使用文档（3）

如何开始开发游戏

# 基础篇

## 开发环境

**必须安装的软件如下**：

* **Redis**：key-value数据库，2.6.x以上版本
* **数据库**：MSSQL 2005以上版本
* **Web服务器**： iis 7.0以上版本
* **.NET运行库**：.Net Framework 4.0以上版本

（**注：6.2.7.0之后版本要使用.NET运行库4.5.1版本** ）

**开发软件**

1. VS2010版本以上开发工具，建议使用VS2013，[下载正式版地址](http://www.cocoachina.com/bbs/read.php?tid=195353)
2. IronPython2.6.1 RC1 for .NET 4.0 RC插件（使用Python脚本开发时安装）

## 协议生成器

协方生成器工具，它提供开发人员在开发阶段快速生成客户端与服务器端的通讯协议代码，减少因通讯协议写错造成两边联调时间，提高开发效率；（注：在生产环境运行时，就不再需要它了）。也可模拟客户端发送Http或Socket请求，测试服务端Action协议，服务端是Console程序则使用Socket请求，Web程序则使用Http请求。

**数据库初始化**

采用批处理方式创建（批处理文件在当前下载Scut目录：E:\Scut.svn\Source\Tools\ContractTools\db\mssql\必备库创建一键运行.bat）；使用Sa帐号（安装MSSQL数据库时的帐号）创建数据库（或用DB维护者权限的帐号），配置需要相应修改；bat文件如下：

|  |
| --- |
| set dbServer=. 数据库服务地址  set dbAcount=sa 帐号（有创建库权限的）  set dbPass=123 密码  set gameuser=game\_user 需要创建游戏服使用的帐号  set gamepass=123 游戏服使用的密码  set dbpath=%cd%\Data 数据库存储位置目录（默认当前目录） |

**站点部署**

1. 打开控制面板->管理工具-> Internet 信息服务(IIS)管理器
2. 在网站结点右击，添加站点contract.scutgame.com
3. 将站点路径指向到Scut:\Source\Tools\ContractTools\release目录
4. 设置应用程序池“Framework v4.0”和“经典”模式
5. 设置Host文件，添加配置“127.0.0.1 contract.scutgame.com”
6. 修改数据库连接，打开Web.config文件（如果使用MySql，设置成providerName=“MySqlDataProvider”）

|  |
| --- |
| <?xml version="1.0"?>  <configuration>  <connectionStrings>  <add name="Contract" providerName="" connectionString="server=.;database=ContractDB;uid=sa;pwd=123" />  </connectionStrings>  <system.web>  <compilation targetFramework="4.0" />  <httpRuntime requestValidationMode="2.0" />  </system.web>  </configuration> |

### 使用说明

增加的参数类型分分：请求参数和返回参数；

请求参数：由Client提供上传的参数定义，遵守URL参数标准，格式：name=Jim&age=1；

返回参数：由游戏服返回给Client的流数据格式（二进制），下发的字段类型（short，int，string，byte）基础类型，Record和End是下发行集合的循环体开始和结尾标记；例：要返回发下学生信息

-------------------------------------------

Id Name Age

1 Jim 20

2 Coco 18

-------------------------------------------

定义格式：

--------------------------------------------

Record

Id Int

Name String

Age Age

End

--------------------------------------------

## 分服中心站点（DirCenter）

DirCenter 是分服中心，负责管理游戏信息与游戏服（区）信息，提供以下接入口：

1. Service.aspx页面提供给Client获得指定游戏的服（区）信息，用户选一个服进入后，访问相应的游戏服（获得Url地址，以Http或Socket方式访问），内容格式是二进制流；
2. OfficialService.aspx页面提供给官网获得游戏与服信息，如官网排行榜，内容是Json格式；
3. DirService.asmx提供WebService服务方式给后台管理使用（增、改、删）等操作，一般需要在IIS设置IP限制访问；

**站点部署**

1. 打开控制面板->管理工具-> Internet 信息服务(IIS)管理器
2. 在网站结点右击，添加站点dir.scutgame.com
3. 将站点路径指向到Scut: \Source\DirCenter\release目录
4. 设置应用程序池“Framework v4.0”和“经典”模式
5. 设置Host文件，添加配置“127.0.0.1 dir.scutgame.com”
6. 修改数据库连接，打开Web.config文件（如果使用MySql，设置成providerName=“MySqlDataProvider”）

|  |
| --- |
| <?xml version="1.0"?>  <configuration>  <connectionStrings>  <add name=" DirData" providerName="" connectionString=" Data Source=.;Database=PayDB;Uid=game\_user;Pwd=123;" />  </connectionStrings>  <system.web>  <compilation targetFramework="4.0" />  </system.web>  </configuration> |

## Lua调试

使用Decoda工具调试Lua代码，首先打开Decoda程序，在Project ->Setting项设置如下：



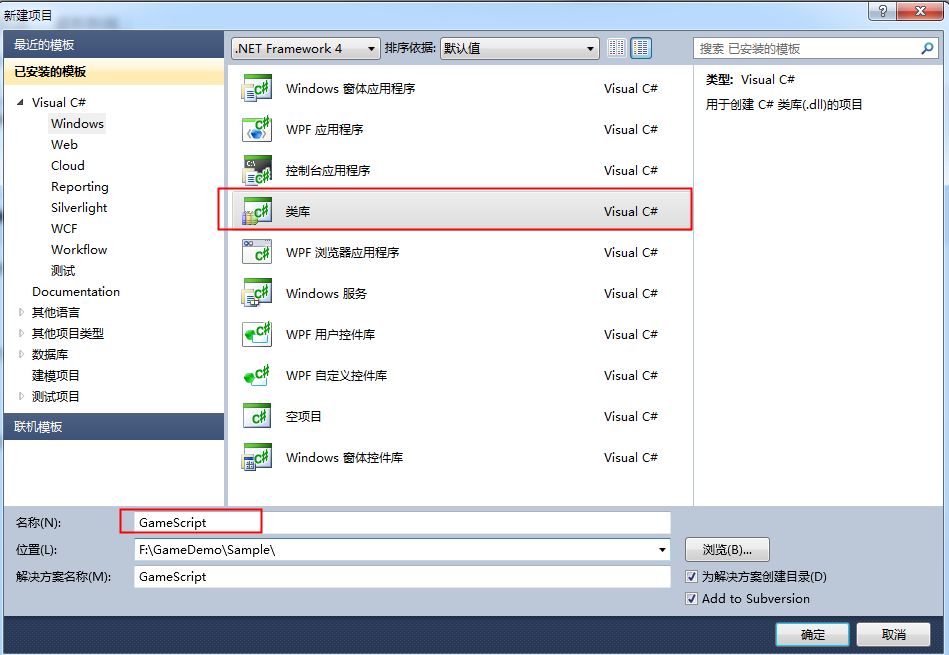
接着选择在Debug-> Start Debuging 启动调试（或按F5）；如果是竖屏的，需要设置Command栏：“460 960”



