# 一.搭建软件环境

## 1.搭建三层架构

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。**

### 3）事件参数e

在控件自动生成的事件中，有些事件参数e有Cancel方法，可以取消这个事件的执行。

## 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。

#### 注意：

ShowDialog方法打开窗体时，窗体将直接占有主线程，若不关闭则原窗体程序无法继续执行。而Show相当于是重开一个线程开启窗体，即使不关闭，原窗体都会继续执行。

ShowDialog一般对应Close方法，而Show一般对应Hide方法。

### 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）添加皮肤

使用IrisSkin2或者IrisSkin4类库，然后在工具箱中选择项即可添加。一般使用IriSkin4，因为4有个方法SkinFormOnly可以设置只是窗体皮肤，否则整个窗体中的控件的背景颜色也会被修改。

可以通过SkinFile添加皮肤文件（.ssk文件），也可以通过SkinStream添加皮肤文件在VS生成的Resources中的缓存。

### 4）控件带动窗体移动

首先，需要一个全局变量判断鼠标是否按下。然后用另外两个全局变量记下按下鼠标时窗体的位置和鼠标的位置（相对于屏幕，可由Control点出）。当鼠标按下且移动发生时，在事件中将鼠标的新位置减去老位置即可得到差量，再由窗体老位置加上这个差量即可。

括号中的内容其实是没必要的，因为若不给窗体和鼠标的老值赋新值的话，窗体老值与新值的差量和鼠标老值与新值的差量还是一样的，只有两个都不重新赋值，那就是一样的！（后面别忘了把新值再次赋值给老值！）

## 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>

Save()保存当前所有节点信息，在此类的实例中innerxml属性就是即将存入xml文档的内容。

**注意：当添加完所有的节点和节点特性后，一定要执行Save()，否则将不会保存到xml文件中**

CreateNode：可以给此类的实例添加节点，但是**不会保存到xml文档，需要执行Save**后才保存。

CreateAttribute：可以生成一个节点特性，生成后将其添加到节点的特性中即可。跟着节点在Save后保存。

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类与XmlAttributes类

XmlNode类：节点类。可通过XMLDocument类对象的CreateNode获得。常用的属性是Attributes，获得所有特性，此属性有方法可以添加特性。常用的方法是AppendChild，添加子节点。

XMLAttributes类：特性类。可通过XMLDocument类对象的CreateAttribute获得。然后可以通过value获取或设置特性值。

## 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）

## 6.GridControl的应用

### 1）复选框

#### 第一种方法：

在其Designer中，有个属性ColumnEdit，此处可以创建一个CheckEdit。创建后，其属性ValueChecked和ValueUnchecked默认为boolean类型的值，此时默认为单选。

若要实现多选，只需要将ValueChecked和ValueUnchecked改为string类型即可，其值还是true和false。

另外，此处可以自定义复选框图片。也可以不使用ColumnEdit，直接自定义复选框图片来实现。可以参见My Study Notes的笔记（不过其实那边不详细…）。

#### 第二种方法：

直接在GridView中将OptionSelection选择MultiSelect为true和选择MultiSelectMode为CheckBoxRowSelect即可。

### 2）全选

复选框第一种方法：

暂未实现表头全选

复选框第二种方法：

DevExpress控件自带的功能，只要那么选择了就有全选功能了。

### 3）数据绑定

首先需要给对应的列绑定字段，用FieldName绑定用于过滤而得到数据表中选中的此字段。然后在后面将具有相同绑定字段的数据源（可以是DataTable）赋值给DataSource即可。

**注意点**：用FieldName绑定的字段需要与数据源中存在的字段相同！

### 4)自动序号列

在Grid控件中，其中最前面一列是空出来的，一般不会有显示。此列一般使用来做序号列。

此列可以认为是命名为Indicator。因此，首先，需要在窗体构造函数中为View的属性IndicatorWidth赋值，即是为此列设置宽度。之后，要使其有自动增长的序号，需要在View的事件CustomDrawRowIndicator中添加自动增长的方法实现：参数e可以点出info获取Indicator的信息，判断info的属性IsRowIndicator为true则表示为自动序号列（还要判断e.RowHandle>=0）；接着只需要将info的属性DisplayText赋值即可（值赋为e.RowHandle+1的字符串）。

