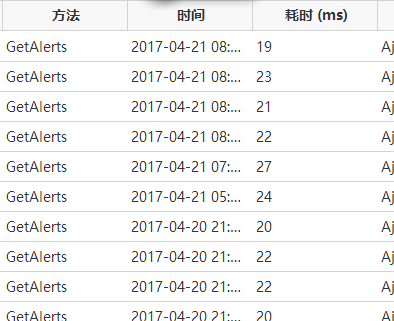
WebFrame 3.0 进行了以下升级：

1. 将.net framework从4.0升级到了4.5.2;
2. 开发环境从vs2013升级到vs2015;
3. MVC版本从MVC4升级到MVC5.2.3;
4. 对模板的初始化接口进行了统一规定;
5. 新增可以自动初始化的多语言类；
6. MonogoDB的支持模块根据知识管理部的成果进行了改写；
7. 对框架自带栏目-文章-元数据表进行了优化，大大提升了查询速度；间接提升消息组件、日程表组件、上传组件的速度。

优化前的消息提醒：



优化后：



1. 对开发方式进行了改进，现在每一个业务部门都有一个单独的分支，可以用来提交修改，并和主分支同步。

框架组件版本对照表

|  |  |  |
| --- | --- | --- |
| 组件名称 | 升级前 | 升级后 |
| ASP.net MVC | 4.0 | 5.2.3 |
| WebAPI | 4.0 | 5.2.3 |
| Newtonsoft.json | 4.5 | 10.0.2 |
| Ninject | 3.2 (.net4.0) | 3.2(.net 4.5 -full) |
| Owin | 无 | 1.0 |
| EntityFramework | 6.1.2 | 6.1.3 |
| AutoMapper | 4.1.1 | 4.2.1 (注) |
| log4net | 2.0.3 | 2.0.7 |

注：由于AutoMapper最新版本不支持Mapper.CreateMap()动态绑定，所以这里是使用最后一个支持这个方法的版本。

从原有项目升级的方法：

1. 下载本部门的框架的svn分支，各部门的svn分支见下表：

|  |  |
| --- | --- |
| 部门 | Svn分支地址 |
| 主分支 | http://61.183.159.222:8443/svn/公用组件库/dotNet/公司统一框架/v3.0/Trunk |
| 知识管理业务部 | http://61.183.159.222:8443/svn/公用组件库/dotNet/公司统一框架/v3.0/Branch/知识管理 |
| 资产评价业务部 | http://61.183.159.222:8443/svn/公用组件库/dotNet/公司统一框架/v3.0/Branch/资产评价 |
| 海外业务部 | http://61.183.159.222:8443/svn/公用组件库/dotNet/公司统一框架/v3.0/Branch/海外 |
| 项目管理部 | http://61.183.159.222:8443/svn/公用组件库/dotNet/公司统一框架/v3.0/Branch/项目管理部 |

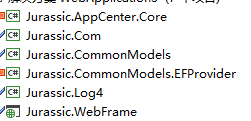
关于如何使用Svn的分支和合并功能，请留意周炜的培训和相关文档;

1. 用vs2015打开新下载的svn分支下的解决方案，确保能正常编译运行;
2. 打开项目解决方案；将解决方案中的启动项目调整为使用.netframework4.5.2.
3. 将主项目使用nuget升级MVC框架和WebAPI到5.2.3; 其他第三方组件根据情况升级， 一般都是升级最新版本;
4. 调整原有项目引用；如果原来项目是DLL引用，将最新版本框架编译后的DLL覆盖原有DLL; 如果原来项目是源码项目引用，将原引用的源码项目调整到同名的新框架下的项目；
5. 调整框架数据库的结构，调整的脚本请参见统一框架svn下的Docs/UpdateToV3.sql。暂只提供sqlserver版本。
6. 运行测试;
7. 升级完成；如果有问题，请联系项目管理部。

考虑到今后统一框架主要是以3.0及后续版本为主， 对于现有正在开发中的Web项目，建议升级到最新版本。

如果是全新开始一个项目，则按照以下步骤：

1. 打开VS2015,新建->项目->Web->ASP.net应用程序，点击确定;
2. 在出现的对话框中,选择Empty模板，在下方勾选MVC,点击确定;
3. 在nuget程序包中，添加最新版本MicroSoft.Owin.Host.SystemWeb程序集。
4. 在新的解决方案中，添加框架的源码项目，浏览到本部门的统一框架svn文件夹中的Code文件夹，添加如下项目：



1. 右击新建的MVC项目，添加以上框架的引用。
2. 将Jurasssic.WebFrame中的3个web.config（根目录下的web.config和Views目录下的web.config,Areas目录下的web.config,如果areas文件夹不存在则新建）复制到新项目，覆盖原有同名文件。
3. 将WebFrame中的App\_Data目录整个复制到新的项目;
4. 修改Global.asax.cs文件如下：

namespace WebApplication1

{

public class MvcApplication:Jurassic.WebFrame.MvcApplication

{

protected override void Application\_Start()

{

base.Application\_Start();

}

}

}

1. 按照Docs中的建库脚本CreateNewV3.sql建立新的库，并执行CreateAdmin.sql创建一个新的管理员账号admin;密码为888888。
2. 修改Web.config中的DefaultConnection连接串为新的数据库连接;
3. 调试运行，在第一次登录时，可能要强制修改密码。

### 升级到WebFrame 3.1：

**WebFrame 3.1有如下功能的调整：**

1. 完善了首页定制功能，用户可以拖动改变布局或取消、重置首页布局。并且可以不同角色进入不同的定制首页
2. 功能管理界面现在可以自动浏览框架中的程序集，自动根据控制器和方法的信息填写表单。
3. 功能管理完善了当两个功能项对应相同控制器和方法时，使用不同的参数控制权限的逻辑。
4. 现在主页面刷新时，菜单可以记忆刷新前的状态。
5. 用户信息增加了TrueName属性，记录用户真实姓名，并且可由用户自行修改。
6. 开始在Tags文件夹下存放编译后的各基线的稳定版本。
7. 删除了长期不用的Echarts、HighCharts和Knockout等第三方js组件。

**修改了如下BUG**

1. 在不同账号在同一浏览器登录时，由于重新打开了上次打开过的页面导致无权限的问题。
2. 在首页不显示真实姓名的问题。
3. 用户管理中，点击保存以后关闭小窗口
4. 其他小BUG

## 数据库安装：

系统自带的标准数据库为SQLServer 2012版本的WebTemplateDB.mdf, 在项目的App\_Data文件夹下，是作为文件型数据库而存在。与标准Sqlserver完全兼容。

框架数据库自身有完整的用户、权限、角色、组织机构管理模块，而标准库中有支持这些模块的相应表结构。可以将其附加到一个正式的sqlserver2012的服务器上。也可以通过生成脚本或导入导出建立到其他版本的数据库服务器上。

在建立库以后，再修改Web.Config配置文件中名为”DefaultConnection”连接字符串的配置即可（不可改名称）。

系统也自带对Oracle数据库的支持。基本步骤和Sqlserver相似，只是除了要修改连接字符串以外，还要在global.asax.cs全局应用程序类中，需要调整一下表空间：

ninjectKernel.Rebind<UserContext>().ToSelf()

.WithPropertyValue("Schema", ""); //要支持Oralce数据库，需要""中是Oralce库的Schema名称

## 公共模块的配置和运行

在项目正常建立以后，系统自带的“系统管理”功能组已经可以使用。它包括：

|  |  |
| --- | --- |
| 功能管理 | 查看并管理系统的功能  模块、页面和按钮的树型结构 |
| 角色管理 | 管理系统的角色，对角色进行功能模块的权限分配 |
| 用户管理 | 管理系统用户并授予角色权限 |
| 语言管理 | 进行多语言资源的管理 |
| 日志管理 | 查看并管理系统日志 |
| 组织管理 | 管理组织机构、岗位和人员 |
| 初始化 | 进行系统初始化操作，扫描程序集，找出功能点，并创建一个全部权限管理员账号 |

关于这些系统自带功能的使用，请参照文档《应用程序组件技术说明》。

## 服务端Controller代码编写方法

框架对服务器的控制器基类，以及执行每个动作的返回值进行了限定，以便于充分和客户端的框架脚本配合工作。

首先每个控制器需要继承自控制器的基类BaseController;

比如:

public class LogController : BaseController

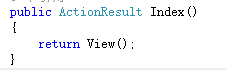
BaseController封装了一般性的异常处理、日志记录以及用户身份等属性和方法。还提供了布局页显示一些公共元素（如面包屑导航）的一些必需的信息。

下面分几种情况来举例说明服务端(主要是控制器Controller和Action)的编写方法

#### 返回视图

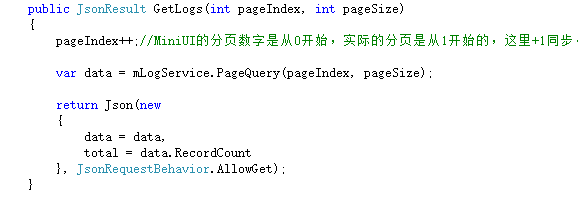
返回视图一般用于打开一个功能的操作页面。由于mini-ui框架以ajax异步操作为主，所以通常不需要在视图中声明数据model。

比如:



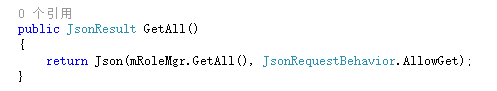
#### 执行GET请求返回列表数据源

比如 分页的情况：



以上用了一个包含data和total的匿名对象封装返回值，是配合mini-ui的datagrid控制显示分页数据的一般写法。这样可以充分利用data-grid的一系列默认值设定。

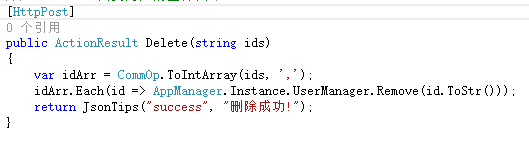
如果查询结果无需分页，则可以写成类似于这样：



即直接将列表序列化到前端，无需作额外的匿名对象包装。

#### 执行POST请求并返回结果。一般用于发起增删改等数据维护操作的场合。

比如：



注意这里调用JsonTips()返回一个约定的对象便于前台框架统一处理返回值。JsonTips()方法有多个重载， 分别是：

protected JsonResult JsonTips(object returnValue = null)

protected JsonResult JsonTips(object returnValue, JsonRequestBehavior method = JsonRequestBehavior.DenyGet)

protected JsonResult JsonTips(string type, string title, string message, object returnValue = null， params string[] formatArgs)

protected JsonResult JsonTips(string type, string message, object returnValue = null, params string[] formatArgs)

其中，type参数的取值范围是”success”、”warning”和“error”, 分别会在浏览器端显示不同强度的提示信息。提示信息主要由title和message两个参数表示。而returnValue可以是任意可用json序列化的类型的返回值。

其中，title和message参数都是默认作为多语言关键字而存在。它们会自动在App\_Data/CustomStrings.xx\_xx.resx中形成一个字符串的Key。这意味着，在切换到不同的语言环境时，它们会自动转成另一种语言的表现形式。

所以，有必要编辑相应语种的资源文件，把它们先翻译为对应的文字。

比如下面例子中的Move\_Success和Move\_Success\_{0}都是多语言关键词。

Message支持带格式化参数，比如例子中的{0}占位符。

格式化参数在最后的可选参数params string[] formatArgs)中提供。

Title不支持带格式化参数。

多语言的关键字没有强制性的语法规定。但推荐用英文+下划线形式。以便于不同语种的人都能识别并编辑它对应的内容。

如：

public JsonResult Move(string ids, string pId)

{

//...

return JsonTips("success", "Move\_Success", "Move\_Success\_{0}", GetTreeJsonObject(movedFunc), movedFunc.Name);

}

## 前端JS代码编写方法

Scripts/webframe-1.0.js是本框架的前端核心js代码库。它的作用是自动从服务端权限系统中读取用户菜单，自动生成用户菜单和页面操作按钮，并统一规定了一般表单的处理流程，将表单处理分解为beforeSubmit、beforeAjax、afterSubmit等几个规定动作。

此框架给前端代码的编写带来如下便利：

1. 分层次的用户菜单和按钮，都可以在上述的《功能管理》界面统一进行配置，无需硬编码每个菜单的页面地址，每个页面中的按钮及按钮回发事件和回发地址。
2. 权限可以很轻松地控制在按钮一级。
3. 前端的视图界面编写得到了简化。开发人员无需考虑引用js、导航栏、工具栏、FORM表单标记等常规元素。所有页面都已通过框架中的\_Layout和\_FormLayout两个布局页进行处理。而开发者只需编辑原来处于<FORM></FORM>表单标记之间的内容即可。
4. 在默认情况下，无需编写任何代码，界面的按钮就可以响应事件并执行回发操作，完成正常的处理流程，并自动返回并显示处理结果。无需再为每个按钮写冗长而重复的验证数据、拼接提交地址、处理提交数据、提交请求、获取结果、显示结果、处理错误、页面跳转的大段代码。所有这些操作框架内部已经自动完成。
5. 如果有需要，提供了规定的扩展点对按钮提交前、提交后公开了某些自定义行为的编程接口，开发人员只需要在这些少数接口点加入自己的特有逻辑即可（如果有的话）。

在下面的一节中，就具体介绍了这方面的扩展点。

### $.regButton(options)扩展

Webframe的js代码核心是$.regButton(options)这个jquery的扩展方法。它的options参数的定义为：

{

id : '按钮ID',

text:'按钮文本',

//以上两者选其一，以ID优先

noSubmit: function(){..} //不执行提交操作的方法,

beforeSubmit: function(ajax) {…} //表单数据验证前方法,

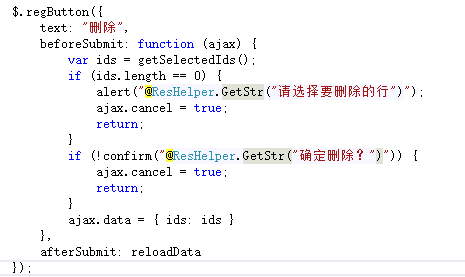
beforeAjax: function(ajax){…} //向服务器发请求之前方法,

afterSubmit:function(data){…}提交后方法

}

它们的执行顺序，就是上面参数的顺序。

比如一个实例：



##### Options中的属性详解：

**noSubmit**: 当一个按钮点击无需经过后台处理，而只执行本地操作时调用。当options中定义了noSubmit时，以下几个属性均为无效。

**beforeSubmit**: 当一个按钮的点击经过服务器后台处理，提交时表单数据验证之前调用此定义的函数。beforeSubmit自带参数ajax. 其中的属性和jquery的.ajax()方法中的属性相同，稍稍做了扩展，如下：

{

//以下都是jquery.ajax自带的属性

url: ‘提交的url, 由框架根据配置文件自动指定’

data: ‘提交的数据，默认是整个表单序列化后的数据‘

success：’提交后执行的方法，已由框架指定‘

error:’出错处理’,

method: ‘提交方法是POST还是GET，由框架根据配置文件自动指定’

//以下是框架本身扩展的属性

id: 按钮的Id, 可以识别按钮

text: ‘按钮文本，在id没有指定时，可以识别按钮，如果指定id则以id优先’,

validate: ‘是否自动验证表单，默认值是true’,

cancel: ‘是否中途停止表单的提交，默认值是false’

