广州爱奇迪软件科技有限公司(http://www.iqidi.com) Mail:wuhuacong@163.com

Winform 开发框架
架构设计说明书

版本： 5.0

编制人：伍华聪

目录

1. 引言

1.1. 背景

本人多年来一直致力于开发一些共享软件，如客户关系管理系统、会员管理系统、病人资料管理系统、送水管理系统、酒店管理系统、仓库管理系统、配电网络可视化管理系统，以及一些小型的软件，如 QQ 搜通天、易博搜搜、赶集小神童、绿苗帮电脑监控系统等，以及在日常中，也是从事一些较为大型系统的开发工作。一直以来，有一个想法，就是尽可能利用好的、经过淬炼的技术，以及日常积累的经验所得，构建一个 Winform 开发框架的生态体系。

这个体系包括有：

1）Database2Sharp 代码生成工具，用来辅助生成复杂的架构基础代码，以及日常的一些琐碎反复的工作。

2）一个稳定成熟、反复应用过的 Winform 开发框架，集成应用程序必备的一些常用操作，新的业务系统只需要在其上面按既定的模式叠加业务操作即可，提供框架的可用性、稳定性以及完善性。

3）提供一个所有业务系统都很常见的权限管理系统以及一个也很常见的字典数据管理模块。这两个模块是组件化的模块，既相互独立，又可以和 Winform 框架进行集成，供业务系统重复调用。

4）一个集成多年经验积累、反复优化提炼的公用类库，类库封装日常开发所应用到的方方面面，如一把瑞士军刀，一个个奇兵，各有用处。

5）一系列界面控件的整合效果，提供各种报表生成的解决方案（普通二维报表、自定义模板报表、复杂报表等高级功能。

6）提供一个封装日常数据显示的分页控件，集成高性能的数据分页显示、数据打印、数据导出、常用数据操作等接口功能。

7）提供一个通用的程序自动更新模块，更快、更便捷实现程序的自动更新，避免挨个给使用者打电话、发信息通知或者发送软件等，要求其对应用程序进行升级。

8）基于上面第 2 点的 Winform 开发框架及整合所有通用模块，还可以开发基于互联网软件发布模式（整合 WCF 服务、Web API 服务）的混合式开发框架，构建一个安全、高效、便捷、分布式的业务管理系统。

1.2. 设计目标

1、使用代码生成工具辅助，快速搭建相关的业务框架系统。

2、在完善的 Winform 框架模板上开发新的业务系统，减少重复劳动。

3、支持多种数据库业务系统应用的解决方案，配置即可自动切换。

4、界面、模块、公用代码复用性高，降低重复代码，提高开发效率。

5、权限管理系统、数据字典模块和业务系统分离，独立成一个系统模块，并能为其他项目系统提供相应功能的服务。

6、只需关注业务系统的核心业务实现，框架、类库、独立模块、控件简化、封装常用操作。

7、良好封装、开发高效、界面友好、强壮完善等特点。

8、所有模块在《Winform 开发框架》、《WCF 开发框架》、《混合式框架 》中都可以独立维护管理，方便维护和独立更新。

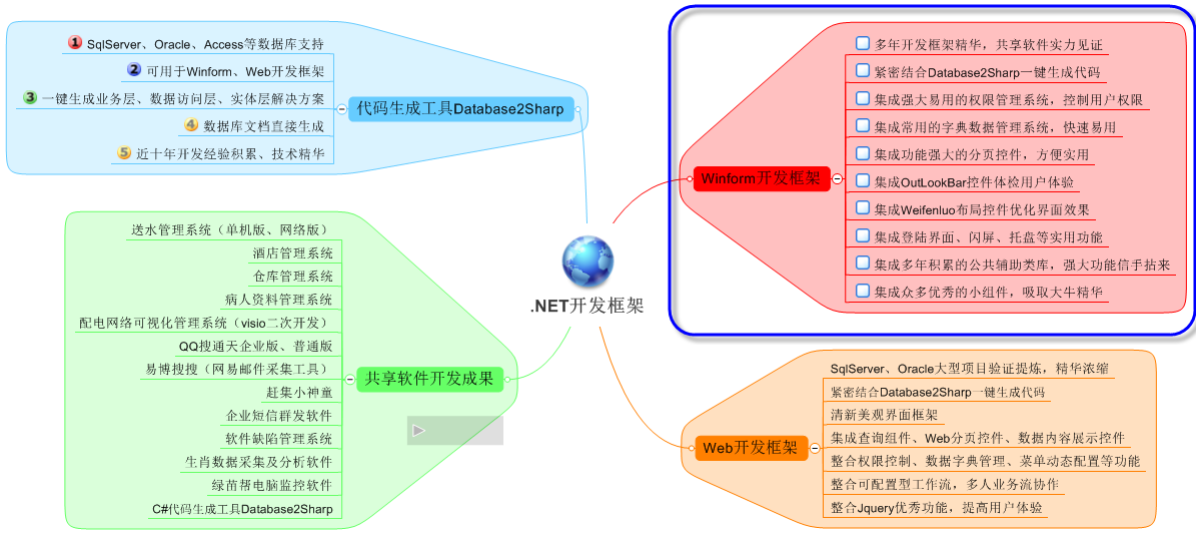
2. 模块列表

编号	名称	说明
MD001	Winform 框架界面处理模块	提供对业务系统界面的展示和整合。集成权限管理系统模块、集成字典数据管理模块、集成强大的分页控件、集成 OutLookBar 界面控件、集成多文档界面 Weifengluo 布局控件、集成美观实用的登陆界面、闪屏展示界面、托盘缩小提示功能、全局运行一次实例限制模块代码、集成多年积累公共组件简化常用功能操作、集成 Apose.Cell 基于模板自定义报表快速生成复杂报表模块、众多美观使用的界面控件整合等等。
MD002	Winform 框架业务及数据处理模块	提供业务层封装、数据处理层、实体类等模块。紧密结合 Database2Sharp 强大代码生成工具生成的代码、各层高度抽象继承及使用泛型支持多数据库的开发框架。 系统所有数据处理模块，均能很好支持多种数据库，包括 Oracle、SqlServer、SQLite、MySql 常规数据库，以及其他数据库等，所有模块开发模型及分层思路相同，实现统一化、高效化，减少学习成本。
MD003	公用类库模块	提供日常各种开发操作的辅助类库。覆盖范围基本上是包罗万象的，从集合（包括同步、排序等的）、设备辅助类（包括声音、照片采集、剪贴板、计算机硬件信息、键盘、鼠标等辅助类库）、加密类库（包括 Base64、Md5、SHA1、可逆与不可逆加密等）、线程（多线程、代理、Timer 计时器等）、以及更多的、更广应用的辅助类库，这些类库包括各个方面，如配置、字节操作、日历、DataTable 操作、打印、目录、文件、Access、Excel、Word 合并、正则表达式、网页采集、压缩算法、图片操作、WInform 窗体动画、INI 文件操作、日志操作、RichTextBox、独立存储、拼音编码、人民币格式、随机字符、POS 打印、反射操作、代理设置、注册表、图标、Windows 消息、字

		符串、XML、提示对话框封装、文件对话框封装等等。
MD004	权限管理模块	提供对业务系统功能控制以及用户角色管理的系统功能。权限管理模块既相对业务系统独立，又可以与业务系统紧密整合。权限管理模块对机构基本信息、机构和用户关系、机构和角色关系进行管理。对角色基本信息、角色和用户关系、角色和机构关系、角色和功能关系进行管理
MD005	数据字典模块	提供给业务系统常用字典管理功能，提供界面对字典进行维护等功能。字典模块在很多系统是必须的，如人员的职称、身份、客户类型、活动分类等等，有了这些维护处理，界面处理只需一行代码就可以绑定，非常方便。 另外，我们在字典模块中，根据全国统计局的数据，整理全国省份、城市、行政区的三级联动数据，方便使用。
MD006	通用自动更新	根据客户端软件版本和服务器的版本对比，自动提示客户对版本进行更新，从服务器下载更新文件解压覆盖现有文件，并自动重启软件。 自动更新模块，能够最大程度减少我们部署软件的成本，使客户能够联网后实现自动更新。
MD007	通用附件管理	该模块其实是很通用的一个模块，例如我们的一些日常记录，可能会伴随着有图片、文档等的附件管理，如果为每个业务对象都做一个附件管理，或者每次开发系统都重新做，那么效率肯定没有直接采用通用的附件管理那么方便快捷了。而且在日益增多的项目管理中，我们不需要维护一大堆相同或者类似的代码。 该模块封装良好，支持多数据库，可以在我们开发的系统模块中整合使用，基本上拖动控件即可使用。
MD008	注册码验证	提供一个注册码生成工具（含源码），以及在例子中结合注册码验证，从而实现对开发的系统使用注册码限制使用，对注册码授权机制进行介绍。
MD009	人员信息管理	通用人员信息管理模块，包括有人员基本信息，学习情况，职称情况，履历情况，出国情况，家庭情况，受奖情况，以及相关的附件信息。这个模块在很多场合都可能用到，如企业员工管理、科室员工管理等等，这些要求登记人员详细资料及图片等信息的系统模块。

3. 架构视图

3.1. 整个.NET 开发框架之间的关系



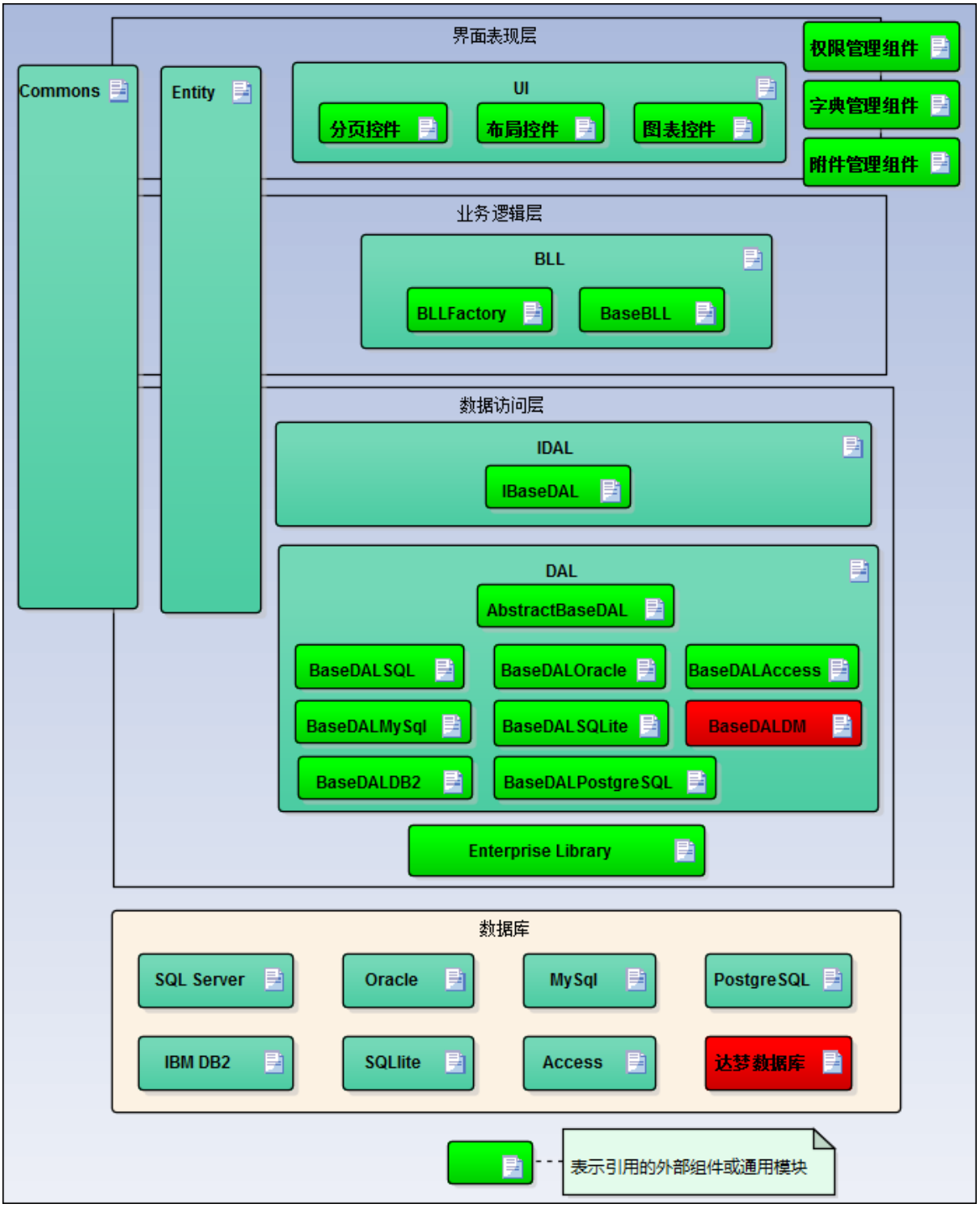
我一直从事相关的.NET 开发工作，一直在想如何整合我所有的开发经验及技术积累。在学习、进步、提升的开发工作中，开发过很多 Winform 共享软件、Asp.Net 的 WebForm 项目、Asp.NET MVC 项目等，发现很多东西是相互关联很紧密的，但往往自己觉得太忙太懒，要好好整理，并整理出一个体系的东西一般比较难，但我一直没有停步，梦想总会慢慢接近并实现。

在做了很多项目之后，发现人的惰性或者惯性很大，因此有机会得好好整理下开发的成功，优化再优化，用的时候就越来越顺手了。

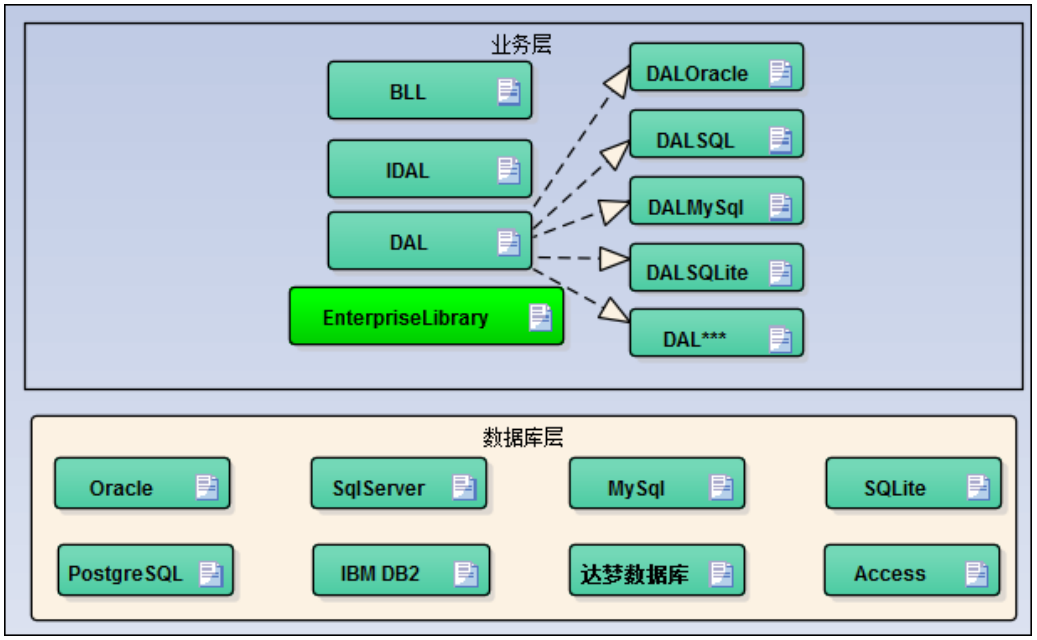
在所有开发过的项目过程，很多如权限管理、字典数据管理、附件管理等模块，都是非常常用的模块，可以将其独立、封装出来；数据展示模块是很多业务系统中常有的功能，可以使用封装良好的分页控件进行处理，简化工作；而整个 Winform 开发框架，不同的业务系统还是有不少差别，在做过送水管理系统、酒店管理系统、仓库管理系统、病人资料管理系统、客户关系管理系统等软件后，发现大多数业务需要处理的东西是基本一致的，而且很多东西如数据展示、报表处理、窗体布局、模块继承等东西都是需要频繁处理的事情。

因此，本人把反复修正过、完善过的仓库管理系统，进一步集成和优化，吸收更多之前的开发经验，并集成更多的一些常见模块，对仓库管理系统进行升级，把它提升到 Winform 开发框架的载体和概念高度上来。