### 5）排序

m\_AppTable.DefaultView.Sort = "status ASC";

如上式所示，对控件的数据源的默认视图进行排序，即可完成控件显示的排序。当然，这个只是实现了视图的排序，在m\_ AppTable中下标所对应的行还是一开始赋值的行。

## 7.限制输入

通过控件的按键事件。

### 1）限制长度

e.KeyChar属性就是指示按下的键对应的字符。其中13表示回车键，8表示左删除键。一般还需要判断选中的文本是否为空，否则可能按不了删除键。

**注意：e.Handled只是是否处理过按压事件，若处理过则无法输入，即e.Handled为true。**

**另外，控件的KeyPress事件是在按下之后才触发的，因此算是已经输入了！**

## 8.数据绑定

### 1）ComboBox数据绑定

首先，可以直接给控件的Items添加元素即可。

其次，可以使用键对应值的方法。将控件的DataSource赋值（一般为DataTable，此表有两列，即为键值），将控件的ValueMember绑定为该数据源的Key列，将控件的DisplayMember绑定为该数据源的Value列。其中ValueMember代表的是这个控件对应的Value的实际值，而DisplayMember代表的是这个控件对应的Value的显示值。

## 9.线程与异步

在窗体显示时，有时窗体会无法绘制，比如等待窗体（如点击上传时弹出来的等待窗体）可能无法显示图片与文字，可能是因为后面执行的代码占用了线程，若出现这个情况，可以把无法绘制的窗体的DoubleBuffer值设置为true，或许能解决此问题。若是问题仍然无法解决，则可以在此窗体Show之后添加Application.DoEvent方法，等待处理完当前线程中目前的所有信息。

解决了显示问题后，若后面的代码还需要占用过多线程（如与服务器通讯）时，将使此窗体卡住，即只是静态效果而非动态效果，这是因为UI线程被堵塞，此时就需要将后面的代码通过多线程执行。

在多线程中，主线程定义的控件是无法使用的，需要使用异步方可在线程外访问。首先通过this.InvokeRequired属性判断此方法被调用时是否在主线程外，若是则通过this.Invoke方法重新执行此方法。其中使用的委托可以为任意委托，参数为object数据但其中是委托参数。如：

if (this.InvokeRequired)

{

this.Invoke(new EventHandler(AsyncDownLoad), new object[] { sender, e });

}

else{}

（还有另一说法：this.InvokeRequired指示的是某一控件（this）是否在线程外。）

使用异步的方法其实是将线程外的程序通过一种特殊的方式在线程中执行，也就是说还是在UI线程中执行，故若把线程中所有程序都在同一个异步方法中执行，则UI还是只显示静态效果。

故，若主线程遇到堵塞，且UI只需要显示静态效果，则可以只使用一个Invoke，若要显示动态效果，则不建议只使用一个，可以把阻塞主线程的那一段代码直接在后台线程中运行，然后通过全局变量的方式把线程外的控件值带入此段代码中，而其他的各段代码分别新建一个异步方法给后台线程调用。

**猜测**：注意：调试时，若在此种异步中产生异常，异步将循环执行，异常也跟着循环产生，不会抛出？异常是会抛出的，但是网络异常的那个类不知为何不会。

DevExpress.Data.CurrencyDataController.DisableThreadingProblemsDetection = true;//不设置此值会导致线程交叉

### 线程池：

线程池其实就是一个存放线程对象的“池子(pool)”，他提供了一些基本方法，如：设置pool中最小/最大线程数量、把要执行的方法排入队列等等。ThreadPool是一个静态类。

方法SetMaxThreads可以设置线程池的最大与最小

方法QueueUserWorkItem将线程加入队列，准备执行。其参数可以为lambda表达式，也可以使用委托。

Lambda表达式可以写成(参数名)=>{方法体}

## 10.获取按键

Enter键按下时keychar值为\r。上下左右键在C#中没有对应的keychar值。若要获得上下左右键，不能使用keypress事件，而要使用keydown或者keyup事件，其中keydown只要按压就会触发即一直按着就一直触发，keyup一般只触发弹出那一次。

