Scut快速开发（3）

# 开发环境

**需要安装的软件**

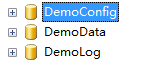
1. 消息队列
2. 数据库，Sql2005以上版本
3. VS2010开发工具（.Net Framework 4.0以上）

# 代码框架

项目划分三层：实体层，组件层，业务逻辑层（脚本层）；模型层主要是数据实体映射，自定义缓存结构；组件层主要负责实现中层层扩展功能；业务逻辑层主要负责实现游戏功能；

## 数据库创建

游戏划分成三个库：DemoConfig库（负责存储游戏配置数据） 、DemoData库(负责存储游戏玩家数据)和DemoLog库(游戏玩家日志记录数据)；如图



这里只为每个库建立一张表：

地图信息配置表（MapInfo）DemoConfig库

|  |
| --- |
| create table MapInfo  (  MapId int not null,  MapName varchar(50) not null,  PointX int,  pointY int,  primary key(MapId)  ) |

玩家表（GameUser）DemoData库

|  |
| --- |
| create table GameUser  (  UserID int not null,  NickName varchar(50) not null,  Pid varchar(50),  UserStatus int,  Property text,  CreateDate datetime,  primary key(UserID)  )  go  --继承BaseUser基类  EXECUTE sp\_addextendedproperty N'MS\_Description', '玩家信息[:BaseUser]', N'user', N'dbo', N'table', N'GameUser', NULL, NULL  Go  --自定义枚举  EXECUTE sp\_addextendedproperty N'MS\_Description', '玩家状态[Enum<UserStatus>]', N'SCHEMA', N'dbo', N'table', N'GameUser', N'column', N'UserStatus'  Go  --自定义实体类  EXECUTE sp\_addextendedproperty N'MS\_Description', '玩家属性[CacheList<Property>]', N'SCHEMA', N'dbo', N'table', N'GameUser', N'column', N'Property' |

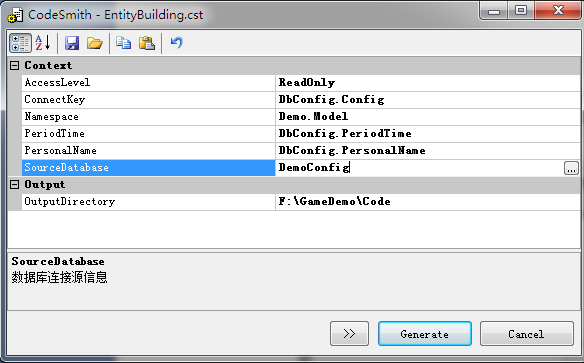
玩家登录日志表（UserLoginLog）

|  |
| --- |
| create table UserLoginLog  (  ID int not null identity(1,1),  UserId varchar(50) not null,  Pid varchar(50),  Ip varchar(50),  CreateDate datetime,  primary key(ID)  ) |

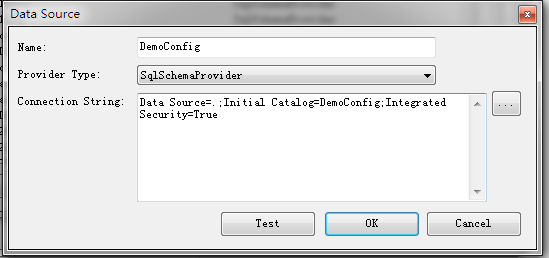
## 数据实体生成

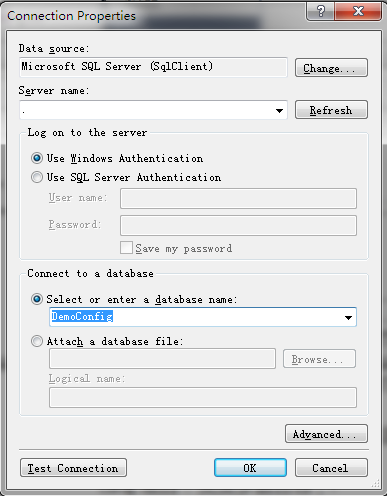
使用CodeSmith工具制作实体模板（[已提供现有模板](#_CodeSmith实体模板)）来生成我们需要的实体类；打开EntityBuilding.cst文件，弹出窗口中进行配置相应属性，配置完成后点击“Generate”生成实体类；如图：