## 搭建游戏Console项目

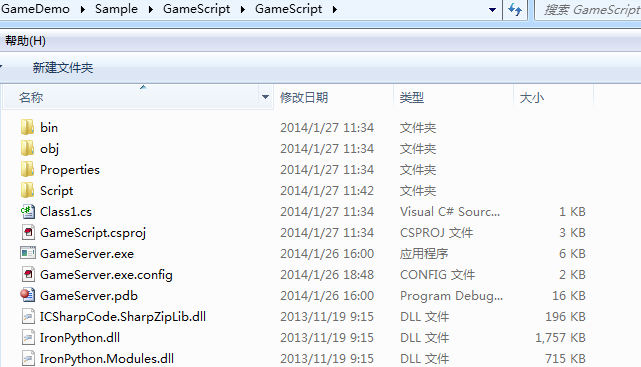
### 新建项目

打开VS2010，新建“类库”项目，输入游戏的名称（如：GameScript），确定生成；如图：

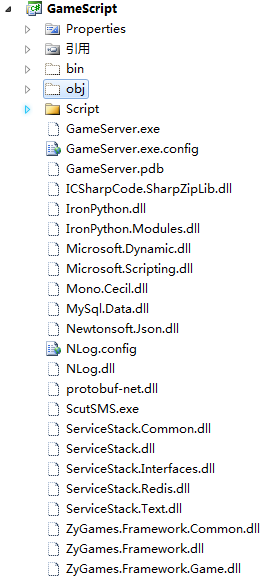


### 拷贝Scut Lib文件

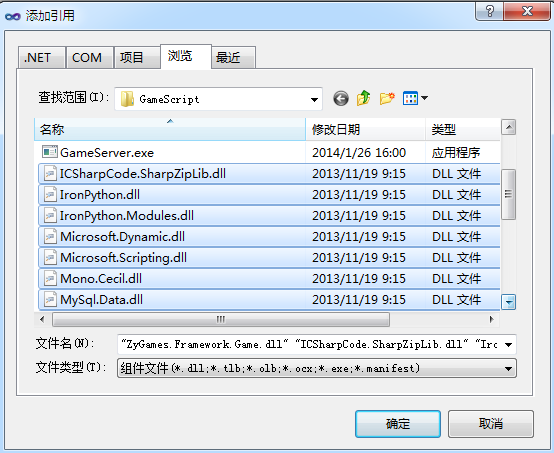
打开Scut.svn\Release\xx\Console目录(注:xx最新版本)（可以从[官网](http://www.scutgame.com/download/)或[GitHub](https://github.com/ScutGame/Scut/tree/master/Release)上下载），将所有文件Copy至项目GameScript目录下，如图：



可以选中GameScript项目点击显示所有文件，将GameServer.exe.config与NLog.config文件和Script目录包括进项目中，如图：

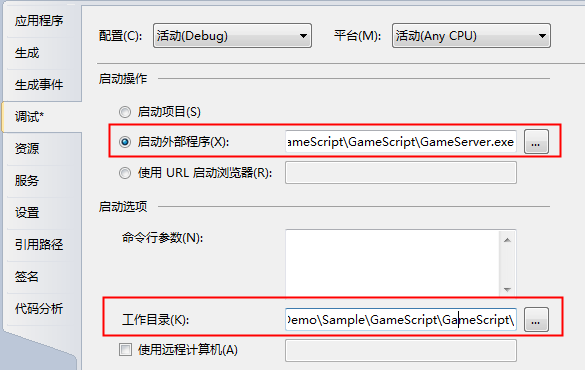


添加项目引用，将GameScript目录下的所有Dll文件全部引用进项目，如图：

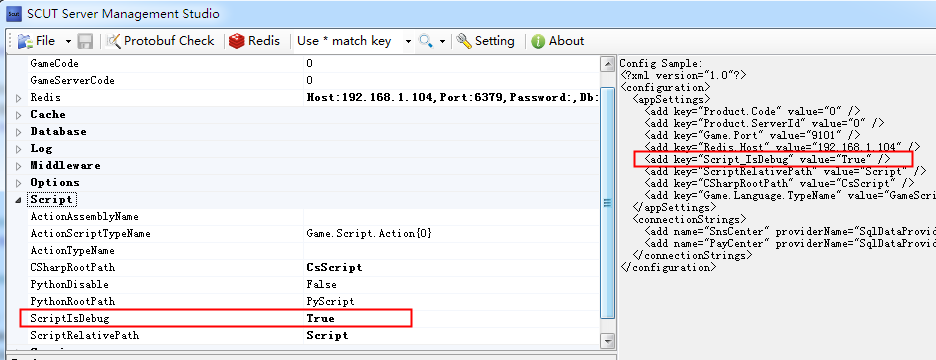


### 配置项目

右击GameScript项目选“属性”，打开“调试”栏选择“启动外部程序”设置为“GameServer.exe”路径，并将工作目录设置GameScrip项目的目录(配置当前相对目录:"GameScript\GameServer.exe"和"..\..\"工作目录)，如图：

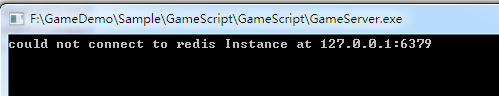


接着修改Config配置，打开项目中的ScutSMS.exe程序，设置ScriptIsDebug属性为True，并点击保存；如图：



### 调试或运行

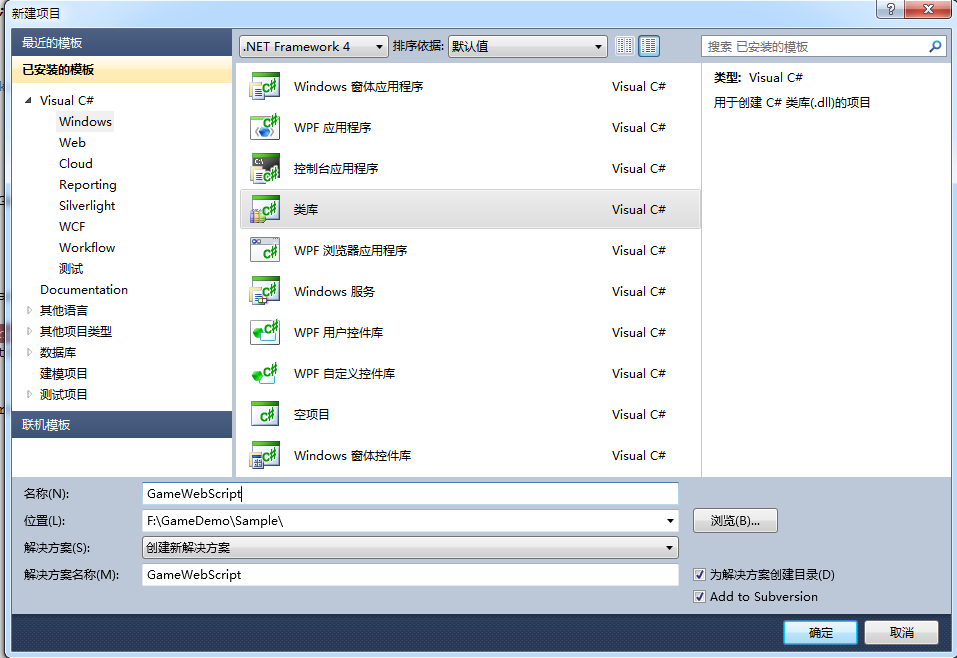
在MainClass类的OnStartAffer方法中设置断点，按F5运行调试，如果出现如下错误修改“GameServer.exe.config”文件的Redis服务器地址（或Redis检查是否启动；