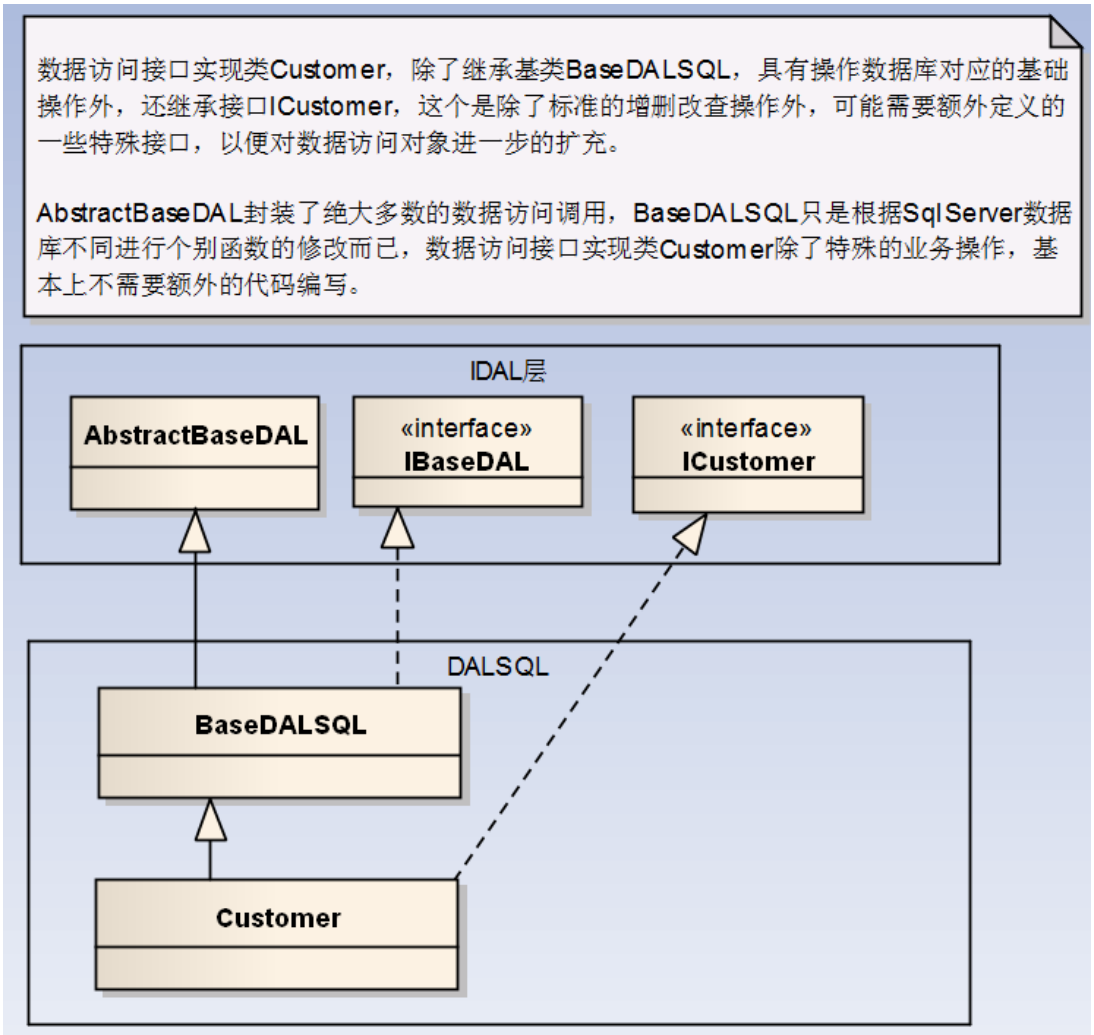
3.2. Winform 开发框架的架构视图



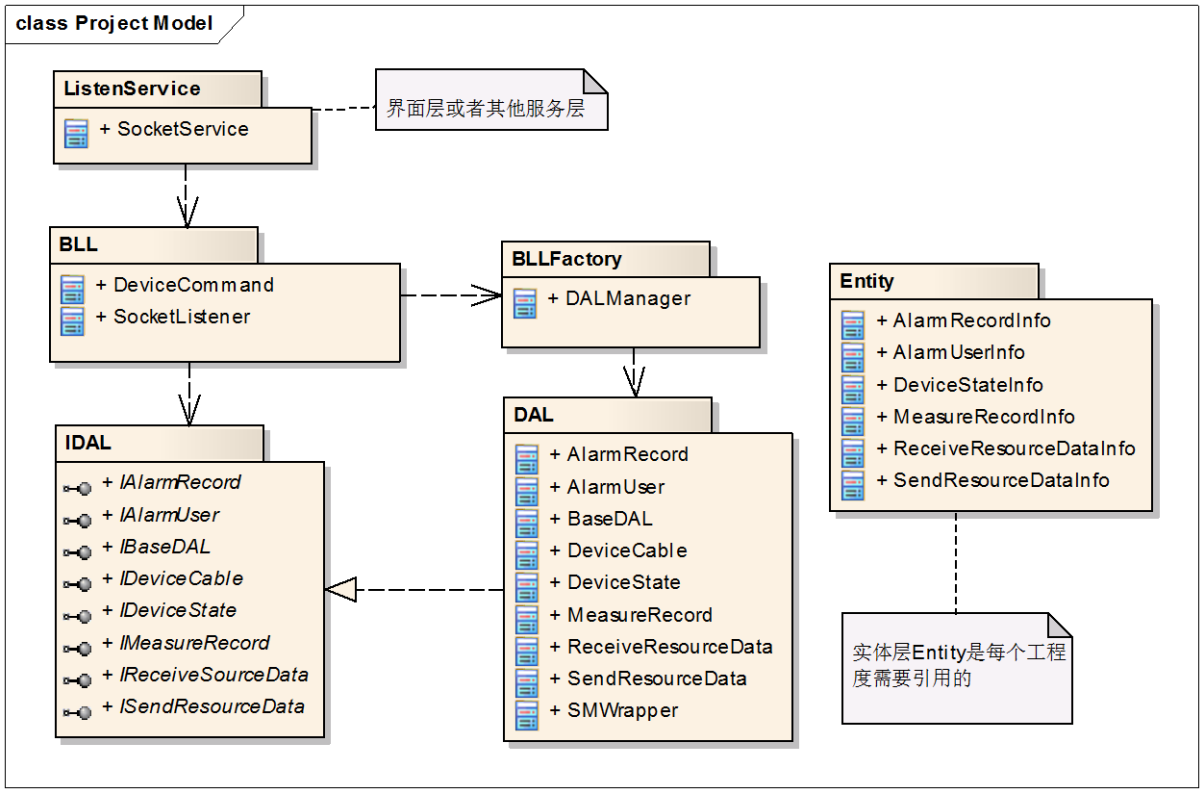
Winform 框架系统架构



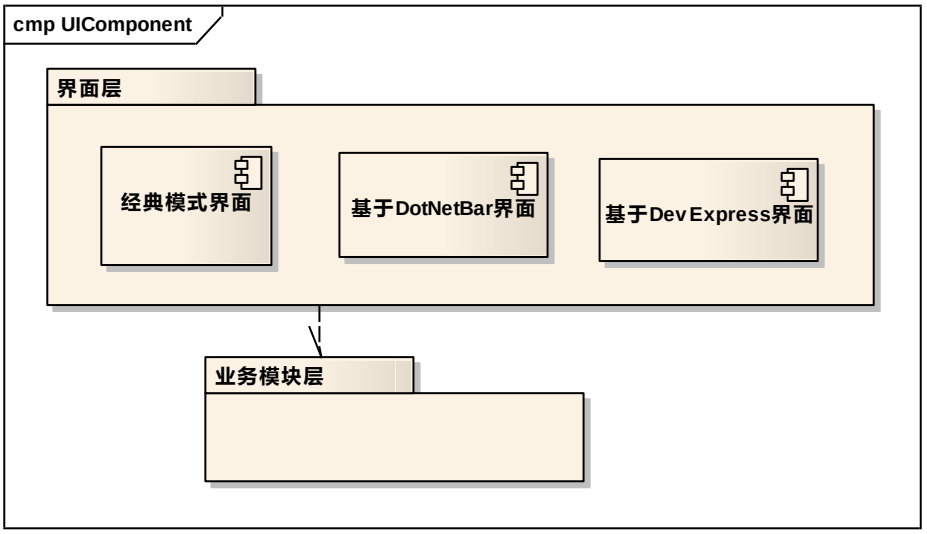
框架界面层以下的架构设计



框架数据访问层类的关系



Winform 框架各层的调用关系



Winform 框架界面扩展视图

3.3. WCF 开发框架的架构视图

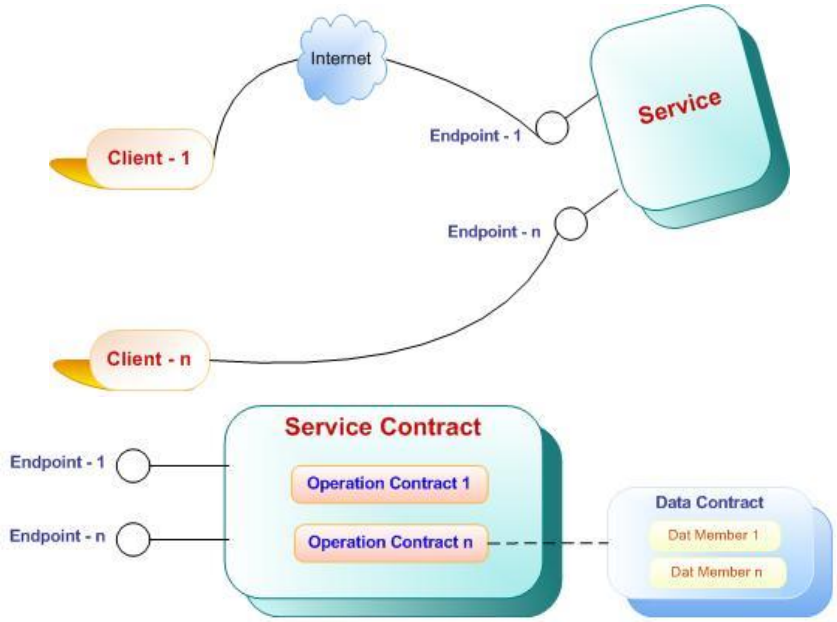
整个 WCF 框架和 Winform 开发框架一样，整合了权限控制管理、字典管理模块、公用类库、通用程序自动更新等模块，具备良好的界面布局和分布式服务应用的特点，支持数据分页、数据导入、Excel 导出、支持多界面样式、支持闪屏、热键控制、多数据库支持，并且和代码生成工具 Database2Sharp 紧密结合等等特点，不一一尽数。

界面层在传统的 Winform 框架界面中和 WCF+Winform 框架界面中表现一致，在局域网中部署测试，客户端 + WCF 服务器 + Oracle 数据库服务器这种部署模

式，非常流畅，由于是基于 WCF 框架结构，可以应用在广域网中，不过可能服务响应及 Winform 的体验要依赖于带宽的大小。

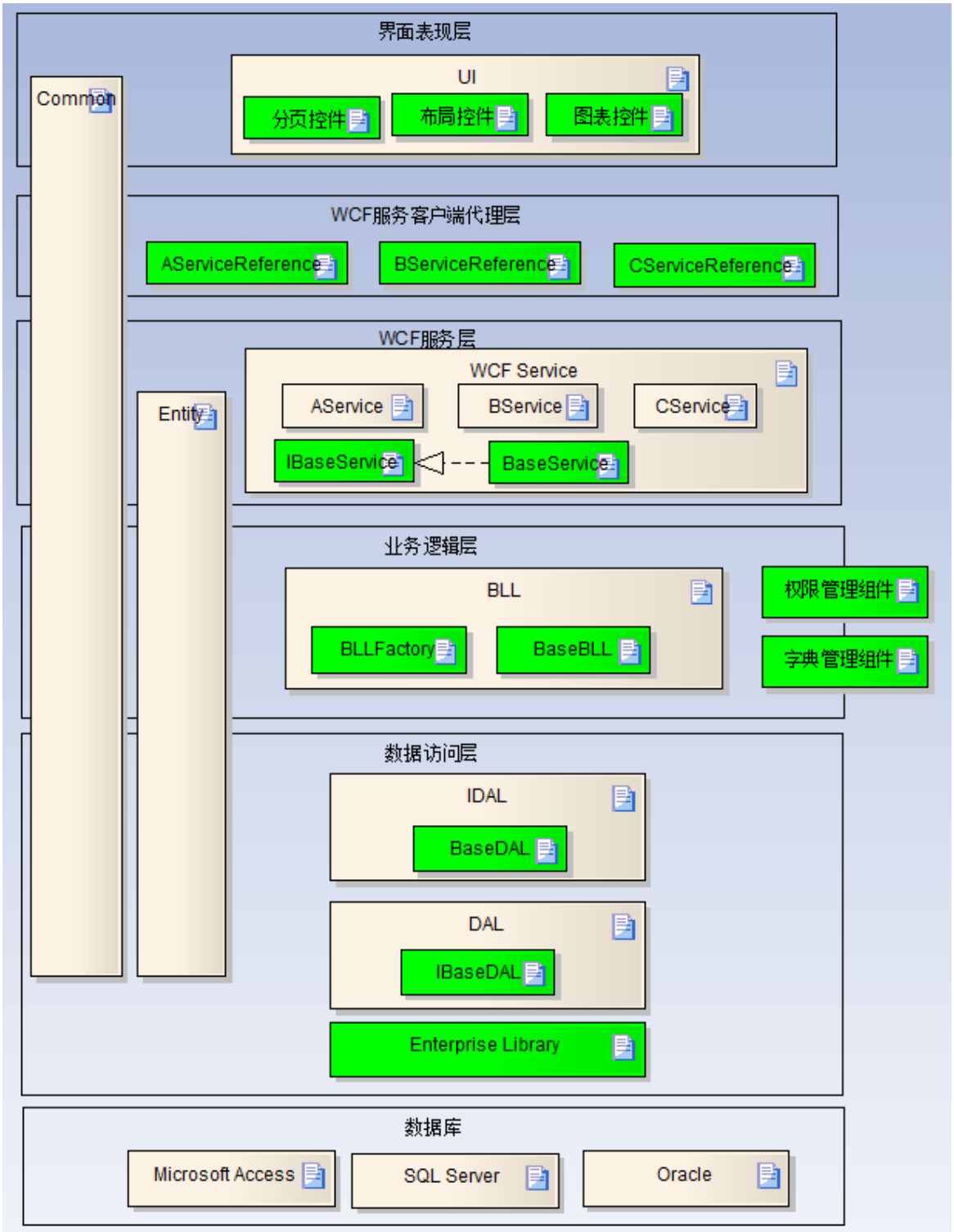
3.3.1. WCF 开发框架特点

WCF 主要是基于客户端-服务端通讯模式来实现分布式应用，并通过服务公布的节点进行访问，实现数据的交换等服务。下面是其中应用的几个示意图。

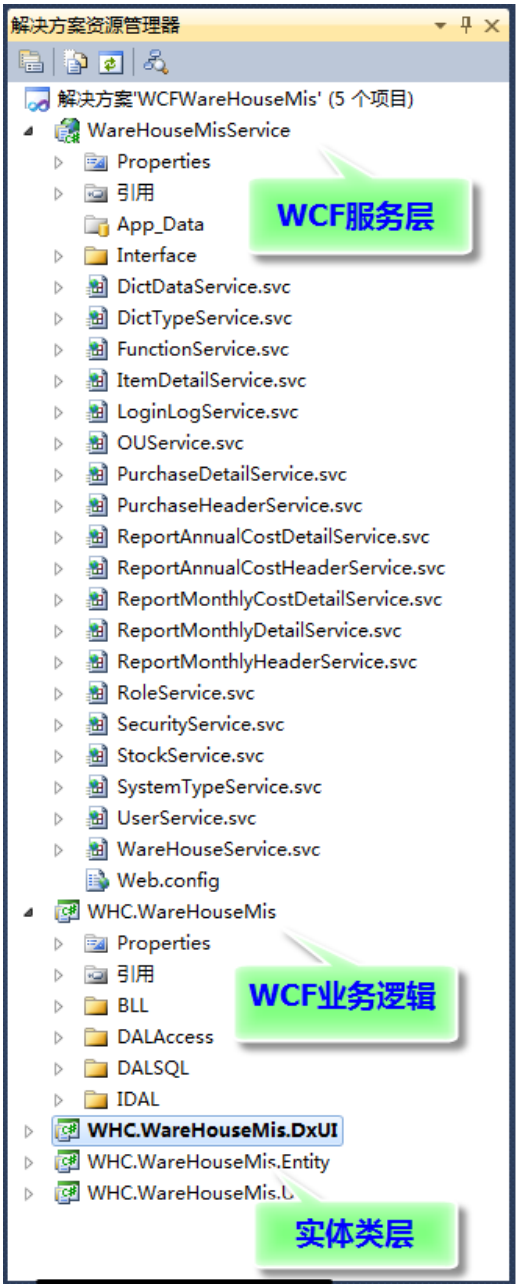


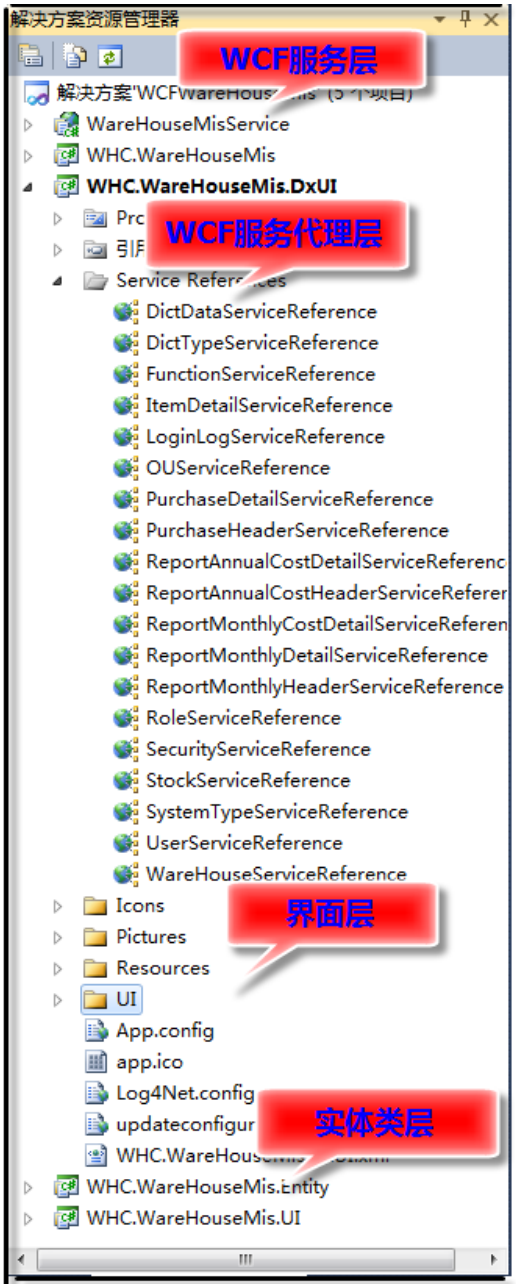
3.3.2. WCF 开发框架架构视图

基于 Winform 框架的 WCF 开发框架扩展，首先在界面层和 BLL 层插入一层 WCF 服务层，界面层 UI 不再业务层 BLL 打交道，而是代之以 WCF 服务层的客户端代理类打交道，而 WCF 服务层则是 BLL 层更进一步的包装，设计图如下所示。