}

以上的每个属性，都可以在beforeSubmit的函数中赋新值。比如，上述的例子中，ajax.cancel=true可以在条件不满足时中止表单的提交。

**beforeAjax**: 在数据提交前执行。这时数据验证已通过，beforeSubmit也以执行完毕。它的参数和beforeSubmit完全相同，所以仍然有机会修改。这个函数一般用于提交时动态显示一些效果，比如等待信息。

**afterSubmit**: 提交成功后执行的函数。它有一个data参数，其中的属性定义如下：  
{

Type: “success”,”warning”或“error”,

Title:”信息标题”,

Message:”详细信息“，

ReturnValue: Object,指具体的返回值

}

其中前三个属性由框架自动显示提示信息，最后一个ReturnValue是业务数据，可以在此函数内进行业务性的处理。注意大小写。

以上三个属性对应的函数中，”this“关键字都表示是按钮本身的HTML DOM对象。

##### 全局性的事件

以上都是每个按钮分开定义自己的响应事件，互不干扰。但有些情况下，需要所有按钮在某个时刻都执行相同的动作，比如刷新页面等。基于此，另外定义了全局性的form表单事件，

它们的触发方法分别是：

$(‘form’).trigger(“beforeSubmit”, function(evt, ajax){….});

$(‘form’).trigger(“beforeAjax”, function(evt, ajax){….});

$(‘form’).trigger(“afterSubmit”, function(evt, data){….});

参数说明:

evt表示事件对象；

ajax表示将要提交的ajax对象；

data表示提交完成后的返回的数据对象

例如：

$('form').bind('afterSubmit', function () {

grid.reload();

});

它们都会紧接在相应的同名按钮事件之后引发。并且**第二个**参数定义和按钮事件相同。

又如：

$('form').bind('beforeSubmit', function (evt, ajax) {

showTips({ Type: 'warning', Title: '没有提交' });

ajax.cancel = true; //防止表单进一步提交

});

此外，为了方便前台JS与后台的交互操作，还额外提供了四个扩展：

强烈建议在框架中前台ajax请求时，使用这4个扩展之一来代替jquery自带的$.ajax()或其他ajax调用写法：

理由是：这4个扩展自带了错误处理机制。

### $.newGET(url, data, callback，dataType) 扩展—无缓存的GET请求

因jquery自带的$.getJSON方法有缓存，但用POST不符合http规范。所以额外定义此扩展，来无缓存用GET方法请求后台数据。

url: 请求地址

data:请求参数，可以省略此参数

callback：成功时的回发事件

datatype: 返回数据的类型，默认是”json”

### $.newPOST(url, data, callback,dataType) 扩展—POST请求

因jquery自带的$.getJSON方法有缓存，但用POST不符合http规范。所以额外定义此扩展，来无缓存用GET方法请求后台数据。

url: 请求地址

data:请求参数，可以省略此参数

callback：成功时的回发事件

datatype: 返回数据的类型，默认是”json”

### $.getSync (url, data, dataType) 扩展 --- 同步GET请求直接返回结果

通常jquery的ajax请求是异步的，此方法可以同步请求直接返回结果值。这样可以简化程序逻辑。

url: 请求地址

data:请求参数，可以省略此参数

datatype: 返回数据的类型，默认是”json”

例如：

var \_userMenus = $.getSync(\_userMenusUrl);

### $.postSync (url, data, dataType) 扩展 --- 同步POST请求直接返回结果

通常jquery的ajax请求是异步的，此方法可以同步请求直接返回结果值。这样可以简化程序逻辑。

url: 请求地址

data:请求参数，可以省略此参数

datatype: 返回数据的类型，默认是”json”

例如：

var \_userMenus = $.getSync(\_userMenusUrl);

就以系统的日志管理页面为例，说明一下前端代码的写法：

### $.lang(key) js端的多语言API

该API是在WebFrame2.1版本推出的。它类似于服务端的Html.Str(key)扩展，可以动态在资源文件中添加原来没有的Key. 如果遇到以前没有定义过的Key.它会返回Key本身，并在稍后自动向服务器发送一个添加Key的请求。

### showTips 非强制提示方法

不同于alert(), showTips不必强求用户点击确定，显示3秒钟会自动消失。

有两种调用方法 一种是showTips(tips)此时tips是一个对象：

{Type:”error/warning/success 中一种”, Title:”消息标题”, Message:”消息内容”}

一种是showTips(type, title, message)

即将上述对象三个字段分开用三个参数表示。

### 出错处理

以上的4个扩展，均已用统一方案考虑出错时的显示和处理情况，所以无需在编程时额外考虑。

## View视图的编写方法

前面已经说过，具体功能页面无需考虑工具栏按钮和面包屑导航，以及表单标记。只需摆放自己的业务组件UI即可。

一般View的结构如下：



由于在Views/Shared中定义了\_FormLayout.cshmtl为默认的布局页，所以，如果View中不作特别声明（如上面的例子），则都会以\_FormLayout作为布局页，其中就包含了默认的面包屑导航、工具栏和表单标记。包含所有框架功能。

如果有特殊需要，不想从\_FormLayout继承得到View,则有一个基本的布局页 \_Layout.cshtml。其中只包含对基本的框架js和样式表的引用，不含\_FormLayout中的附加元素。类似的元素需要自行编写。也必须显示在开头声明页面的Layout为 \_Layout.cshtml.如下：

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

特殊的样式

frame-fit 一般用于页面中的最后一个控件（比如数据网格）自动调整该控件高度到页面底部。

frame-submit：如果手写一个按钮，这个按钮不在工具栏，又想走框架的按钮事件流程，则在按钮的class中加上这样一个样式名称（比如登录页的登录按钮）

frame-default: 如果一个按钮需要接收回车键，则设定此样式名称。

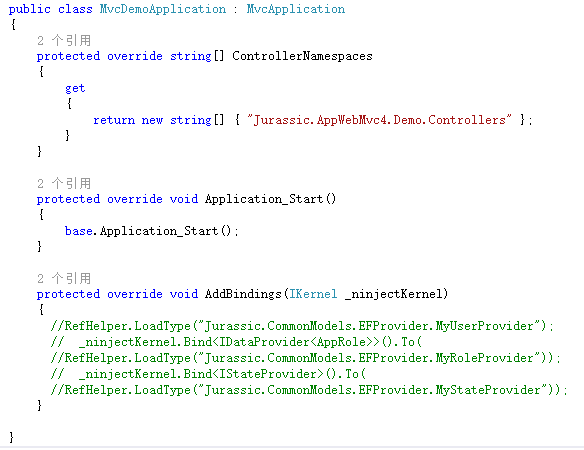
frame-search-grid: 如果一个DataGrid控件需要支持搜索，则声明此样式

frame-search-tree: 如果一个tree控件需要支持搜索，则声明此样式

## Global.asax的编写方法

具体项目中的global.asax不能省略，并且要继承自Jurassic.WebFrame.MvcApplication.

例如：

  
其中，ControllerNameSpaces重写以返回自身的控制器所在的命名空间。(在2.0版本之后不再需要重写)。

AddBindings()方法用Ninject依赖注入容器注入系统或自身定义的接口实现。

## 工具栏和搜索框的编程

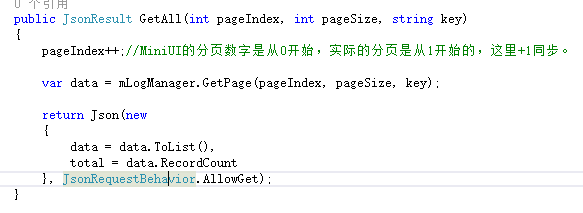
在标准的FormLayout中，在工具栏右侧有一个搜索框。如图：



在框架中的定义是当前页面的搜索，此搜索框在默认情况下，典型页面中，是配合页面中的数据展示控件（比如mini-grid或mini-tree）进行条件查询。

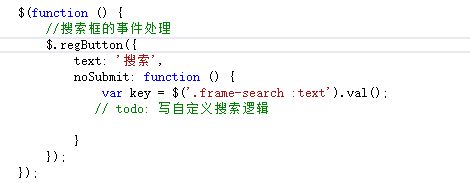
具体操作方式如下：

1. 在View视图的数据展示控件代码加一个伪样式名称:frame-serach-grid:
2. 在服务端Controller中，对应返回数据的方法，加上参数’key’, 比如：



至此，一个支持简单条件搜索的功能就完成了。当用户在搜索框输入关键词并回车，在相应的grid控件会自动展示搜索结果。

以上是基于框架支持的自动化搜索编程方式。在此方式下，无需在视图端写任何脚本逻辑。但有时如果需要自定义搜索逻辑，可以在视图端脚本中添加以下内容：



### 如何自定义工具栏？

工具栏按钮一般是通过AppFunctions.json文件中的配置自动生成的。在某些无关权限的特殊页面，可能不需要通过配置，直接硬编码工具栏按钮。为此，框架中的\_FormLayout.cshtm布局页中定义了一个名为UserToolBar的section. 通过在视图中定义这个section,即可以加上自己定义的工具栏按钮。

比如：

@section UserToolBar

{

<a class="mini-button" plain="true">@Html.Str("Refresh")</a>

}

在上面的代码中，定义了一个刷新按钮。

然后，就可以在js代码中绑定这个工具栏按钮事件：

$.regButton({

text: "@Html.Str("Refresh")",

noSubmit: function () {

gridDeptUsers.load();

}

});

以上代码的完整示例，参考：

Jurassic.WebQuery/Views/SelectDepUser/Index.cshtml.

如果完全不需要显示工具栏，则使用开关变量：

ViewBag.ShowToolBar=false;

### 如何控制搜索框是否显示？

在视图页或控制器返回视图的相关方法中，加上：

ViewBag.ShowSearchBox = false;

如果没有上述声明，搜索框默认展示。

类似的，还有如下开关：

ViewBag.SearchEmptyText = “搜索框为空时的文本”;

控制面包屑导航：

ViewBag.ShowBreadCrumb = false;

控制是否重新加载菜单：

ViewBag.ReloadMenu = true;

这是因为有时程序出于某种需要，在控制器中用代码对自身的菜单或按钮进行了增删改，但这种数据不会立即同步到主框架中存储的本地js菜单数据，会导致新增的按钮点击无效。使用以上代码可以强制让主框架更新其菜单数据。

## Web框架可选组件 Jurassic.Web\*\*\*\*

目前框架提供了以下可选组件.

|  |  |
| --- | --- |
| Jurassic.WebSignalR | 支持服务端主动推送消息的组件 |
| Jurassic.WebReport | 基于RDLC Report的报表组件 |
| Jurassic.WebUpload | 通用文件上传组件，支持大文件和断点续传 |
| Jurassic.WebHtmlEditor | 基于KindEditor的在线HTML编辑器组件 |
| Jurassic.WebRepeater | 分页展示数据的MVC组件 |
| Jurassic.WebQuery | 基于数据模型的数据增删改查的套件 |

在使用它们时，可以直接引用其DLL或项目，或者不显式引用，将其DLL拷贝到实际项目的bin目录下，都可以应用组件功能。它们所要用到的脚本资源，也延用WebFrame的同样的打包方式 。

以上组件都在WebTemplate的模板项目中给出了应用的实例Demo。下面几节结合DEMO分别讲解使用方法和编程接口。

### Jurasssic.WebUpload 文件上传组件

#### 概述

Jurassic.WebUpload是基于Jquery.FileUpload组件(见<http://blueimp.github.io/jQuery-File-Upload/>)二次封装的一个MVC环境下的文件上传组件。

对比普通的HTTP文件上传技术方案，它有以下特点：

1. 支持大文件上传，文件大小只受磁盘空间限制；
2. 上传文件大小不受IIS限制；
3. 支持断点续传；
4. 上传有进度条显示；
5. 有统一的前台和后台编程调用接口和扩展点。

#### 运行环境

该组件必须结合WebFrame框架，不能单独使用。

浏览器支持HTML5的FileAPI。否则不能实现分段上传和断点续传。

#### 使用方法

下面以WebTemplate框架模板项目中相关demo为例，也说明WebUpload组件在项目中的使用方法(可参照WebTemplate中UploadImage.cshtml中的写法)：

1. 依据前述建立项目的方法，依据WebTemplate模板建立一个基于WebFrame框架的MVC项目。
2. 引用Jurassic.WebUpoad项目，它的源码项目地址是: svn/公用组件库/dotnet/AppCenter/Jurassic.WebUpoad.csproj. 如果使用编译后的DLL，则在 svn/公用组件库/dotnet/AppCenter/packages/Jurassic.FrameWorks/Jurassic.WebUpload.DLL。该DLL中，打包了所有组件资源，包括前台脚本和后台处理逻辑，所以只需要引用，就会自动在项目中生成运行组件所需要的软件环境。
3. 在自已开发的项目中，在需要引用组件的视图中，添加如下代码：  
   在视图文件的开头，引用命名空间：

@using Jurassic.WebUpload;

在视图中需要引用上传组件的位置，写如下调用：

@Html.Upload("uploadresult1", "UploadGetName", "HelloWorld", "uploadDone1")

其中的参数含义是：

"uploadresult1" 是表单中，上传后文件的ID保存到的表单对象名称。该表单对象（通常是隐藏域）会随表单的整体提交到服务器。

"UploadGetName": 上传后执行的后台方法所在控制器中的方法名称

"HelloWorld": 上传后执行的后台方法名称。

“uploadDone1”: 后台指定的控制器中指定方法返回处理后的数据后，前台回调的js函数名称。

当然，如果你不需要上传完毕后，后台立即执行回调函数，只需要表单整体提交时，传文件ID给服务器端，也可以将后面三个参数都省略。

1. 后台回调方法的写法：

在前面一步中，已经申明后台的回调方法，那么在服务端控制器中（如HellowWorld控制器）的方法对应写法如下：

public ActionResult UploadGetName(ResourceFileInfo[] results)

{

//TODO： 这里写自己的处理逻辑，在DEMO中，只简单演示一下返回上传文件的文件名

return Json(results.Select(r => r.FileName));

}

注意到上面的方法参数签名是固定的。Results这个参数集合里包含了一切上传文件的相关信息，包括文件名、长度、媒体类型，甚至还包括文件流。

1. 前台js回调函数的写法：

<script type="text/javascript">

mini.parse();

function uploadDone1(r) {

alert("此处表示组件1上传后的回调，它显示上传的文件名： Name=" + r);

}

//…

</script>

1. 表单的整体提交后的写法：如果不需要回调的话，上面的代码可以省略。但通常表单的整体提交是不可少的。所以，在表单整体提交的后台处理逻辑里，需要获取到视图传递的上传文件的ID：

[HttpPost]

public ActionResult UploadImage(string uploadresult1, string uploadresult2)

{

//表单其他数据处理

//

return JsonTips("success", "提交成功",

"上传的文件ID分别为：组件1：{0}, 组件2：{1}", (object)null, uploadresult1, uploadresult2);

}

1. 在表单中上传文件的ID传递到后台后，通常假想的做法，是将此ID保存在实际系统业务表的某个记录文件信息的字段中。在今后可以通过此ID访问到上传文件的各种信息。

为配合此任务，框架提供了一系列API, 它们都封装在Jurassic.CommonModels. ResourceFileService这个类中，只需要将这个类实例化，就可以调用其中的方法，常用的有：

/// <summary>

/// 获取文件信息（包括文件流），不存在返回null

/// </summary>

/// <param name="fileId">文件信息ID</param>

/// <returns></returns>

public ResourceFileInfo GetFile(int fileId)

/// <summary>

/// 删除文件信息

/// </summary>

/// <param name="fileId">文件信息ID</param>

/// <returns>被删除的文件信息，如果没有找到文件返回null</returns>

public ResourceFileInfo DeleteFile(int fileId)

/// <summary>

/// 获取文件流，不存在返回null，存在返回文件流Stream

/// </summary>

/// <param name="fileId">文件信息ID</param>

/// <returns></returns>

public Stream GetFileStream(int fileId)

此外，如何通过ID，来生成一个可供用户下载的URL?

