# 一.搭建软件环境

## 1.搭建三层架构（不知道是不是，听说的）

三层分为数据访问层DAL、业务逻辑层BLL、表现层（窗体界面）

当然在三层之间，还有需要一个对象实例游走，即为模型层Model

为了使软件更加灵活，可以支持多种数据库，可以在DAL层之上添加一个IDAL层，作为数据访问接口层。为何要这么做呢？因为层层互联，BLL层要访问DAL则需要引用它，而当多个数据库时将会引用很多，所以使用接口层可以让BLL只调用IDAL，而实现在DAL。

层层互联要实现，则需要添加引用。模型层Model需要被DAL、BLL和表现层都引用。BLL还需要引用DAL（若有IDAL则引用IDAL，此时DAL也需要引用并实现IDAL）。表现层还需要引用BLL。这样就把三层架构创建并联系起来了。

## 2.若使用IDAL层，如何使解决方案知道调用那个DAL来实现呢？

浅显的理解，哪个实例就使用的是哪个实现：

1）直接new一个实例

2）定义接口类型的变量，用某个实例去初始化

## 3.配置文件的读写

### 1）Configuration类与ConfigurationManager类的区别

Configuration类一般以实例化后使用，设置打开对象的权限后，可以对配置文件进行读写，之后还要记得保存。此外，还要一个重要的注意点，在保存后需要重新加载配置文件，RefreshSection方法的参数必须为字符串的appSettings。

ConfigurationManager一般调用静态方法来使用，可以直接设置与获取配置文件内容，但是其Add方法执行时会出现只读问题，即Add方法无法写入内容。

### 2）写入后，如何判断写入哪个文件？

网络解释是：调试时，写入的配置文件是.vshost.exe.config。只有运行.exe文件时才是写入到.exe.config。

但是在我的实践中，写入后代码中可以通过ConfigurationManager的方法读取到，然而却没有保存。故而，重启调试后依然读取不到数据。

#### PS：程序在调试模式下，当配置管理器为Debug时，编译出来的exe无法写配置文件（原因未知），当配置管理器为Release时，编译出来的exe直接运行可以写配置文件。直接运行exe文件时，读写的配置文件是.exe.config，但是在调试时，读写的配置文件是.vshost.exe.config，而且当停止调试时，VS会默认将.vshost.exe.config的内容还原为.exe.config。因此，若要使配置文件保存信息，需要直接运行使用Release编译出来的exe文件！！！

### 3）如何使软件读写配置文件app.config，而不是.exe.config或者.vshost.exe.config？

从网络查询知，无法实现！可以将需要的配置信息手动录入到app.config中，编译或者运行后将会复制到.exe.config中

### 4）续Configuration类：注意点

首先，一般的，此类的实例从ConfigurationManager.OpenEXEConfiguration方法中获得。

此类中已提供了几个固定的标签供开发者使用，如appSettings、connectionStrings等标签，无法自行添加，需要使用到这些标签时，便访问对应属性。

在使用此类时，添加后需要调用保存方法，但是此时只是保存到系统缓存中，还需要调用ConfigurationManager的刷新方法刷新对应标签（不对应似乎也没问题）。

## 4.数据库访问层

### 1）调用直接定义的DBHelper层

程序中直接使用类（如SqlConnection和SqlCommand），即定义一个连接类实例并传入连接字符串，然后把连接实例及sql语句传入命令类实例中，通过命令类的方法访问数据库。而数据库访问层直接调用其中的方法，传入写好的sql语句。

### 2）MVC

基本步骤：定义IDAL接口层的基接口IBaseDAL，并使用泛型，规定可以为class与new()；再定义其他IDAL实现这个基接口；接着在DAL层定义BaseDAL基类（此类不知道为什么不需要实现基接口），使用泛型并规定为class；最后在DAL层添加其他DAL并继承基类实现对应接口即可。

搭建好MVC后，可以使用不同的框架来实现数据存取，如NHibernate框架等，下面给出的链接的方案使用的是另一个框架。

详情可见<http://blog.csdn.net/anjingyatou/article/details/18234829>

在链接的方案中，BaseDAL需要使用System.Data.Entity命名空间下的DBContext类，而由于每次建立一个BaseDal继承的类的时候都会实例化一个DbContext。因此，优化为使用简单工厂模式，使用CallContext类存储DbContext实例，CallContext类可以理解为就是把数据存放到单独一个内存空间中，随时为你提供调用。当你想把数据放入数据槽中调用CallContext.SetData方法，当你想从数据槽读取数据是调用CallContext.GetData方法。主要作用：保证线程内唯一！

此外，该方案中IBaseDAL中使用到了System.Linq.IQueryable接口、System.Linq.Expressions.Expression类和Func委托，需要深入了解一下。

在BaseDAL类的实现中，增删改时，需要调用DbContext的SaveChanges()保存变动数据。链接中暂时未给出。

### 3）edmx实体模型（EF框架）

通过XML阅读器查看edmx文件，大致由SSDL、CSDL、C-S三部分组成，分别对应着对于数据库、实体、数据库表与实体之间的映射，SSDL中有对数据库表、字段等的规定，CSDL中有对实体名、实体属性等的规定，C-S中有对数据库表与实体之间的映射。