## 搭建游戏Web项目

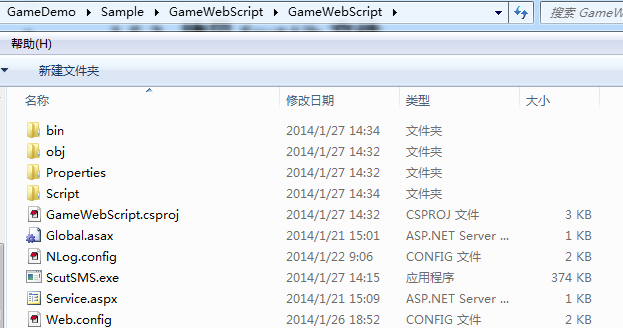
### 新建项目

打开VS2010，新建“类库”项目，输入游戏的名称（如：GameWebScript），确定生成；如图：

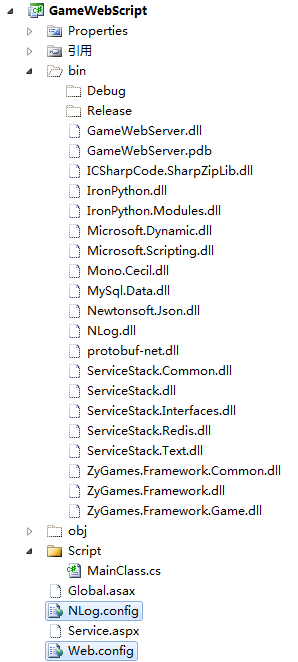


### 拷贝Scut Lib文件

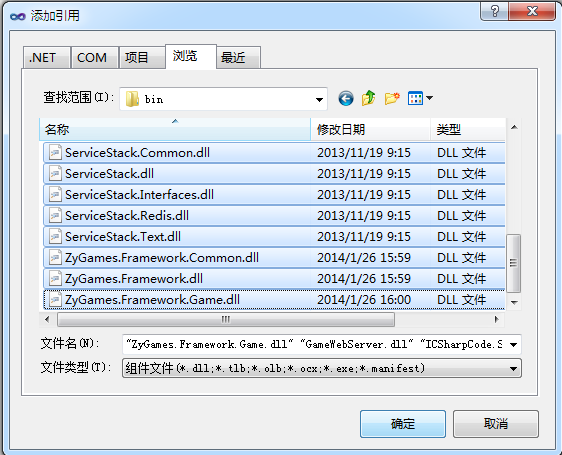
打开Scut.svn\Release\6.1.5.6\Web目录（可以从[官网](http://www.scutgame.com/download/)或[GitHub](https://github.com/ScutGame/Scut/tree/master/Release)上下载），将所有文件Copy至项目GameWebScript目录下，如图：



可以选中GameWebScript项目点击显示所有文件，将Web.config与NLog.config文件和Script目录包括进项目中，如图：

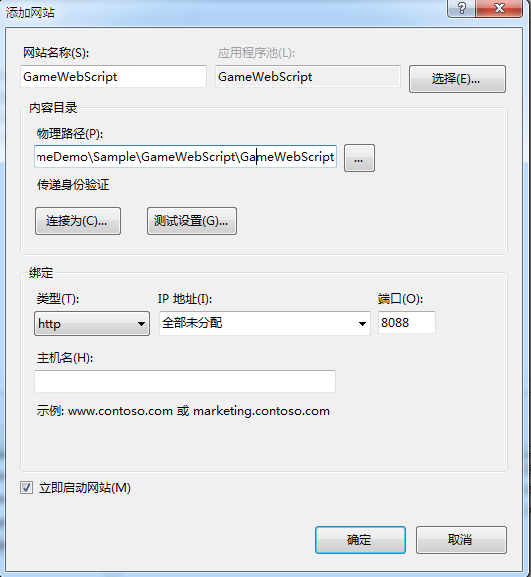


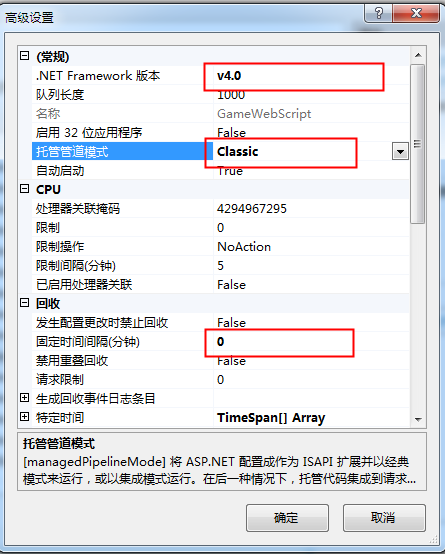
添加项目引用，将GameScript目录下的所有Dll文件全部引用进项目，如图：



### IIS建立站点

打开IIS管理器，建立“GameWebScript”站点，并设置应用程序池为4.0，如图：



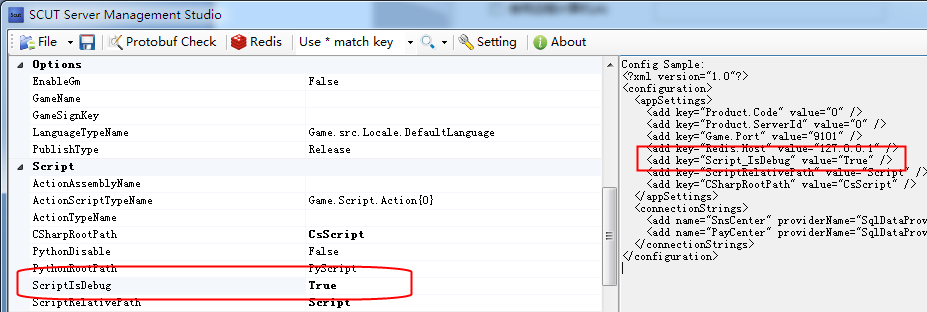


### 配置项目

右击GameWebScript项目选“属性”，打开“调试”栏选择“使用URL启动浏览器”设置为“GameServer.exe”路径，并将工作目录设置GameScrip项目的目录，如图：