此下载的格式为 ~/Files/Download/ID。它将提供给用户一个具有原始文件名的下载。

### Jurasssic.WebReport Web报表组件

#### 概述

本组件是对RDLC报表的MVC风格的封装。

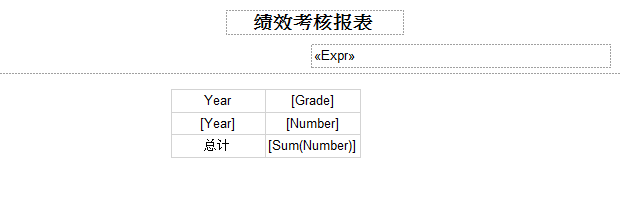
使用本组件，可以有以下好处:

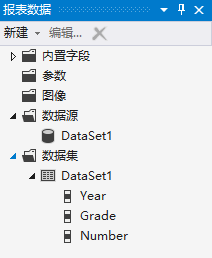
1. 使用本组件，可以减少后台数据绑定代码的编写。
2. 可以在代码中设置报表的常用样式。

#### 使用方法

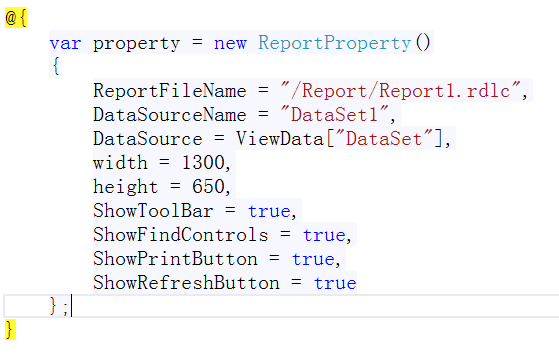
可以参考WebTemplate框架模板项目中示例Report Demo，以及项目管理规范/12-项目模板中的Demos中的WebReport报表示例。

1. 在项目引用中添加WebReport引用, 它的源码项目地址是: svn/公用组件库/dotnet/AppCenter/Jurassic.WebReport.csproj. 如果使用编译后的DLL，则在 svn/公用组件库/dotnet/AppCenter/packages/Jurassic.FrameWorks/Jurassic.WebReport.DLL。
2. 添加报表设计器，设计报表，添加数据集





1. [在视图中使用@Html.Report(property)](mailto:在视图中使用@Html.Report(property));来渲染报表，参数property可以用来进行数据绑定和报表的样式的定制，property作为一个对象包括多个属性。如图：



### Jurasssic.WebHtmlEditor 在线HTML编辑器组件

#### 概述

本组件是基于对KindEditor4.1.10的开源组件（http://www.kindeditor.net）进行MVC风格的封装。

使用本组件，可以有以下好处:

1. 在本组件中，封装了kindeditor的js引用和初始化逻辑，用户无需考虑这些细节。
2. 由于所有组件所需资源已经打包到组件DLL中，所以只需要引用本组件DLL，就可以自动获取所有这些资源，而不需要手动添加到项目中。
3. 提供了规范化的调用接口，用户可以对组件初始化的方式进行有限的选择。

#### 运行环境

该组件必须结合WebFrame框架，不能单独使用。

浏览器支持IE8以上，及其他非IE最新版本的浏览器。

#### 使用方法

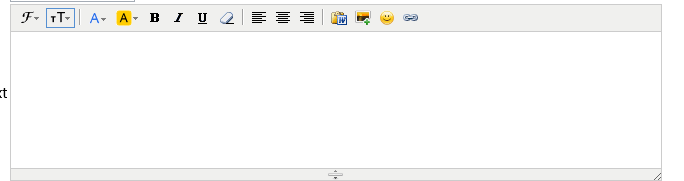
该组件需要和一个标准的TextArea控件绑定。TextArea的大小就是生成的Editor控件的大小。

所以在cshtml页面中，首先要有一个textarea:

@Html.TextAreaFor(model=>model.Text) //假定数据模型有Text属性。

@Html.HtmlEditor(“Text”) //因上面的代码生成的TextArea自动生成了name=”Text”属性，所以可以直接用“Text”名称为参数初始化编辑器

如图所示：

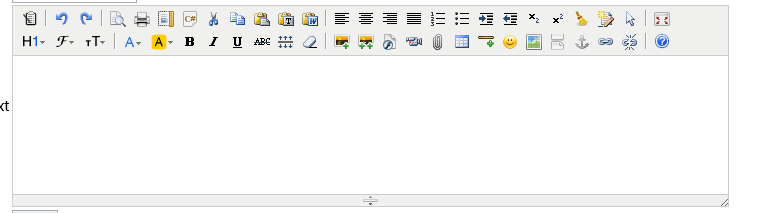


以上默认是工具栏简化以后的结果，

如果想显示全工具栏，则可用以下方式调用：

@Html.HtmlEditor("Text", null, true)

则会显示以下编辑器：



对于编辑器的精确编程，比如说，要调用此编辑器的原生api(具体可查阅该编辑器的文档，

地址是：<http://kindeditor.net/doc.php>）

则可采用以下方式 调用：

@Html.HtmlEditor("Text", “myEditor”)

其中，myEditor就是编辑器初始化后，生成的全局对象的变量名称。

可以基于此对象对编辑器调用它的原始API。

比如：

myEditor.html(“<p>Hello world!</p>”);

关于此组件的演示DEMO，可以参见Jurassic.WebTemplate这个模板项目中的 HelloWorld/HtmlEditor 这个具体实现。

此编辑器已经实现了表单提交时的数据同步到TextArea。所以无需再调用它的同步代码。

在控制器一端，需要注明处理逻辑方法的标记为：

[ValidateInput(false)]

否则，由于MVC的安全机制，将不允许提交带html标记的内容。

### Jurasssic.WebRepeater 分页数据展示组件

#### 概述

虽然框架集成的mini-ui组件中已经有功能非常强大的datagrid控件，并且已被广泛使用，但框架组认为，还是要补充一种基于传统的服务端直接推送的数据展示组件。类似于asp.net webform的Repeater控件。但是基于MVC的方式实现。

它的特性如下：

1. 可以用cshtml提供的HTML数据标签来定义展示模板。这样可以回避复杂的js前端模板处理。充分利用razor引擎本身的强大的数据定义和处理能力。
2. 显示方式比mini-datagrid更灵活，单项的显示完全基于单项模板ItemTempalte.
3. 支持传统的分页以及目前移动端流行的“点击展示更多”这种数据逐步加载的功能。

#### 使用方法

该组件是基于框架定义的Pager<T>泛型分页对象来实现的。

典型的后台逻辑（参考WebTemplate/RepeaterController中的实现）：

public ActionResult Index(PageModel pageModel)

{

return View(new Pager<HelloWorldModel>(allData, pageModel.Page, pageModel.PageSize));

}

前台cshtml视图逻辑：

@using Jurassic.WebRepeater;

@model Pager<Jurassic.WebTemplate.Controllers.HelloWorldModel>

@{

ViewBag.Title = "Index";

}

<link rel="stylesheet" href="~/Scripts/bootstrap/css/bootstrap.min.css" />

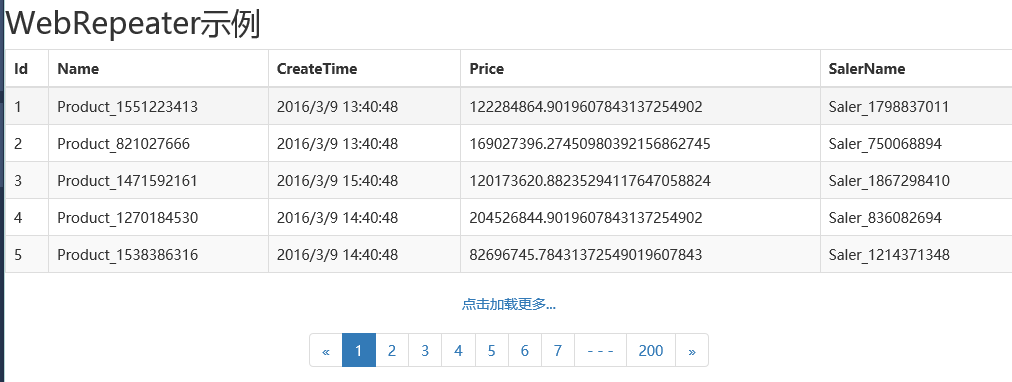
<script src="~/Scripts/bootstrap/js/bootstrap.min.js"></script>

<h2>WebRepeater示例</h2>

@Html.Repeater()

说明，在此处，用了一个无参数的调用，意味着所有参数取默认值。

运行所得到的结果为：



关于参数的设置：

@Html.Repeater(new RepeaterFormData

{

Id = "repeater1",

HeaderTemplate = GetHeaderTempalte,

ItemTemplate = GetItemTemplate,

})

在以上的调用中，HeaderTemplate和ItemTemplate分别定义了首部模板和分项模板。这些模板是在页面上定义的MVC helper方法：如：

@helper GetItemTemplate(object obj)

{

<tr>

………….

</tr>

}

有关使用的DEMO，请参见WebTemplate中的实例：

其中，RepeaterContoller展示了最简单的用法。

NotebookController展示了稍复杂的带有实际数据CURD操作的用法。

### Jurasssic.WebQuery 高级查询与数据采集组件

#### 概述

在诸如采购管理一类的管理软件中，大量的开发工作都是基础数据的增删改查的常规和重复的任务。这些代码相似度极高，浪费了大量的有限的项目时间又无法保证质量。本组件试图用一个通用的模块来解决这个问题。

本组件并不是一个类似于CodeSmith的代码生成器，而是一个具有从界面展示到数据存储的一系列功能的完整解决方案。

该组件分为高级查询(AdvQuery)和数据采集(AdvData)两个部分。其中，AdvData依赖于AdvQeury。它们共同实现了常规数据的增删改查操作。

#### 运行环境

基于ASP.net MVC4 和EntityFramework 6， 前端的展示和录入界面基于mini-ui, 需要公司的WebFrame框架支持才能运行。

而且模型的定义必须基于框架预先定义的接口。

#### 功能特点

高级查询：

1. 支持空格隔开的多关键字查询;
2. 可以配置复杂的AND OR 组合条件查询；
3. 可以将复杂的查询命名保存以便于再次调用；
4. 极大地简化了常规则列表页的查询的编写方法：只需要打一个[AdvQuery]标记即可。

数据采集：

1. 以实体类为核心，以打标记的方法描述实体类各个属性的规则、约束和录入方式；
2. 可以实现主子表的复杂场景，可以一个主表多个子表；
3. 主表和子表可以有显式的主外键关系，也可以没有；
4. 业务实体定义和数据实体分开，业务实体只用于展示和接收用户输入，数据实体只用于数据持久化，符合通常的编程套路；
5. 自动生成列表页、新增页、编辑页；
6. 支持日期时间、枚举、布尔值、下拉列表等的自动生成对应录入界面；
7. 通过简单的配置，即可以实体数据编辑时下拉列表复杂的联动；
8. 自带通用的数据验证；

#### 适用场合

由于这套组件是以实体模型为核心，所以只适用于那些表结构比较稳定和规整，有完整的业务模型的场合，不适用于预先不知表结构的全动态数据场景。

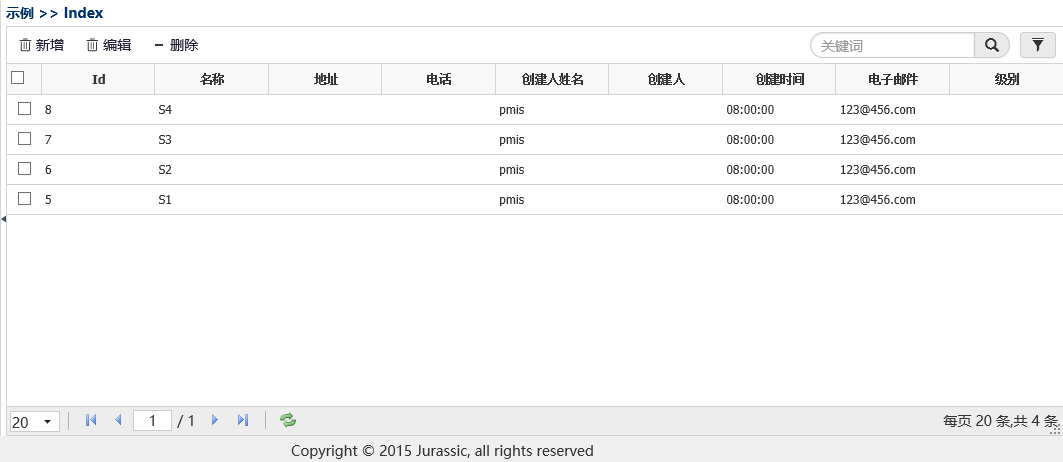
#### 使用方法

##### 数据采集：

下面是一个假设的业务场景，以一个简单的供应商和产品之间的关系为例，来说明数据采集组件的用法。

首先展示一下界面：

列表页：



详情编辑页：



下面分步骤来说明数据采集组件AdvData的使用方法。

###### 定义数据模型和业务模型

通常我们使用EF构造一个应用程序时，可以用VS自带的生成工具，根据指定的数据库生成相关的数据实体类。（使用VS2013 R4以上，选择新建 -> 数据->ADO.net实体数据模型）。系统会提示勾选数据库和表，并生成对应的实体类代码。当然你也可以手写数据实体代码。

而我们需要根据业务展示和用户输入的特点，在此基础上再建立一层业务实体类。

比如: 以一个简单的供应商和产品之间的关系为例，数据实体假设是这样的：

public class Supplier : CUDEntity, INamedEntity

{

public Supplier()

{

Products = new List<Product>();

}

public string Name { get; set; }

public string Address { get; set; }

public string Tel { get; set; }

public SupplierLevel Level { get; set; }

public string Email { get; set; }

[ForeignKey("MasterId")]

public virtual ICollection<Product> Products { get; set; }

}

public class Product : INamedEntity, IDetailEntity<Supplier>

{

public int Id { get; set; }

[ForeignKey("Master")]

public int MasterId { get; set; }

public virtual Supplier Master { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public string Unit { get; set; }

}

以上是EF用以保存数据的实体类，和表结构是一一对应的。

**解释： CUDEntity, 是框架定义的一个基类，如果一个实体需要记录完整的增删改信息，并且该实体需要逻辑删除，则继承此类。这不是使用此组件的必要条件。**

**INamedEntity: 是框架定义的一个具有ID和Name两个属性的实体类。**

**IDetailEntity<Supplier> 这是定义了此子表是Supplier的子表。MasterId是外键指向主表的ID。**

相应的业务模型如下：

[CatalogExt(EntityType = typeof(Supplier))]

public class SupplierModel : INamedModel

{

public SupplierModel()

{

Products = new List<ProductModel>();

}

/// <summary>

/// 定义该属性在录入界面不显示

/// </summary>

[CatalogExt(DataSourceType = ExtDataSourceType.Hidden)]

public int Id { get; set; }

public string Name { get; set; }

/// <summary>

/// 定义本属性用下拉列表输入，用分号隔开的关键词将作为下拉列表的下拉选项

/// </summary>

[CatalogExt(DataSource = "北京;上海;武汉;西安")]

public string Address { get; set; }

/// <summary>

/// 定义本属性用一个用户自定义方法来返回一个下拉列表

/// </summary>

[CatalogExt(DataSource = "GetTel", DataSourceType = ExtDataSourceType.UserDefine)]

public string Tel { get; set; }

public string GetTel()