当根据数据库生成edmx时，会有.tt文件（T4模板文件）和对应的实体类生成，这些实体类是.tt文件根据edmx配置文件生成的，可通过设计界面视图拖动添加并保存在.tt文件下。T4文件可用T4 Editor查看，用文本查看无格式显示。而当生成了空白的edmx时，.tt文件需要通过右键edmx设计界面-添加代码生成页获得，同时实体类也会自动生成。另外还有一个.Context.tt文件，里面定义了一个继承自DbContext类的类，里面还定义了当前上下文的模型（即对实体类之间的联系）。

我的第一次尝试：使用Model First模式。出现了很多问题：A. Microsoft SQL Server，错误: 5120——修改文件使用者Authenticated Users权限为完全控制即可。B. 基础提供程序在 Open 上失败——第一次是因为表结构未生成，而未生成是因为5120错误；第二次显示登录失败（Windows和sa登录都失败），未知原因。

因此，使用Model First模式（先在可视化设计器中创建实体和它们间的关联，然后设计器生成SQL命令并保存于一个SQL文件中，通过执行这一SQL文件完成数据库的创建和修改工作）失败！应该是对三种模式不了解，导致无法解决问题。三种模式还有DataBase First（先设计并建好数据库，然后使用Visual Studio的向导创建EF数据模型并生成实体类代码）和Code First。

# 二.界面与逻辑设计

## 1.委托与事件的应用

### 1）直接通过EventHandler定义事件

比如，定义了一个新窗体或者新控件被调用时，需要在点击确定时退出窗体，但是button的点击事件只在内部完成，此时可以通过EventHandler定义一个确定事件，而且在button的点击事件中判断新定义事件不为空时，调用方法Invoke，暂时理解为被调用。

EventHandler定义的事件方法Invoke传入的参数就是button点击事件的参数，即说明EventHandler是与button关联的事件定义关键字。

### 2）使用委托定义事件

当在一个新窗体或者新控件中，需要把某个值传出去，除了使用公开的属性外，还可以通过委托参数。实际上，使用EventHandler定义的事件也是一种委托，参数是button点击事件的参数。所以说，事件是一种特殊的委托。

定义了委托后，也需要在定义此委托的类中对委托进行被调用。但是，委托不通过Invoke实现被调用，而是和函数调用一样。

**注意：不管是委托还是事件，都需要在外部调用此类，实例化此类后使用+=给委托或者事件注册一个可执行方法。但是此方法不会立即执行，而是委托或者事件被调用的时候才执行。**

## 2.新窗体与新控件（自定义）

### 1）展示时，窗体和控件都有Show方法，但是控件没有ShowDialog方法。

当控件使用Show方法，程序将继续执行，而不会弹出控件。窗体则会等待关闭后才进行下面的代码。而且ShowDialog可以返回一定的操作结果。

### 2）自定义属性

自定义窗体或者控件时，我们会定义一些特有属性，但是当调用新窗体或者控件时，在VS的属性窗体不会显示。若要使其显示，则需要在定义时前面加上[Browsalbe(true)]。

自定义控件时，BackgroundImage是用来设置控件背景图片的，但是此属性是将指定图片布满整个控件，当图片较小时会通过复制多个图片的形式布满控件，即控件较大时出现的效果是背景图片由多个指定图片组成，而非一个。因此，需要设置图片布局，使它占据整个控件的中央，这样即使再大的控件小图片也依然占据中间，而且只由一个图片组成（暂时这样理解）。可以设置默认图片布局，重写BackgroundImage属性并在前面加上[DefaultValue(默认值)]设置属性的默认值。**注意：此处设置的默认值若是对重写的属性或者方法，将可能无法生效，因为每次编译后实例的属性值都为基类属性的默认值，所以遇到此情况时需要在构造函数中设置。**

**拓展：**在属性或者方法重写或者定义时加上[EditorBrowsable(EditorBrowsableState.Never)]，表示该属性或方法始终不能在编辑器中浏览，意思就是说当别人调用这个类时无法通过对象.出这个属性或者方法。当然，只有在生成dll时才生效，同个解决方案下无效。

## 3.SSH登录

### 1）使用SharpSsh类库

默认端口号为22

SshStream类，传入host的ip，用户名，密码。（设置属性Prompt为#，设置属性RemoveTerminalEmulationCharacters为true），方法ReadResponse()可以获取回显（理解为返回的消息），方法Write()可以写入命令，（后面接方法Flush，还要再调用一次传入换行符？）。

**注意**：当执行完命令调用ReadResponse()方法时，可能获取不到回显，需要在前面加上线程延迟给予执行命令的时间。

SshShell类，可传入ip，name，password。属性IdentityFile可查看登录证书，AddIdentityFile方法可以传入登录证书。Connect方法可传入端口号。GetStream可以获取登录信息。

## 4.XML文档的操作

### 1）FileInfo类

属性：FullName可以获得对应文件的全路径（不管存不存在）

Exists判断是否存在对应文件

### 2）XmlDocument类

方法：Load(文件全路径)可以加载xml文档到类对象实例

LoadXml(文档的头字符串)也可以加载xml文档到对象实例，文档头字符串示例：<?xml version=\"1.0\" encoding=\"utf-8\"?><Resource><Form><Name>MainForm</Name><Controls></Controls></Form></Resource>

属性：DocumentElement获取文档的根节点，类型继承XmlNode类（实例化此类接收）

### 3）XmlNode类

## 5.界面多语言显示

### 1）Type类：获取对象的类型

后可获取对象的公共属性（GetProperty方法）

### 2）PropertyInfo类：可定义指向对应的公共属性

可以访问此属性的数据（SetValue方法）

# 三.逻辑的实现