当按下enter键时，不知为何，即使将handled设置为true，也依然产生换行，而且这个换行是在句首！目前的解决方案是当text改变（句首换行时）重新赋值。另外，光标位置不知为何也会到句首去了，需要对光标做处理

当按下向上或向下键时，除了将handled设置为true，最好在后面也加上光标位置处理：

txt\_url.Select(txt\_url.Text.Length, 0);

txt\_url.ScrollToCaret();

当需要用回车键代替某个按钮，如登录时的登录按钮，则只需要把窗体的AcceptButton设置为那个按钮即可。

## 11.一些关键字

const：定义常数，与static不可连用，但自带static功能

## 12.一些常用类

Char：此类可对字符做很多判断，可用于输入字符判断。其中IsLetter类做判断时，输入的是中文也可通过。因此，若要使字符为纯字母，则需要使用IsLower和IsUper，判断是否为大小写的字母。

# 三.逻辑的实现

## 1.数据库访问

之前是希望可以通过某个框架实现数据库访问的，尝试了EntityFramework框架，尝试了modelfirst和codefirst模式，都无法成功实现。因此，以微软的类库SQLClient来完成数据库访问。

### 1）使用SQLClient类库搭建SqlDbHelper类

使用SqlConnection类和SqlCommand类，不过这两个类使用后需要释放或者直接using使用。传入连接字符串、命令字符串后。

调用SqlConnection类实例的open方法，打开数据库连接。

然后调用SqlCommand类实例的方法ExecuteNonQuery方法即可执行对应命令，并返回影响的记录数。此方法一般用来执行增、删和改。

查询的话，会通过其他方法。如要查询一条数据，使用SqlCommand类实例的ExecuteScalar方法，返回非0的数字表示成功。而查询多条数据的时候，不使用SqlCommand类，而是使用SqlDataAdapter类实例，使用其方法Fill，传入DataSet类实例即可将数据传给这个DataSet实例。

### 2）参数配置

若要执行需要参数赋值的sql语句，此时需要定义一个方法，此方法给SqlCommand类的实例配置必要的值。必要值的属性：Connection、CommandText、Transation、CommandType以及Parameters（使用Add方法添加）。而且当其参数的属性Direction要是只输出的才行，否则将是不存在的值（使用DBNull类）。

命令执行后，一般要将参数清空。

### 3）SQL语句相关

delete后面不能接\*，直接from即可

### 4）MySQL数据库

与SqlServer设置基本一致，使用的类库是MySql.Data.dll。

远程连接数据库时，会出现不被允许连接的情况。此情况可能是对方电脑防火墙拦截了3306端口，也可能是因为本机ip未被授权。可执行授权语句：

mysql> grant all PRIVILEGES on db\_name.\* to 'username'@'xxx.xxx.xx.x' identified by 'password' WITH GRANT OPTION;

若语句无效则执行：Mysql> flush privileges

### 5）Access数据库

这是一个轻量级数据库，与SQLite数据库类似。即此数据库不需要配置服务器（用户密码），直接打开文件使用即可。

**注意**：在Access数据库中，长文本类型的字段之间是无法建立连接的，可认为无法做等号，所以一般不采用长文本。

## 2.文件的操作

### 1)SaveFileDialog类

必要参数：FilterIndex（0），RestoreDirectory（设置是否按上一次打开的路径打开），CreatePrompt（文件不存在，是否提示创建）,FileName（不确定一定要有）Title（对话框显示的名称）,Filter（打开后可显示的文件）

设置了必要参数后，即可使用ShowDialog打开保存对话框。

### 2）FileInfo

续二.4.1）

属性Extension可以获取文件的类型，即后缀名

### 3）导出文件

导出文件一般依赖于DevExpress控件GridView，此时在导出前需要先过滤check列。只需将此列设置为不可见即可。当然，在导出结束后，需要将此列重新设置为可见！！！**问题：GridView自动生成的selection无法过滤？**