{

return "123;456;789";

}

public string CreaterName { get; set; }

/// <summary>

/// 定义本属性用一个SQL语句来返回下拉列表选项，其中SQL语句返回两列，第一列成为下拉的显示值，第二列是实际的值

/// </summary>

[CatalogExt(DataSource = "SELECT UserName,UserId FROM USERPROFILE", DataSourceType = ExtDataSourceType.SqlQuery)]

public int CreaterId { get; set; }

/// <summary>

/// 定义只需要录入时间

/// </summary>

[CatalogExt(DataType = ExtDataType.Time)]

public DateTime CreateTime { get; set; }

public string Email { get; set; }

public SupplierLevel Level { get; set; }

/// <summary>

/// 该集合对应数据实体中的集合

/// </summary>

public IEnumerable<ProductModel> Products { get; set; }

}

[CatalogExt(EntityType = typeof(Product))]

public class ProductModel : INamedModel

{

public int Id { get; set; }

public string Name { get; set; }

public decimal Price { get; set; }

public string Unit { get; set; }

[Browsable(false)]

public int MasterId { get; set; }

}

在通常情况下，数据模型和业务模型有很高的相似性。但数据模型一般直接对应数据库表，

结构非常严谨。而业务模型可以根据展示的需要，额外多加一些冗余数据。

注意业务模型的定义是使用本组件的重点。

注意到上面很多属性都打了[CatalogExt()]标签用于定义该列的显示和编辑属性。

比如，在类名上加上了：

[CatalogExt(EntityType = typeof(Supplier))]

用以声明该业务实体对应的数据实体类型。

其中三个属性(Address、Tel、CreaterId)的录入都是采用的下拉列表。

但它们声明的下拉列表数据源是不同的。

Address是用一个；号隔开的字符串来表示下拉列表项:

[CatalogExt(DataSource = "北京;上海;武汉;西安")]

public string Address { get; set; }

Tel是用一个用户自定义方法来表示下拉列表项。

[CatalogExt(DataSourceType = ExtDataSourceType.UserDefine)]

public string Tel { get; set; }

CreaterId是用一个SQL查询来表示下拉列表项。

[CatalogExt(DataSource = "SELECT UserName,UserId FROM USERPROFILE", DataSourceType = ExtDataSourceType.SqlQuery)]

public int CreaterId { get; set; }

对于DataTime类型，可以声明DataType=Date/DateAndTime/Time来分别表示显示或输入日期、日期时间或只显示或输入时间。如果不加标记，则默认只显示日期控件。

[CatalogExt(DataType = ExtDataType.Time)]

public DateTime CreateTime { get; set; }

对于枚举类型，组件会自动生成一个包含所有枚举值的下拉列表。

public SupplierLevel Level { get; set; }

CatalogExtAttribute这个标记，一般用于标记业务模型的定义信息。完整的定义如下：

|  |  |
| --- | --- |
| Name | 名称，等于标记上的属性名，当作用类上时，即为类名 |
| DataType | 数据类型名称枚举，枚举定义：  SingleLineText, //单行文本（默认值）  MultiLineText, //多行文本  Html, //HTML文本  SingleNumber, //整数  FloatNumber, //小数  Currency, //货币  Bool, //真假  Date, //日期  DateAndTime, //日期时间  Image, //图片  File, //文件  Address, //地图上的地点  Time //时间  UserId //用户ID，表示呈现为用户选择控件，当标签打在字符串型上时，为多选，当打在整型上时，是单选  ButtonEdit // 显示一个ButtonEdit控件  MultiLanguage // 表示该属性需要多语言支持  Auto // 初始化时的默认值，因为0=SingleLineText  所以用这个，该值在分析属性类型后，会被其他值代替 |
| DefaultValue | 默认值 |
| DataSourceType | 下拉数据源类型，类型定义有  None, //无下拉，默认值  DirectList, //简单下拉列表  MultipleList, //多选下拉列表  SqlQuery, //来自SQL语句查询的下拉列表  SqlQueryMultipleList, //SQL语句下拉列表，多选  UserDefine, 用户自定义数据下拉列表,自定义方法写在控制器中  Hidden, 隐藏，使用此选项时，在表单用一个隐藏域表示数据  Custom， 用户使用扩展方法自定义的控件  注意：UserDefine选项，开发者只负责提供下拉数据源，而Custom选项则完全由开发者自定义展示控件的代码 |
| DataSource | 数据源内容，如果数据源类型是DirectList或MulltiList,则用户可以用以下两种方式写表数据源的字符串：  DataSource="Name1;Name2;Name3;..."  或DataSource="Name1=Id1;Name2=Id2;Name3=Id3;..."  第一种表示下拉列表的显示值就是返回值，第二种表示下拉显示值对应ID值  如果是SQlQuery或SqlQueryMultipleList,则是一个SQL语句查询字符串，查询如果只返回一列数据，则类似于第一种  如果返回两列数据，则是第二种，第一列是下拉显示值，第二列是下拉返回值。 |
| MaxLength | 最大长度 |
| MinLength | 最小长度 |
| AllowNull | 可为空 |
| MaxValue | 允许的最大值 |
| MinValue | 允许的最小值 |
| RegExpr | 正则表达式验证规则 |
| Ord | 排序位，暂时未用，以原始属性声明为序 |
| ForceUpdate | 未用 |
| Browsable | 是否在列表页显示该列 |
| Editable | 是否可编辑，默认为True，为False时，在编辑界面只读 |
| EntityType | 当标记在在类名上时，表示该类对应数据实体的类型 |
| LinkedProperty | 关联的属性名称，用于下拉列表联动, 用于被联动的属性上 |
| DisplayProperty | 用于联动的下拉列表时，初始显示哪个属性的值 |
| DisplayFormat | 当显示日期/数值型数据时，显示的格式串，参考.net的格式串规则。比如yyyy-MM-dd或#0.00。当为null时，系统自动根据DataType确定。 |
| InputFormat | 当输入日期/数值型数据时，输入的格式要求。当为null时，系统自动设置为DisplayFormat的值。 |

关于数据模型和业务模型需要实现的接口列表如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 接口名称 | 内容 | 描述 | 业务模型 | 数据模型 | 业务子表模型 | 数据子表模型 |
| IId<int> | int Id | 定义自增长的int型Id主键 | 要 | 要 | 要 | 要 |
| ICanLogicalDeleteEntity | bool IsDeleted | 定义可逻辑删除的数据模型 | 不需要 | 如果逻辑删除则要 | 不需要 | 如果逻辑删除则要 |
| IDetailEntity | int MasterId | 定义强关联数据子表的模型 | 不需要 | 不需要 | 不需要 | 如果强关联(有外键)则要 |
| IAttachmentEntity | int BillId;  string ModuleCode | 定义弱关联的子表模型 | 不需要 | 不需要 | 不需要 | 如果弱关联则要 |
| IMultiLanguage |  | 定义需要数据多语言支持的热闹口 | 可实现可不实现 | 如果需要多语言支持则要 |  |  |

只要数据模型和业务模型有选择地实现以上接口，组件就能完成基本的工作。

此外，为了简化模型的设计工作，组件还提供了以下辅助接口和基类，以便于某些具有相同属性的实体编写工作，但这不是必需的。

业务实体类的接口和基类 (命名空间：Jurassic.CommonModels.ModelBase)

|  |  |  |
| --- | --- | --- |
| 接口(基类)名称 | 代码 | 说明 |
| IIdModel | 直接继承IId<int> | 只标记它是一个业务实体而不是数据实体的接口 |
| ICodedModel | string Code { get; set; } | 带编码属性的实体接口 |
| ICUModel | string CreaterName { get; set; }  DateTime CreateTime { get; set; }  string UpdaterName { get; set; }  DateTime UpdateTime { get; set; } | 标记了创建和修改时间和用户的实体接口 |
| INamedModel | string Name { get; set; } | 带名称的实体接口 |
| NamedModel | public int Id { get; set; }  public string Name { get; set; } | 带ID和名称Name的实体基类 |
| CUModel | 参见ICUModel | 直接实现ICUModel的实体基类 |
| CUCodedModel | CUCodedModel : CUModel, ICodedModel | CUModel的子类，带修改时间以及用户信息和Code属性基类 |
| CUNameCodedModel |  | 带修改时间及修改用户信息、Code和名称属性的实体基类 |

数据实体类的辅助接口和基类(命名空间：Jurassic.CommonModels.EntityBase)

|  |  |  |
| --- | --- | --- |
| 接口(基类)名称 | 代码 | 说明 |
| IIdEntity | 直接继承IId<int> | 只标记它是一个数据实体而不是业务实体的接口 |
| ICUEntity | int CreaterId { get; set; }  DateTime CreateTime { get; set; }  int UpdaterId { get; set; }  DateTime UpdateTime { get; set; } | 标记了创建和修改时间和用户ID的数据实体接口 |
| ICUDEntity | ICUDEntity:ICUEntity, ICanLogicalDeleteEntity | 标记了创建和修改时间和用户ID的数据实体接口，并支持逻辑删除 |
| CUDCodedEntity |  | 除上述外，还带Code属性 |
| INamedEntity | string Name { get; set; } | 带名称的数据实体接口 |
| ICodedEntity |  | 带ID和Code的数据实体基类 |

###### 定义模型之间的转换规则并依赖注入

如果使用了AutoMapper, 则这些转换都是自动进行的。在默认情况下无需进行任何配置。

因为只要是使用本组件，在找不到转换类的情况下，组件会使用Mapper.CreateMap()自行创建一个默认的转换规则。

有特殊要求除外，此时可以参考AutoMapper的相关文档，参考：

<https://github.com/AutoMapper/AutoMapper/wiki/Queryable-Extensions>

当然我们也可以用框架提供的“笨”办法，来实现完全可控的转换。

框架中使用IModelEntityConverter<TModel, TEntity>接口来定义业务模型和数据模型之间的互转。

class SupplierConverter : IModelEntityConverter<SupplierModel, Supplier>

{

public Expression<Func<SupplierModel, Supplier>> ModelToEntity

{

get

{

return s => new Supplier

{

Id = s.Id,

CreaterId = s.CreaterId,

Name = s.Name,

Address = s.Address,

Email = s.Email,

IsDeleted = false,

Tel = s.Tel,

Level = s.Level,

};

}

}

public Expression<Func<Supplier, SupplierModel>> EntityToModel

{

get

{

return s => new SupplierModel

{

Id = s.Id,

CreaterId = s.CreaterId,

CreaterName = s.Creater == null ? null : s.Creater.UserName,

Name = s.Name,

Address = s.Address,

Email = s.Email,

Tel = s.Tel,

Level = s.Level,

};

}

}

}

class ProductConveter : IModelEntityConverter<ProductModel, Product>

{

public Expression<Func<ProductModel, Product>> ModelToEntity

{

get

{

return p => new Product

{

Id = p.Id,

MasterId = p.MasterId,

Name = p.Name,

Price = p.Price,

Unit = p.Unit

};

}

}

public Expression<Func<Product, ProductModel>> EntityToModel

{

get

{

return p => new ProductModel

{

Id = p.Id,

Unit = p.Unit,

Price = p.Price,

Name = p.Name,

MasterId = p.MasterId

};

}

}

}

这部分代码比较冗长和枯燥。所以如果数据实体和业务实体极为相似，推荐还是用AutoMapper。

定义完转换规则以后，需要在Global.asax.cs中AddBindings()方法中，依赖注入以上的实现：

SiteManager.Kernel.Bind<IModelEntityConverter<SupplierModel, Supplier>>().To<SupplierConverter>();

SiteManager.Kernel.Bind<IModelEntityConverter<ProductModel, Product>>().To<ProductConveter>();

需要注意的是，如果按以上方法自定义转换规则，则AutoMapper会自动调用这些规则，而所有用Mapper.CreateMap()创建的规则会失效。

###### 添加控制器和视图

最简单的控制器只需要如下写法：

using Jurassic.WebQuery;

namespace Jurassic.WebTemplate.Controllers

{

public class AdvDataMasterDetailsDemoController : AdvDataController<SupplierModel, Supplier>

{

}

}

从以上代码看出，继承自组件的控制器在不需要作额外扩展的情况下是不需要写任何方法的。

但是为了在VS中在控制器和视图之间来回切换，推荐如下写法：

using Jurassic.WebQuery;

namespace Jurassic.WebTemplate.Controllers

{

public class AdvDataMasterDetailsDemoController : AdvDataController<SupplierModel, Supplier>

{

public override ActionResult Index()

{

return base.Index();

}

public override ActionResult Edit(int id = 0)

{

return base.Edit(id);

}

}

}

这样，我们就可以直接右击Index或Edit方法，直接创建或转到相应的Index(列表页）或Edit(编辑页）视图。

分别右击Index和Edit添加对应的视图，视图中代码如下：

Index.cshtml:

@using Jurassic.WebQuery;

@{

ViewBag.Title = "Index";

}

@Html.AdvDataGrid()

<script type="text/javascript">

mini.parse();

</script>

Edit.cshtml:

@using Jurassic.WebQuery;

@Html.AdvDataEdit()

<script type="text/javascript">

mini.parse();

</script>

从以上代码可以看出，虽然在模型定义和转换上比常规开发方法额外多花了点精力，但在最浪费开发者时间的前端代码编写和后端数据保存的方法上得到了极大的简化。

而业务模型和数据模型的建立本来就是要开发者多花精力设计的地方。

###### 配置菜单按钮

列表页上方的操作按钮，和编辑页上的操作按钮，都可以在框架的功能管理页面里面配置。除了标准的增删改方法，还可以配置其他非标准操作的按钮，标准操作不需要额外写事件绑定代码。只需要在配置按钮时，按钮的ID命名前缀为DefaultXXX即可。

比如：



系统支持如下标准的按钮前缀（区分大小写）：

|  |  |  |  |
| --- | --- | --- | --- |
| **前缀** | **对应控制器方法** | **GET/POST** | **功能** |
| DefaultAdd | - |  | 进入新增界面 |
| DefaultEdit | - |  | 进入编辑界面 |
| DefaultDelete | Delete | POST | 删除 |
| DefaultReturn | 无 | 无 | 从编辑页返回列表页 |
| DefaultSave | Edit | POST | 保存 |
| DefaultCopy | Copy | POST | 在编辑页，复制一条新单据 |

###### 配置编辑页的子项Grid菜单按钮

在编辑页的下方，有一排Tab页，每个Tab代表主对象中的一对多关系的子对象显示和编辑区。在这个编辑区的GRID上方也有自己的操作工具栏按钮。如下图所示:



（1）在左侧树菜单结点中，在编辑页下方，和编辑页按钮同级新增一个结点，作为第一个Tab页的根结点，ID和名称任意。唯一要注意的是，右侧的“快速访问”需要打勾，而“按钮”选项则不能打勾，否则无从判断这是一个子项Grid的操作按钮



(2)在下面再新增一个子结点，比如上图的“DefaultDelete-attachment”