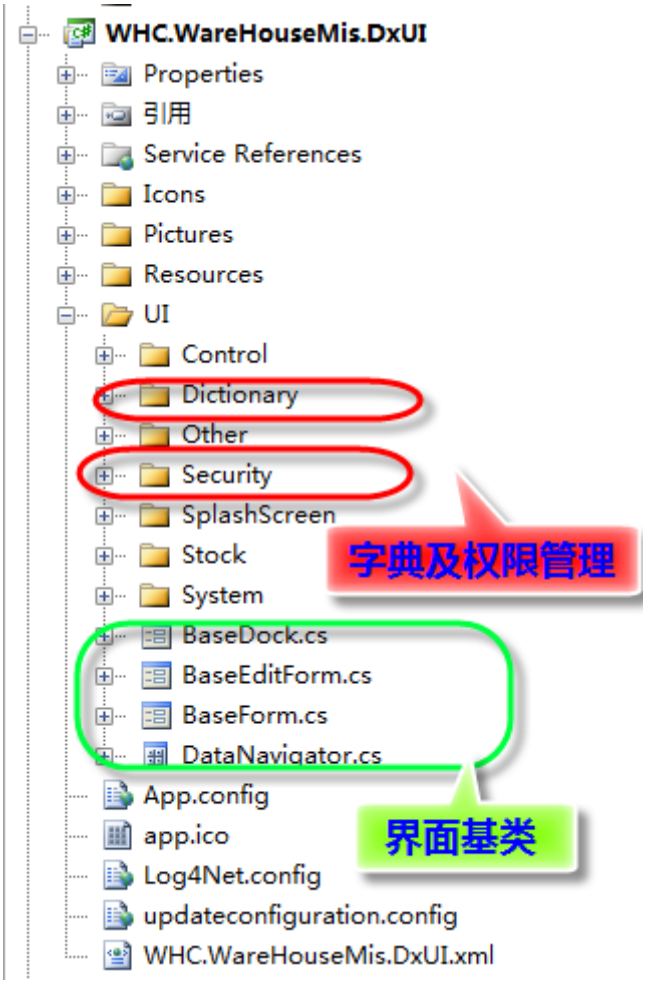
3.3.3. WCF 开发框架项目视图





下面的图示界面层工程项目展开的截图，我们可以看到，在 Winform 框架中独立的通用权限管理模块、通用字典管理模块，在这里做了一个整合，不再是独立应用的程序集模块，所谓合久必分，分久必合，就是这样的道理。

其中红色部分就是字典和权限管理的控制界面模块，这里把它作为界面的一部分，方便服务层的统一部署，统一使用或者统一修改配置等。 另外绿色部分是界面层的基类，这个和 Winform 框架是一样的，都是为了达到统一、合理封装的目的。



3.4. 混合型开发框架的架构视图

混合型框架可以看成是 Winform 框架高级版本，除了它本身是一个完整的业务系统外，它外围的所有辅助性模块均（如通用权限、通用字典、通用附件管理、通用人员管理。。。。）都实现了这种混合型的框架，因此使用非常方便，整个框架如果简化来看，就是在原有的 Winform 界面层，用接口调用方式，避免和业务逻辑类的紧耦合关系。由于他是通过接口方式的调用方式，它本身又可以通过配置指定指向 WCF 的实现（既适应 Winform 集成，也适应 WCF 集成），因此也囊括了 WCF 框架的一切特点。

3.4.1. 混合型框架的特点

混合型框架具有下面几个特点：

1）环境适应性强，模块可重用性高。由于混合型框架，既可以用于传统 Winform 系统开发，也可以用于 WCF 分布式系统开发，因此环境适应性强；而且由于模块具有这些特点，可重用性更高，特别对于通用性的模块，更是具有无可替代的优越性。

2）响应性能更好。如果是 Winform 程序，那么就使用直接访问数据库方式，如果是 WCF 调用方式，就使用 WCF 的专有通道进行数据处理，更好利用系统资源，高效进行数据处理。

3）独立配置，更少的代码修改。所有通用模块，全部通过独立配置文件进行配置 WCF 的连接，减少主配置文件的复杂性；WCF 服务逻辑独立类库，可采用多种服务寄宿方式。

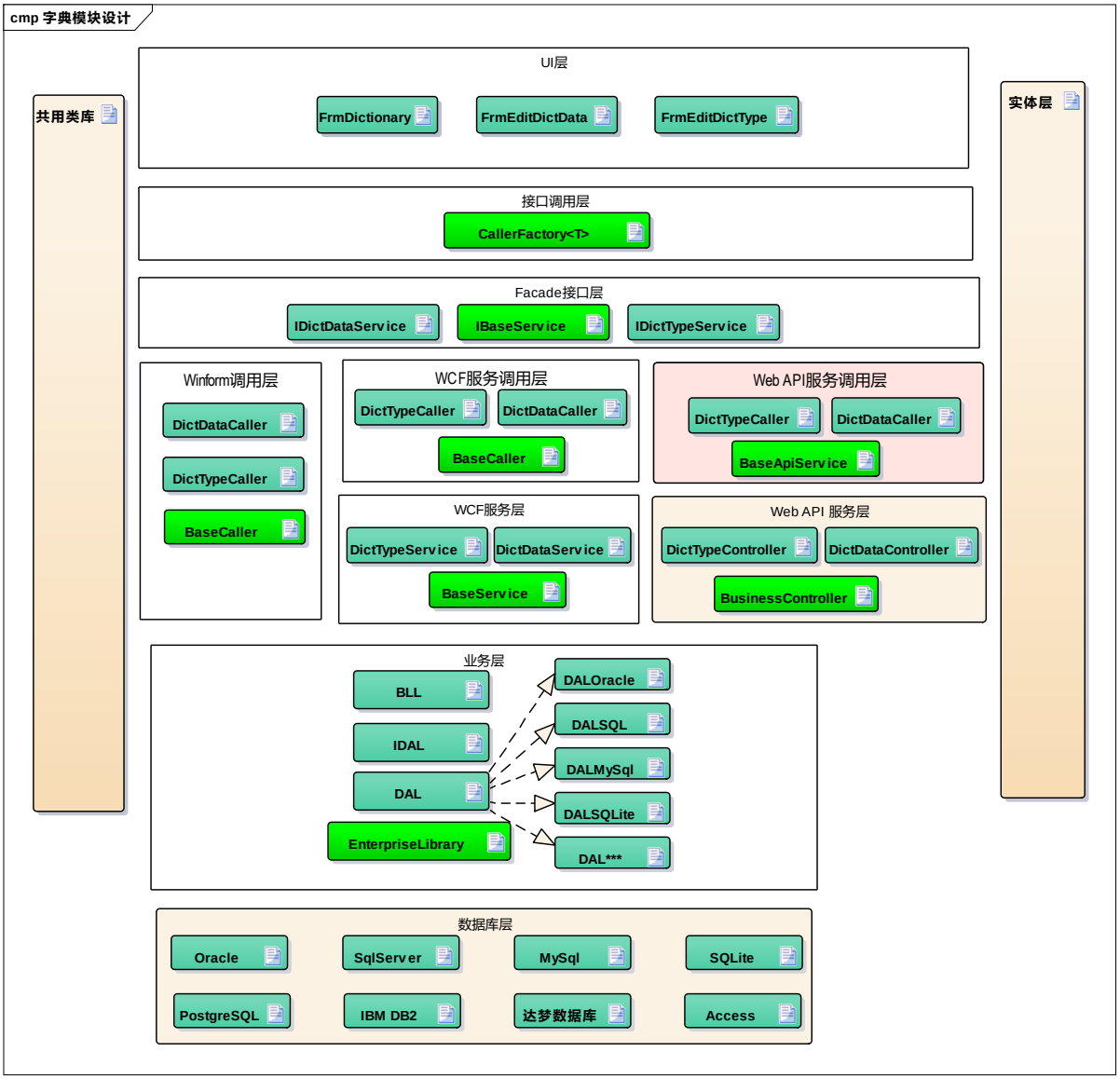
3.4.2. 混合型框架总体设计思路

Winform 开发框架之混合型框架，还是秉承模块化的思路，可以把这个框架分为两大块，一块是主要业务系统模块（如备件管理系统），一块是各种辅助性模块（如通用权限、通用字典、通用附件管理、通用人员管理。。。。），这种两块组合，就是一个完美的系统了。

整个系统的业务系统模块和辅助性模块，都是基于一个思路，通过接口调用开关，决定调用的是 WCF 服务层，还是 Winform 业务层（直接访问数据库），当然界面层的调用不管是调用 WCF 服务层还是 Winform 业务层，都是基于相同的接口，我们可以把它称为 **Facade 层**。辅助性模块则是多种常用模块的组合，它们可能是下面几种的常见模块：通用权限模块、通用字典模块、通用附件管理模块、通用人员管理模块等等。

3.4.3. 混合式框架的代码生成工具支持

当然，虽然混合型框架比传统的 Winform 框架和 WCF 开发框架更为通用，不过由于它引入了多一层，而且为了实现更多模块的分离，增加了一些设计上的复杂性，整个项目工程看起来显得复杂了一点，如下面就是一个以字典模块为例的混合型框架的内部结构。



从上图我们可以看到，整个混合型框架的架构，分为了 UI 层、接口调用层、Facade 接口层、Winform 调用层、WCF 服务调用层、业务层、实体层、以及数据库层等；其中的业务层还可以细化为 BLL 业务逻辑层、数据接口层、数据访问层、实体层等，整个模块通过实体层进行数据的传输载体。

为了实现更简单化的开发，更快更高效的完成混合型框架的开发工作，我扩展了我的代码生成工具 Database2Sharp，使其支持这种混合型框架的代码生成工作，这样开发混合型框架就和开发其他两种 Winform 开发框架、WCF 开发框架一样，非常方便了。

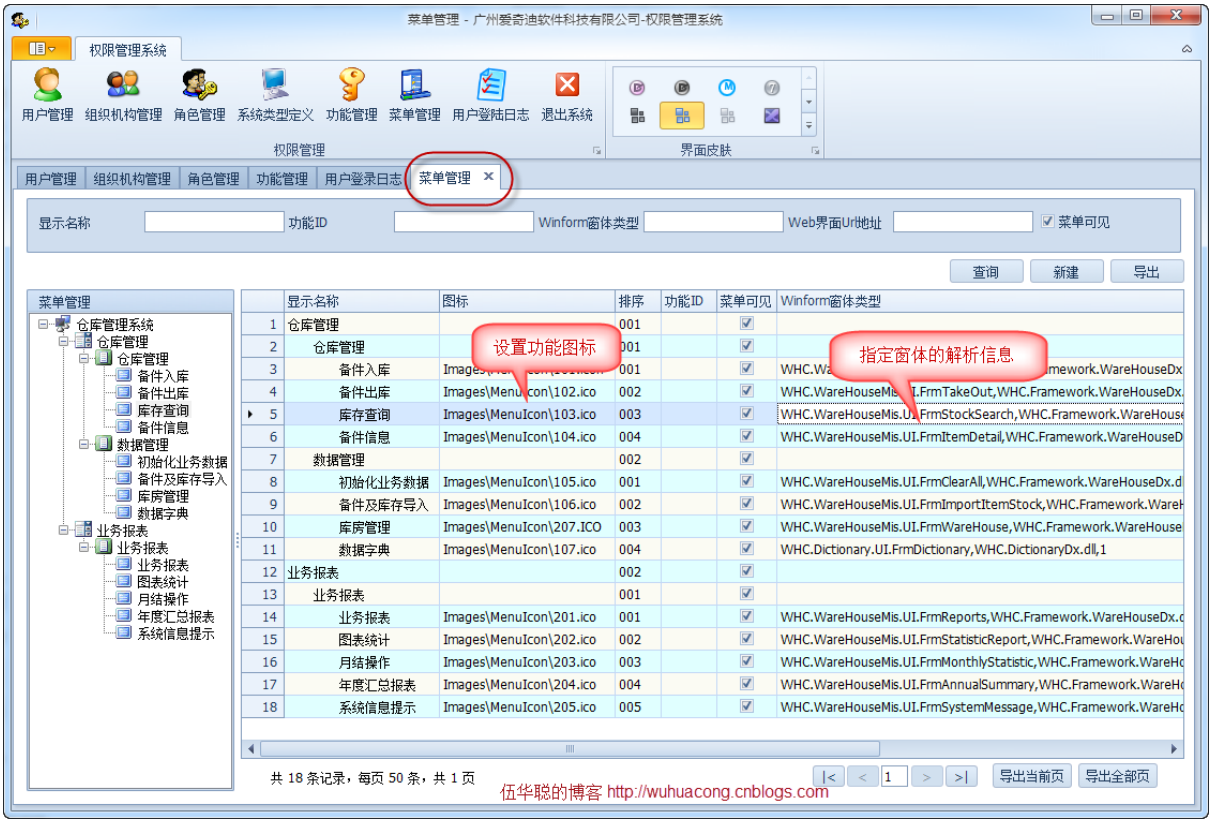
4. 插件化框架设计

支持插件化应用的开发框架能给程序带来无穷的生命力，也是目前很多系统、程序追求的重要方向之一。插件化的模块，在遵循一定的接口标准的基础上，可以实现快速集成，也就是所谓的热插拔操作，可以无限对已经开发好系统进行扩展，而且不会影响已有的功能，不在需要的模块，通过修改配置移除即可。目前在 Winform 开发框架、WCF 开发框架、混合型开发框架，均实现插件化的框架设计。

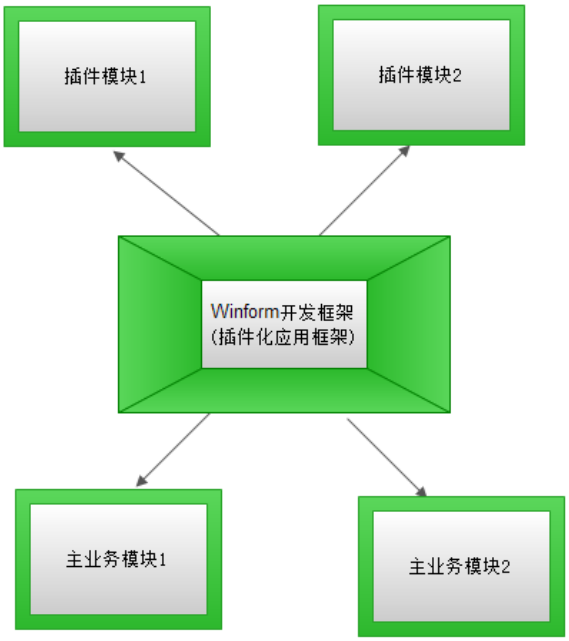
我的 Winform 开发框架一直以来，来源于多年的项目积累以及客户的反馈，已经具备了众多很好的特性以及相关的模块组合，为了更好拥抱变化，提高基于 Winform 开发框架基础上开发新系统的效率，以及为框架融入更多好的特性，故此把我的 Winform 开发框架在原来的基础上进行扩展，实现基于插件化应用的框架特性。

为了引入插件化的应用框架特点，首先需要对通用权限管理系统进行了改进，其中增加了菜单管理模块就是为了做插件化做准备的，通过权限管理系统配置好菜单的相关信息，然后在应用框架中动态加载菜单功能即可实现。这个菜单模块，是用来配置基于 Winform 开发框架、WCF 开发框架、混合式开发框架或者 Web 开发框架的菜单，通过预先的配置，框架程序的动态加载解析，就能实现插件模块的热插拔功能

了。



最终在 Winform 开发框架的程序中，实现基于插件化的应用，如下所示。



4.1. 插件化框架的项目工程规划

为了减少框架整体的复杂性以及提高重用，对插件化的应用框架的项目工程进行了划分，包括《框架基础界面模块》、《插件应用框架启动模块》、《仓库管理系统模块业务逻辑》、《仓库管理系统模块窗体界面》等几个部分。

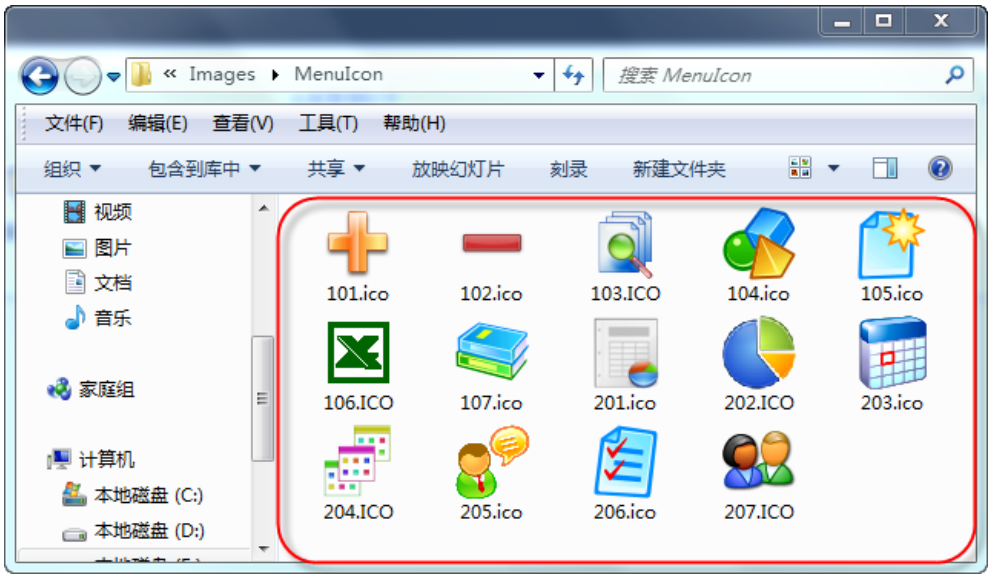
前面两个部分是插件化框架的核心，可以认为是不需要变化的模块，提供所有插件应用动态创建以及使用的框架支撑；后面两个是具体的主业务模块，这里以 Winform 开发框架中的仓库管理系统作为主业务模块，它本身也是插件应用之一，具体的项目工程结构以及说明如下所示。

项目名称	项目说明
WHC.Framework.BaseUIDx	框架基础界面模块，定义窗体界面基类、通用 Excel 导入模块、通用高级查询模块等
WHC.Framework.StarterDx	插件应用框架启动模块，集成权限登录、动态菜单创建、插件应用动态加载、基础框架功能等
WHC.WareHouseMis	仓库管理系统模块的业务逻辑
WHC.Framework.WareHouseDx	仓库管理系统模块的窗体界面

从上面的表格说明中，我们可以看到“WHC.Framework.StarterDx”项目工程，是“插件应用框架启动模块”，它基本上只和权限管理系统模块有关联关系，因为权限系统是框架底层支撑的模块，包括用户登录、菜单管理、权限控制等都需要从权限管理系统中获取数据，具体的主要业务功能如下所示。

4.2. 框架的菜单动态加载

本文第一张图片里面，介绍了菜单的定义信息，其中包括了图标的配置，这些图片为了方便管理，以及插件需要动态添加菜单图标，把它放置在了程序目录的相对路径下面，如下所示，动态创建菜单的时候，从指定的路径去获取图标并加载即可。



动态加载菜单是指在插件化应用框架启动，用户登录后进入主界面后，在主界面中动态创建相应的菜单（菜单在权限管理系统中进行配置管理），如下代码所示。

```
/// <summary>
/// 初始化用户相关的系统信息
/// </summary>
private void InitUserRelated()
{
    根据权限显示对象的初始化窗体

    初始化系统名称

    //动态创建界面菜单对象
    RibbonPageHelper helper = new RibbonPageHelper(this, ref this.ribbonControl);
    helper.AddPages();

    //根据权限屏蔽菜单对象
    InitAuthorizedUI();
    if (this.ribbonControl.Pages.Count > 0)
    {
        ribbonControl.SelectedPage = ribbonControl.Pages[0];
    }
}
```