GridView控件有自带导出方法。可以直接导出txt和pdf文档。而要导出xls文档的话，还需要一个XlsExportOptions类型的实例参数，配置此实例的ShowGridLines为true，SheetName为data。

### 4）获取文件的类——流

FileStream可以获取文件，使用默认编码格式，即文件自身的编码。

### 5）文件夹的操作

Environment.SpecialFolder.Desktop：桌面路径

### 6）压缩文件操作

使用SharpCompress类库，这是一个C#开源框架。支持压缩解压rar 7zip, zip, tar, tzip和bzip2格式的文件包。

使用方法：

using (Stream stream = File.OpenRead(文件路径))

{

MemoryStream sreader = new MemoryStream();

var reader = SharpCompress.Reader.ReaderFactory.Open(stream);

while (reader.MoveToNextEntry())

{

if (!reader.Entry.IsDirectory)

{

reader.WriteEntryTo(sreader);

}

}

## 3.id的唯一性

UUID 是 通用唯一识别码（Universally Unique Identifier）的缩写。其目的，是让分布式系统中的所有元素，都能有唯一的辨识信息，而不需要通过中央控制端来做辨识信息的指定。

比如，在不同设备有相同的表结构，但是若都使用自动增长的id，那么即使相同表结构，不同表对应的表id也有可能存在不同，若使用UUID则可以使id变成唯一的，因为不是自动增长，而是在代码中生成id后存入所有表，并保证不重复。

C#中使用System.Guild.NewGuild().Tostring()可以获得不同的UUID

## 4.获取本机IP

System.Net.Dns.GetHostEntry(System.Net.Dns.GetHostName()).AddressList.FirstOrDefault<System.Net.IPAddress>(a => a.AddressFamily.ToString().Equals("InterNetwork")).ToString();

测试发现获取的是虚拟机的ip地址，将FirstOrDefault改为LastOrDefault即可获取本机ip，但是获取到的是内网ip，要获取外网ip目前只能通过访问某个特定网站获得。

## 5.服务器通讯

### 1）上传数据（与web服务器）：

先将需要上传的数据参数字符串转化为字节数组（Encoding.ASCII.GetBytes），数据参数字符串的格式：”参数1名称=参数1值&参数2名称=参数2值”。

接下来一般需要对证书进行检查：

ServicePointManager.ServerCertificateValidationCallback += CheckValidationResult;

private static bool CheckValidationResult(object sender, System.Security.Cryptography.X509Certificates.X509Certificate certificate, System.Security.Cryptography.X509Certificates.X509Chain chain, System.Net.Security.SslPolicyErrors sslPolicyErrors)

{

return true;

}

证书检查才实例化HttpWebRequest类，否则可能出现证书错误问题。

实例化HTTPWebRequest类，使用其自己的静态方法Create强转类型得到，并将Web服务器的上传接口（类似一个Http地址）作为参数。

实例化之后，需要对这个对象进行一些必要属性的设置：设置操作方法——Method=”POST”，表示上传数据；设置Http标头的值——ContentType=”application/x-www-form-urlencoded”；设置标头——ContentLength=参数字节数组长度。

[HttpRequest中常见的四种ContentType](http://www.cnblogs.com/xiaozong/p/5732332.html)：application/x-www-form-urlencoded、multipart/form-data、application/json和text/xml。其中，第一种一般是对网页数据传输时。C#客户端与Java服务端通讯可见：http://blog.csdn.net/gavin\_dyson/article/details/50526940

#### 注意：

有时在传输时传输的实际长度大于数组长度，因而会导致未传输完毕就关闭流的异常，所以最好不要给http设置标头ContentLength，使用默认规则即可。

猜测：若使用操作方法使用”GET”，则不需要标头等信息，但是传参数时需要在接口（Http地址）中添加，即在此字符串后面加上”?参数1名=参数1值&参数2名=参数2值”。

之后调用HTTPWebRequest对象的方法GetRequestStream方法请求向服务器写入数据的流，对流进行Write即是想服务器上传数据。可以用using释放资源。

上传结束后，调用HTTPWebRequest对象的GetResponse方法返回一个WebResponse类的对象，新对象包含服务器所有的响应信息，用一个流接收后即可得到相应信息的字节数组。可以使用using释放资源。