ID的命名方式与上面第4条中的命名规则相同。如果是默认行为按钮，要加上Default作前缀。保存后再进入原来的编辑页，就得到了子项操作的一系列按钮。

子项的默认操作有

DefaultAdd-xxxx: 代表新增一行的操作，点击此按钮后，会在grid中新增一行，然后用户手动输入数据。

DefaultDelete :代表删除一行的操作，会将打勾选中的行都删除。

注意子项操作都只限于客户端的操作，对服务端的数据不会有影响。最后点击主工具栏的“保存”，修改才会生效。

做完以上工作以后，就可以直接启动运行了。

配置菜单按钮工作比较枯燥，因此，只要配置了初始的Index页面对应的菜单，在其他菜单缺失的情况下，系统将默认为增删改查操作添加默认的操作按钮，然后用户在运行一次该页以后，将看到默认缺省的按钮。然后可以回到功能管理的界面，对默认的按钮进行编辑修改，或增加新的按钮。（非默认的按钮要写自定义的处理逻辑）。

###### 界面微调

界面微调的目的有两个，一个是调整表单中元素的显示先后顺序，二是调整表格列的宽度，二是调整表单中每个数据控件的大小。

这两个工作，都可以直接交由配置文件完成。

配置文件在解决方案的App\_Data目录下的AdvDataConfigs.json.（App\_Data目录主要是存框架运行所需的配置文件，比如菜单的配置文件）。

它的结构是这样的：

[

{

"ClassName": "SupplierModel",

"Items": [

{

"OverWrite": true,

"DataType": "Auto",

"DataSourceType": "None",

"DataSource": "",

"MaxLength": 0,

"MinLength": 0,

"AllowNull": true,

"MaxValue": "",

"MinValue": "",

"PropertyName": "Text",

"FormOrder": 1,

"GridOrder": 1,

"Rows": 1,

"Cols": 1,

"ColumnWidth": 0,

"Fixed": false

},

{

"OverWrite": true,

"DataType": "SingleLineText",

"DataSourceType": "None",

"DataSource": "",

"MaxLength": 0,

"MinLength": 0,

"AllowNull": false,

"MaxValue": "",

"MinValue": "",

"PropertyName": "Id",

"FormOrder": 1,

"GridOrder": 1,

"Rows": 1,

"Cols": 1,

"ColumnWidth": 0,

"Fixed": false },

// ……………

]

ClassName是对应的业务实体类名。

Items代表下面的属性配置集合

其中，有些属性和在实体类属性上的标签CatalogExt(…)中定义的属性名相同，系统运行时决定采用哪一个（是实体属性上的标签还是这个配置）取决于Overwirte属性的值，为true时表示用配置文件中的值替代标签值。为false则忽略配置文件中的相同属性值。

另外还有一些补充的属性如下：

PropertyName:是业务实体下某个属性的名称。

FormOrder: 在详细界面表单下的顺序

GridOrder: 在列表界面中的列顺序

Cols: 该属性生成录入控件时，一个控件（包括提示文字和录入框）占几个单元的宽度

1是默认值。代表占一个单元的宽度。2代表占两个单元的宽度。

表单中一行分成几个单元是根据屏幕宽度自适应的。如果是大屏幕，一行可以分成4个单元。

中等屏幕占3个，小屏幕占2个，超小屏幕一行只能有1个单元。

所以，无论你在Cols里填几，在手机上始终一行只能有一个控件。

如果你在Cols里填2， 则代表，超小屏幕和小屏幕上，它是占满整行的。在中等屏幕上，它占一行的2/3宽度，在大屏幕上，它占1/2宽度。

以下表格是Cols=1,2,3,4时，对布局的影响：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Cols=1 | Cols=2 | Cols=3 | Cols=4 |
| 超小屏幕(xs) | |  | | --- | |  | | |  | | --- | |  | | |  | | --- | |  | | |  | | --- | |  | |
| 小屏幕(sm) | |  |  | | --- | --- | |  |  | | |  |  | | --- | --- | |  |  | | |  |  | | --- | --- | |  |  | | |  |  | | --- | --- | |  |  | |
| 中等屏幕(md) | |  |  |  | | --- | --- | --- | |  |  |  | | |  |  |  | | --- | --- | --- | |  |  |  | | |  |  |  | | --- | --- | --- | |  |  |  | | |  |  |  | | --- | --- | --- | |  |  |  | |
| 大屏幕(lg) | |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | | |  |  |  |  | | --- | --- | --- | --- | |  |  |  |  | |

Rows代表控件占几行的高度。这通常用于设置多行文本框或HTML编辑器。

ColumnWidth: 在DataGrid中显示时，数据列的宽度，默认是100.

Fixed: 是否在DataGrid中冻结该列，在横向拖动时，位置不变。

###### 多语言支持

本组件基于框架的多语言机制，在运行中会动态生成表格和表单文本元素的Key。然后可以根据这些生成的Key在语言管理中进行编辑，翻译成目标语言的文字。

Key的命名规则如下：

|  |  |  |
| --- | --- | --- |
| 界面元素 | 命名前缀 | 示例 |
| 主/从表中的列名 | Column.类名.属性名 | Column.Product.Price |
| 表单中的Label | Label.类名.属性名 | Label.Product.Price |
| 表单中空白提示文字 | Empty.类名.属性名 | Empty.Person.CompanyName |
| 明细表的选项卡文字 | Details.类名.属性名 | Details.Deploy.Attachments |
|  |  |  |

这其中有许多可能都是相同的。但不排除有些场景下代表同一对象的提示文字不同的情况。

如果空白提示文本觉得不需要，可以在语言管理界面中对应语言的文本填上“-”减号，表示留空。

为了减轻维护各类前缀Key的麻烦，对于Column.,Label.,Empty三种前缀的Key, 如果后面的类名属性名都相同，对应的文本也相同，可以只用维护一个Column.开头的Key对应文字就行了。其他的可以留空，当留空时，默认取Column.开头的Key对应的值。

#### 如何扩展

##### 视图界面上的扩展

在上面讲到CatalogExt()的配置属性时提到， DataSourceType属性实际上决定了控件的显示方式。 其中DataSourceType=ExtDataSourceType.Custom的情况下，开发者可以自行决定控件的显示方式。 比如：

@using Jurassic.WebQuery;

@Html.AdvDataEdit(CustomEmailControl)

@helper CustomEmailControl(string propName, object value)

{

if (propName == "Email")

{

<label>电子邮件：</label>

@Html.TextBox("Email", value, new{ vtype="email", @class="mini-textbox"})

}

}

<script type="text/javascript">

mini.parse();

</script>

在以上代码中，就自定义了Email的显示方式。

但这样的方法，极有可能会破坏整体布局。

基于mini-ui的界面元素Id有一套固定的命名规则。这可以从生成的HTML代码查出来。

可以基于这个命名过的id获取对应的miniui对象，从而调用miniui的API对其进行自定义操作。

具体的命名规则如下（区分大小写）：

|  |  |  |
| --- | --- | --- |
| 界面UI元素 | Id | name |
| mini-datagrid 主表 | grid\_类名 |  |
| mini-datagrid 从表 | detailsgrid\_集合名 |  |
| 编辑界面表单元素 | master\_属性名 | 属性名 |
| 主表/从表的列名name |  | 属性名 |

##### 控制器上的扩展

在控制器端，可以重写BeforeShowing、BeforeSaving、AfterSaved方法来实现对数据显示前、保存前和保存后作一些操作。

比如：

/// <summary>

/// 此处重写检测员工所获荣誉数量，如果一个也没有，就给一个优秀员工,

/// 如果超过三个，则不让保存并返回错误。

/// </summary>

/// <param name="t"></param>

/// <returns></returns>

protected override bool BeforeSaving(PersonModel t)

{

if (t.Honors.Count == 0)

{

//如果一个也没有，则添加一个默认的

t.Honors.Add(new HonorModel

{

GetDate = DateTime.Now,

HonorName = "优秀员工",

});

}

if (t.Honors.Count > 3)

{

SetTips("error", "Honor数量太多", null);

//终止下步操作，返回错误

return false;

}

return true;

}

以上例子，大家可以下载svn/项目管理规范/12项目模板/AddinDemo.

关于数据查询和采集组件的介绍，就先到这里。

## 关于Model, 推荐的做法

MVC只是一个表现层的框架。无论是Controller,还是View都是为表现层服务的。在MVC中对Controller和View都有规定的写法，而体现业务逻辑的Model层MVC本身并没有给出一个类似的规范。但在本框架中，推荐这么做：

1. “瘦”的Controller, “胖“的Model。

Contoller是控制器，我们理解为它负责验证和接收界面的输入，并交给业务逻辑处理，并将处理结果以合适的格式推送给浏览器。所以，它的职责只是”上传下达“， 而不要写成一个臃肿的Controller, 把业务逻辑等也通通写在里面。

Controller只是MVC这个特定表现层框架的类型，而我们要实现的业务逻辑，从本质上说，是和采用哪种技术框架一点关系都没有的。所以在业务逻辑的编写上，要体现出和具体UI无关的特点。让它在任何场合都能发挥作用。

所以，一定要把业务逻辑从Controller中独立开来。

1. 将业务逻辑进一步细分为”业务操作“和”数据操作“

”纯“业务逻辑和具体的数据持久化操作，两者需要解耦，各自不依赖于对方的具体实现，使用接口进行通信。

关于数据访问类， 在Jurassic.AppCenter.Core项目中，已经定义了数据访问操作的基本接口IAuditDataService,和两个默认实现，一个是基于EF的EFAuditDataService<T>，一个是基于ADO的AdoAuditDataService<T>。开发者可以直接使用这两个泛型接口实现，很容易实现基本的数据访问操作。也可在此基础上派生。也可单独实现IAuditDataService接口。

1. 不要试图去catch异常

因为框架已经对异常做了完善的处理并记录在日志中。这样即可以减少你的代码量, 又能使异常处理无遗漏。而不要每个方法都来一个大段的try…catch。这样的代码即不美观，又难以对异常作完整的记录，更难以在调试状态纠错。

特别不要试图用try…catch来隐藏代码中的逻辑错误。

当然有一种例外，就是你明确知道哪个地方会出现外部异常，比如文件没找到，网络通信不能连接等， 并且有意识地有替代方案去回避这个错误（比如重试几次）而不影响整体逻辑。

1. 从控制权限的角度去规划Contoller和Controller中的每个Action。

本框架的权限控制核心是基于Contoller的Action的执行过程来实现的。所以在编写控制器代码时，要注意，自己提供给前台调用的Action是否是实际要控制权限的粒度。判断标准就是这个Action方法的签名（包括而且仅限于 控制器类名、方法名以及方法参数，不包括返回值）。这个粒度可以小到点击按钮时的处理操作。并且层次要合理，尽量把相关的功能操作放一起。

## Web框架对多语言的支持

### 界面文字多语言支持

所有的界面需要多语言的文本信息都对应一个Key.

在程序层面，只需要考虑什么地方需要显示什么Key对应的文字 ，在界面层HTML中写：

@Html.Str(“Key”)或 @ResHelper.GetStr(“Key”)这种语法来代表固定的语言文字。

框架会自动将参数Key值添加到资源文件（如果没有的话）。

然后，系统维护人员可以打开系统管理-》资源管理界面，去维护这个Key所对应的语言信息。

当然，前提是引用过这个Key的页面或代码至少已经运行过一次。

### 数据内容多语言支持

前述在介绍Jurassic.WebQuery数据组件时，如果要在数据层面上实现框架级的多语言支持，在特定的实体属性上打标记和实现指定接口。这里再复述一次：

1. 需要在特定业务实体类的需要多语言支持的属性上打如下标记：

public partial class PersonModel : IId<int>

{

//….

[CatalogExt(DataType = ExtDataType.MultiLanguage)]

public string Name { get; set; }

//…

}

1. 需要对应的数据实体实现接口IMultiLanguage:

public partial class Person : IMultiLanguage, IId<int>, ICanLogicalDeleteEntity

{

//…其他属性

/// <summary>

/// 该对象中所有多语言的属性文本集合

/// </summary>

[ForeignKey("BillId")]

public ICollection<Sys\_DataLanguage> LangTexts { get; set; }

}

1. 实现以上两点以后，如果使用数据组件Jurassic.CommonModels.ModelBase.ModelDataService<TModel, TEntity>这个数据服务，则该服务会自动处理多语言所需要的数据访问。该组件实现了通用的TModel和Tentity之间的相互转换和持久化。
2. 如果没有使用上述数据服务，则需要调用LangDataMapper<TModel,TEntity>这个泛型类来实现多语言数据的访问。
3. 如果使用EF的业务实体里有外键指向其他多语言实体类，则该多语言实体类需要满足上述规则，并手动写转换。

这里举个例子：

问题：

部门组织

Public class Department :IMultiLanguage

{  
public int Id {get;set;}  
public string Name {get;set;}

….

}

1.需要在普通頁面的编辑页显示部门名称中英文，如何配  
2.需要在普通页面的列表页显示部门名称中英文

首先需要业务实体里有一个表示部门名称的属性

比如：

//数据实体

Public class XXX

{

Public int Id{get;set;}

….

Public int DeptId{get;set;}

Public Department Department{get;set;}

}

//业务实体

Public class XXXModel

{

Public int Id{get;set;}

….

Public int DeptId{get;set;}

Public string DeptName {get;set;}

}

然后可以用两个方案：第一个是沿用以前的转换类，里写

return new XXXModel  
{  
   ...  
   DeptName = s.Department.LangTexts.FirstOrDefault(l=>l.Name == "Name" && l.Language == ResHelper.CurrentCultureName && l.BillType ==”Department” ).Text  
}

s.Department是数据实体里的部门对象

因为部门对象是多语言支持的，所有它要继承IMultiLanguage接口, 有可查询的LangTexts集合属性  
ResHelper.CurrentCultureName是框架里定义的当前语言简称

第二个方案是用AutoMapper,就是不写转换类，需要特殊映射的地方写automapper转换：

Mapper.CreateMap<XXX, XXXModel>()  
                .ForMember(s => s.DeptName,   
                opt => opt.MapFrom(s => s.Department.LangTexts.FirstOrDefault(l=>l.Name == "Name" && l.Language == ResHelper.CurrentCultureName && l.BillType == “Department”).Text));

## Web框架对模块化开发的支持

WebFrame框架支持各个模块以单独的应用程序形式进行开发，并可以独立运行，也可以作为单独的模块在一个更大的应用程序中运行。

WebFrame本身，以及一些可重用组件，比如WebUpload、WebSchedule都是以模块化的方式开发的。

开发一个独立模块的方法与开发一个单独的应用程序方式几乎完全相同，只是在做为一个单独的模块放在一个更大的应用程序中时，需要做以下处理：

### 初始化

如果模块有初始化动作，并且原来是写在global.asax.cs中的，但是模块放在其他程序中时，该文件不会起作用，所以需要将该文件中的初始化代码放在另一个类中：

using Owin;

using Jurassic.WebFrame;

public class StartupConfig : IStartupConfig

{

public void Config(IAppBuilder app)

{

//模块初始化代码

}

}

在应用程序启动时，会自动寻找这个类并执行Config方法。

在这个方法执行时，应用程序的Global.asax.cs中的Application\_Start()已经执行完毕，所有在Application\_Start()中初始化过的对象都可以在此方法中使用。

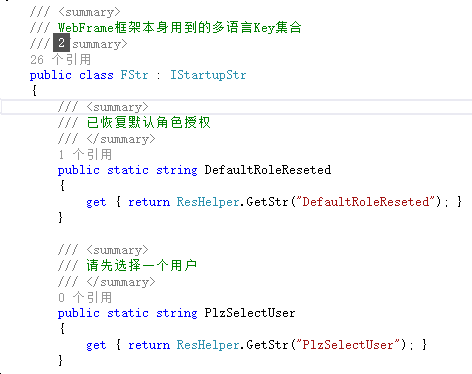
还有一种方式就是利用MVC注册域的方法来初始化，即继承AreaRegistration

这个类，并重写其中的RegisterArea方法。

多语言包的初始化：

在WebFrame 3.0以后，新增了IStartupStr接口，实现这个接口的类中的静态属性，将会被自动初始化为多语言资源文件中的一个Key:

如，在Jurassic.WebFrame框架中，实现以下类：



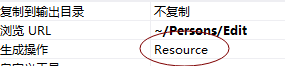
[使用这种方法初始化的好处是可以减少以往在页面中滥用@Html.Str(Key)](mailto:使用这种方法初始化的好处是可以减少以往在页面中滥用@Html.Str(Key))造成的多语言Key不规范和重复，并且在需要使用时，在代码中有智能提示。

### 打包所有资源

首先分清哪些是框架本身有的资源，比如常用的jquery,在webframe中已经有，就不需要再打包。只打包项目中那些独特的资源。

包括脚本文件、css文件, App\_Data中的语言文件、cshtml视图等。\*.cs源代码不要打包。

打包的方式是：单击要打包的以上资源文件名称，按F4调出属性面板，【生成操作】选择Resource。



然后重新编译即可。

### 在主程序中注册

如果模块的名称**不是**以Jurassic.Web\*.dll的格式命名的，那么在承载该模块的的主项目中global.asax.cs中，需要重写以下方法：

protected override IEnumerable<string> ControllerNameSpaces

{

get

{

var list = base.ControllerNameSpaces.ToList();

list.Insert(0, "AddinDemo"); //这里写不带.DLL后缀的模块名称

return list;

}

}

将模块编译过的DLL及其依赖项COPY到应用程序的bin目录。如果不刻意强调松藕合，也可以直接在主程序中添加对这个模块的引用。

### 配置菜单

运行主程序，同时在功能管理界面中配置该模块的菜单和权限。（如果对AppFunctions.json文件熟，也可以直接把模块中的相同文件内容和主程序合并）。

## 常见问题及解答 (FAQ)

Q: 为什么在日志管理中看不到最新的日志

A: Log4net的日志是异步记录的，有一个缓存区。可以在web.config中JLogDbAppender这个配置节中配置 :

<bufferSize value="100" />

将数值改小一点。

Q: 我不想用框架规定的导航条-》工具栏这样的典型页面，应该怎样写View?