其中是 RibbonPageHelper 为了方便动态创建菜单而创建的辅助类。

```
/// <summary>
/// 动态创建 RibbonPage 和其下面的按钮项目辅助类
/// </summary>
public class RibbonPageHelper
{
    private RibbonControl control;
    public MainForm mainForm;

    public RibbonPageHelper(MainForm mainForm, ref RibbonControl control)
    {
        this.mainForm = mainForm;
        this.control = control;
    }

    public void AddPages()
    {
        //约定菜单共有 3 级，第一级为大的类别，第二级为小模块分组，第三级为具体的菜单
        List<MenuNodeInfo> menuList =
WHC.Security.BLL.BLLFactory<SysMenu>.Instance.GetTree(Portal.gc.SystemType);
        if (menuList.Count == 0) return;

        int i = 0;
        foreach(MenuNodeInfo firstInfo in menuList)
        {
            //如果没有菜单的权限，则跳过
            if (!Portal.gc.HasFunction(firstInfo.FunctionId)) continue;

            //添加页面（一级菜单）
            RibbonPage page = new DevExpress.XtraBars.Ribbon.RibbonPage();
            page.Text = firstInfo.Name;
            page.Name = firstInfo.ID;
            this.control.Pages.Insert(i++, page);

            if(firstInfo.Children.Count == 0) continue;
            foreach(MenuNodeInfo secondInfo in firstInfo.Children)
            {
```

```
                    //如果没有菜单的权限，则跳过
                    if (!Portal.gc.HasFunction(secondInfo.FunctionId)) continue;


                    //添加 RibbonPageGroup（二级菜单）
                    RibbonPageGroup group = new RibbonPageGroup();
                    group.Text = secondInfo.Name;
                    group.Name = secondInfo.ID;
                    page.Groups.Add(group);

                    if(secondInfo.Children.Count == 0) continue;
                    foreach (MenuNodeInfo thirdInfo in secondInfo.Children)
                    {

                        //如果没有菜单的权限，则跳过
                        if (!Portal.gc.HasFunction(thirdInfo.FunctionId))
continue;


                        //添加功能按钮（三级菜单）
                        BarButtonItem button = new BarButtonItem();
                        button.PaintStyle = BarItemPaintStyle.CaptionGlyph;
                        button.LargeGlyph = LoadIcon(thirdInfo.Icon);
                        button.Glyph = LoadIcon(thirdInfo.Icon);

                        button.Name = thirdInfo.ID;
                        button.Caption = thirdInfo.Name;
...................
                        group.ItemLinks.Add(button);
                    }
                }
            }
        }

..............
```

菜单为了方便管理，约定分为 3 级菜单，三个层级的菜单示意图如下所示。

启动顶部的选项卡级别为第一级，下面的 Ribbon 分组为第二级，具体的功能菜单（或者按钮）为第三级，以上就是通过菜单数据动态创建的菜单界面图。

4.3. 框架的用户信息和权限控制

基础框架需要传统的登录进行验证，登录成功后，把用户关联的具有的权限下载到本地，然后由系统逻辑统一判断即可。

插件应用框架系统的登录代码和普通的差别不大，登录后把相关信息存储在框架变量中，如下所示。

为了使框架记录的权限信息、用户数据、以及系统的一些配置信息能够传递到每个插件应用的窗体中，设计了一个插件应用界面需要实现的接口，放在了 BaseUI 项目工程中。

```
namespace WHC.Framework.BaseUI
{

    /// <summary>
    /// 父窗体实现的权限控制接口

    /// </summary>
    public interface IFunction
    {

        /// <summary>
        /// 初始化权限控制信息

        /// </summary>
        void InitFunction(LoginUserInfo userInfo, Dictionary<string, string>
functionDict);


        /// <summary>
        /// 是否具有访问指定控制 ID 的权限

        /// </summary>

        /// <param name="controlId">功能控制 ID</param>

        /// <returns></returns>
        bool HasFunction(string controlId);


        /// <summary>
        /// 登陆用户基础信息

        /// </summary>
        LoginUserInfo LoginUserInfo { get; set; }


        /// <summary>
        /// 登录用户具有的功能字典集合

        /// </summary>
```

```
        Dictionary<string, string> FunctionDict { get; set; }

        /// <summary>
        /// 应用程序基础信息
        /// </summary>
        AppInfo AppInfo { get; set; }

    }
}
```

然后在 BaseUI 的项目中，界面基类 BaseForm 实现这个接口。

```
namespace WHC.Framework.BaseUI
{
    public partial class BaseForm : DevExpress.XtraEditors.XtraForm, IFunction
    {

        public BaseForm()
        {
            InitializeComponent();
        }


.................
```

最后，就是我们如何传递用户信息以及权限信息到窗体本身，传递到窗体作为其本身的变量后，就可以很方便使用这些关键的信息了。

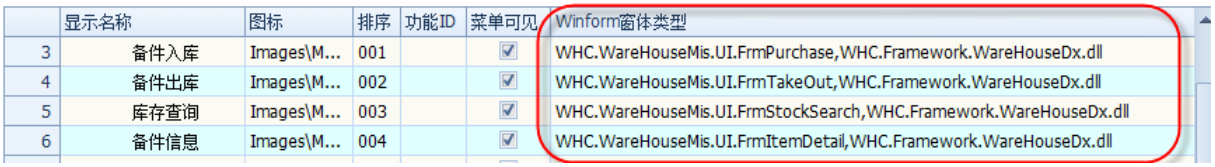
在我们动态加载插件应用的后，我们会创建对应的 Form 对象，然后转换为 IFunction 接口，赋予该接口相关的变量属性即可实现用户信息及权限信息的传递，如下代码所示。

```
      Form tableForm = (Form)Activator.CreateInstance(formType);

      //如果窗体集成了 IFunction 接口(第一次创建需要设置)
      IFunction function = tableForm as IFunction;
      if (function != null)
      {
          //初始化权限控制信息
          function.InitFunction(Portal.gc.LoginUserInfo,  Portal.gc.FunctionDict);

          //记录程序的相关信息
          function.AppInfo = new AppInfo(Portal.gc.AppUnit, Portal.gc.AppName,
                    Portal.gc.AppWholeName, Portal.gc.SystemType);

      }
```

4.4. 插件应用的动态加载

上面说到，只要是实现基于 Form 的，都可以动态创建方式调用显示插件的界面出来，而如果界面实现了 IFucntion 的权限控制接口，那么框架就能够传递给它相应的数据，实现更加完善的控制功能。

在关于权限系统的菜单管理图片中，可以看到了有个 Winform 的窗体类型的字段，里面就是用来动态构造插件的配置信息，这些信息主要是用来构造插件的窗体，并传递给它相关数据即可，下图是菜单管理里面的 “Winform 窗体类型” 信息的具体内容。



	显示名称	图标	排序	功能ID	菜单可见	Winform窗体类型
3	备件入库	Images\M...	001		☑	WHC.WareHouseMis.UI.FrmPurchase,WHC.Framework.WareHouseDx.dll
4	备件出库	Images\M...	002		☑	WHC.WareHouseMis.UI.FrmTakeOut,WHC.Framework.WareHouseDx.dll
5	库存查询	Images\M...	003		☑	WHC.WareHouseMis.UI.FrmStockSearch,WHC.Framework.WareHouseDx.dll
6	备件信息	Images\M...	004		☑	WHC.WareHouseMis.UI.FrmItemDetail,WHC.Framework.WareHouseDx.dll

当系统完成菜单的动态创建后，菜单按钮的响应事件就是触发动态加载插件的事件。在添加菜单的时候，对它的响应事件也做了处理，具体代码如下所示。

```
//添加功能按钮（三级菜单）
BarButtonItem button = new BarButtonItem();
.................
button.Caption = thirdInfo.Name;
button.Tag = thirdInfo.WinformType;
button.ItemClick += (sender, e) =>
{
   if (button.Tag != null && !string.IsNullOrEmpty(button.Tag.ToString()))
   {
         LoadPlugInForm(button.Tag.ToString());
   }
   else
   {
         MessageDxUtil.ShowTips(button.Caption);
   }
 };

group.ItemLinks.Add(button);
```

单击事件的响应处理就是动态构建插件应用的事件，其中就是根据“Winform 窗体类型”的数据进行解析的。

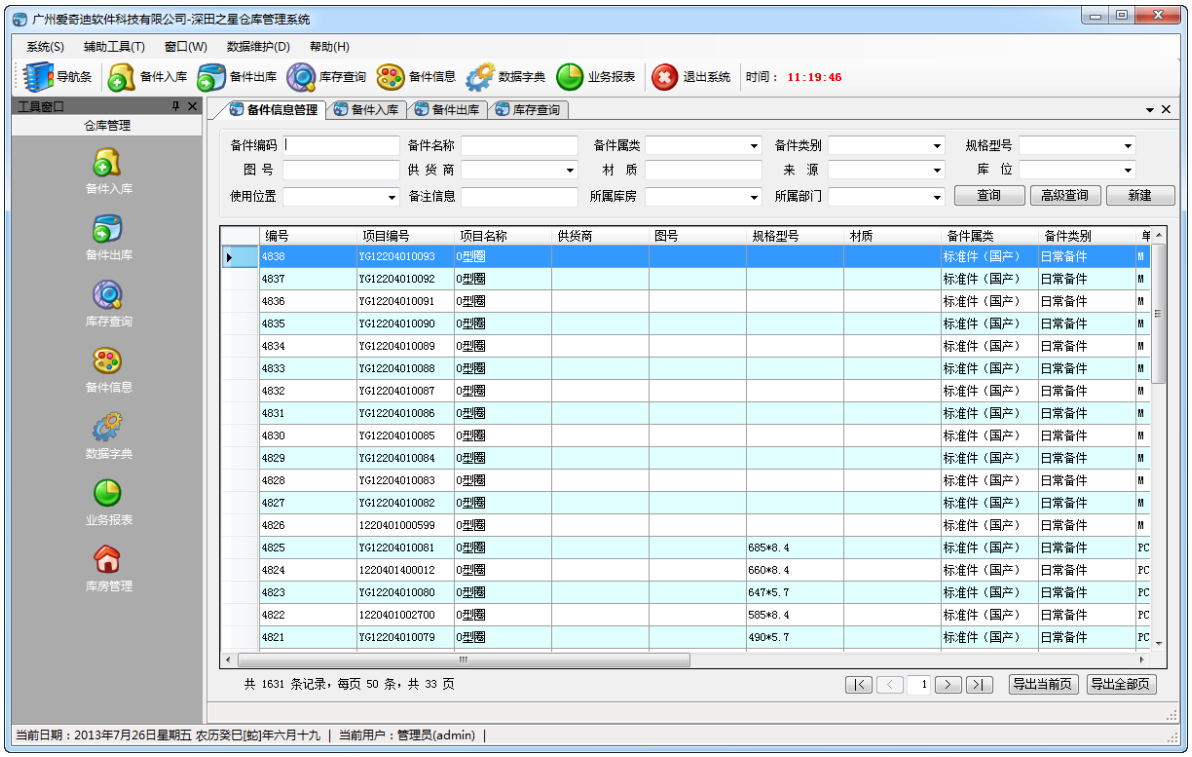
```
string dllFullPath = Path.Combine(Application.StartupPath, filePath);
Assembly tempAssembly = System.Reflection.Assembly.LoadFrom(dllFullPath);
if (tempAssembly != null)
{
   Type objType = tempAssembly.GetType(type);
   if (objType != null)
    {
         LoadMdiForm(this.mainForm, objType, isShowDialog);
    }

}
```

通过动态创建菜单模块，动态加载插件应用，以及权限控制等管理，我们就能隔离框架本身和插件应用模块之间的耦合性关联，所有后续开发或者别人开发的业务模块，都可以很方便的通过权限管理系统配置数据、自动更新模块更新程序应用的方式，把一个高效、易于扩展、动态管理的系统应用弄得丰富多彩，有声有色。

基于插件化应用框架的 Winform 开发框架改造，使得今后开发业务系统，只是基于一定的接口协议，开发插件应用即可，整体性的框架本身可以有专门的人员进行维护，提高团队对业务模块的横向切割和快速开发的效率，更好、统一、高效完成企业化应用框架的搭建和使用。

5. 界面设计

5.1. 经典模式界面设计



基于传统经典模式的界面，采用了 OutLookBar 工具条以及鼎鼎有名的 Weifengluo 多文档布局控件，集成了分页控件、使用基于 Apose.Cell 控件的自定义报表等功能，能适应大多数业务系统的引用。框架数据编辑界面、普通查询窗体界面均采用窗体集成模式，简化开发代码，提高窗体开发效率以及统一界面的一致性。

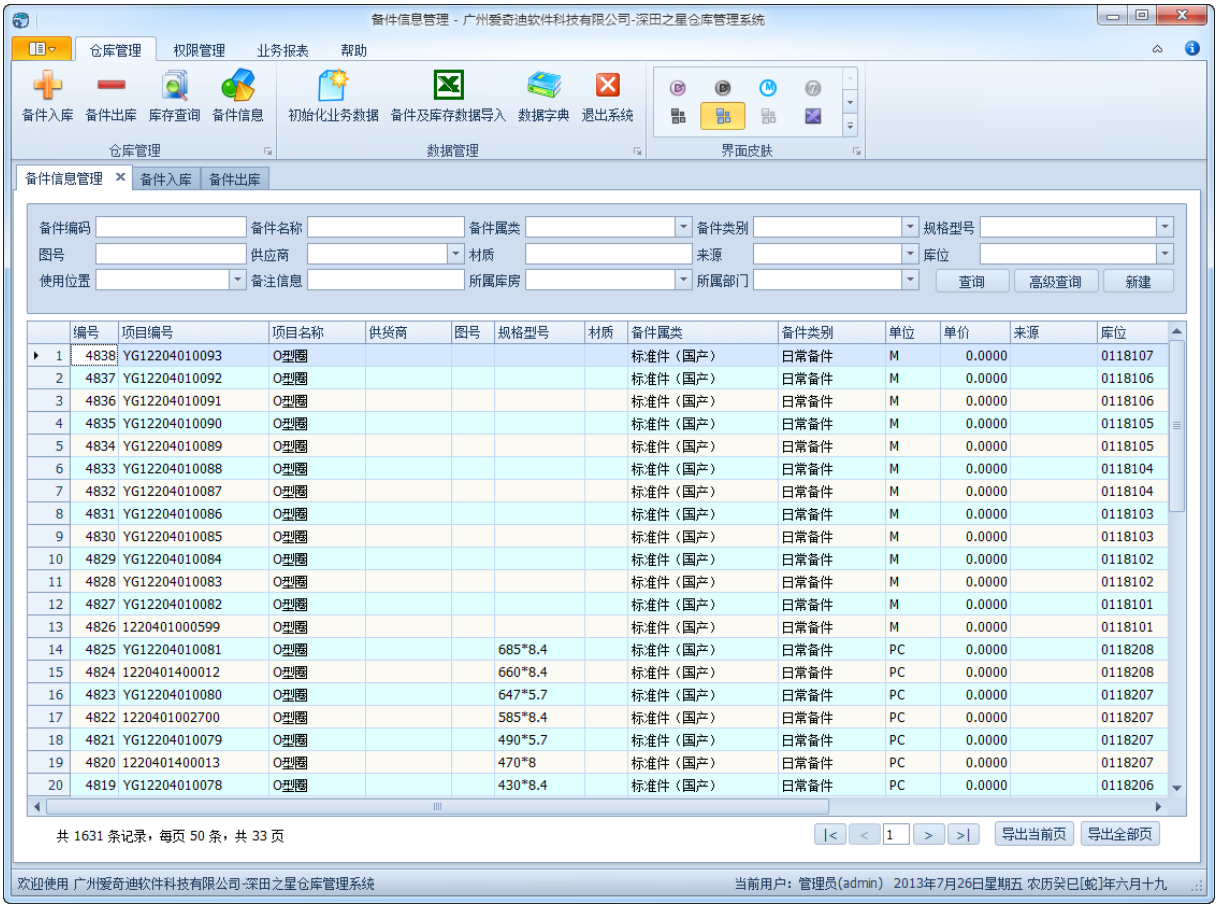
权限管理系统模块、常用字典管理模块，直接在界面调用即可实现管理控制和数据调用处理。

5.2. 基于 DotNetBar 界面设计

在基于传统经典模式的 Winform 框架基础上，引入 DotNetBar 优秀的界面组件，对界面的样式，布局均由很大程度的提升。工具栏统一集中放置在 Ribbon 工具条上，可以折叠分组等，另外也支持多文档的界面操作，非常方便和美观。

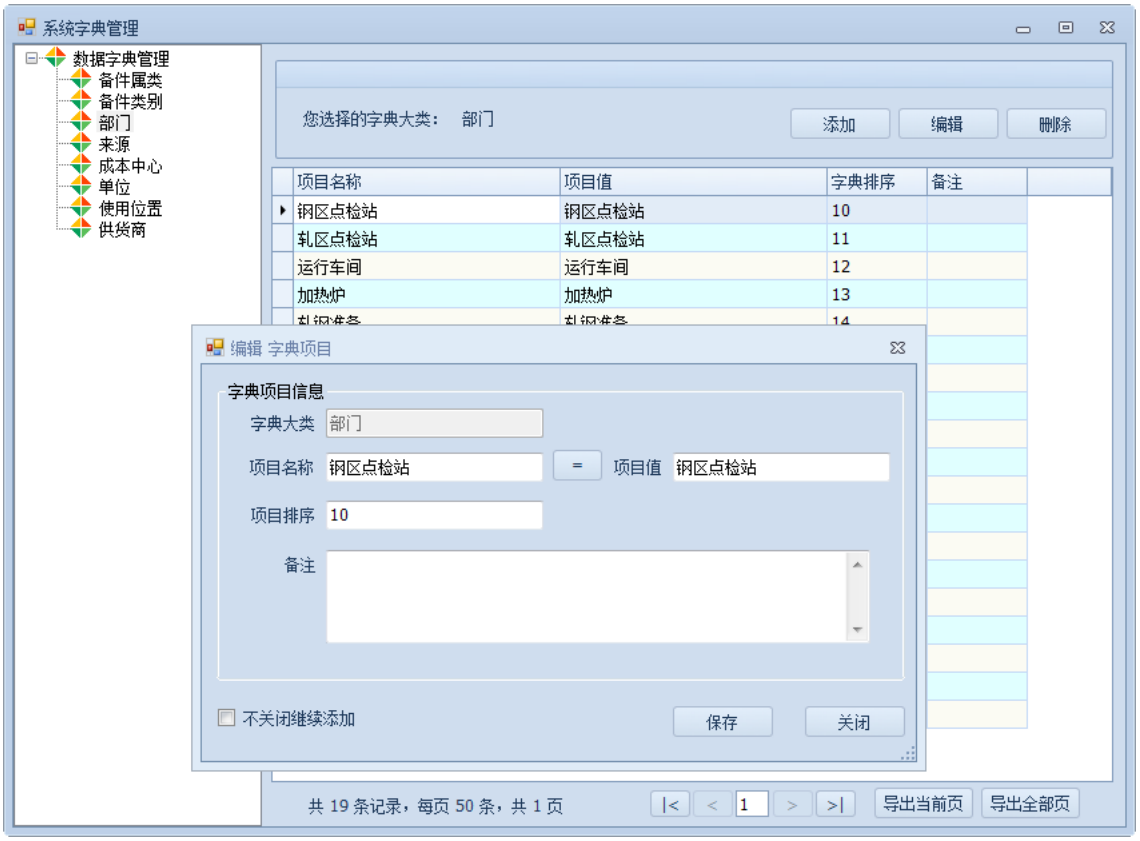
由于整合需要，我为这个框架开发了一套基于 DotNetBar 样式的分页控件，使得界面整合更加一致美观，同时使用上是无缝切换。