Config库配置，设置只读权限

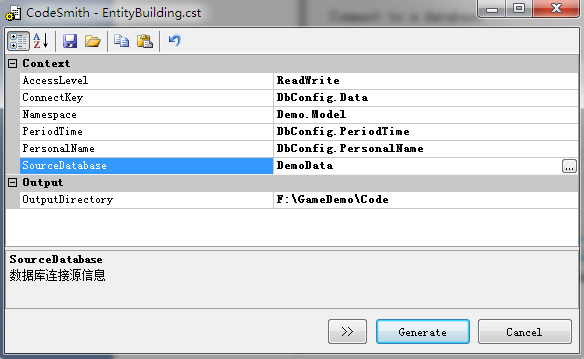


增加数据库连接

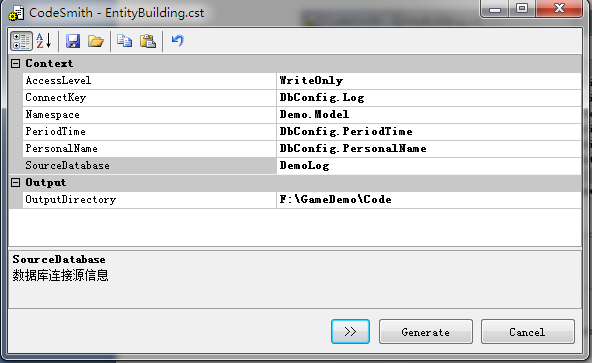




Data库配置，设置成读写权限



Log库配置，设置成只写权限

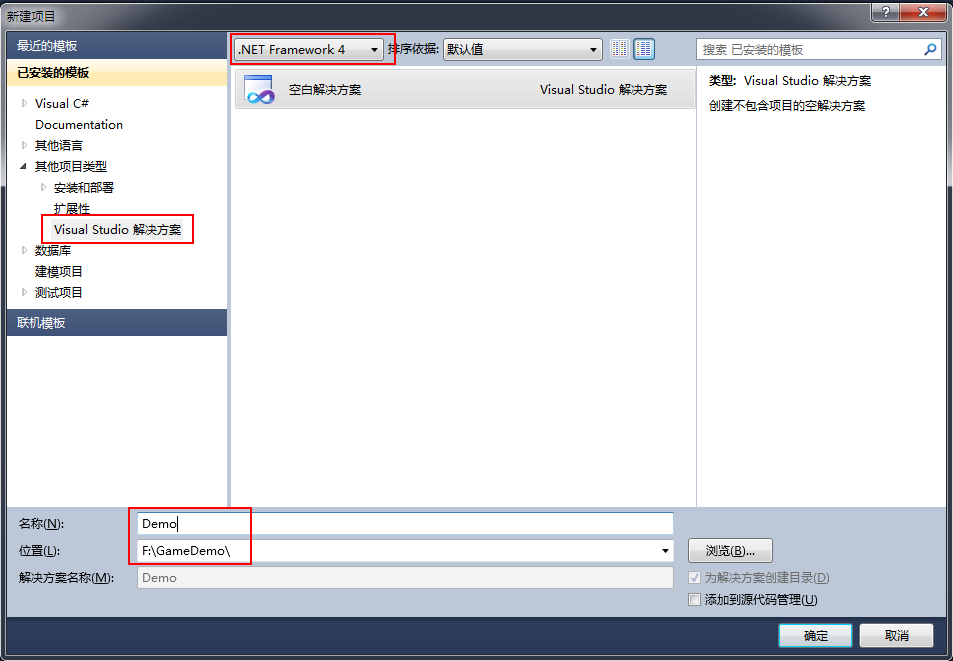


打开F:\GameDemo\Code目录，可查看生成的数据实体类，创建好项目后将实体类文件Copy到项目对应的目录下

## 项目搭建

### 创建解决方案

打开VS2010 在菜单上选择 文件 -> 新建 -> 项目；弹出“新建项目”对话窗口，在左则展开“其它项目类型”，选择“Visual Studio解决方案”，选择“.Net Framework 4.0”后，输出项目名称及位置，再点击“确定”；如图：