A: 在页面开头声明使用如下布局页：

@{

Layout = "~/Views/Shared/\_Layout.cshtml";

}

它只包含基本的js和css引用和body标记。没有其他的控件。

为了保持整体框架风格的一致性，建议不要完全抛弃基本的\_Layout.cshtml自己另外搞一套。

Q：如果我的按钮不在工具栏中定义，而是在页面其他地方手动写的标记，怎样和框架页的默认提交事件结合？

A: 按钮需要声明一个特殊的样式 frame-submit: 如下：

<a class=”mini-button frame-submit”>自写的按钮</a>

这样，你仍然可以这么写：

$.regButton({ text: ‘自写的按钮’, beforeSubmit: function(ajax){….} …..}

但是此时提交到的url默认情况下相同于页面的自提交操作。即提交地址和页面地址相同。除非你中途修改ajax.url

Q: 为什么我照例子写了一个控制器和相应页面，并且在功能配置界面配置了相应的按钮，但点击按钮报错“Get请求出错”

A: 检查控制器里相应的事件处理方法是否标记了[POST]方式 那样，在配置界面点击按钮提交的Method应该也配置成”POST“。

Q：为什么我修改了Web.config中的连接字符串，系统登录不了而且报错

A：是否修改了”DefaultConnection”这个Key？根据惯例，连接串的名称“DefaultConnection”是不能修改的，它指向的数据库类型和位置可以任意，但就是名称不能改。

Q：框架是否支持传统的WebForm和服务器控件?

A：同为asp.net大框架下的分支，你可以在站点加上传统的aspx页面和后台代码。但不建议这么做。

Q: 为什么我在功能管理里配置了页面和按钮，但进该页面显示不了按钮？

A: 首先要排除是否是权限问题， 其次页面所对应的ControllerName和ActionName必须填写而且必须有效。而不是仅仅只配置一个页面地址(Location), 这样框架才能正确识别当前页面所对应的菜单树节点配置信息。

Q: 为什么在MVC中在AreaRegistration中配置的路由无效？打开首页会显示 http404 not found?

A: 继承自AreaRegistration的写法有问题，应该是这么写：

public override void RegisterArea(AreaRegistrationContext context)

{

RouteTable.Routes.MapRoute({…});

}

而不是：

context.MapRoute({..});

Q：使用框架的网站，为什么启动时显示：

HTTP 错误 403.14 - Forbidden

Web 服务器被配置为不列出此目录的内容。

A： 可能是没有引用Owin程序集和Mircosoft.Owin.Host.SystemWeb程序集。所以在网站启动时没有执行路径映射。引用这两个程序集就好了。可以从nuget获取最新版本。

## 附录

### 在编写本框架的过程中，参考了以下网址：

[MVC 自定义AuthorizeAttribute实现权限管理](http://www.cnblogs.com/jyan/archive/2012/07/24/2606646.html)

[ASP.NET MVC实用技术：自定义AuthorizeAttribute与ActionLink的隐藏](http://www.cnblogs.com/daxnet/archive/2012/03/23/2413260.html)

[Understanding and Using Simple Membership Provider in ASP.NET MVC 4.0](http://www.codeproject.com/Articles/689801/Understanding-and-Using-Simple-Membership-Provider)

[Setting up the ASP.NET MembershipProvider](http://www.codeproject.com/Articles/607307/Setting-up-the-ASP-NET-MembershipProvider)

[ASP.NET MVC：多语言的三种技术处理策略](http://www.cnblogs.com/happyframework/p/3228979.html)

[在ASP.NET MVC中通过URL路由实现对多语言的支持](http://www.cnblogs.com/artech/archive/2012/05/04/localization-via-url-routing.html)

### MVC的学习网站

<http://www.asp.net/mvc/overview/older-versions/getting-started-with-aspnet-mvc4/intro-to-aspnet-mvc-4>

### Ninject 官网及开源代码

<http://ninject.org/>

<https://github.com/ninject/ninject>

### Mini-UI官网

官网: <http://www.miniui.com/>

示例：<http://www.miniui.com/demo/>

教程：<http://www.miniui.com/index.html#tutorial>

文档：<http://www.miniui.com/docs/>

API参考：<http://www.miniui.com/docs/api/index.html>l

# WinFrame框架

## 概述

Jurassic.WinFrame框架（以下简称框架）分为服务端（WCF）和客户端两个子框架，其中，客户端是基于微软官方的Smart Client Software Factory(SCSF)框架和Infragistics 14.2的前端组件构建, C/S架构的应用程序框架。它的特点是完全的插件式开发和高扩展性。服务端是基于WCF以及自定义身份验证和实例工厂。它可以将符合要求的任何独立类库发布为WCF服务。

### 开发目的

开发此框架的目的是为对未来公司各个基于Winform平台和Wcf服务的项目统一技术框架，并在此框架基础上建立统一的开发规范，减少各项目之间各自技术选型和框架不统一的情况。

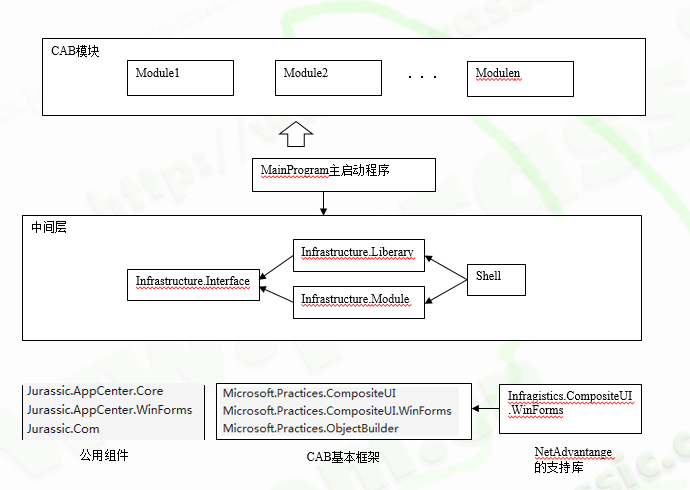
主要以下几点:

* 统一Winform平台的前后台框架；
* 在此框架基础上建立统一开发规范；
* 减小各项目在基础框架和组件实现上的差别；
* 新进员工快速培训，缩短适应期；
* 框架层和应用层分离，各自独立发展；
* 为各业务组件的抽提和复用提供公共的平台和技术支持；
* 框架和各组件本身作为公司技术资产，对提升生产力发挥持久的作用。

## 框架简介

### 框架结构图

客户端：



框架主体

公用组件

具体项目

服务端（WCF）：

服务宿主实例

WebApp for WCF

客户端

用户业务接口

Jurassic.AppCenter.Wcf

服务核心框架

接口实现

## 开发及运行环境

开发平台是vs2013, 需要安装infragistics 14.2的winform套件。其安装文件在武汉公司服务器：<ftp://192.168.1.135>上。其他第三方组件，依赖项均已nuget上下载并且已经集成到源代码管理中。

在部署到生产环境时，Wcf服务器端只需发布一个本地版本并拷贝到服务器相应目录即可。Web.config中的连接字符串根据服务器实际情况自行修改。服务器要求装有.net framework 4.0.

客户端根据情况，可以打包为绿色版本或制作一个完整的安装包。客户端打包后，就框架本身而言不需要客户端安装除了.net 4.0运行环境以外的其他软件。

## 各版本升级注意事项

### 从1.1或1.0升级到1.3

升级方法：

此版本升级100%兼容老版本框架，除了获取最新框架源码（对于引用源码项目）或最新DLL（对于引用DLL的项目），并做一下简单运行测试外，并不需要做任何额外操作。

1.3版本的主要改进是

* 1. **增强了创建窗体或侧边栏的API。**

在1.1或之前版本中，创建或激活窗体或侧边栏对应的API分别是：

CreateAndActiveChildForm<T>(id)

CreateAndActiveSideBar<T>(Id)

以上两个方法的泛型参数T必须是可实例化的具体窗体或控件类型, 这在实际应用中不太灵活。 基于面向接口编程的原则， 在1.3版本中, 新增一个API：

CreateAndActive<T>(id)

此方法的类型参数T将可以用接口来代替。框架在运行过程中会根据接口和实现类的关系，来选择合适的类进行实例化。并且自动判断UI类型，选择是显示为子窗体还是侧边栏。

注意：上一版本的API含义和用法保持不变。

为了配合新增的API功能实现，对于WorkItem对象，新增了它的扩展方法WorkItem.RegisterType<Tinstance, TregisterAs>()， 用于定义接口和实现类的关系。

* 1. **优化了以UserControl作为基类的MDI子窗体显示支持。**

在以前版本的框架中，UserControl也是可以单独以窗体形式在MDI界面中展示的。但是有Ribbon工具栏的用户界面限制为必须继承自Form基类，否则工具栏无法和主工具栏合并。

在此次修订中，取消了这一限制，主要体现在：

如果UserControl带有自己的Ribbon工具栏，并且实现了框架定义的IRibbonForm接口，那么该UserControl的Ribbon工具栏也将和主窗体Ribbon工具栏合并显示。并且不会影响到工具栏按钮的事件绑定逻辑。

* 1. **改写了对话框显示API：WorkItem.ShowDialog<T>(), 可以参过参数定义对话框起始位置。**

改写后的API多了两个可选参数：

/// <summary>

/// 显示指定类型控件的模态对话框

/// </summary>

/// <typeparam name="T"></typeparam>

/// <param name="workItem">当前的WorkItem</param>

/// <param name="startPosition">起始位置</param>

/// <param name="onCreated">生成时的委托</param>

/// <returns>对话框结果</returns>

public static DialogResult ShowDialog<T>(this WorkItem workItem, FormStartPosition startPosition = FormStartPosition.CenterScreen,

Action<T> onCreated = null) where T : class, IDialogView

* 1. **对于所有的WorkItem扩展出的API，如果有泛型<T>的版本，相应地提供了非泛型版本的API**

例如：CreateAndActivateSideBar<T>(this WorkItem workItem, string id)

增加了CreateAndActivateSideBar(this WorkItem workItem, Type t, string id)

* 1. **对于侧边栏的显示或隐藏，去掉了动画效果，加快缩进和展示的速度。**
  2. **在 项目模板中，添加了使用编译后的框架DLL的模板**

分别是WinTemplate\_Lite和WinTemplate\_Local\_Lite, 分别对应于网络版本和单机版本。使用此模板创建项目和使用源码的模板完全相同。但是项目编译和调试速度更快。

### 升级到WinFrame 2.0:

## 项目建立步骤

### 解决方案建立步骤

在SVN上有一模板文件夹在：

<http://221.232.134.250:8443/svn/项目管理规范/12-项目模板/WinTemplate>

1. 建立和“项目管理规范”同级的项目文件夹 比如 “XX系统”
2. 将WinTemplate文件夹整体COPY到 “XX系统/系统开发”文件夹下面，再重新命名相关文件。
3. 将WinTemplate文件夹重命名为实际解决方案名称。
4. 打开解决方案文件夹，将\_WinTemplate\_.sln重命名为 解决方案名称.sln
5. 重新命名解决方案中的项目文件夹\_WinTemplate\_为实际项目名称。
6. 打开Global.asax，调整其中的名称和命名空间。
7. 直接可编译运行。
8. 用户名密码是pmis 888888

### 数据库安装：

Wcf服务器端数据库安装：

与WebFrame框架的数据库完全一致。

本地数据库：

本地数据库推荐使用Sqlite。在基础框架中已经包含对Sqlite的ado.net和EF实体的支持。请参见DBHelper的相关文档。

## 公共模块的配置和运行

在项目正常建立以后，系统自带的“系统管理”功能组已经可以使用。它包括：

|  |  |
| --- | --- |
| 功能管理 | 查看并管理系统的功能  模块、页面和按钮的树型结构 |
| 角色管理 | 管理系统的角色，对角色进行功能模块的权限分配 |
| 用户管理 | 管理系统用户并授予角色权限 |
| 语言管理 | 进行多语言资源的管理 |
| 日志管理 | 查看并管理系统日志 |

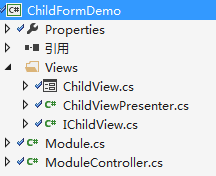
关于这些系统自带功能的使用，请参照文档《应用程序组件技术说明》。

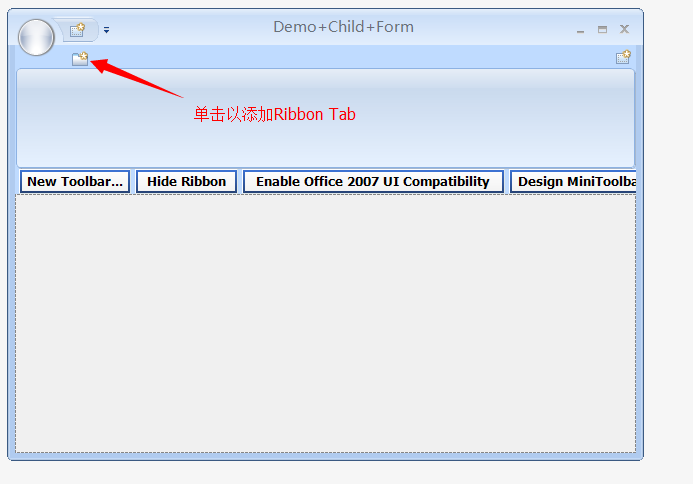
## 客户端模块编写方法

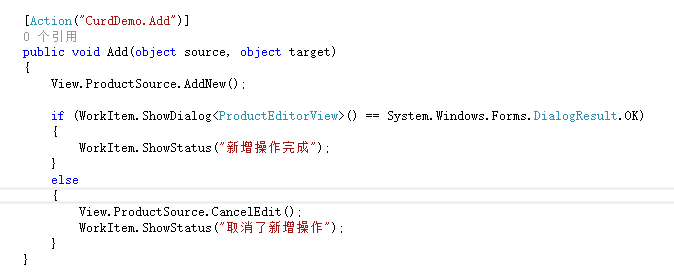
所有客户端模块都是基于CAB标准和MVP设计模式的插件式开发方式，下面分几种模板来分别描述客户端模块的编写方法。

### 一般性窗体

1. 根据标准的CAB模板，一般性窗体的ChildForm\_Template\_， 复制项目文件夹，再重命名文件夹和项目文件，形成一个新的CAB项目。
2. 在解决方案资源管理器中右击添加现有项目，将刚才复制并重命名过的项目添加到整体解决方案中。
3. 一个标准的CAB项目结构如图所示：



1. 将已有的ChildView、IChildView、IChildViewPresenter重命名为实际名称，包括相应的命名空间。
2. 直接打开改名后的ChildView.应该出现Form设计器，应该如图所示：  
   
3. 在Ribbon中用设计器添加Tab、Group和ToolBar。它是和后台要执行的Action的标签直接关联。
4. 在窗体的主要工作区摆放业务相关的控件和布局。
5. 为每个ToolBar设置图标、文字和Key。其中Key非常重要，在写和这些按钮绑定的事件代码时，需要用[Action(Key)]来标注。比如：

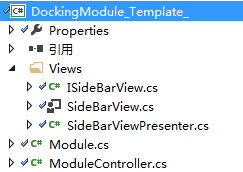


上图中“CurdDemo.Add”就是Ribbon菜单中”添加”按钮的Key.