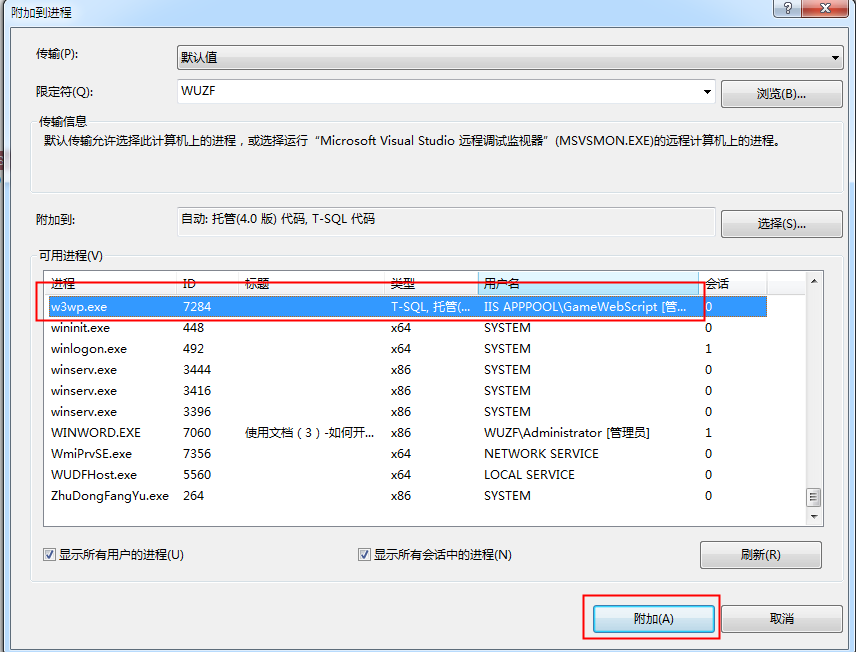


接着修改Config配置，打开项目中的ScutSMS.exe程序，设置ScriptIsDebug属性为True，并点击保存；如图：



### 调试或运行

在MainClass类的Start方法中设置断点，使用附加到进程调试（如果未找到w3wp.exe进程需要先在IE访问站点，再附加），如图；



## 自定义通讯结构（Unity3D接入）

通讯协议可以自己定制，以Byte字节数据方式传递数据，实现自己需要的接收消息格式和响应格式，以下例子使用Protobuf序列化对象成Byte传递数据，定义字节流格式：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 第一段（4位长度） | 第二段 | 第三段 |
| 字节流段 | Byte[L] | Byte[N] | Byte[M] |
| 说明 | headLength | headBytes | contentBytes |

L代表头部信息字节数组长度的字节数据（int占4位）

N代表头部信息字节数组长度

M代表内容序列化后的字节数组长度

以下结构仅供参考（在Lib 6.3.7.5版本提供此功能），Sample在GitHub上的目录Sample\GameRanking\Server\src\GameRanking.HostU3D4\_5

### 定义通讯结构包模型

新建立GameRanking.Pack类库项目，定义自定义的通讯协议结构Model，定义请求包头部结构MessagePack类和响应包头部结构ResponsePack类，并在GameServer所在项目的配置文件中的配置键ScriptAsmReferences属性值为Game.Pack.dll），MessagePack类实现如下：

|  |
| --- |
| [ProtoContract]  public class MessagePack  {  [ProtoMember(1)]  public int MsgId { get; set; }  [ProtoMember(2)]  public int ActionId{ get; set; }  [ProtoMember(3)]  public string SessionId{ get; set; }  [ProtoMember(4)]  public int UserId{ get; set; }  } |

ResponsePack类实现如下：

|  |
| --- |
| [ProtoContract]  public class Request1001Pack  {  [ProtoMember(101)]  public int PageIndex { get; set; }  [ProtoMember(102)]  public int PageSize { get; set; }  } |

业务协议结构

|  |
| --- |
| [ProtoContract]  public class Response1001Pack  {  [ProtoMember(101)]  public int PageCount { get; set; }  public List<RankData> Items { get; set; }  }  [ProtoContract]  public class RankData  {  [ProtoMember(101)]  public string UserName { get; set; }  [ProtoMember(102)]  public int Score { get; set; }  } |

将Game.Pack.dll类库提供给Unity客户端（同为C#语言才使用此模型）

GameServer.exe.config文件如下配置：

|  |
| --- |
| <?xml version="1.0"?>  <configuration>  <appSettings>  <add key="ScriptAsmReferences" value="GameRanking.Pack.dll" />  </appSettings>  </configuration> |

### 实现IActionDispatcher分发器接口

实现自定的Action分发器管理类（CustomActionDispatcher），实现对byte数据解析包TryDecodePackage方法，业务接口包结构如下

|  |
| --- |
| public class CustomActionDispatcher : IActionDispatcher  {  public bool TryDecodePackage(ConnectionEventArgs e, out RequestPackage package)  {  //这里解出头部信息根据ActionId来分发请求到相应的Action子类  package = null;  byte[] content;  MessagePack head = ReadMessageHead(e.Data, out content);  if (head == null)  {  return false;  }  package = new RequestPackage(head.MsgId, head.SessionId, head.ActionId, head.UserId) { Message = content };  return true;  }  /// <summary>  /// 定义byte[]格式：headLength + headBytes + contentBytes  /// </summary>  /// <returns></returns>  private MessagePack ReadMessageHead(byte[] data, out byte[] content)  {  MessagePack headPack = null;  content = new byte[0];  try  {  //解头部(解之前当然还需要对byte[]解密，这里跳过这步)  int pos = 0;  byte[] headLenBytes = new byte[4];  Buffer.BlockCopy(data, pos, headLenBytes, 0, headLenBytes.Length);  pos += headLenBytes.Length;  int headSize = BitConverter.ToInt32(headLenBytes, 0);  if (headSize < data.Length)  {  byte[] headBytes = new byte[headSize];  Buffer.BlockCopy(data, pos, headBytes, 0, headBytes.Length);  pos += headBytes.Length;  headPack = ProtoBufUtils.Deserialize<MessagePack>(headBytes);  //解消息的内容  if (data.Length > pos)  {  int len = data.Length - pos;  content = new byte[len];  Buffer.BlockCopy(data, pos, content, 0, content.Length);  //内容数据放到具体Action业务上处理  }  }  else  {  //不支持的数据格式  }  }  catch (Exception ex)  {  //不支持的数据格式  }  return headPack;  }  public ActionGetter GetActionGetter(RequestPackage package)  {  // 可以实现自定的ActionGetter子类  return new ActionGetter(package);  }  public void ResponseError(BaseGameResponse response, ActionGetter actionGetter, int errorCode, string errorInfo)  {  //实现出错处理下发  ResponsePack head = new ResponsePack()  {  MsgId = actionGetter.GetMsgId(),  ActionId = actionGetter.GetActionId(),  ErrorCode = errorCode,  ErrorInfo = errorInfo  };  byte[] headBytes = ProtoBufUtils.Serialize(head);  byte[] buffer = BufferUtils.AppendHeadBytes(headBytes);  response.BinaryWrite(buffer);  }  } |

启用CustomActionDispatcher对象

|  |
| --- |
| [ProtoContract]  public class MainClass : GameSocketHost, IMainScript  {  public MainClass()  {  ActionDispatcher = new CustomActionDispatcher();  }  } |