5.3. 基于 DevExpress 界面设计

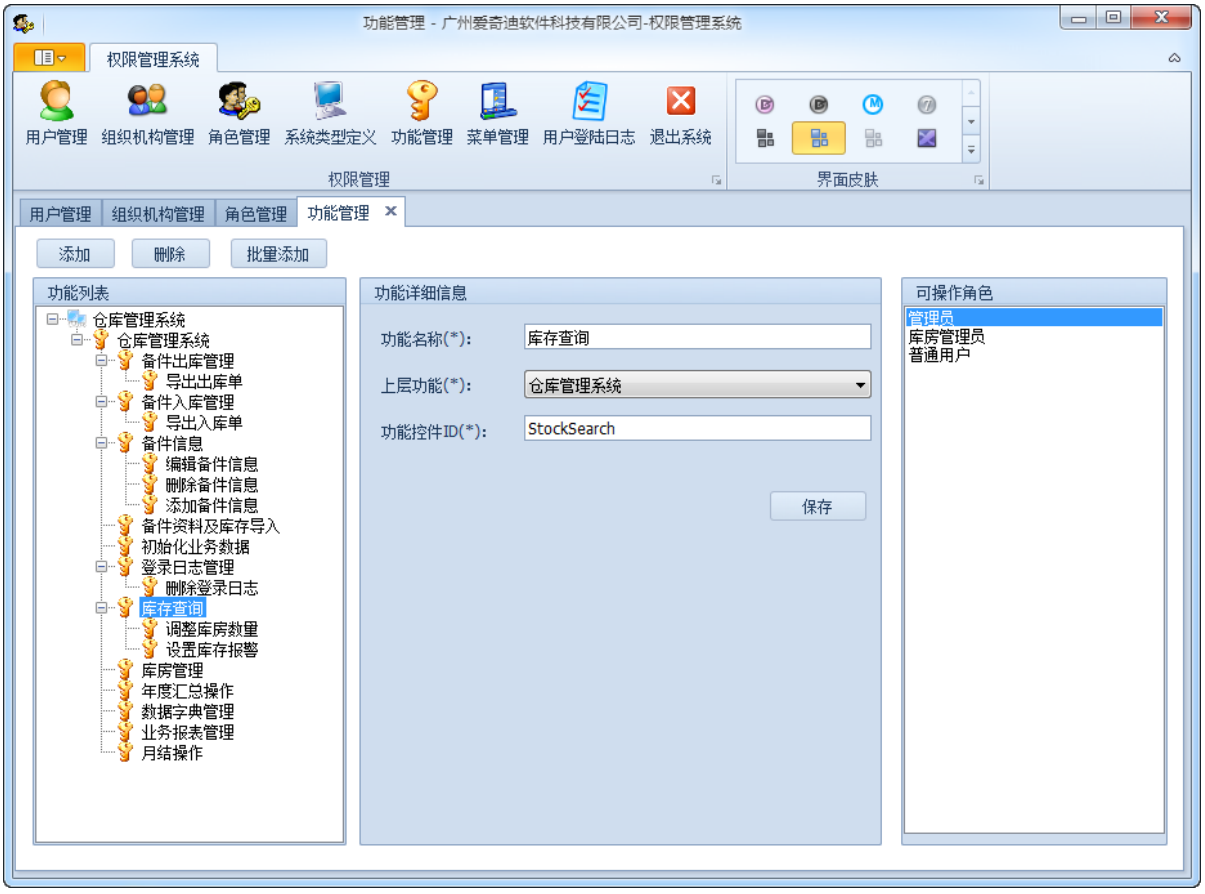


基于 DevExpress 界面设计也在基于传统经典模式的 Winform 框架基础上，引入 DotNet 最为优秀的界面组件 DevExpress，对界面布局、样式等模块提升到一个极高的高度，同时提供该样式的分页控件，使得整合更加完美。

由于 WCF 框架整合了字典模块和权限模块的界面，因此同时更新了界面效果，其中 DevExpress 界面效果的字典模块如下所示。



其中 DevExpess 界面效果的权限控制模块如下所示

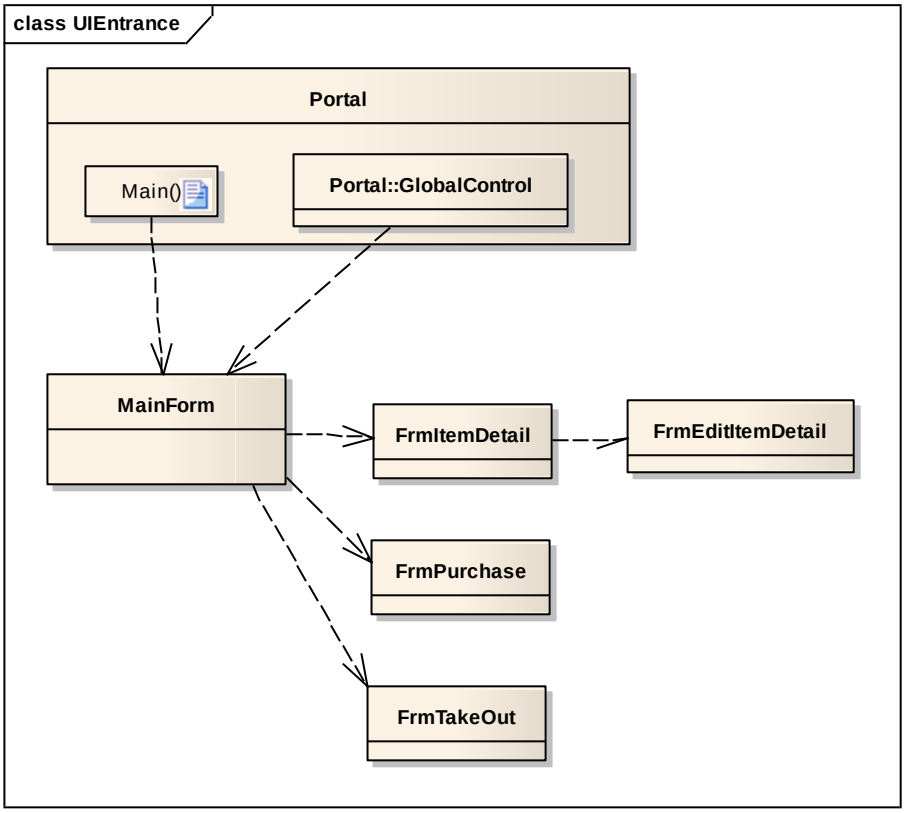


6. 模块详细设计

6.1. 界面层入口类设计

程序入口类是 Portal 类，提供一个 Main 函数，Main 函数将启动一个名为 MainForm 的窗体。另外 Portal 类有一个静态类 GlobalControl，这个类包含一些常用到窗体的模块功能，同时也会放置一些常用到的属性放到这里，这些功能及属性，可以给系统整个生命周期内任何窗体使用。GlobalControl 类还含有 MainForm 类的实例，在 Main 函数启动的时候构造传递过来。

整个 Winform 框架只包含一个 MainForm 窗体，这个是系统的主界面，而主界面则可以包含无数个数据查询显示窗体对象，如 FrmItemDetail、FrmPurchase、FrmTakeOut 等窗体，这些窗体主要通过多文档界面来展示，方便管理而且美观实用。另外每个对应的业务窗体，如 FrmItemDetail 窗体，它可能还会有一些数据编辑、新建数据等窗体，这些窗体类之间的关系如下图所示。



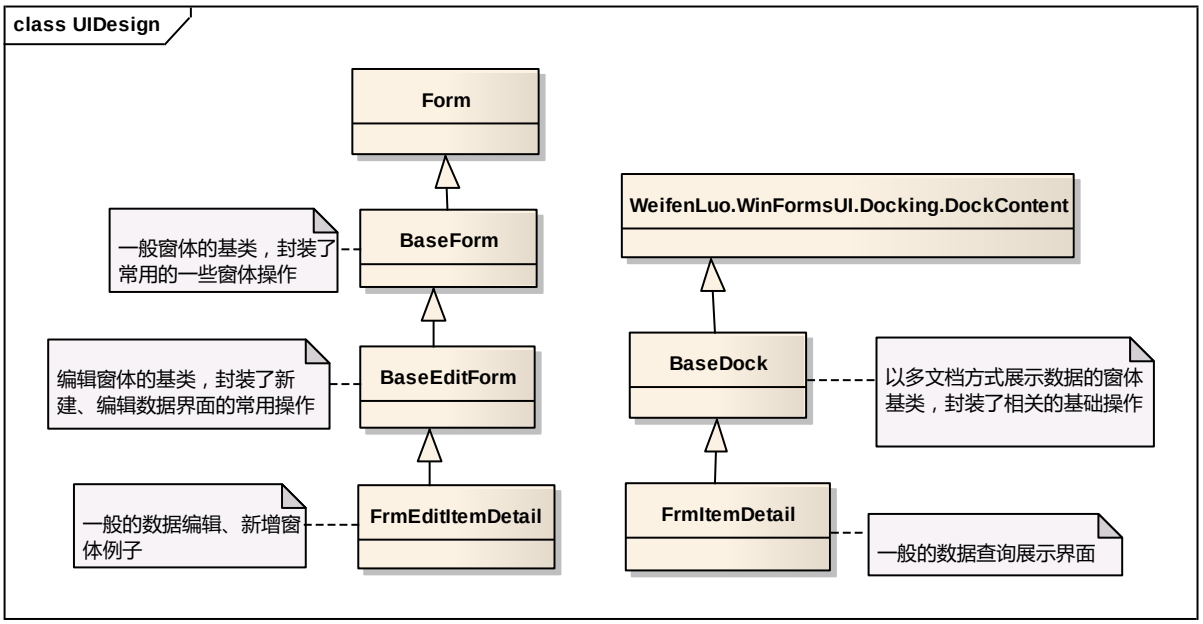
程序入口类关系

6.2. 界面层模块设计

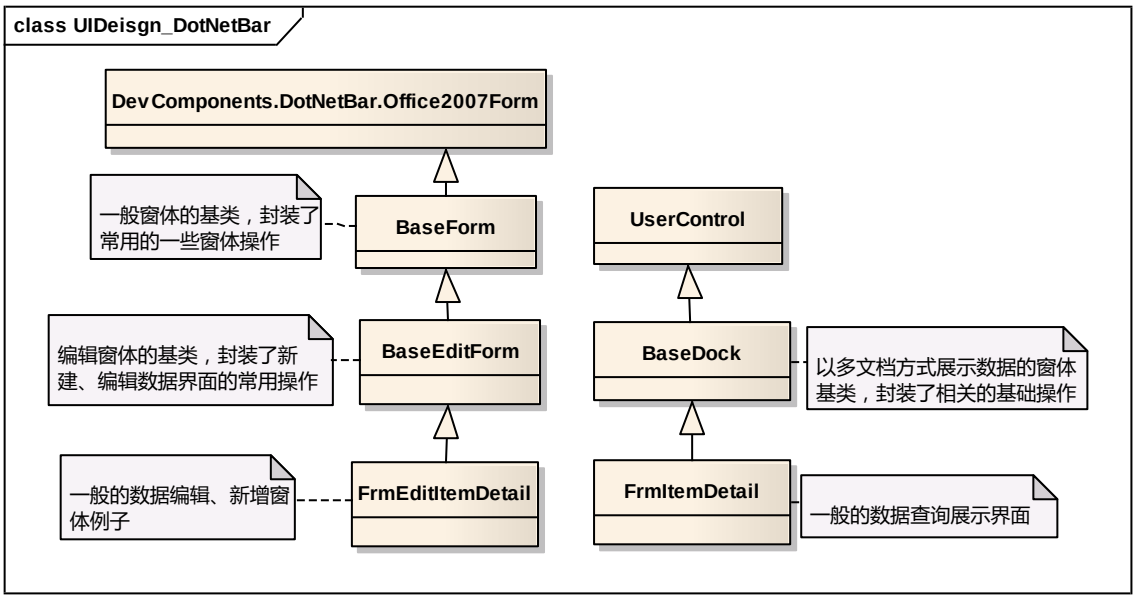
采用窗体继承，极大程度上统一了界面，并且对常用的界面操作，提供了良好的封装，如基础数据编辑、新增窗体积累封装了对回车、方向键、数据刷新、异常处理、数据检查、数据保存、数据更新等接口，为窗体的数据处理提供了很大的方便性。

而数据查询显示窗体则考虑到多文档展示的需要，一般继承合适的基类，封装一些常用到的界面布局，以便实现相应的界面处理效果。

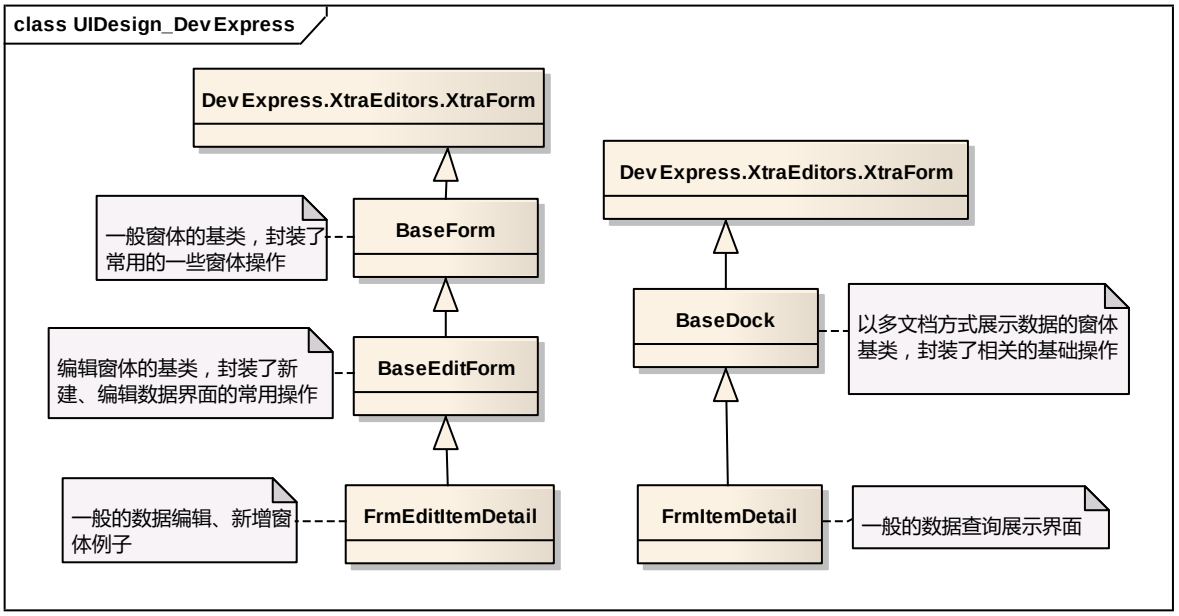
1）基于传统界面的窗体继承



2）基于 DotNetBar 界面的窗体继承



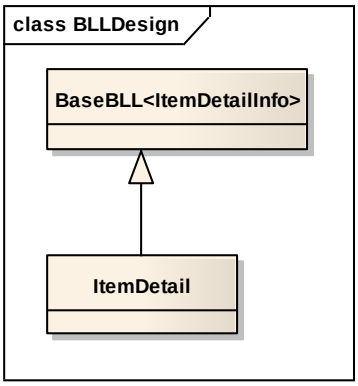
3）基于 DevExpress 界面的窗体继承



6.3. 业务层模块设计

6.3.1. 业务层继承关系

具体的备件管理业务类 ItemDetail 继承自 BaseBLL 业务基类，该基类封装了对数据对象的常用操作，通过给基类传递一个对应的业务实体类，从而实现具体业务逻辑的处理。



业务规则基类 BaseBLL，它是采用了泛型和和反射的方式来实现，另外通过不同的数据库配置信息，从不同的程序集中构建相应的数据库访问，实现多种数据库访问的支持及切换。虽然在实际项目开始后，切换数据库较少，但对于 Winform 开发框架，能开发不同的数据库应用，这个也是很重要，我们只需要提供相应数据库访问层就可以了。

```
public class BaseBLL<T> where T : BaseEntity, new()
{
    private string dalName = "";
    protected IBaseDAL<T> baseDal = null;

    public BaseBLL() : this("")
    {
    }

    public BaseBLL(string dalName)
    {
        #region 根据不同的数据库类型，构造相应的DAL层
        AppConfig config = new AppConfig();
        string dbType = config.AppConfigGet("ComponentDbType");
        if (string.IsNullOrEmpty(dbType))
        {
            dbType = "sqlserver";
        }
        dbType = dbType.ToLower();

        string DALPrefix = "";
        if (dbType == "sqlserver")
        {
            DALPrefix = "DALSQL.";
        }
        else if (dbType == "access")
        {
            DALPrefix = "DALAccess.";
        }
        else if (dbType == "oracle")
        {
            DALPrefix = "DALOracle.";
        }
        #endregion

        if (string.IsNullOrEmpty(dalName))
        {
            this.dalName = GetType().FullName.Replace("BLL.", DALPrefix);
        }
        else
        {
            this.dalName = dalName;
        }

        baseDal = Reflect<IBaseDAL<T>>.Create(this.dalName, "WHC.WareHouseMis");
    }
```