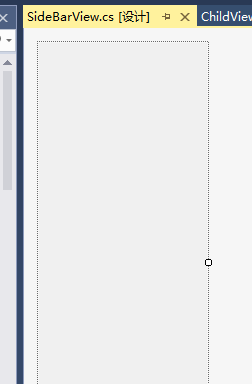
1. 写相关代码，注意以下原则：
2. 在View中写和界面的具体控件耦合的代码。
3. 在Presenter中写和具体控件无关，但是和Iview接口相关的界面逻辑代码以及和Ribbon中的按钮事件绑定的Action动作代码。
4. 相关的业务服务对象，或WCF的代理服务对象，用依赖注入的方法写在Presenter的构造函数中。用[ServiceDependency]标注。
5. 在ModuleController中写打开View（以及和本View相关的UI组件，如侧边栏）的代码。

### 可停靠面板

1. 根据标准的CAB模板，可停靠面板的DockingModule\_Template\_， 复制项目文件夹，再重命名文件夹和项目文件，形成一个新的CAB项目。
2. 在解决方案资源管理器中右击添加现有项目，将刚才复制并重命名过的项目添加到整体解决方案中。
3. 一个标准的可停靠面板项目结构如图所示：



从中可以看出，它和一般性窗体的项目结构大体一致。

1. 将已有的SideBarView、ISideBarView、ISideBarViewPresenter重命名为实际名称，包括相应的命名空间。
2. 直接打开改名后的SideBarView.应该出现UserControl设计器，如图所示：  
   

可以看出，它基本上是一块白板。你可以在上面根据项目需求自由发挥，设计面板界面。

对于可停靠面板的编程规范，也和窗体大体一致。在此就不重复了。

需要强调的是，可停靠面板一般和窗体配合工作，所以它大多数情况下必须先于窗体加载，而且一般也不和窗体处于同一模块。那么在插件式环境下，保证可停靠面板所在模块预加载的方法是在主启动程序所在的ProfileCatalog.xml中进行配置 :如下图

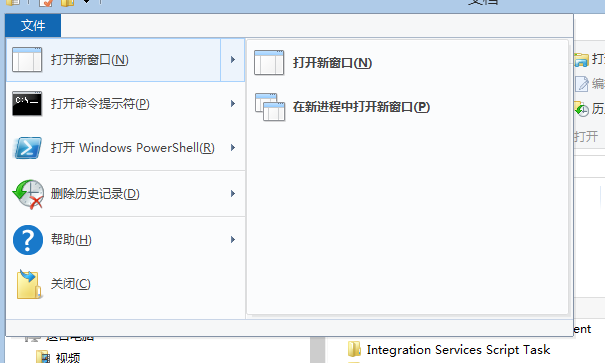


可停靠面板和窗体通信，推荐使用事件发布（EventPublication）和订阅（EventSubscription）的编程模型。具体可见公用 组件库/dotnet/AppCenter/WinFrame.sln示例项目。

### 开始菜单中的自定义控件

当我们打开Word或其他Office类程序，或者最新的Windows8文件窗口，总有一个“文件”之类的开始菜单，点开以后会出现一个特殊的“开始”屏幕，如下图所示：

Windows 8:



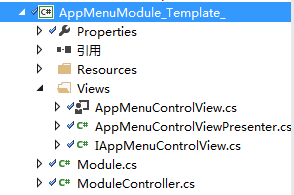
Word2013:



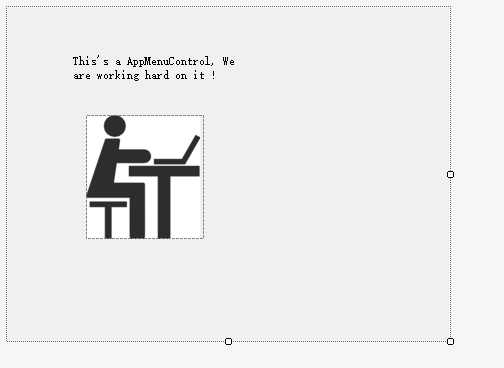
本框架对于这种“开始”菜单的编程也提供了充分的支持。

下面是大致的开发和配置步骤：

1. 根据标准的CAB模板，可自定义菜单容器控件的AppMenuModule\_Template\_， 复制项目文件夹，再重命名文件夹和项目文件，形成一个新的CAB项目。
2. 在解决方案资源管理器中右击添加现有项目，将刚才复制并重命名过的项目添加到整体解决方案中。
3. 一个标准的自定义菜单容器控件项目结构如图所示：



从中可以看出，它和一般性窗体的项目结构大体一致。

1. 将已有的AppMenuControlView、IAppMenuControlView、IAppMenuControlViewPresenter重命名为实际名称，包括相应的命名空间。
2. 直接打开改名后的AppMenuControlView.应该出现UserControl设计器，如图所示：  
   

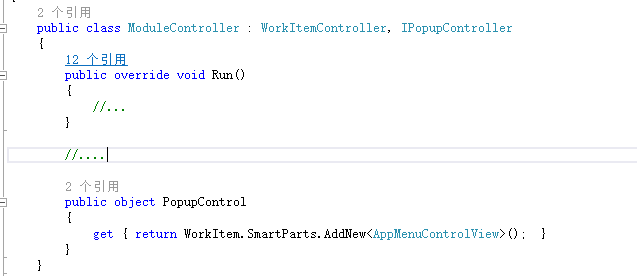
可以看出，它上面象征性地摆放了一个Label和一个PictureBox。你可以删除它们，在上面根据项目需求自由发挥，设计控件界面。

对于菜单容器控件的编程规范，也和窗体大体一致。在此就不重复了。

设计好的界面如何摆放到实际的窗体中？

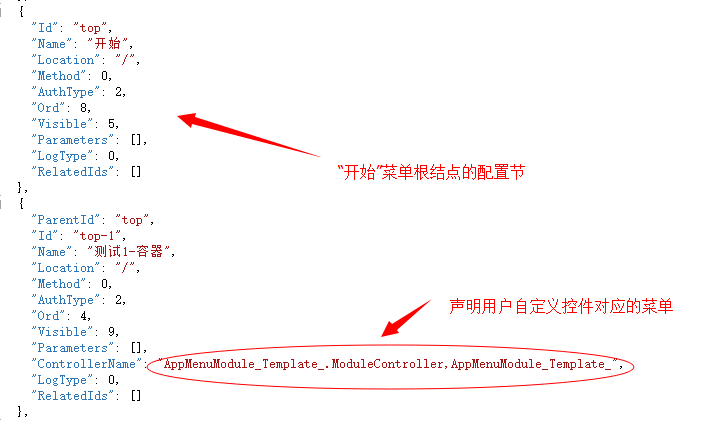
这就需要借助于配置和ModuleController中的接口实现。

首先看ModuleController的写法：



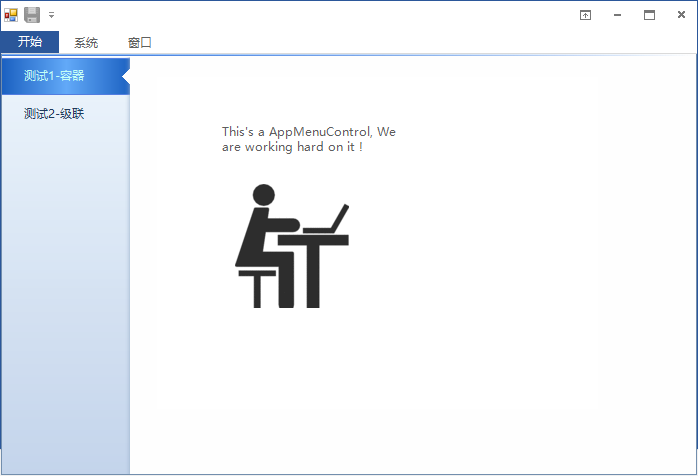
从中看出，自定义菜单容器控件所在的ModuleController类需要实现IPopupController接口，该接口用于返回实例化后的控件对象。

其次，需要在AppFunctions.json这个系统功能模块配置文件中，写如下配置:



注意：AppFunctions.json文件根据运行的架构，如果是C/S架构，则是在服务端的App\_Data文件夹，如果是单机版，则是在本地的App\_Data文件夹。

做好以上功课后，运行就会得到自定义控件在右侧容器区显示：



## 本框架的配置文件使用。

保证本框架能正常运行的关键配置点有以下几个：

1. **App.config**

App.config是应用程序本身的配置文件。它在编译后会形成和可运行程序名称.config文件。在App.config中存有系统自身运行所需的基础信息。

作为开发人员需要特别关注的配置节是<CompositeUI>

比如：

<CompositeUI>

<!--<visualizer>

<add type="SampleVisualizations.WorkItemVisualization, SampleVisualizations" />

</visualizer>-->

<!--在此配置节可以配置一些SCSF的服务-->

<services>

<add

instanceType="WCFLoginModule\_Template\_.StartupService, WCFLoginModule\_Template\_"

serviceType="Jurassic.AppCenter.SmartClient.Infrastructure.Interface.Services.IStartupService, Jurassic.AppCenter.SmartClient.Infrastructure.Interface" />

</services>

</CompositeUI>

在<services>中可以配置在系统初始化时需要加载的模块和服务。注意此时，用户尚未登录取得合法身份。所以不要在写此类服务时使用有关用户相关上下文的信息。

**2．ProfileCatalog.xml**

ProfileCatalog.xml是SCSF框架默认用来保存系统需要加载的插件式模块信息的配置文件。它的读取并加载是在用户身份验证通过以后，所以在此配置文件中注册的模块可以使用相关用户上下文。

为了保证系统的启动加载速度，建议不需要把系统用到的所有模块都写在此文件中，只写那些系统运行所需的一些公共模块。

下面是ProfileCatalog.xml的示例：



**3．AppFunctions.json**

AppFunctions.json是所有不同架构的框架（不管是C/S还是B/S，不论客户端还是WCF服务）所通用的描述系统整个模块/页面/功能按钮的综合配置文件。

在本框架中，它主要描述各插件式模块的信息。

包括类名、程序集名、可见性，规则约束等。

在此配置文件中注册的模块是随系统的菜单操作而动态加载。这点和ProfileCatalog.xml中注册的不同（ProfileCatalog.xml是一次性全部加载）。

本框架目前的配置点比较多，未来的版本中，有可能会统一到一个配置（**AppFunctions.json）**中。

## WCF服务的编写方法

本框架的WCF服务框架内置了用户身份验证、日志、异常处理模块，在一般情况下，开发者只需要按WCF的规定编写自己的业务接口和接口的实现，和编写普通的类没有任何区别。然后在配置文件中进行服务的注册，就可以将业务接口作为WCF服务发布了。

下面简要说明步骤：

1. 服务的协议

服务协议:接口的名称用[ServiceContract]特性来标识，接口的具体操作用[OperationContract]来标识，如下图：

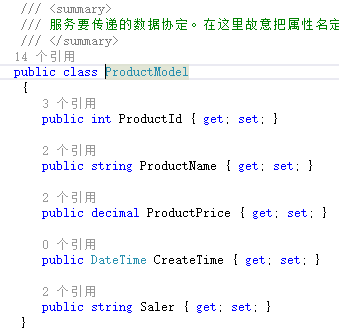


数据协议:

数据协议可以用普通的POCO实体类定义，请保证它们的public属性都是可序列化的。

注意：这里省去了[DataContract]之类的标记工作。因为服务端和客户端都将引用相同的接口定义所在类库，默认情况下是可行的。如果有特殊要求，还是要按WCF的规则来定义【DataContract】和【DataMebmer】.