### 定义Action协议接口

定义BaseStruct子类，实现父亲GetUrlElement、TakeAction、WriteResponse方法，如排行榜列表1001接口下：

|  |
| --- |
| using System;  using System.Collections.Generic;  using GameServer.Model;  using ZyGames.Framework.Cache.Generic;  using ZyGames.Framework.Common;  using ZyGames.Framework.Game.Contract;  using ZyGames.Framework.Game.Service;  using GameRanking.Pack;  using ZyGames.Framework.Common.Serialization;  using ZyGames.Framework.RPC.IO;  namespace GameServer.CsScript.Action  {  public class Action1001 : BaseStruct  {  private Request1001Pack requestPack;  private Response1001Pack responsePack;  public Action1001(ActionGetter actionGetter)  : base(1001, actionGetter)  {  }  public override bool GetUrlElement()  {  byte[] data = (byte[])actionGetter.GetMessage();  if (data.Length > 0)  {  requestPack = ProtoBufUtils.Deserialize<Request1001Pack>(data);  return true;  }  return false;  }  public override bool TakeAction()  {  int pageCount;  var cache = new ShareCacheStruct<UserRanking>();  var rankingList = cache.FindAll(false);  rankingList = MathUtils.QuickSort<UserRanking>(rankingList, compareTo);  rankingList = rankingList.GetPaging(requestPack.PageIndex, requestPack.PageSize, out pageCount);  responsePack = new Response1001Pack();  responsePack.PageCount = pageCount;  responsePack.Items = new List<RankData>();  foreach (var item in rankingList)  {  responsePack.Items.Add(new RankData() { UserName = item.UserName, Score = item.Score });  }  return true;  }  public override void WriteResponse(BaseGameResponse response)  {  var packageHead = new ResponsePack()  {  MsgId = actionGetter.GetMsgId(),  ActionId = actionGetter.GetActionId(),  ErrorCode = 0,  ErrorInfo = ""  };  byte[] headBytes = ProtoBufUtils.Serialize(packageHead);  byte[] contentBytes = ProtoBufUtils.Serialize(responsePack);  byte[] buffer = BufferUtils.MergeBytes(  BufferUtils.AppendHeadBytes(headBytes),  contentBytes  );  //需要对字节数据加密处理，这里跳过  response.BinaryWrite(buffer);  }  private int compareTo(UserRanking x, UserRanking y)  {  int result = y.Score - x.Score;  if (result == 0)  {  result = y.UserID - x.UserID;  }  return result;  }  }  } |

## Model实体类设计

游戏的实体数据设计主要划分成四类：玩家私有数据，玩家共享数据，配置数值数据和日志数据；

### 玩家私有数据

配置参数CacheType.Dictionary类型：配置实体是玩家私有的，任何玩家都可以访问此数据，且实体类需要继承BaseEntity类；玩家任务示例如下：（注：” **ConnData**“是Config配置数据库连接串的Name值）

|  |
| --- |
| <connectionStrings>  <add name="**ConnData**" providerName="SqlDataProvider" connectionString="Data Source=localhost;Database=PHData;Uid=game\_user;Pwd=123;"/>  </connectionStrings> |

|  |
| --- |
| namespace Demo.Model  {  [Serializable, ProtoContract]  [EntityTable(CacheType.Dictionary, "**ConnData**")]  public class UserTask : BaseEntity  {  public UserTask ()  : base(false)  {  }  [ProtoMember(1)]  [EntityField(true)]  public int UserID  {  get;  set;  }  [ProtoMember(2)]  [EntityField(true, ColumnDbType.Text)]  public CacheList<UserTaskInfo> TaskPackage  {  get;  set;  }  protected override int GetIdentityId()  {  return UserID;  }  }  [Serializable, ProtoContract]  public class UserTaskInfo : EntityChangeEvent  {  public UserTaskInfo()  : base(false)  {  }  [ProtoMember(1)]  Public int TaskId { get; set; }    [ProtoMember(2)]  Public int Status{ get; set; }  }  } |

### 玩家共享数据

配置参数CacheType.Entity类型，配置实体是全局可读写的，实体类需要继承ShareEntity类；如排行榜示例：

|  |
| --- |
| namespace Demo.Model  {  [Serializable, ProtoContract]  [EntityTable(CacheType.Entity, "ConnData")]  public class UserRanking : ShareEntity  {  public UserRanking ()  : base(false)  {  }  [ProtoMember(1)]  [EntityField(true)]  public int UserID  {  get;  set;  }  [ProtoMember(2)]  [EntityField]  public string UserName  {  get;  set;  }  }  } |

### 配置数值数据

配置参数AccessLevel.ReadOnly类型，配置实体是只读的，实体类需要继承ShareEntity类；如地图信息示例：

|  |
| --- |
| namespace Demo.Model  {  [Serializable, ProtoContract]  [EntityTable(AccessLevel.ReadOnly, "ConnConfig")]  public class MapInfo : ShareEntity  {  public MapInfo ()  : base(true)  {  }  [ProtoMember(1)]  [EntityField(true)]  public int MapID  {  get;  set;  }  [ProtoMember(2)]  [EntityField]  public int PointX  {  get;  set;  }  [ProtoMember(3)]  [EntityField]  public int PointY  {  get;  set;  }  }  } |

### 日志数据

配置参数AccessLevel.WriteOnly类型,且实体类需要继承LogEntity类；如玩家登录日志示例：

|  |
| --- |
| namespace Demo.Model  {  [Serializable, ProtoContract]  [EntityTable(AccessLevel.WriteOnly, "ConnLog")]  public class UserLoginLog: LogEntity  {  public UserLoginLog ()  {  }  //配置自增列  [ProtoMember(1)]  [EntityField(true, IsIdentity = true)]  public int ID  {  get;  set;  }  [ProtoMember(2)]  [EntityField]  public string SessionID  {  get;  set;  }  [ProtoMember(3)]  [EntityField]  public DateTime LoginDate  {  get;  set;  }  }  } |

### 组合类结构

所有孩子类需要继承EntityChangeEvent类，并且需要标记Serializable和ProtoContract扩展属性，成员和属性成员需要标记ProtoMember扩展属性；

**EntityChangeEvent子类结构**

**父亲与子类是一对一的关系**

|  |
| --- |
| namespace Demo.Model  {  [Serializable, ProtoContract]  public class ChildItem : EntityChangeEvent  {  public ChildItem() : base(false)  {  }  [ProtoMember(1)]  public int Name{ get; set; }  }  [Serializable, ProtoContract]  [EntityTable(CacheType.Entity, "ConnData")]  public class MyClass : ShareEntity  {  public MyClass () : base(false)  {  Item = new ChildItem();  }  [ProtoMember(1)]  [EntityField(true)]  public int ID  {  get;  set;  }  [ProtoMember(2)]  [EntityField(true, ColumnDbType.Text)]  public ChildItem Item  {  get;  set;  }  }  } |