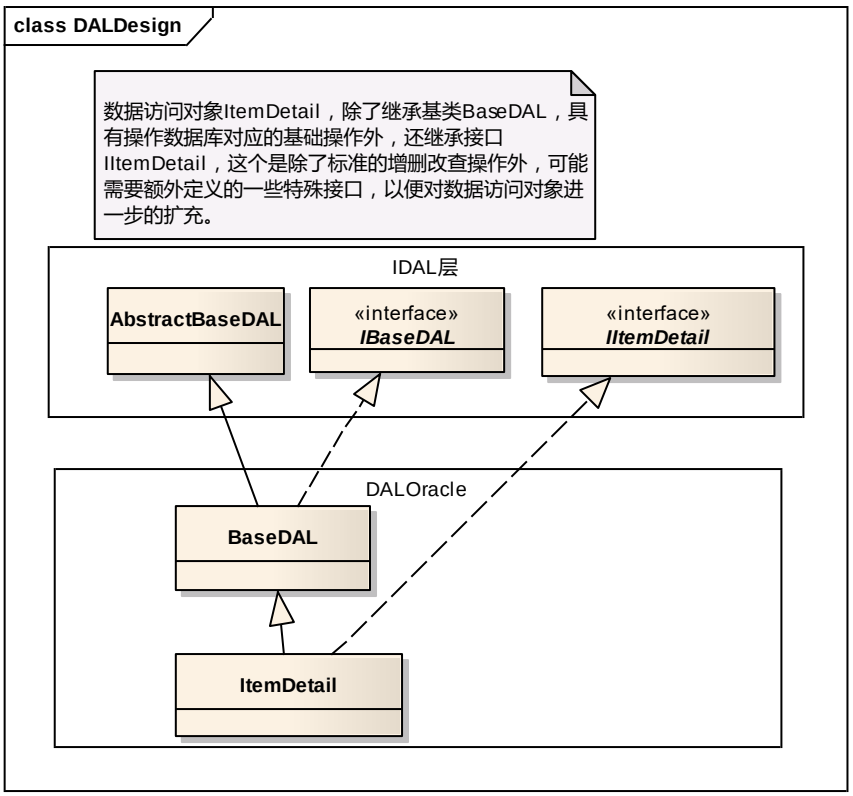
在界面层，如果需要实例化一个业务对象，只需要通过一个业务类构造工厂类进入即可，简单易记，而且统一。传递给 BLLFactory 对象的参数是对应具体的业务实体类，该工厂类将负责构造该业务类相关的对象资源，使用代码如下所示。

```
ItemDetailInfo info = BLLFactory<ItemDetail>.Instance.FindByID(ID);
```

整个框架是面向对象的数据处理方式，所有的业务类通过类似 BLLFactory<ItemDetail>.Instance 出来的对象都是强类型的，具有所有 ItemDetail 业务类的一切智能提示，非常方便操作，而不是一个通用的数据库操作类。使用业务实体类来传递数据有很多方便之处，不需要记住各种字段名称，而且在业务操作中也不容易出错，当然整体框架也提供了基于 DataTable 等数据信息的返回，以适应一些特殊的情况。

6.3.2. 数据访问层继承关系

数据访问层的对象，除了继承自基类 BaseDAL 外，还实现自己的特殊接口（如果需要），这样可以很好对数据访问对象的基本操作功能和扩展操作功能实现分离，并且有很好的扩展性。



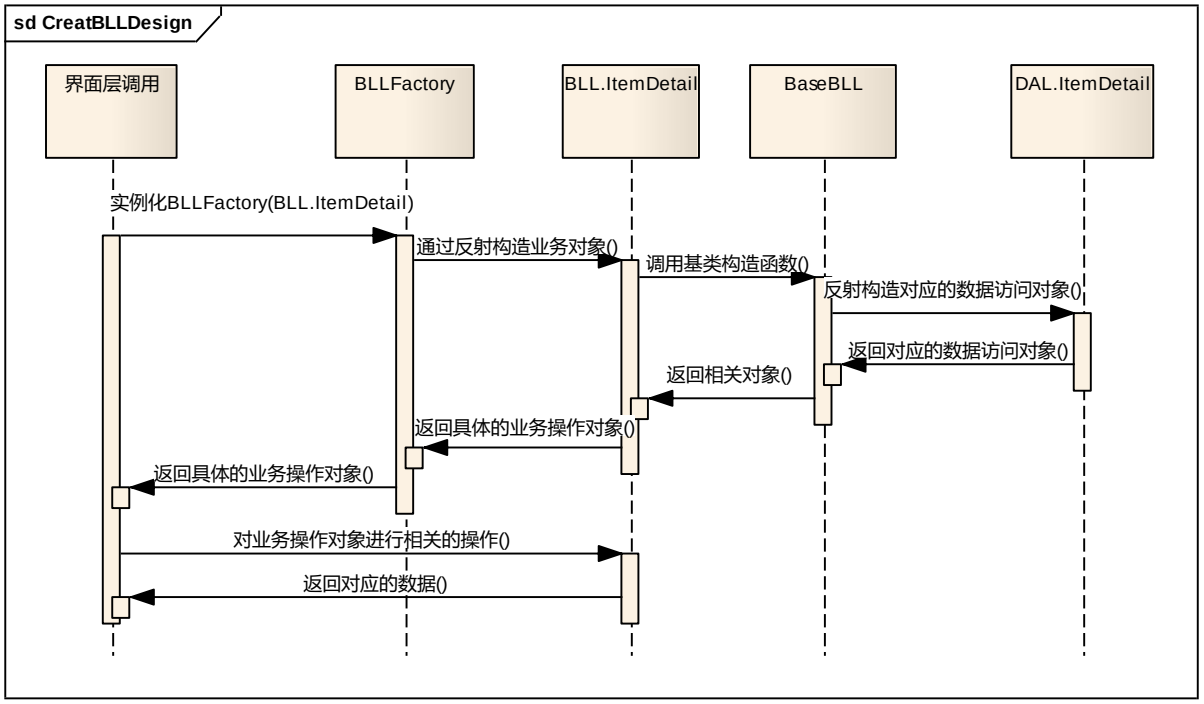
不过由于基类采用了泛型的封装，在构建对象的时候，属于强类型的对象，智能提示也比较好。数据访问基类 BaseDAL 的代码设计所示。

```
/// <summary>
/// 数据访问层的基类
/// </summary>
public abstract class BaseDAL<T> : IBaseDAL<T> where T : BaseEntity, new()
{
……
}
```

BaseDAL 基类封装了各种数据库操作方法（几乎能满足各种要求的方法集合），因此具体的数据库访问对象，基本上不需要做数据访问的代码编写了，如果需要额外的数据访问接口，也只需要做一些扩展就可以了。

6.3.3. 操作对象创建逻辑

对象构造及接口调用的序列图如下图所示。



6.3.4. 基于事务的处理

在事务管理方面，框架提供了很好的支持，几乎每个方法都有事务参数接口，一旦使用事务，所有的方法均需要使用事务对象。事务对象在保证操作完整性方面，提供了很好的支持，在一些原子性的逻辑里面，事务必不可少。
一般如果涉及到多个逻辑操作的，可以考虑使用事务处理，事务处理通过业务对象的基类或数据访问层的基类创建，一般在业务层进行组装各种不同的事务逻辑。

```
/// <summary>
/// 删除订单及订单明细信息
/// </summary>
/// <param name="id"></param>
/// <returns></returns>
public bool DeleteOrderRelated(string id)
{
    bool result = false;
    DbTransaction trans = CreateTransaction();
    if (trans != null)
    {
        SellInfo info = baseDal.FindByID(id, trans);
        if (info != null)
        {
            List<OrderDetailInfo> detailList = BLLFactory<OrderDetail>.Instance.FindByOrderNo(info.OrderNo, trans);
            foreach (OrderDetailInfo detailInfo in detailList)
            {
                BLLFactory<OrderDetail>.Instance.Delete(detailInfo.ID, trans);
            }

            //最后删除主表订单数据
            baseDal.Delete(id, trans);

            try
            {
                trans.Commit();
                result = true;
            }
            catch (Exception ex)
            {
                trans.Rollback();
                LogTextHelper.Error(ex);
                throw;
            }
        }
    }
    return result;
}
```

7. 接口设计

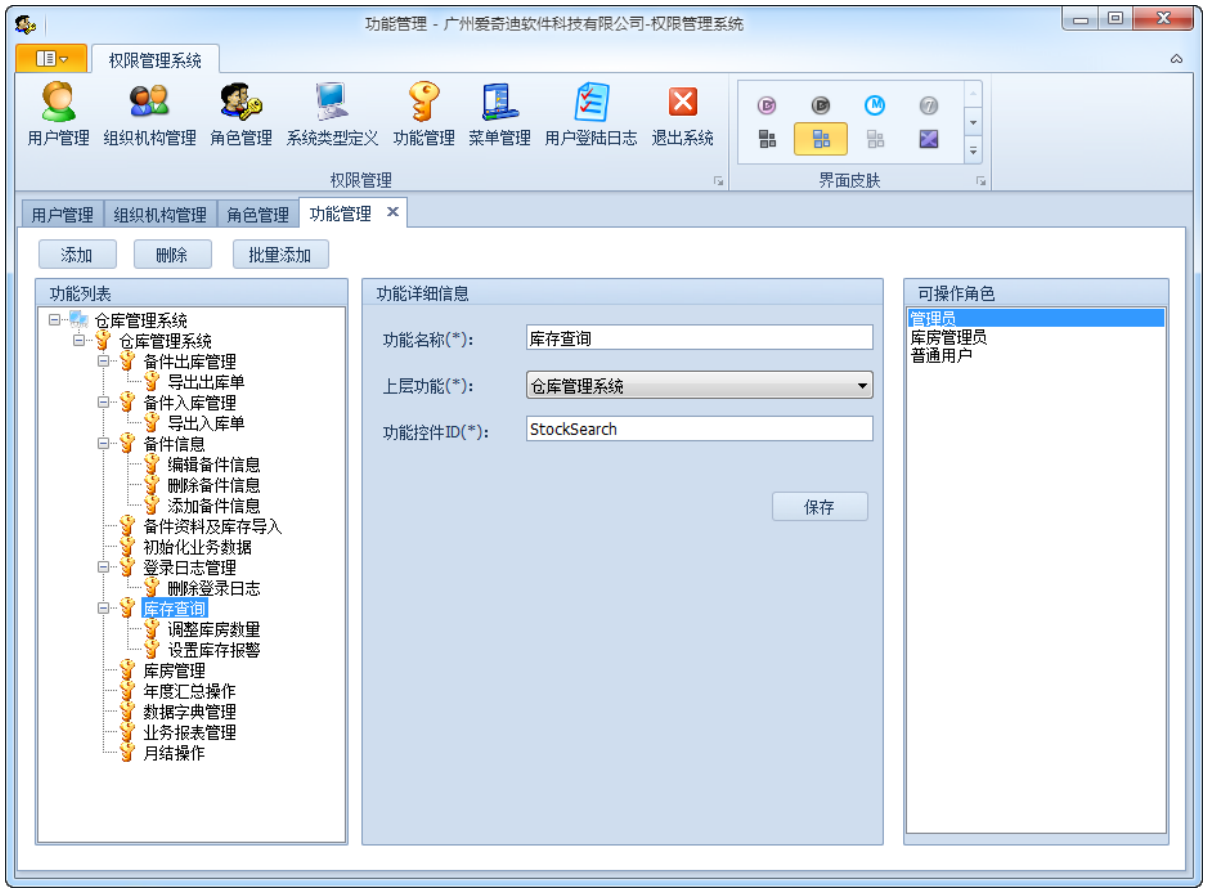
在 Winform 框架中，其中权限管理系统、字典管理系统，都是可以做成独立的程序来使用，而且应该可以在程序中引用来查询或者获取相关的字典数据，如找某个键值的字典列表作为下拉列表，而且由于实际项目中，有的是 SqlServer、有的是 Access 数据库的或者其他数据库，所以支持多数据库是最好的选择。

通用权限管理系统具有和业务系统既独立又有整合性特点，既相互独立，有相互整合，方便重用，又不需重新开发，非常方便、更提高效率。 由于权限系统精简而又能满足日常绝大多数的需要，不会复杂的难于管理，而且也是基于角色的授权访问机制（RBAC），最重要是非常适合软件的整合使用。

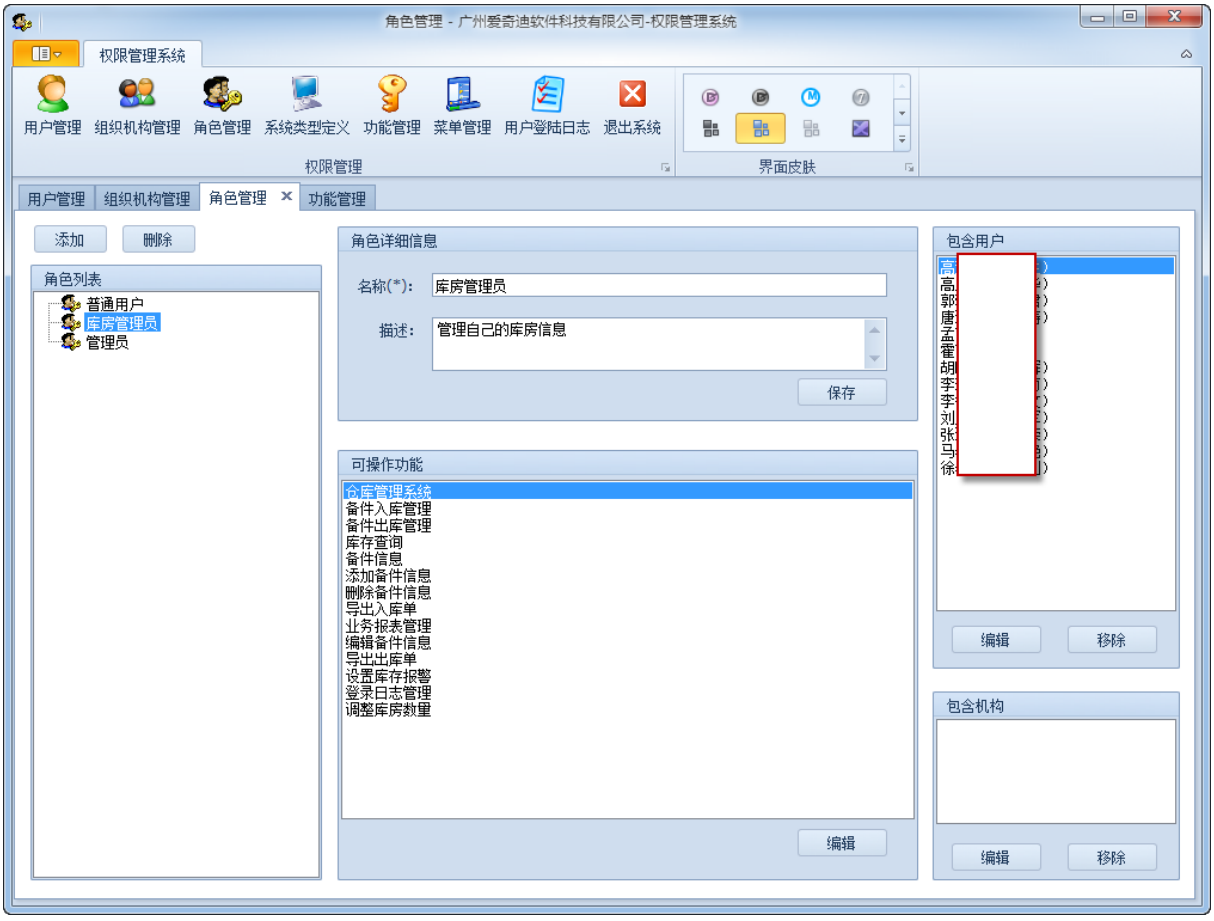
权限管理系统作为一个独立的模块，其主要由登陆界面、权限管理主界面（管理用户、角色、机构、功能，以及控制角色的权限等操作功能

7.1. 权限系统接口

权限系统登录


功能管理 - 广州爱奇迪软件科技有限公司-权限管理系统
权限管理系统
用户管理
组织机构管理
角色管理
系统类型定义
功能管理
菜单管理
用户登陆日志
退出系统
权限管理
界面皮肤
用户管理
组织机构管理
角色管理
功能管理
添加
删除
批量添加
功能列表
仓库管理系统
仓库管理系统
备件出库管理
导出出库单
备件入库管理
导出入库单
备件信息
编辑备件信息
删除备件信息
添加备件信息
备件资料及库存导入
初始化业务数据
登录日志管理
删除登录日志
库存查询
调整库房数量
设置库存报警
库房管理
年度汇总操作
数据字典管理
业务报表管理
月结操作
功能详细信息
功能名称(*):
库存查询
上层功能(*):
仓库管理系统
功能控件ID(*):
StockSearch
保存
可操作角色
管理员
库房管理员
普通用户

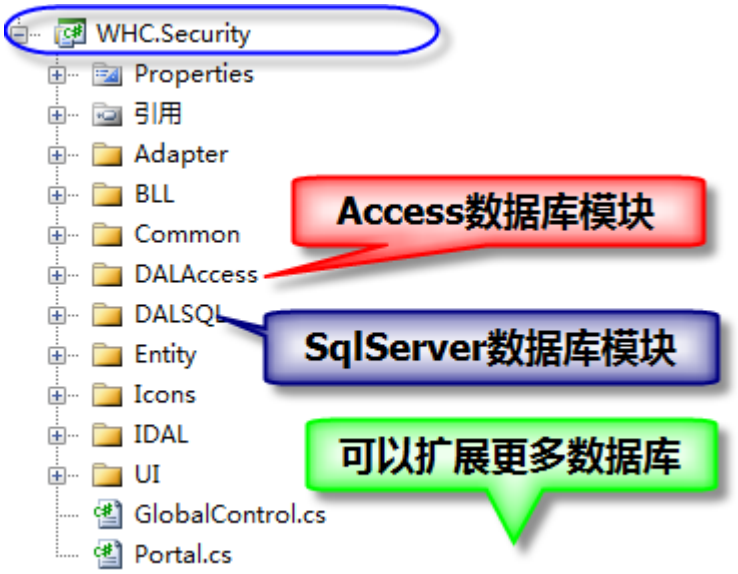
权限系统功能定义


角色管理 - 广州爱奇迪软件科技有限公司-权限管理系统
权限管理系统
用户管理
组织机构管理
角色管理
系统类型定义
功能管理
菜单管理
用户登陆日志
退出系统
权限管理
界面皮肤
用户管理
组织机构管理
角色管理
功能管理
添加
删除
角色列表
普通用户
库房管理员
管理员
角色详细信息
名称(*):
库房管理员
描述:
管理自己的库房信息
保存
可操作功能
仓库管理系统
备件入库管理
备件出库管理
库存查询
备件信息
添加备件信息
删除备件信息
导出入库单
业务报表管理
编辑备件信息
导出出库单
设置库存报警
登录日志管理
调整库房数量
编辑
包含用户
编辑
移除
包含机构
编辑
移除