### 创建项目

在“资源管理器”中添加新的几个项目Model、Lang、Com、Bll、HostServer；**Model项目：**负责从数据库中导出表的数据实体类映射；

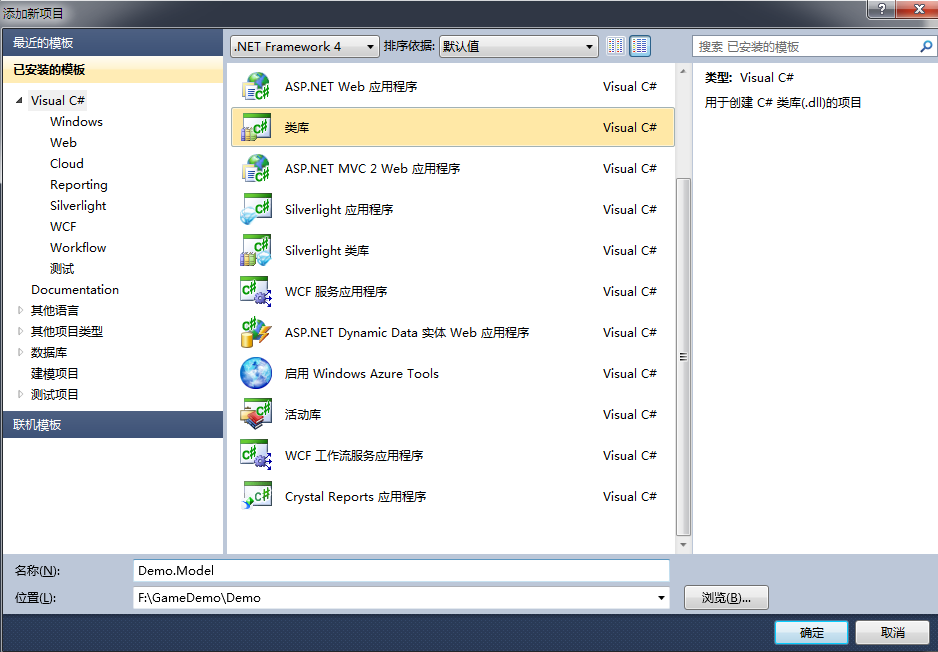
**Lang项目：**负责多语言包；

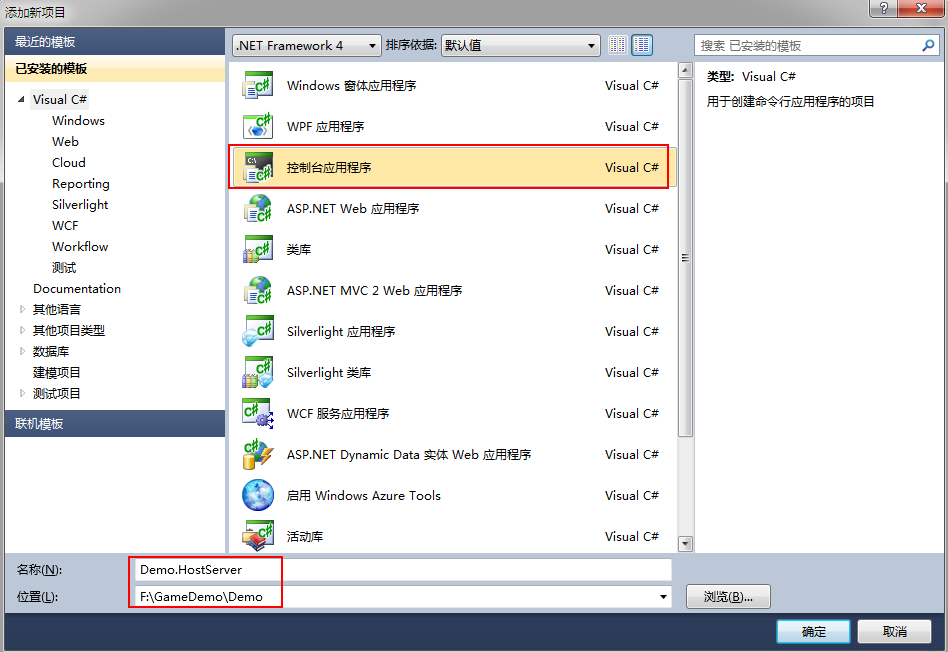
**Com项目：**负责中间层组件扩展处理，及中间层业务实体类；

**Bll项目：**负责业务逻辑处理；

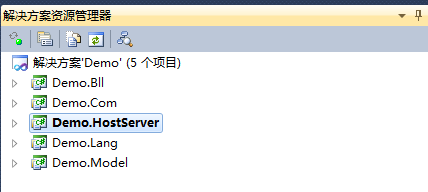
**HostServer项目：**控制台启动程序，及脚本（Python）业务逻辑处理；