**CacheList链表结构**

**父亲与子类是一对多的关系**

|  |
| --- |
| namespace Demo.Model  {  [Serializable, ProtoContract]  public class ChildItem : EntityChangeEvent  {  public ChildItem() : base(false)  {  }  [ProtoMember(1)]  public int Name{ get; set; }  }  [Serializable, ProtoContract]  [EntityTable(CacheType.Entity, "ConnData")]  public class MyClass : ShareEntity  {  public MyClass () : base(false)  {  Items = new CacheList<ChildItem>();  }  [ProtoMember(1)]  [EntityField(true)]  public int ID  {  get;  set;  }  [ProtoMember(2)]  [EntityField(true, ColumnDbType.Text)]  public CacheList<ChildItem> Items  {  get;  set;  }  }  } |

**CacheDictionary字典结构**

**父亲与子类是一对多的关系**

|  |
| --- |
| namespace Demo.Model  {  [Serializable, ProtoContract]  public class ChildItem : EntityChangeEvent  {  public ChildItem() : base(false)  {  }  [ProtoMember(1)]  public int Name{ get; set; }  }  [Serializable, ProtoContract]  [EntityTable(CacheType.Entity, "ConnData")]  public class MyClass : ShareEntity  {  public MyClass () : base(false)  {  Items = new CacheDictionary<int, ChildItem>();  }  [ProtoMember(1)]  [EntityField(true)]  public int ID  {  get;  set;  }  [ProtoMember(2)]  [EntityField(true, ColumnDbType.Text)]  public CacheDictionary<int, ChildItem> Items  {  get;  set;  }  }  } |

## 使用CacheStruct缓存

CacheStruct缓存分为：ShareCacheStruct和PersonalCacheStruct类；

ShareCacheStruct类：全局的缓存结构类型，从Redis内存或DB中加载全部玩家的数据驻留内存；

PersonalCacheStruct类：玩家私有的结构类型，一般以UserId为主键，只会从Redis内存或DB中加载属于自己的数据；

### 查找缓存实体

ShareCacheStructs查找方法包括：Find、FindKey、FindAll方法（若为空会自动从Redis或DB中加载数据），而Foreach方法不会加载，只在所有玩家的数据缓存中查找

|  |
| --- |
| //ShareCacheStruct  int Id = 1;  intt userId=1380001  var userRankCache = new ShareCacheStruct<UserRank>();  UserRank userRank = userRankCache.FindKey(Id);  userRank = userRankCache.Find(t=>t.UserName=="xxx");  List<UserRank> userRankList = userRankCache.FindAll(t => t.UserId == userId); |

PersonalCacheStruct查找方法包括：TryFind、TryFindKey、TryFindAll、Find、FindKey、FindAll方法（若为空会自动从Redis或DB中加载数据），而FindGlobal、Foreach方法不会加载，只在所有玩家的数据缓存中查找；

|  |
| --- |
| //PersonalCacheStruct  int userId = 1380001;  string pessionId = userId.ToString();  var itemCache = new PersonalCacheStruct<UserItem>();  UserItem userItem;  if (itemCache.TryFindKey(pessionId, out userItem, userId) == LoadingStatus.Success)  {  }  List<UserItem> userItemList;  if (itemCache.TryFindAll(pessionId, t => t.Userid == userId, true, out userItemList) == LoadingStatus.Success)  {  }  //FindGlobal只查询，不会从Redis中加载数据  userItemList = itemCache.FindGlobal(t => t.Userid == userId); |

### 更新缓存实体

|  |
| --- |
| //ShareCacheStruct  int userId = 1380001;  var userRankCache = new ShareCacheStruct<UserRank>();  UserRank userRank = new UserRank();  userRank.Id = userRankCache.GetNextNo();  userRankCache.Add(userRank);  userRank = userRankCache.FindKey(Id);  if (userRank != null)  {  userRank.ModifyLocked(() =>  {  userRank.UserName = "xxxx";  userRank.Sore = 100;  });  } |

|  |
| --- |
| //PersonalCacheStruct  int userId = 1380001;  string pessionId = userId.ToString();  var itemCache = new PersonalCacheStruct<UserItem>();  UserItem userItem;  if (itemCache.TryFindKey(pessionId, out userItem, userId) == LoadingStatus.Success)  {  userItem.ModifyLocked(() =>  {  userItem.Name = "xxx";  });  } |

### 删除缓存实体

|  |
| --- |
| //ShareCacheStruct  int userId = 1380001;  var userRankCache = new ShareCacheStruct<UserRank>();  UserRank userRank = userRankCache.FindKey(userId);  userRankCache.Delete(userRank); |

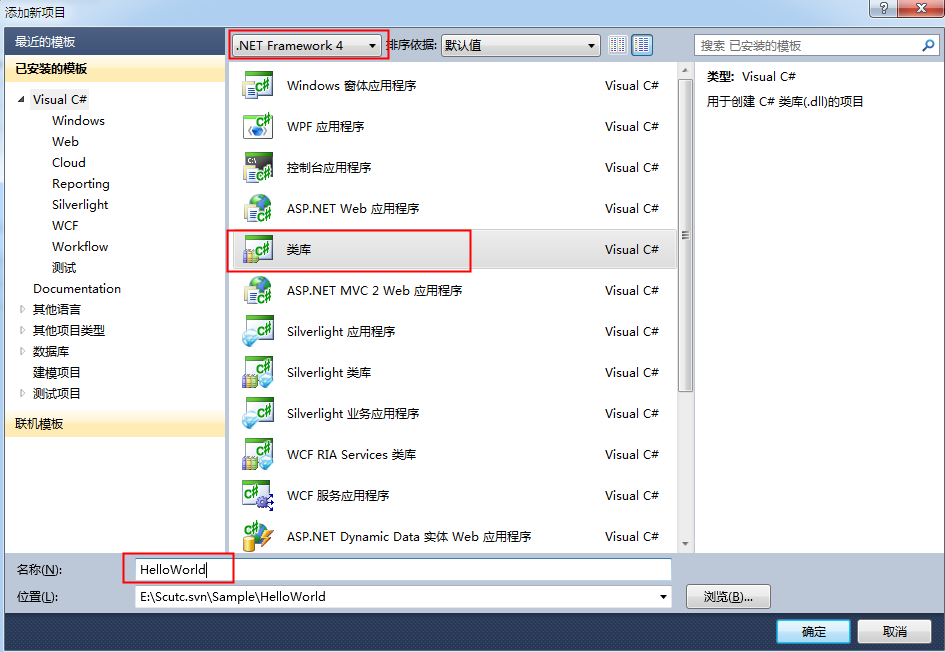
|  |
| --- |
| //PersonalCacheStruct  int userId = 1380001;  string pessionId = userId.ToString();  var itemCache = new PersonalCacheStruct<UserItem>();  UserItem userItem;  if (itemCache.TryFindKey(pessionId, out userItem, userId) == LoadingStatus.Success)  {  itemCache.Delete(userItem);  } |