权限系统角色权限分配

7.1.1. 项目视图

经过优化后的权限管理系统，界面及业务逻辑封装到一个类库中，我们开发的业务管理系统中集成就很简单了，主要的项目工程界面如下所示：



7.1.2. 系统调用代码

权限系统模块的相关的代码很简单，这也是利用权限管理系统后简化很多代码的根本所在：

```
        private void Form1_Load(object sender, EventArgs e)
        {
            //获取所有权限管理系统的用户，并在下拉列表中展示
            List<UserInfo> userList = BLLFactory<User>.Instance.GetAll();
            this.txtLogin.Items.Clear();
            foreach (UserInfo info in userList)
            {
                this.txtLogin.Items.Add(info.Name);
            }
        }

        private void btnSecurity_Click(object sender, EventArgs e)
        {
            //独立启动权限管理系统，只需一行代码即可
            WHC.Security.UI.Portal.StartLogin();
        }
```

权限控制的精髓就是，用户登录后，通过把用户拥有的权限获取出来，放到一个功能列表中，然后在每一个窗体中，根据用户的功能列表，显示或者屏蔽对应的功能即可，获取功能列表代码如下所示：

```
UserInfo info = userBLL.GetUserByName(loginName);

#region 获取用户的功能列表
 Function functionBLL = new Function();
```

```
 List<FunctionInfo> list = functionBLL.GetFunctionsByUser(info.ID, "WareMis
");
 if (list != null && list.Count > 0)
 {
     foreach (FunctionInfo functionInfo in list)
     {
        if (!Portal.gc.FunctionDict.ContainsKey(functionInfo.ControlID))
         {
           Portal.gc.FunctionDict.Add(functionInfo.ControlID, functionInfo);
         }
     }
 }
#endregion
```

判断的时候，放在一个函数，判断用户访问的功能是否在列表中即可，代码如下：

```
        /// <summary>
        /// 看用户是否具有某个功能
        /// </summary>
        /// <param name="controlID"></param>
        /// <returns></returns>
        public bool HasFunction(string controlID)
        {
            bool result = false;
            if (FunctionDict.ContainsKey(controlID))
            {
                result = true;
            }

            return result;
        }
```

7.2. 数据字典接口

7.2.1. 调用代码

字典数据模块做成独立的程序后，一个可以独立运行，也可以在宿主程序中通过 DLL 方式调用类库来获取字典数据，如下所示：

```
        private void tool_Dict_Click(object sender, EventArgs e)
        {
            FrmDictionary dlg = new FrmDictionary();
            dlg.ShowDialog();
        }
```

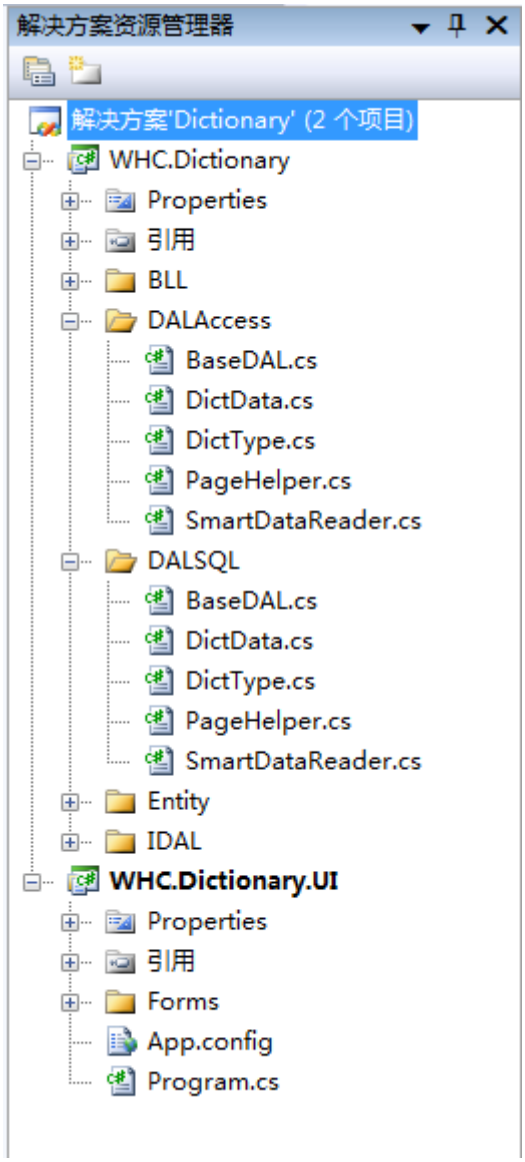
调用字典类库获取对应的字典内容，代码如下所示：

```
    private void InitDictItem()
    {
      this.txtManufacture.Items.Clear();
      this.txtManufacture.Items.AddRange(DictItemUtil.GetDictByDictType("供货商"));

      this.txtBigType.Items.Clear();
      this.txtBigType.Items.AddRange(DictItemUtil.GetDictByDictType("备件属类"));

      this.txtItemType.Items.Clear();
      this.txtItemType.Items.AddRange(DictItemUtil.GetDictByDictType("备件类别"));

      this.txtSource.Items.Clear();
      this.txtSource.Items.AddRange(DictItemUtil.GetDictByDictType("来源"));

      this.txtWareHouse.Items.Clear();
      this.txtWareHouse.Items.AddRange(DictItemUtil.GetAllWareHouse().ToArray());
```

```
        this.txtDept.Items.Clear();
        this.txtDept.Items.AddRange(DictItemUtil.GetDictByDictType("部门"));
    }
```

7.2.2. 项目视图

字典也是一个小型的框架系统，它本身也支持多数据库的配置，另外由于它的功能模块相对独立和通用，一般是把它作为一个独立模块进行使用，表和业务表放在一起就可以了，他的项目视图如下所示。



8. 通用自动更新模块

8.1. 自动更新模块介绍

通用自动更新模块，它根据客户端软件版本和服务器的版本对比，自动提示客户对版本进行更新，从服务器下载更新文件解压覆盖现有文件，并自动重启软件。软件支持文件 Zip 解压缩，支持进度更新过程，支持参数化启动等。

下面是我的 Winform 开发框架中集成通用自动更新模块的截图，如下所示。

更新过程中会先关闭主程序，把更新的 Zip 文件下载后进行自动解压，然后启动主程序。

8.2. 自动更新模块的使用

自动更新一般需要设置一些参数，如程序标题、更新地址路径、版本号等，本通用自动更新客户端配置文件的 XML 文件 updateconfiguration.config 内容如下所示。

updateconfiguration.config

```
<?xml version="1.0" encoding="utf-8"?>
<applicationUpdater applicationId="c60ed4af-74bf-40db-b444-736c0832a3d7"
                    manifestUri="http://localhost/iqidi/Client/WHC.WareHouseMis.DxUI.xml"
                    version="2.0.0"
                    title="深田之星仓库管理系统-版本更新">
</applicationUpdater>
```

服务端的配置文件 WHC.WareHouseMis.DxUI.xml 内容如下所示。

WHC.WareHouseMis.DxUI.xml

```xml
<?xml version="1.0" encoding="utf-8" ?>
<manifest>
  <version>3.0.0</version>
  <description>V3.0.0 本版本更新了系统显示布局及部分功能的完善，修正一些小Bug。</description>
  <!--applicationId运用程序ID，需要与客户端配置一样，否则不会进行更新-->
  <myapplication applicationId="c60ed4af-74bf-40db-b444-736c0832a3d7">
    <!--暂时没有使用，重新启动exe名称，parameters启动时传入的参数-->
    <entryPoint file="WHC.WareHouseMis.DxUI.exe" parameters="-S" />
    <!--更新文件的目录，相对于更新系统，默认为同级目录-->
    <location>.</location>
  </myapplication>
  <!--base表示存放该文件的url-->
  <files base="http://localhost/iqidi/Client/">
    <!--文件名称-->
    <file source="WHC.WareHouseMis.DxUI.zip" unzip="true" />
  </files>
</manifest>
```

自动更新虽然可以独立进行执行并更新，不过一般会在主程序中加入对自动更新的判断（毕竟使用客户大多数都是会运行主程序的），实现自动更新判断及执行，我们可以把其放到一个后台线程中执行判断，这样可以提供用户的体验，不会中断界面操作。

我一般倾向于把自动更新放到登录界面的首页上，这样用户每次登录的时候，可选择性进行更新，登录后一般要进行业务操作，如果更新退出可能会导致客户的一些重要数据没有保存而丢失，这样影响不好。

```
    #region 更新提示/判断是否自动更新
    updateWorker = new BackgroundWorker();
    updateWorker.DoWork += new DoWorkEventHandler(updateWorker_DoWork);
    updateWorker.RunWorkerCompleted += new RunWorkerCompletedEventHandler(updateWorker_RunWorkerCompleted);

    string strUpdate = config.AppConfigGet("AutoUpdate");
    if (!string.IsNullOrEmpty(strUpdate))
    {
        bool autoUpdate = false;
        bool.TryParse(strUpdate, out autoUpdate);
        if (autoUpdate)
        {
            updateWorker.RunWorkerAsync();
        }
    }
    #endregion
}

#region 更新提示线程处理
private void updateWorker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)...

private void updateWorker_DoWork(object sender, DoWorkEventArgs e)
{
    try
    {
        UpdateClass update = new UpdateClass();
        bool newVersion = update.HasNewVersion;
        if (newVersion)
        {
            if (MessageUtil.ShowYesNoAndTips("有新的版本，是否需要更新") == DialogResult.Yes)
            {
                Process.Start(Path.Combine(Application.StartupPath, "Updater.exe"), "121");
                Application.Exit();
            }
        }
    }
    catch (Exception ex)
    {
        MessageUtil.ShowError(ex.Message);
    }
}

#endregion
```

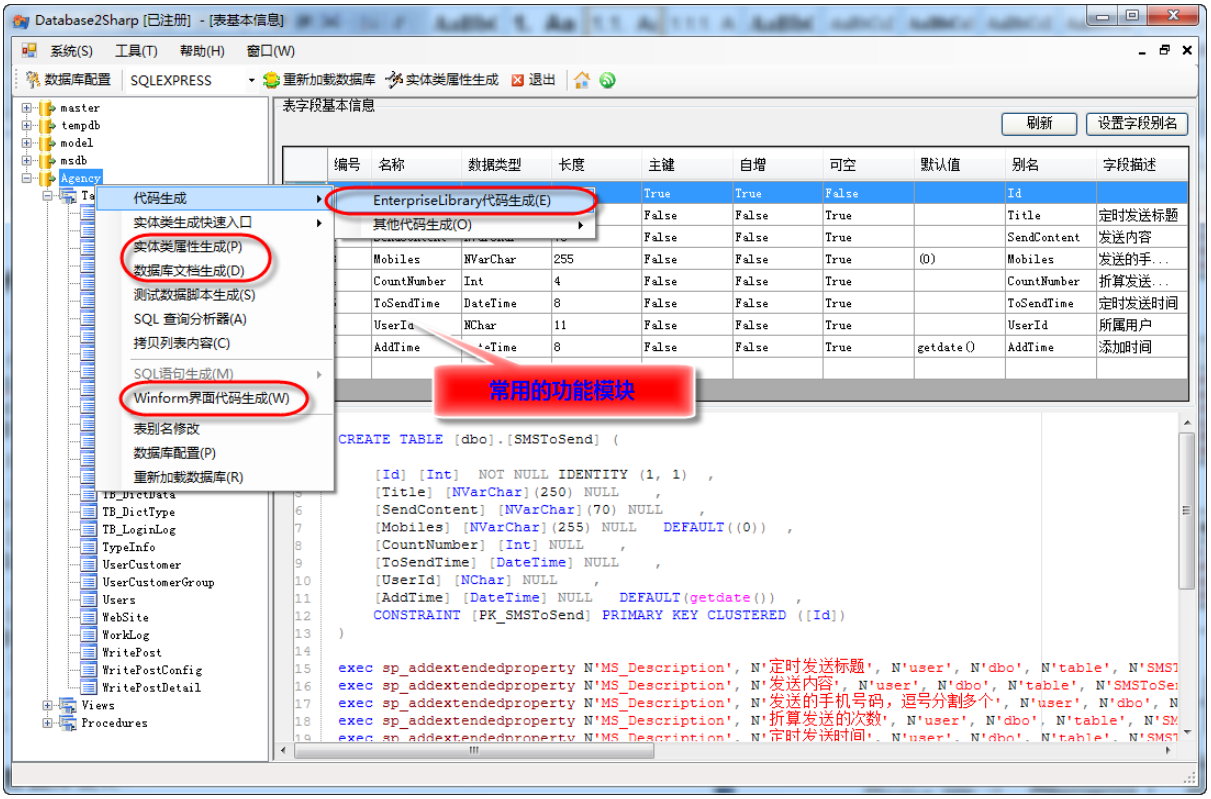
9. 代码生成工具的使用

9.1. 代码工具介绍

代码生成工具 Database2Sharp，是一个主要基于 Enterprise Library 和 PetShop 架构的 C#代码生成工具，提供了对 MS Sql2000、MS Sql2005、Oracle、Mysql、Access 的支持；可以生成各种架构代码，导出数据库文档、浏览数据库架构、查询数据、生成 Sql 脚本等。

Enterprise Library 代码生成，生成整个项目工程框架，包含实体类、数据访问类、业务类、Asp.net 页面类，利用泛型及缓存机制，良好的架构极大简化代码，强大完善的基类使你甚至不用编写一行代码。

Database2Sharp 采用了方便灵活、功能强大的 NVelocity 模板引擎作为代码生成的一部分，你可以程序目录中修改相应的代码模板，实现部分自定义的代码生成。Database2Sharp 除了可以生成主体的 Winform 开发框架源码外，还可以生成数据库设计文档、实体类快速生成、各种样式的 Winform 界面代码（逻辑处理代码）生成，极大方便您的开发工作。



9.2. 使用代码工具生成框架代码

使用 Database2Sharp 来生成框架代码，虽然直接生成的代码，就是一个整体方案的代码，基本上可以直接运行。不过购买了该 Winform 开发框架后，利用该工具主要是增量开发，不需要从头来过。下面提供几个注意的地方。

1）代码生成工具生成的代码是基于 Project 的，而 Winform 开发框架为了项目数量，方便管理，是把业务层、数据访问层、数据接口层、实体层放到一个工程项目中了，因此生成的代码我们复制到对应的目录位置就可以了，默认命名空间不需要改动。

2）为了代码生成方便，代码生成工具需要把数据库字段的中文说明作为代码注释或者说明的一部分，因此，设计数据库（SqlServer、Oracle 等）的时候，我们强烈要求把注释添加到字段说明里面去。

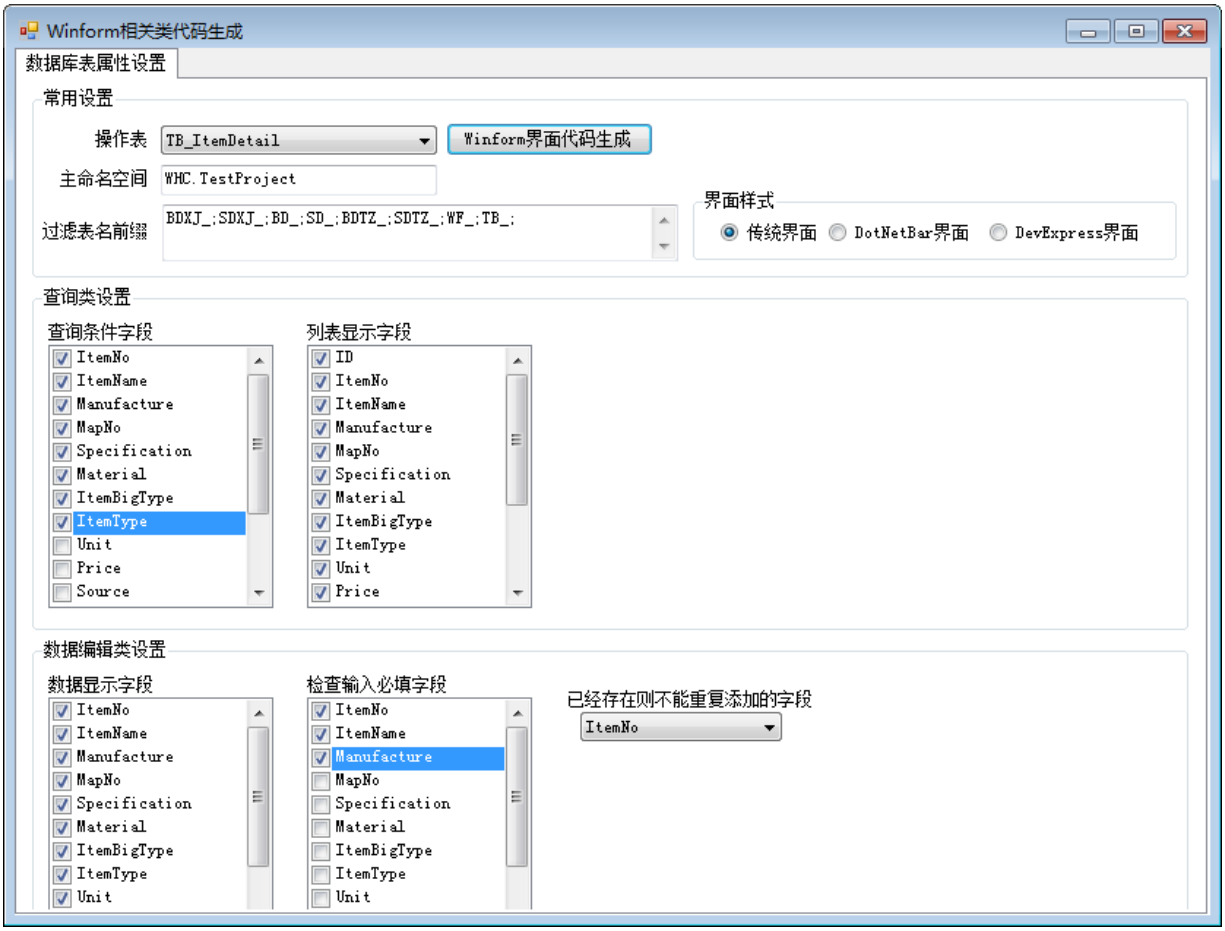
3）数据库表一般需要提供一个主键关键字（建议取名为 ID），主键字段可以为自增长的整形类型，也可以是任意字符型。建议 SqlServer 一般采用自增长整形、Oracle 采用 Number 类型，并为每个表指定一个部分同名的序列名称，如 Seq_ABC，其中 ABC 代表对应的表名。