示例：



1. 服务的具体实现

服务的具体实现，就是上述服务协议所规定的接口的一个具体实现。它的写法和一般的类并无不同。

**注意：服务必须继承自Jurassic.AppCenter.Wcf.WcfBaseService**

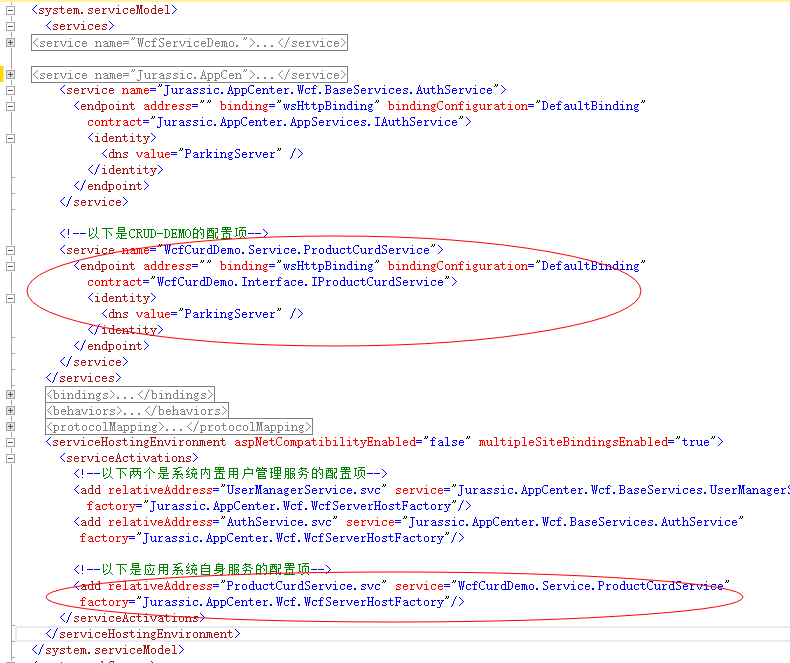


如果是涉及到数据库操作，推荐使用两层:一层是直接实现接口的上层业务逻辑，一层是底层数据访问逻辑。两者使用业务实体来交换数据。

上面的代码就是使用了两层结构的例子。其中IAuditDataService<Demo\_Product>就是框架提供的通用实体数据访问接口的一个标准实现。它使用框架提供的依赖注入功能在运行时动态实例化。

1. 配置

在以上工作完成以后，只需要通过在作为宿主程序的web.config文件中作简单配置，服务就可以发布了：



## 如何实现客户端无缝调用WCF服务(或本地服务)

在本框架中，集成了自动生成WCF代理并调用WCF的一系列API。因此，在编写客户端时，结合SCSF框架本身的服务加载机制，可以做到：在使用服务接口定义的服务时，完全不用考虑该服务是如何生成、来自哪里，是本地还是远程。只需要在注入对应的服务实例时，针对不同的情况稍做做改即可。下面就以下3步说明如何生成WCF的客户端代理：

1. 根据提供的主程序配置文件模板，增加服务端的终结点配置：



上图表示，只需要在框架的配置文件内，增加一个(或多个)endpoint配置节，每个提向一个wcf的服务。

1. 创建一个无界面的Module用于动态生成这些服务代理。

在该Module中的Module.cs中，写如下代码：



当然，如果你确信这些代码的加载方式不太可能会变，可以直接把它写在和相应的调用方模块的Moudle.cs中也可以。

1. 上述完成后，就可以在具体的调用方发起服务调用了:



## 自动更新组件AutoUpdateModule使用方法

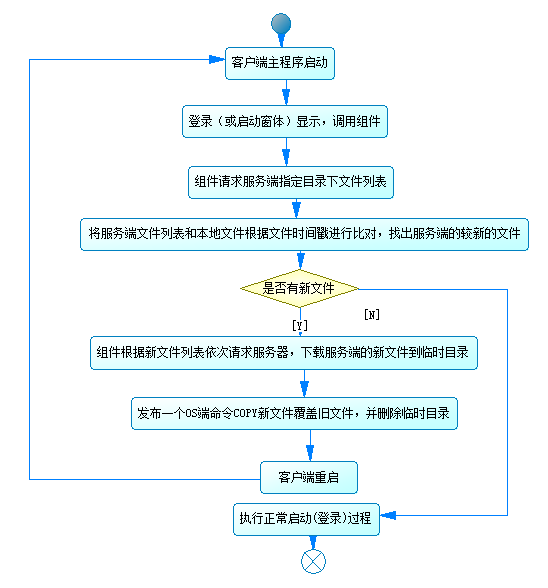
注意：本节所有的关于类库的描述都是基于 **公用组件库/dotnet/AppCenter/Winframe.sln**下的项目。

### 实现原理

自动更新组件AutoUpdateModule是本框架中，以MVP模式规范编写的一个CAB模块。

它需要配合自动更新服务器一起工作，服务器端的部署方式是基于本公司WCF服务端框架编写的，本公司的WCF服务端框架自带了支持自动更新所需要的后台服务模块。

这个简单的C/S结构的子系统的主要工作流程如下：



### 服务端编译、配置和安装

首先服务端的WCF服务是不需要任何额外的C#编码，因为只要是基于公司WCFFrame框架的系统都自带了对应的WCF服务实现。具体的实现请参见: Jurassic.AppCenter.Wcf.BaseServices.UpdateService这个类。

如果你只用于自动更新的话，可以直接编译AppCenter目录下的WcfServiceDemo这个项目并将其和依赖项组成的WCF应用程序部署到指定服务器IIS上即可。

服务端的Web.config配置文件中关于WCF的配置节一定要参照WcfServiceDemo的配置节。

### 客户端调用和配置

客户端是以用户控件的形式提供的UI。在AutoUpdateModule中有一个默认实现AutoUpdateView。可以将其加入到系统显示的启动界面中。**但是不能直接在设计器中从工具箱拖进去**。因为如果拖进去的控件是不走CAB初始化的步骤的。控件将不能正常初始化诸如WorkItem之类的框架对象。 因此正确的加载方法是这样：

pnlUpdate.ClientArea.Controls.Add(\_rootWorkItem.SmartParts.AddNew<AutoUpdateView>());

其中, pnlUpdate是预先放好的一个容器。 \_rootWorkItem是窗体能获取到的workitem.

代码层面只需要这一句就够了。

配置方法：

全面参考在WinFrame.sln中的mainprogram项目中App.config中关于WCF的配置节即可。

另外，如果需要指定服务端的文件更新目录，可以在<appSettings>的配置节中加上一句：

<add key=”remoteDir” value=”服务端目录名“/>

注意这个服务端目录名是单个名称，它会自动建立在服务端运行目录下。这意味着多个客户端程序可以共享一个服务端WCF服务来发布自动更新。因为它们各自的目录可以不同。

如果没有这个配置节，则服务端默认认为是UpdateFiles目录。

### 使用和维护

从以上描述可以得知，服务器上有对应的专门的目录用以存放所有客户端运行所需的完整版本的文件。当这些文件有新版本时，将新版本文件直接COPY到服务端目录中，这样客户端会在启动时自动获取这些更新。

唯一需要注意的是，服务端这个专门目录下的目录结构和客户端运行目录的目录结构要保持一致。

如果在自动更新过程中发生了无法预料的异常，会在客户端根目录有一个Update.log的日志文件，在其中记录了异常的相关信息。

### 扩展接口

可以重新定义用于展示客户端自动更新的View。方法是另起一个项目，在其中仿照标准的AutoUpdateView的程序骨架建立一个新的View, 并且在初始化代码中加上

pnlUpdate.ClientArea.Controls.Add(\_rootWorkItem.SmartParts.AddNew<自定义的View名称>())。

这新的扩展方式 是使用MVP的特点，它的Presenter依然是原有的，只是View的实现不同了而已。

当然后期根据应用要求也可能提供其他方向的扩展。

## 常见问题及解答 (FAQ)

Q: DBHelper能否支持Oracle数据库？

A:DBHelper这个数据库访问帮助类理论上可以支持任一种关系型数据库，因为DBHelper是基于与具体数据库无关的IDBComm的接口的组件进行数据库操作。目前框架中已经实现了基于SqlServer、Oracle、Access、SQLite的IDBComm接口组件。

Q：为什么我在模块代码中打了断点，但是调试时进不了断点？

A：排除逻辑原因的话，极大可能是因为Winframe框架采用的是插件式开发，所有的业务模块项目都作为插件动态加载，而主程序对其没有显式依赖。这也造成了如果不单独编译要调试的项目，只启动主程序的话，该模块不会自动编译，自然也就不能形成调试文件，进不了断点。

解决方案是每次运行前单独编译一下修改过的项目。如果觉得这样麻烦，可以手动设置主程序的依赖项。方法是右击主程序所在项目，选择“生成依赖项-》项目依赖项”，再勾选你的业务模块。

Q：为什么我的窗体在启动时，没有显示窗体的标题，我明明在窗体设计器里输入了标题？

A: 因为你没有在窗体的类前加上[SmartPart]标记。这会导致窗体的初始化不能按照SCSF框架的套路进行，导致窗体行为异常。

## 附录

### 在编写本框架的过程中，参考了以下网址：

对话框显示

<http://www.codeproject.com/Articles/28927/Displaying-a-Dialog-Box-in-SCSF>

SCSF中的IoC

<http://www.cnblogs.com/flyabroad/archive/2008/06/17/1223741.html>

SCSF中Action的使用

<http://blog.csdn.net/lee576/article/details/12155317>

StackOverflow上关于SCSF的提问与解答

<http://stackoverflow.com/questions/tagged/scsf>

### SCSF的学习网站

官网

<https://msdn.microsoft.com/en-us/library/ff709809.aspx>

Quick Starts

<https://msdn.microsoft.com/en-us/library/ff699186.aspx>

Publishing Events

<http://msdn.microsoft.com/en-us/library/ff699223.aspx>

MVP设计模式

<http://msdn.microsoft.com/en-us/magazine/cc188690.aspx>

# Jurassic.Frameworks的VS模板插件

## 概述

Jurassic.Frameworks的VS模板插件是基于Visual Studio 2013开发平台创建的一套VS可以快速创建公司框架体系项目和项目文件的可安装插件程序。

### 开发目的

开发此模板插件的目的是为了可以快速准确的创建集成公司框架的项目和对应的项目文件。创建的模板项目和项文件包含了对应的文件和引用，省去了一系列的操作，节约了开发时间。

主要以下几点:

* 保证新建项目的正确性和一致性；
* 无需手动修改保证项目命名空间和项目命名的一致性；
* 可以选择性的加载框架可扩展模块；
* 便于公司框架后期的升级和维护；
* 插件本身作为公司技术资产，简化操作，节约了开发时间。

## 运行环境

插件的安装使用只针对Visual Studio 2013版本，暂不支持Visual Studio其它开发版本。其它所必须的环境参考公司框架所必须的开发环境即可。

## 安装及使用

SVN上有一模板插件安装文件夹在：

[http://192.168.1.3:8443/svn/公用组件库/dotNet/AppCenter/packages/  
Jurassic.Frameworks/JurassicVSIX](http://192.168.1.3:8443/svn/公用组件库/dotNet/AppCenter/packages/Jurassic.Frameworks/JurassicVSIX)

1. 确认已经安装VS2013开发环境。
2. 确认C:\Program Files (x86)\Microsoft SDKs\Windows\v8.1A\bin\NETFX 4.5.1 Tools\gacutil.exe文件存在。
3. 用系统管理员身份执行安装目录下的registerWizard.bat注册脚本，确认注册成功。

直接双击安装可能失败（如图1），用管理员身份执行就会成功（如图2）。

1. 执行jurassic.FrameworksVSIX.vsix安装程序，安装成功后重启VS2013开发工具。
2. 在VS2013新建项目中出现，已安装->模板->Visual C#->Jurassic.Framework选项。
3. 选中Jurassic.Framework选项，会在右侧窗体中出现WebFrame Project Template、 WinFrame WCFProject Template和WinFrame LocalProject Template三个选项（如图3），根据实际情况选择对应的项目模板。
4. 新建项目的位置应在 项目管理规范\12-项目模板 文件夹之中，以确保新建项目引用程序集的路径正确。
5. 勾选“为解决方案创建目录”这个选项。
6. 确定之后就可以创建对应的项目，自动生成必须的项目文件。
7. 在项目创建后的实际开发的过程中，还有简化开发项模板（如图4）
8. EnptyControllerItemTemplate会创建空的Controller文件。
9. GeneralControllerItemTemplate会创建带有基本增删改查方法的Controller文件。

注: 增删改查方法的实现需要开发人员编写代码完成，模板只提供了空方法。

1. MVPItemTemplate会创建基于MVP开发模式所需的View，IView，ViewPresenter三个文件。



图1

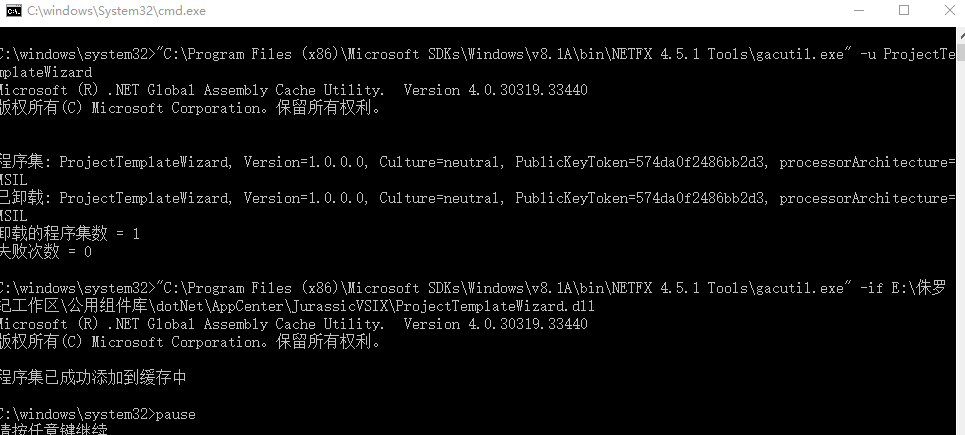


图2

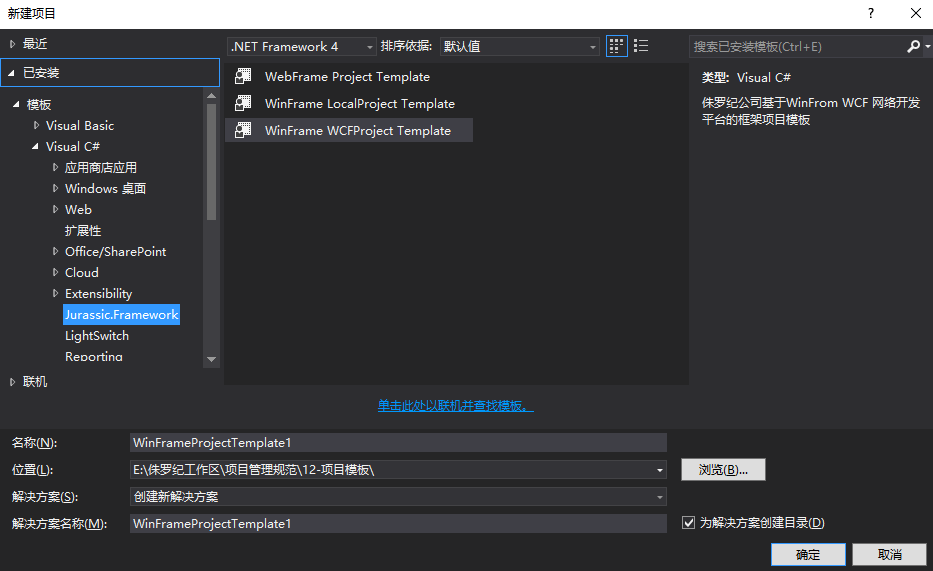


图3

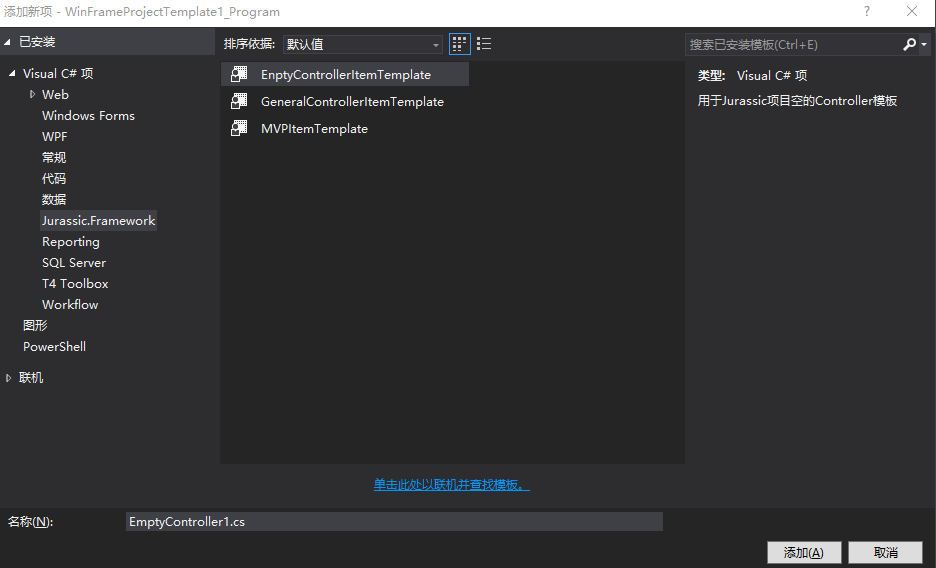


图4