# 示例篇

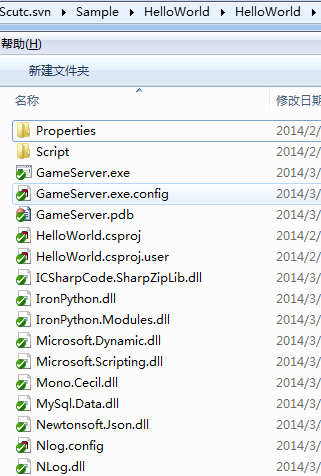
## HelloWorld程序（Console C#版本）

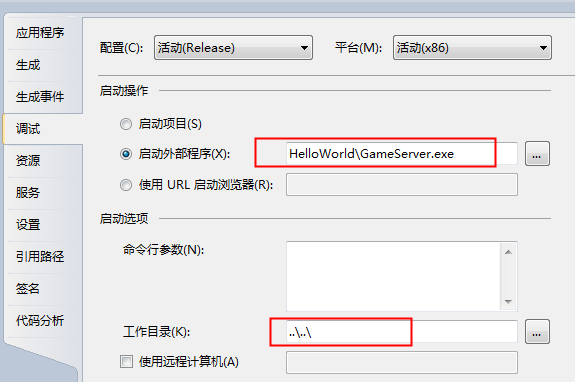
### 创建项目

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“Visual C#”，选择“类库”，选择“.Net Framework 4.0”后，输出项目名称（HelloWorld）及位置，再点击“确定”；如图：



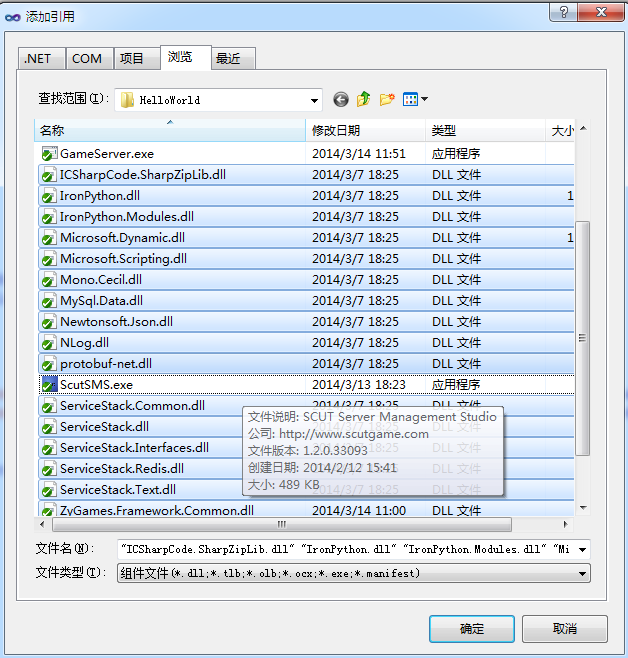
接着把Scut.svn\Release\xx\Console目录(注:xx最新版本)下的所有文件（包括Script目录）Copy到项目”HelloWorld“目录下；在HelloWorld项目，右击点“属性”，在“调试”栏设置启动操作为“启动外部程序”路径：”HelloWorld\GameServer.exe“和工作目录 "..\..\"；如图：





### ****Lib引用****

添加引用HelloWorld目录下的所有dll文件；



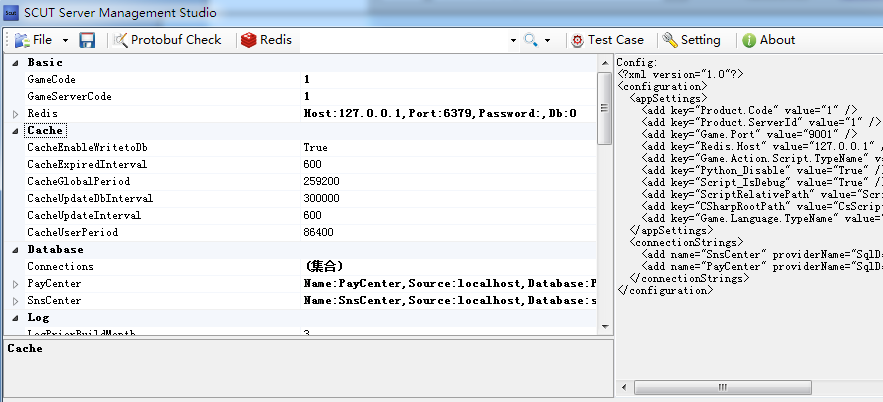
### ****添加项目协议****

打开协议生成器，增加“HelloWorld”项目方案，并定义Action1001接口协议；如下：（参考[例子](#_使用说明)）



### ****使用C#脚本****

在项目“HelloWorld”下，新建“Script/CsScript/Action”目录存放C#脚本文件；并设置脚本的解析命名空间，使用ScutSMS工具打开GameServer.exe.config配置文件设置如下：



|  |
| --- |
| <?xml version="1.0"?>  <configuration>  <appSettings>  <add key="Product.Code" value="1" />  <add key="Product.ServerId" value="1" />  <add key="Game.Port" value="9001" />  <add key="Redis.Host" value="127.0.0.1" />  <add key="Game.Action.Script.TypeName" value="HelloWorld.Script.CsScript.Action.Action{0}" />  <add key="Python\_Disable" value="True" />  <add key="Script\_IsDebug" value="True" />  <add key="ScriptRelativePath" value="Script" />  <add key="CSharpRootPath" value="CsScript" />  <add key="Game.Language.TypeName" value="GameServer.CsScript.Locale.SimplifiedLanguage" />  </appSettings>  <connectionStrings>  <add name="SnsCenter" providerName="SqlDataProvider" connectionString="Data Source=localhost;Database=snscenter;Uid=sa;Pwd=123;Pooling=true;" />  <add name="PayCenter" providerName="SqlDataProvider" connectionString="Data Source=localhost;Database=PayDB;Uid=sa;Pwd=123;Pooling=true;" />  </connectionStrings>  </configuration> |

（注：使用脚本的优先级Python>C#，禁用Python脚本配置"Python\_Disable"）

接着可以在Script/Action目录下新建脚本文件如下：

ActionIDDefine.cs

|  |
| --- |
| using System;  namespace HelloWorld.Script.CsScript.Action  {  public class ActionIDDefine  {  ///<summary>  ///hello  ///</summary>  public const Int16 Cst\_Action1001 = 1001;  }  } |

BaseAction.cs

|  |
| --- |
| using ZyGames.Framework.Game.Contract;  using ZyGames.Framework.Game.Service;  namespace HelloWorld.Script.CsScript.Action  {  public abstract class BaseAction : BaseStruct  {  protected BaseAction(short aActionId, HttpGet httpGet)  : base(aActionId, httpGet)  {  }  }  } |