#### 注意：

在C#中byte一般都是无符号的，即一个字节是0~255。而在java中，byte是有符号的，即一个byte是-128~127。

### 2）下载数据（与web服务器）：

与上传一模一样，但是主要处理的数据不是传入的参数，而是服务器的相应信息。

不论上传还是下载，一般也可以采用base64转码。Convert.ToBase64String与Convert.FromBase64String

### 3）传输数据注意

直接通过接口与服务器通讯，服务端不能通过把接收到的东西转化成string再取byte的方法，这样将导致所接收的内容均先变为文本后才转成byte，这样有可能导致乱码（非文本文件的时候）。

通过流传输数据时，若传输的是纯文本文件，那么可以通过参数的方法传输，而若是非文本文件，则不能使用通过参数的方法，因为Java中通过参数获取数据的方法是先将数据转化成string获取的。因此，最好使用某个定义好的协议来传输，如定义长度标志位，指定每一段数据的长度。

#### 注意：

因C#与Java的byte不同，C#中是0~255，而Java中是-128~127。按理说，在传输过程中，都是以二进制传输的，因此C#的byte传输到Java中也不会出现错误，C#中超过127的byte会被自动保存为负数的。

但是，因不知原因，超过127的在Java中有时会被解析成-17，-65，-67，因此在C#与Java通讯时一般采用base64编码的字符串来传输，因为base64在传输中byte不会超过127。

另外，在获取base64字符串时，会产生+号，而Java接收时将变成空格，所以需要将+号替换成%2B才可被Java正常接收，且显示为+号

#### 传输数据大小问题

使用Post传输时，默认最大只能传输2MB大小的数据，当传输的数据过大时，要么服务端接收到的数据为null，要么客户端出现基础连接已断开或者无法写入流数据的异常。网上提到的解决方法很多，但暂时仍未见凑效：

1. C#代码中设置HttpWebRequest对象的AllowWriteStreamBuffering为false，表示不使用缓冲处理，此设置下标头不能不设置即需给ContentLength赋值。此方法总出现“请求被中止的错误”。
2. 在服务端做修改，将TomCat的Server.xml文件中的Connector标签的maxPostSize设置为足够大的值。此种方法并未凑效，传输稍大于2MB的数据时服务端仍然无法接收到。
3. 类似修改.net环境，即修改C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG路径下的web.config.comments文件中的executionTimeout节点下的maxRequestLength的值。此方法也无法凑效。另外，此方法本来是使用于调用.net上传控件的情形下，或许不适应HttpWebRequest。

### 4）使用Socket自建服务器

用于监听的IP要使用类IPAddress的Any属性，即表示本机所有ip都被监听。加上输入的端口号Port实例化一个终结点对象。

实例化一个Socket作为监听使用，（IPv4、流、TCP）

## 6.加密方式

PBEWithMD5AndDES加密算法，使用时应确保所使用的盐（即密钥，应用于DES）一致。一般地，PBE中协助加密的字符串采用用户名。

http://bbs.csdn.net/topics/380176831

## 7.调用dll类库

C#调用Dll类库，若此类库为C#所写，则可以直接添加引用；若此类库是C++所写，则需要使用DLLImport导入外部引用。（不全是）

C++编写的类库：一般以指针作为参数，此时若是出参时，分三种情况，即ref和out以及引用类型。当接口出参为值类型时，我们将根据接口中是否需要传入值来确定ref或者out。当接口出参为引用类型时，我们可以不添加ref和out，而是直接写参数即可。

### 注意：

string是一种特殊的引用类型！！！当SDK接口出参时，使用out或者ref都无法给string出参，而且string也无法以直接写参数的方式出参！此时一般的解决方案是使用一个字节数组出参，之后才转化为string。也可以使用StringBuilder，不过这个类型似乎是C#特有的。

## 8.调试

在启动项目的属性中，将调试模块的启用承载去掉，即可让软件调用exe.config文件，而不是vshost那个文件。

Project Address：http://admin@code.xmzkteco.com/cooperation-project/new-firmware/CSSofeWare