如图：

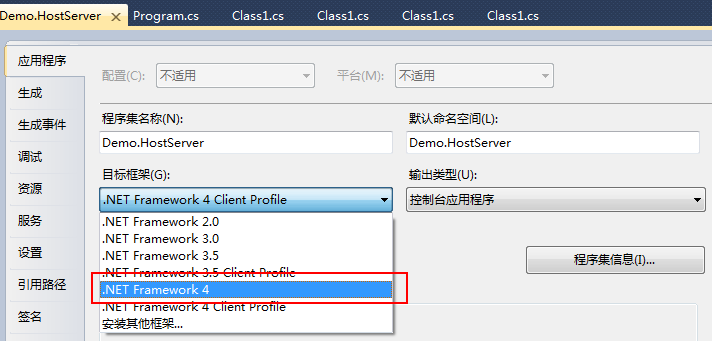




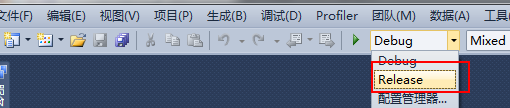
创建项目结果如下：



设置控制台程序“HostServer”属性为“.Net Framework 4.0”



设置成“Release”编译方式



### Model项目

**组件引用**

|  |  |
| --- | --- |
| ****项目**** | **引用路径** |
| Model | **Plugin\Redis\protobuf-net.dll**  **Framework\V5.1\ZyGames.Framework.Common.dll**  **Framework\V5.1\ZyGames.Framework.dll**  **Game\V2.6\ZyGames.Framework.Game.dll**  [组件详情](#_Scut开发包) |

**目录划分**

**划分配置库（ConfigModel）、玩家信息库（DataModel）、玩家日志库（LogModel）和自定义类型数据实体（Model）目录存储数据实体类； Enum目录存储自定枚举；**

**步骤**

接着将之前使用CodeSmith生成好的实体类复制到相应的ConfigModel、DataModel和LogModel目录下，并在VS Model项目中将其包括在项目中；

接着编译Model项目会发现在GameUser类中缺少些命名空间引用，UserStatus枚举和Property类没有定义（注意到没，这些缺少的类和枚举就是在创建表时增加的配置）；

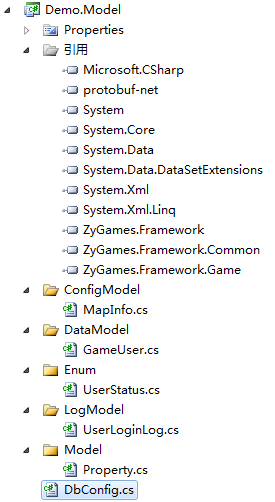
在Model项目中分别在Enum目录与Model目录下增加UserStatus自定义枚举和Property自定义类；接收编译就可以成功了。（建议在GameUser类的构造函数中实例化Property属性及其它引用类型属性，在业务层使用中减少判断Null语句）

**DbConfig类**

**在生成实体类时，生成配置ConnectKey项中使用**

|  |
| --- |
| public class DbConfig  {  public const string Config = "DemoConfig";  public const string Data = "DemoData";  public const string Log = "DemoLog";  public const int GlobalPeriodTime = 0;  public const int PeriodTime = 0;  public const string PersonalName = "UserId";  }  //UserId：是玩家库中表的主键字段名称; 红色部分是需要修改的 |

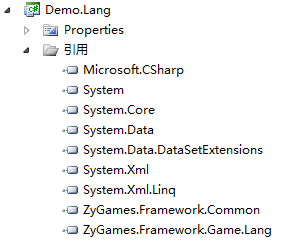
项目建立完成如图：



### Lang项目

**组件引用**

|  |  |
| --- | --- |
| ****项目**** | **引用路径** |
| Lang | **Framework\V5.1\ZyGames.Framework.Common.dll**  **Game\V2.6\ZyGames.Framework.Game.Lang.dll** |