9.3. 使用代码生成工具生成 Winform 界面代码

该代码工具除了生成整体项目源码外，还提供生成对应表的 Winform 界面处理代码。由于界面布局部分要好看，设计一般要自己多次调整，因此工具只是生成较为稳定的逻辑处理代码。提供了 Winform 开发框架中常见的传统界面、DotNetBar 界面、DevExpress 界面的代码生成。

说起 Winform 界面的代码生成，一般来说就两种界面比较典型，一个是查询列表显示界面，一个是数据查看编辑界面。本功能也主要是提供这两类界面代码的生成，通过配置查询列表中的条件字段以及查询列表字段显示信息，就可以合理生成符合我的

WInform 框架要求的界面代码，查询列表显示界面类继承自 BaseDock 基础类。另一方面，通过配置数据查看编辑界面的编辑字段，数据检查字段，判断关键数据重复的字段等参数，可以生成较为完善的数据查看编辑界面代码，生成界面如下所示。



Winform 界面代码生成后，会直接在代码编辑窗体中打开，用户可以复制或者保存起来放到 VS 的编辑器中进行相应的修改，后续的工作应该较为轻松了。

9.4. 使用代码生成工具生成数据库设计文档

除了生成较高质量的 C#项目代码外，工具还提供了一个非常不错的数据库文档的生成，方便数据库定型后，统一生成数据库设计文档，然后再在此基础上做一些修饰就非常不错的了。

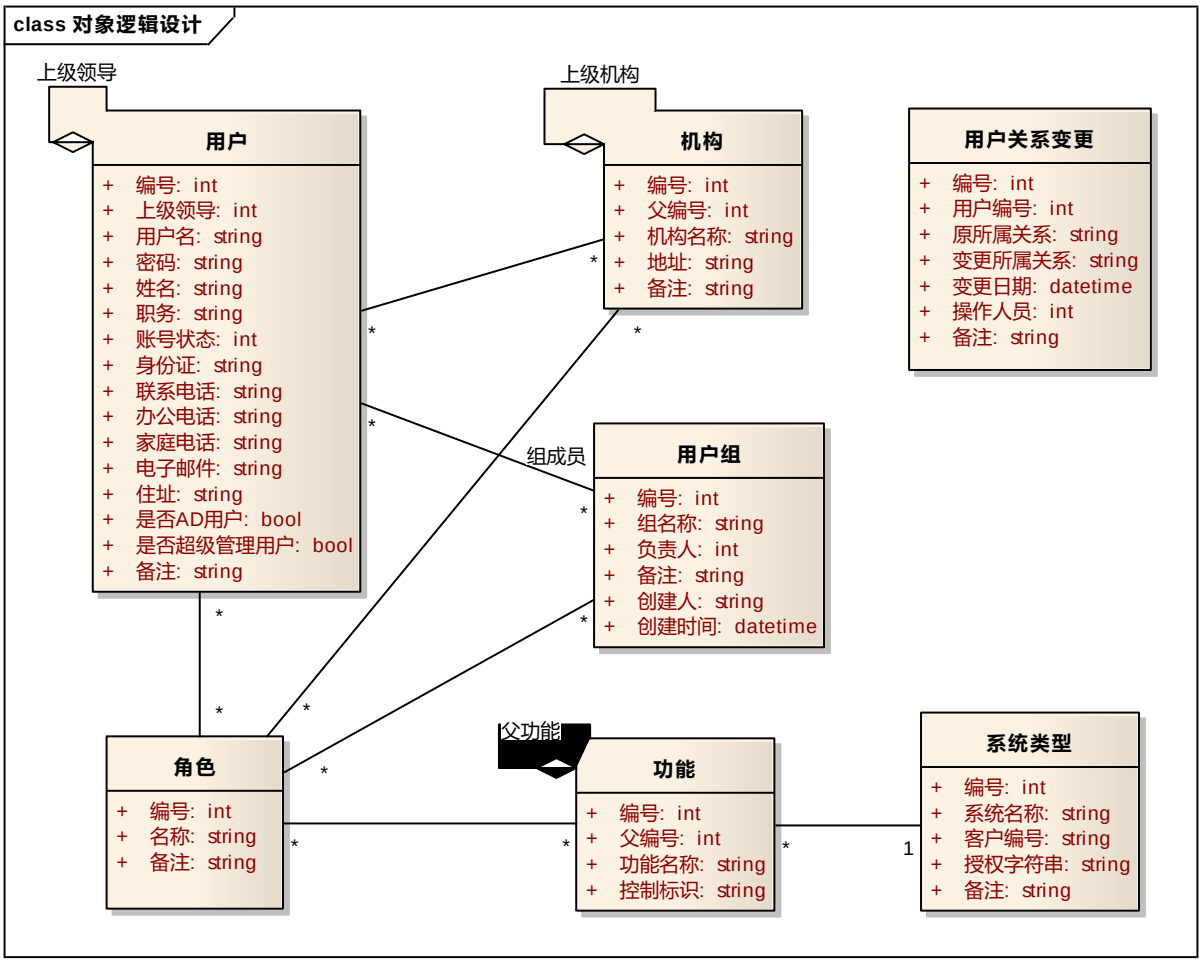
10. 数据库设计

10.1. Winform 框架数据库设计

由于内容比较多，请参考另外的独立数据库设计文档《**Winform 开发框架_数据库设计说明书**》。

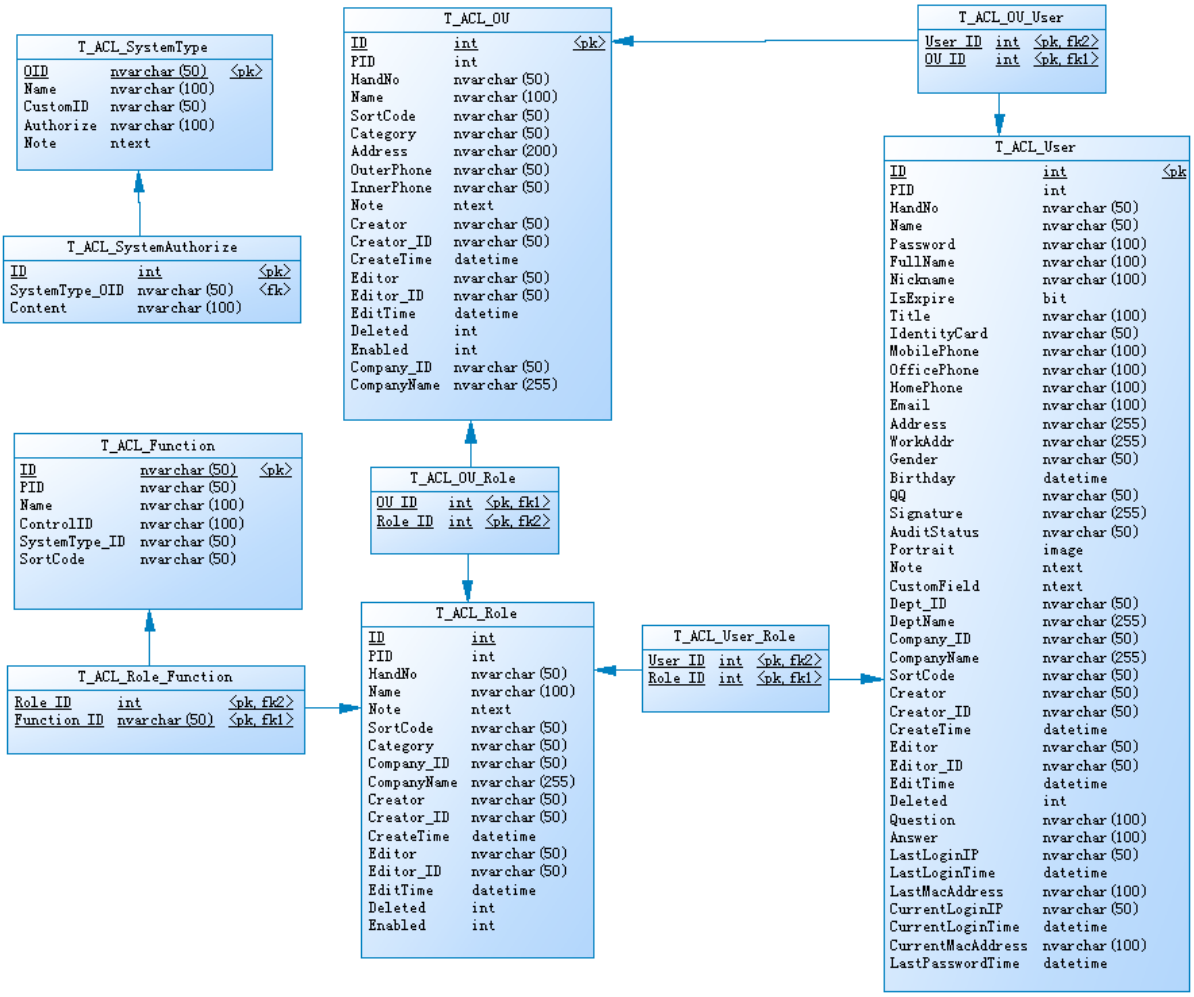
10.2. 权限数据库设计

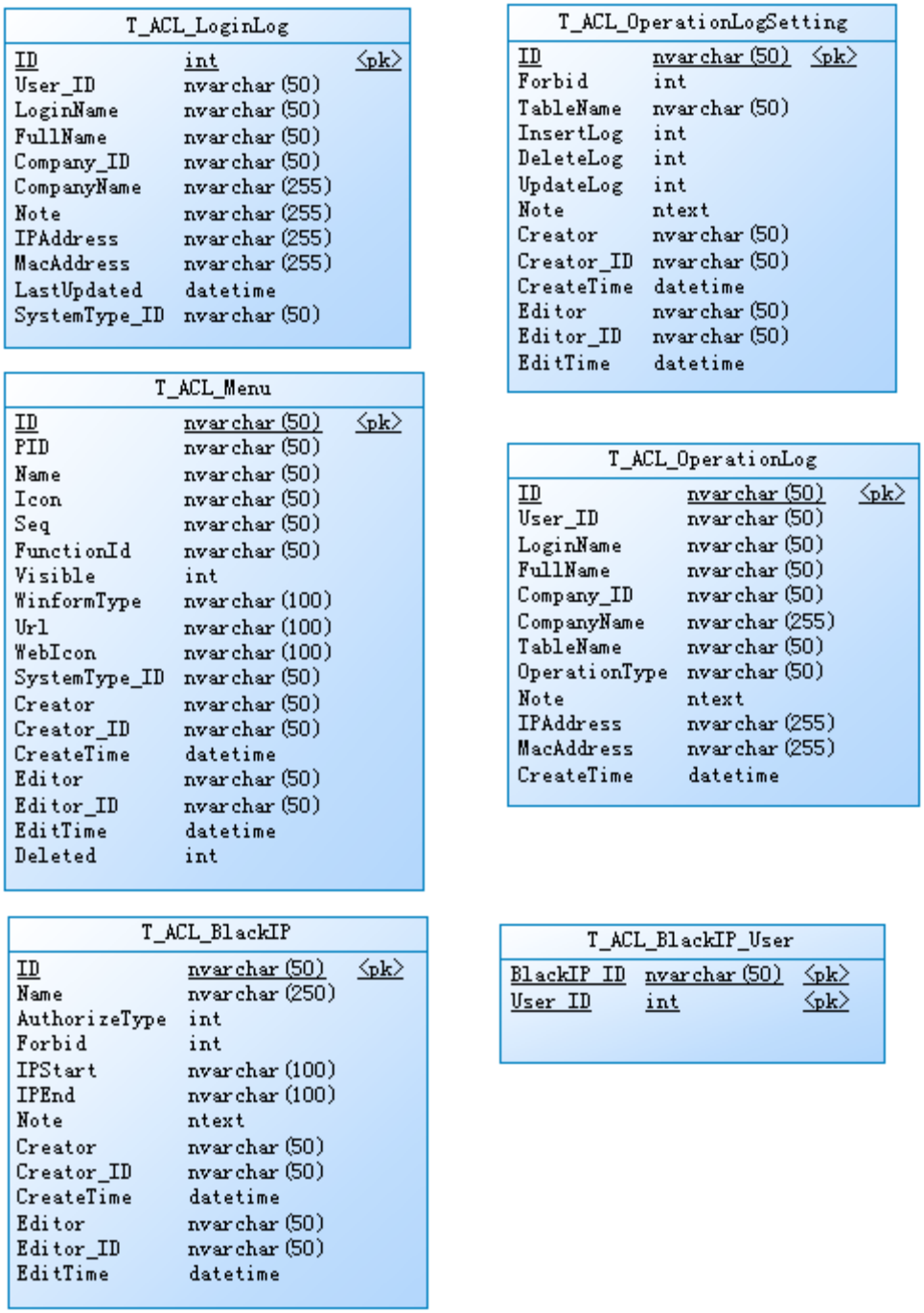
10.2.1. 逻辑设计



10.2.2. 物理设计

数据库分为两部分：一部分是权限系统模块的核心关系表，包括用户、角色、机构、功能、系统类型以及它们之间的中间表等关系；另一部分是权限系统的拓展内容，包括菜单管理、登陆日志、关键操作日志及配置表、系统黑白名单表等内容。


T_ACL_SystemType
OID nvarchar(50) <pk>
Name nvarchar(100)
CustomID nvarchar(50)
Authorize nvarchar(100)
Note ntext
T_ACL_SystemAuthorize
ID int <pk>
SystemType_OID nvarchar(50) <fk>
Content nvarchar(100)
T_ACL_OU
ID int <pk>
PID int
HandNo nvarchar(50)
Name nvarchar(100)
SortCode nvarchar(50)
Category nvarchar(50)
Address nvarchar(200)
OuterPhone nvarchar(50)
InnerPhone nvarchar(50)
Note ntext
Creator nvarchar(50)
Creator_ID nvarchar(50)
CreateTime datetime
Editor nvarchar(50)
Editor_ID nvarchar(50)
EditTime datetime
Deleted int
Enabled int
Company_ID nvarchar(50)
CompanyName nvarchar(255)
T_ACL_OU_User
User_ID int <pk, fk2>
OU_ID int <pk, fk1>
T_ACL_User
ID int <pk
PID int
HandNo nvarchar(50)
Name nvarchar(50)
Password nvarchar(100)
FullName nvarchar(100)
Nickname nvarchar(100)
IsExpire bit
Title nvarchar(100)
IdentityCard nvarchar(50)
MobilePhone nvarchar(100)
OfficePhone nvarchar(100)
HomePhone nvarchar(100)
Email nvarchar(100)
Address nvarchar(255)
WorkAddr nvarchar(255)
Gender nvarchar(50)
Birthday datetime
QQ nvarchar(50)
Signature nvarchar(255)
AuditStatus nvarchar(50)
Portrait image
Note ntext
CustomField ntext
Dept_ID nvarchar(50)
DeptName nvarchar(255)
Company_ID nvarchar(50)
CompanyName nvarchar(255)
SortCode nvarchar(50)
Creator nvarchar(50)
Creator_ID nvarchar(50)
CreateTime datetime
Editor nvarchar(50)
Editor_ID nvarchar(50)
EditTime datetime
Deleted int
Question nvarchar(100)
Answer nvarchar(100)
LastLoginIP nvarchar(50)
LastLoginTime datetime
LastMacAddress nvarchar(100)
CurrentLoginIP nvarchar(50)
CurrentLoginTime datetime
CurrentMacAddress nvarchar(100)
LastPasswordTime datetime
T_ACL_Function
ID nvarchar(50) <pk>
PID nvarchar(50)
Name nvarchar(100)
ControlID nvarchar(100)
SystemType_ID nvarchar(50)
SortCode nvarchar(50)
T_ACL_OU_Role
OU_ID int <pk, fk1>
Role_ID int <pk, fk2>
T_ACL_Role
ID int
PID int
HandNo nvarchar(50)
Name nvarchar(100)
Note ntext
SortCode nvarchar(50)
Category nvarchar(50)
Company_ID nvarchar(50)
CompanyName nvarchar(255)
Creator nvarchar(50)
Creator_ID nvarchar(50)
CreateTime datetime
Editor nvarchar(50)
Editor_ID nvarchar(50)
EditTime datetime
Deleted int
Enabled int
T_ACL_User_Role
User_ID int <pk, fk2>
Role_ID int <pk, fk1>
T_ACL_Role_Function
Role_ID int <pk, fk2>
Function_ID nvarchar(50) <pk, fk1>



10.3. 字典数据库设计

字典数据库部分只包含两个表，一个是字典数据类型表、一个是字典信息表，字典类型表包括一个可以无限递归的树形结构，可以嵌套很多类型，只是一般字典模块不需要太多的层次。字典数据信息表，则是实际的数据字典内容，有一个键值指向具体的所属类型。

详细的表结构如下所示。

1） 字典数据信息表：TB_DictData

字段列表						
编号	字段列名	字段描述	数据类型	可空	默认值	约束类型

1	ID	编号	NVarChar(50)			主键
2	DictType_ID	字典大类	NVarChar(50)	√		
3	Name	字典名称	NVarChar(50)	√		
4	Value	字典值	NVarChar(50)	√		
5	Remark	备注	NVarChar(255)	√		
6	Seq	排序	NVarChar(50)	√		
7	Editor	编辑者	NVarChar(50)	√		
8	LastUpdated	编辑时间	DateTime(8)	√	getdate()	

2） 字典数据类型表：TB_DictType

字段列表						
编号	**字段列名**	**字段描述**	**数据类型**	**可空**	**默认值**	**约束类型**
1	ID		NVarChar(50)			主键
2	Name	类型名称	NVarChar(50)	√		
3	Remark	备注	NVarChar(255)	√		
4	Seq	排序	NVarChar(50)	√		
5	Editor	编辑者	NVarChar(50)	√		
6	LastUpdated	编辑时间	DateTime(8)	√	getdate()	
7	PID	分类	NVarChar(50)	√		

11. 版本修订历史记录

日期	作者	内容	版本
2011-08-12	伍华聪	初稿	1.0
2011-12-25	伍华聪	完善内容，并添加 WCF 框架介绍及相关视图	2.0
2013-07-26	伍华聪	完善内容，并增加插件化设计介绍内容	3.0
2013-09-29	伍华聪	完善内容和插图	3.1
2014-03-13	伍华聪	对整体文档进行完善修正	4.0
2016-11-30	伍华聪	对整体文档进行完善	5.0