负责处理多语言包配置，需要实现中层提供的语言包；以下是定义类

**LanguageManager类**

|  |
| --- |
| public class LanguageManager  {  private static object thisLock = new object();  private static Dictionary<LangEnum, IGameLanguage> \_langTable = new Dictionary<LangEnum, IGameLanguage>();  private static LangEnum \_langEnum;  static LanguageManager()  {  \_langEnum = (ConfigUtils.GetSetting("LanguageType", "0")).ToEnum<LangEnum>();  LanguageHelper.SetLang(\_langEnum);  }  public static IGameLanguage GetLang()  {  return GetLang(\_langEnum);  }  public static IGameLanguage GetLang(LangEnum langEnum)  {  IGameLanguage lang = null;  if (!\_langTable.ContainsKey(langEnum))  {  lock (thisLock)  {  if (!\_langTable.ContainsKey(langEnum))  {  switch (langEnum)  {  case LangEnum.ZH\_CN:  \_langTable.Add(langEnum, new GameZhLanguage());  break;  default:  throw new Exception("Language is error.");  }  }  }  }  lang = \_langTable[langEnum];  return lang;  }  } |

**IGameLanguage接口**

|  |
| --- |
| public interface IGameLanguage : ILanguage  {  #region  /// <summary>  /// 君主帐号  /// </summary>  int SystemUserId { get; }  /// <summary>  /// 玩家名称  /// </summary>  string KingName { get; }  string Date\_Yesterday { get; }  string Date\_BeforeYesterday { get; }  string Date\_Day { get; }  string St1002\_GetRegisterPassportIDError { get; }  string St1005\_NickNameOutRange { get; }  string St1005\_NickNameExistKeyword { get; }  string St1005\_NickNameExist { get; }  string St1006\_PasswordTooLong { get;}  string St1006\_ChangePasswordError { get;}  string St1006\_PasswordError { get;}  string St1066\_PayError { get; }  #endregion  } |

**GameZhLanguage类**

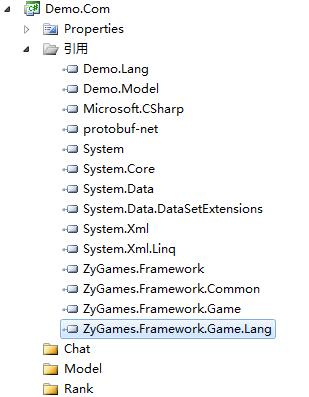
|  |
| --- |
| class GameZhLanguage : BaseZHLanguage, IGameLanguage  {  public int SystemUserId  {  get { return 1000000; }  }  public string KingName  {  get { return "系统"; }  }  public string Date\_Yesterday { get { return "昨天"; } }  public string Date\_BeforeYesterday { get { return "前天"; } }  public string Date\_Day { get { return "{0}天前"; } }  public string St1002\_GetRegisterPassportIDError { get { return "获取注册通行证ID失败!"; } }  public string St1005\_NickNameOutRange { get { return "您的昵称输入有误，请重新输入!"; } }  public string St1005\_NickNameExistKeyword { get { return "您输入的昵称存在非法字符，请重新输入!"; } }  public string St1005\_NickNameExist { get { return "您输入的昵称已存在，请重新输入!"; } }  public string St1006\_PasswordTooLong { get { return "输入错误，请输入4-12位数字或字母!"; } }  public string St1006\_ChangePasswordError { get { return "修改密码失败!"; } }  public string St1006\_PasswordError { get { return "密码格式错误!"; } }  public string St1066\_PayError { get { return "充值失败"; } }  } |

### Com项目

**组件引用**

|  |  |
| --- | --- |
| ****项目**** | **引用路径** |
| **Com** | **Plugin\Redis\protobuf-net.dll**  **Framework\V5.1\ZyGames.Framework.Common.dll**  **Framework\V5.1\ZyGames.Framework.dll**  **Game\V2.6\ZyGames.Framework.Game.Lang.dll**  **Game\V2.6\ZyGames.Framework.Game.dll**  Model  **Lang** |

划分中间件业务实体（Model），聊天组件（Chat）与排行榜（Rank）等目录，如图：

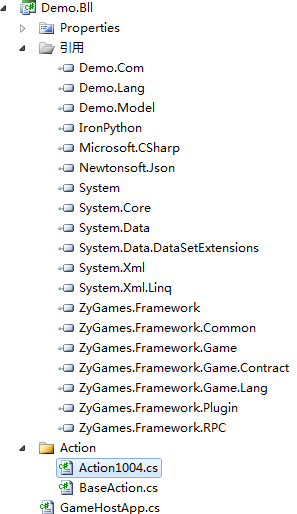


需要使用中间层的功能，请参考《中间层使用文档》

### ****Bll项目****

**组件引用**

|  |  |
| --- | --- |
| ****项目**** | **引用路径** |
| **Bll** | **Plugin\Newtonsoft.Json.dll**  **Plugin** **\Python\IronPython.dll**  **Framework\V5.1\ZyGames.Framework.Common.dll**  **Framework\V5.1\ZyGames.Framework.dll Framework\V5.1\ZyGames.Framework.Plugin.dll**  **Framework\V5.1\ZyGames.Framework.RPC.dll**  **Game\V2.6\ZyGames.Framework.Game.Lang.dll**  **Game\V2.6\ZyGames.Framework.Game.dll**  **Game\V2.6\** **ZyGames.Framework.Game.Contract.dll**  Model  **Lang**  **Com** |



**功能划分**

创建Action目录划分接口协议处理逻辑；主要提供中间层定义的固定协议接口，如：登录（1004）与建角（1005）及充值中间层接口

**ActionIDDefine类**

|  |
| --- |
| **public class ActionIDDefine**  **{**  **///<summary>**  **///客户端注册Socket**  **///</summary>**  **public const Int16 Cst\_Action100 = 100;**  **///<summary>**  **///错误日志**  **///</summary>**  **public const Int16 Cst\_Action404 = 404;**  **///<summary>**  **///注册通行证ID获取接口**  **///</summary>**  **public const Int16 Cst\_Action1002 = 1002;**  **///<summary>**  **///用户注册**  **///</summary>**  **public const Int16 Cst\_Action1003 = 1003;**  **///<summary>**  **///用户登录**  **///</summary>**  **public const Int16 Cst\_Action1004 = 1004;**  **///<summary>**  **///创建角色**  **///</summary>**  **public const Int16 Cst\_Action1005 = 1005;**  **}** |

**BaseAction类**

|  |
| --- |
| **public abstract class BaseAction : AuthorizeAction**  **{**  **protected BaseAction(short actionID, HttpGet httpGet)**  **: base(actionID, httpGet)**  **{**  **}**  **protected override bool IgnoreActionId**  **{**  **get**  **{**  **//排除不需要登录授权的协议接口**  **return actionId == ActionIDDefine.Cst\_Action404;**  **}**  **}**  **}** |

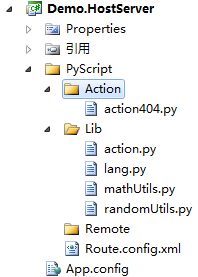
### HostServer项目

**组件引用**

|  |  |
| --- | --- |
| ****项目**** | **引用路径** |
| **HostServer** | **Plugin\Newtonsoft.Json.dll**  **Plugin\NLog.dll**  **Plugin\Redis\protobuf-net.dll**  **Plugin** **\Python\IronPython.dll**  **Plugin** **\Python\IronPython.Modules.dll**  **Plugin** **\Python\Microsoft.Dynamic.dll**  **Plugin** **\Python\Microsoft.Scripting.dll**  **Plugin\Redis\ServiceStack.dll**  **Plugin\Redis\** **ServiceStack.Common.dll**  **Plugin\Redis\** **ServiceStack.Interfaces.dll**  **Plugin\Redis\** **ServiceStack.Redis.dll**  **Plugin\Redis\** **ServiceStack.Text.dll**  **Framework\V5.1\ZyGames.Framework.Common.dll**  **Framework\V5.1\ZyGames.Framework.dll Framework\V5.1\ZyGames.Framework.Plugin.dll**  **Framework\V5.1\ZyGames.Framework.RPC.dll**  **Game\V2.6\ZyGames.Framework.Game.Lang.dll**  **Game\V2.6\ZyGames.Framework.Game.dll**  **Game\V2.6\** **ZyGames.Framework.Game.Contract.dll**  Model  **Lang**  **Com**  **Bll** |

**功能划分**

**划分PyScript目录，存放Python脚本文件；层次如图：**

****

**Action目录：处理请求与响应的脚本，可以协议生成器工具中Copy部分模板；**

**Lib目录：Python中间层脚本，复制Scut开发包中的PythonLib目录；**

**Remote：应用程序之间内通讯，访问时有IP访问限制；**

**Route.config.xml：是请求路由配置表，格式如下：**

|  |
| --- |
| <?xml version="1.0" encoding="utf-8" ?>  <config>  <!--Python安装类库路径-->  <lib path="D:\Python\Lib" />  <route-list>  <!--配置Action路由  action：映射的Action代码  path：指定执行的脚本路径  ignoreAuthorize:是否不需要登录授权，true：不需要登录授权  -->  <route action="404" path="Action\action404.py"/>  <route action="1009" path="Action\action1009.py"/>  </route-list>  </config> |

**GameHostApp类**

|  |
| --- |
| public class GameHostApp : GameHost  {  //public override void Listen()  //{  //这里处理请求与响应模式，设置连接超时等些参数  //ServiceProxy.Listen(IpAddress, Port, conn, recv, inacv, connectCount);  //ListenAfter();  //}  protected override void DoBindAfter()  {  //这里处理程序启动时逻辑  }  protected override void OnRequested(HttpGet httpGet, IGameResponse response)  {  //这里处理请求与响应模式  }  protected override void OnCallRemote(string route, HttpGet httpGet, MessageHead head, MessageStructure structure)  {  //这里处理服务器应用程序间通信  }  protected override void OnClosed(ChannelContext context, string remoteaddress)  {  //这里处理服务被关闭逻辑  }  protected override void OnSocketClosed(ChannelContext context, string remoteaddress)  {  //这里处理Socket服务被关闭逻辑  }  protected override void OnServiceStop(object sender, EventArgs eventArgs)  {  //这里处理服务被停止逻辑  }  } |

**Program类**

|  |
| --- |
| class Program  {  static void Main(string[] args)  {  GameHost.Start();  }  } |

**宿主程序Config配置**

|  |
| --- |
| <?xml version="1.0"?>  <configuration>  <configSections>  <section name="zyGameBaseBll" type="ZyGames.Framework.Game.Configuration.ZyGameBaseBllSection,ZyGames.Framework.Game"/>  <section name="zyGameBase-GM" type="ZyGames.Framework.Game.Command.GmSection,ZyGames.Framework.Game"/>  <section name="sdkChannel" type="ZyGames.Framework.Game.Sns.Section.SdkChannelSection,ZyGames.Framework.Game"/>  </configSections>  <appSettings>  <!--消息队列异步写库配置-->  <add key="MessageQueuePath" value=".\private$\DemoCmdSql"></add>  <add key="MessageQueueNum" value="10"></add>    <!--宿主程序启动配置-->  <add key="Game.Host.TypeName" value="Demo.HostServer.GameHostApp,Demo.HostServer"/>  <add key="Game.Action.TypeName" value="Demo.Bll.Action.Action{0},Demo.Bll"/>  <!--Port:接收请求的端口  connectTimeout:建立连接和传送数据的超时间(默认10秒)  receiveTimeout:保持连接的情况下，空闲超时触发Closing事件,尽量最大,默认10分钟  inactivityTimeout:连接断开的情况下，空闲超时触发Faulted事件,要小于receiveTimeout时间间隔(默认5秒)  connectCount:允许连接的并发数，默认100  -->  <add key="Game.Port" value="9001" />  <add key="Game.wcf.ConnectTimeout" value="10" />  <add key="Game.wcf.ReceiveTimeout" value="1800" />  <add key="Game.wcf.InactivityTimeout" value="5" />  <add key="Game.wcf.ConnectCount" value="1000"/>  <!--Code：游戏代码；ServerId：游戏分区代码-->  <add key="Product.Code" value="1"/>  <add key="Product.Name" value="游戏名称"/>  <add key="Product.ServerId" value="1"/>  <!-- 配置语言版本，ZH\_CN：简体,EN\_US：英文,BIG5\_TW：繁体 -->  <add key="LanguageType" value="ZH\_CN"/>  <!--发布的版本类型，Debug：测试，Release：正式-->  <add key="PublishType" value="Debug"></add>  <!--GM功能是否开启，true：开启-->  <add key="EnableGM" value="true"></add>  <!--协议接口请求超时监控,单位毫秒-->  <add key="ActionTimeOut" value="600"></add>  <!--Python脚本配置  Python\_IsDebug：是否开启调试功能  PythonRootPath：脚本路径，相对于程序运行目录  -->  <add key="Python\_IsDebug" value="true"/>  <add key="PythonRootPath" value="..\..\PyScript"/>    <!--游戏缓存配置  global.period：全局缓存生命周期24小时  user.period：玩家缓存生命周期8小时  CacheManager\_Interval：缓存监控时间间隔  -->  <add key="Cache.global.period" value="86400"/>  <add key="Cache.user.period" value="28800"/>  <add key="CacheManager\_Interval" value="6000"/>    <!--通用组件配置开始-->  <add key="PayDB\_ConnectionString" value="Data Source={0};Database=PayDB;{1}; Pooling=true;"/>  <add key="PayDB\_Server" value="."/>  <add key="PayDB\_Acount" value="加密密码"/>  <add key="Snscenter\_ConnectionString" value="Data Source={0};Database=snscenter;{1}; Pooling=true;"/>  <add key="Snscenter\_Server" value="."/>  <add key="Snscenter\_Acount" value="加密密码"/>  <!--通用组件配置结束-->  </appSettings>  <connectionStrings>  <add name="DemoConfig" providerName="" connectionString="Data Source=.;Database=DemoConfig;Uid=;Pwd=; Pooling=true;"/>  <add name="DemoData" providerName="" connectionString="Data Source=.;Database=DemoData;Uid=;Pwd=; Pooling=true;"/>  <add name="DemoLog" providerName="" connectionString="Data Source=.;Database=DemoLog;Uid=;Pwd=; Pooling=true;"/>  </connectionStrings>  <!--业务层自定义配置-->  <zyGameBaseBll>  <login defaultType="ZyGames.Framework.Game.Sns.Login36you,ZyGames.Framework.Game">  <retailList>  <add id="0000" type="ZyGames.Framework.Game.Sns.Login36you,ZyGames.Framework.Game" args="Pid,Pwd,DeviceID"/>  </retailList>  </login>  </zyGameBaseBll>  <zyGameBase-GM>  <command>  <!--GM:cache-->  </command>  </zyGameBase-GM>  <startup>  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>  </startup>  </configuration> |

## 定义协议

打开协议生成器工具，增加一个“Demo”项目方案，接着在增加协议（或从现有项目中Copy相同的接口协议），接着定义客户端提供的请求参数和服务器下发的参数（支持多行的格式，字段类型：Record与End组合）；定义好客户端与服务器之前通讯的接口后，使用自动生成的服务端Python代码（客户端使用Lua脚本代码）复制到已创建的Python接口文件，如图：



## 项目部署

需要架设Socket方式的分发器（或Http方式的分发器，两都可一起架设），将客户端请求先通过Socket或Http分发器再转发到游戏服务器（HostServer程序）上；

## Socket分发器

使用OA部署功能架设分发器；在本机使用批处理文件，如下：

|  |
| --- |
| C:\Windows\System32\schtasks.exe /Create /RU system /TN DemoSocketApp /SC ONSTART /TR " F:\Demo Socket\Doudizhu\ZyGames.Framework.Game.SocketServer.exe" /F |

## HostServer程序

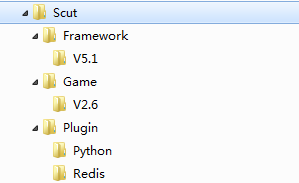
使用OA部署功能架设游戏服；在本机使用批处理文件，加在任务计划中启动，如下：

|  |
| --- |
| C:\Windows\System32\schtasks.exe /Create /RU system /TN DemoHostApp /SC ONSTART /TR "F:\Demo\Demo.HostServer.exe" /F |

# 参考

## Scut开发包

组件目录结构：（注：Plugin目录下是第三方组件）



Framework目录下是框架核心组件；Game目录下是中间层组件，可提供业务层扩展；

## CodeSmith实体模板