Action1001.cs

|  |
| --- |
| using ZyGames.Framework.Game.Service;  namespace HelloWorld.Script.Action  {  public class Action1001 : BaseAction  {    private string content;  public Action1001(HttpGet httpGet)  : base(ActionIDDefine.Cst\_Action1001, httpGet)  {  }  public override bool GetUrlElement()  {  return true;  }  public override void BuildPacket()  {  this.PushIntoStack(content);  }  public override bool TakeAction()  {  content = "Hello World";  return true;  }  }  } |

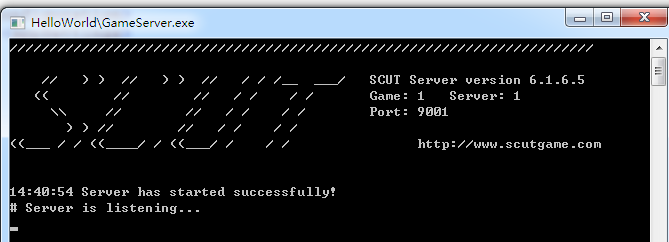
### ****项目启动类****

打开MainClass.cs文件，OnStartAffer方法是项目启动时处理逻辑，OnServiceStop是项目关闭时处理逻辑；如图：

|  |
| --- |
| using ZyGames.Framework.Cache.Generic;  using ZyGames.Framework.Game.Context;  using ZyGames.Framework.Game.Contract;  using ZyGames.Framework.Game.Runtime;  using ZyGames.Framework.Script;  namespace Game.Script  {  public class MainClass : GameSocketHost, IMainScript  {  protected override BaseUser GetUser(int userId)  {  return (BaseUser)CacheFactory.GetPersonalEntity("GameServer.Model.GameUser", userId.ToString(), userId);  }  protected override void OnStartAffer()  {  }  protected override void OnServiceStop()  {  GameEnvironment.Stop();  }  }  } |

### 运行与调试

1）在Action1001类中设置断点，按F5运行程序，结果如下：



（如果不能调试，检查配置项Script\_IsDebug属性是否为True）

2）查看日志，Nlog.config设置项目Log输出日志（这里是"D:\NLog"目录下）；

3）打开协议生成器的“单元测试”功能，设置服务器地址：”192.168.1.104:9001“,协议ID选择1001，点击"socket测试"发送请求；

4）VS2010中会捕获请求定位到断点位置，再按F5继续运行

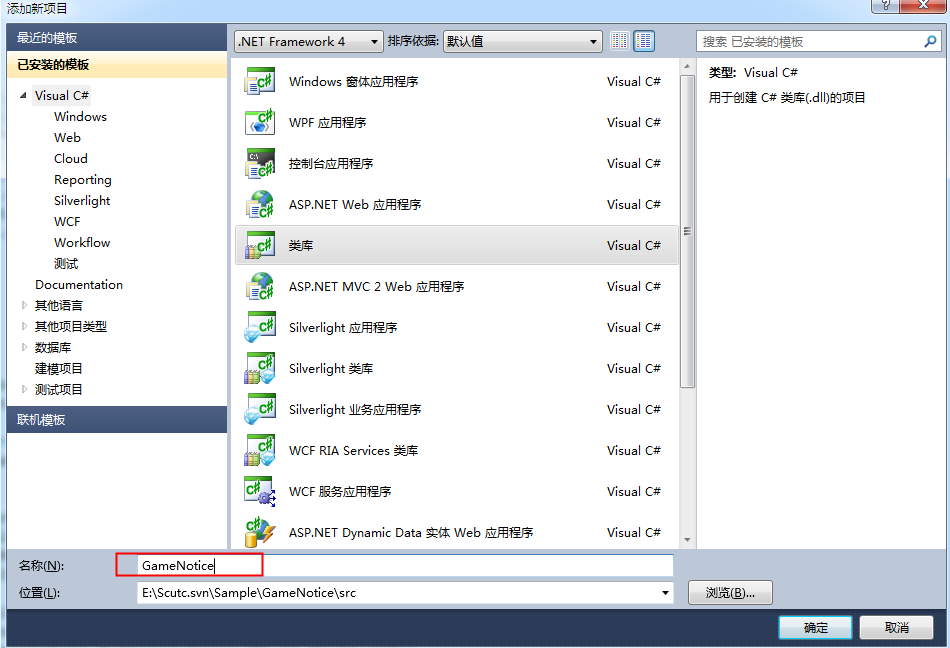
5）协议生成器返回结果（状态值为不0，响应出错，在项目LOG是查看），如下：



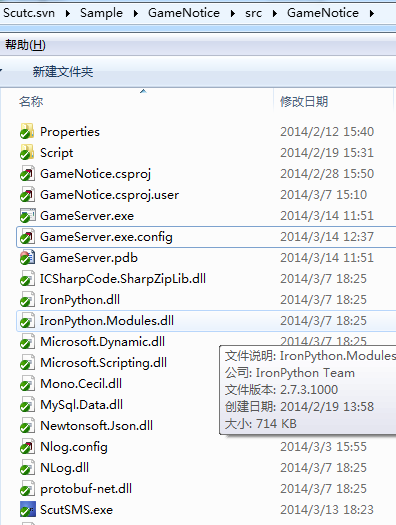
## 游戏公告程序（Console Python版本）

### 创建项目

打开VS2010，新建一个控制台项目命名为GameNotice类库，如图：



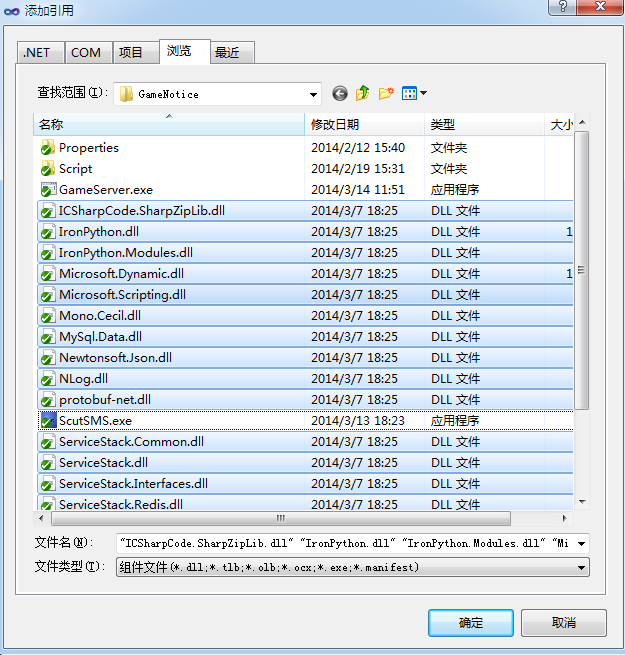
接着把Scut.svn\Release\xx\Console目录(注:xx最新版本)下的所有文件（包括Script目录）Copy到项目”GameNotice“目录下；在GameNotice项目，右击点“属性”，在“调试”栏设置启动操作为“启动外部程序”路径：”HelloWorld\GameServer.exe“和工作目录 "..\..\"；如图：





### Lib引用

添加引用项目目录下的所有dll文件；



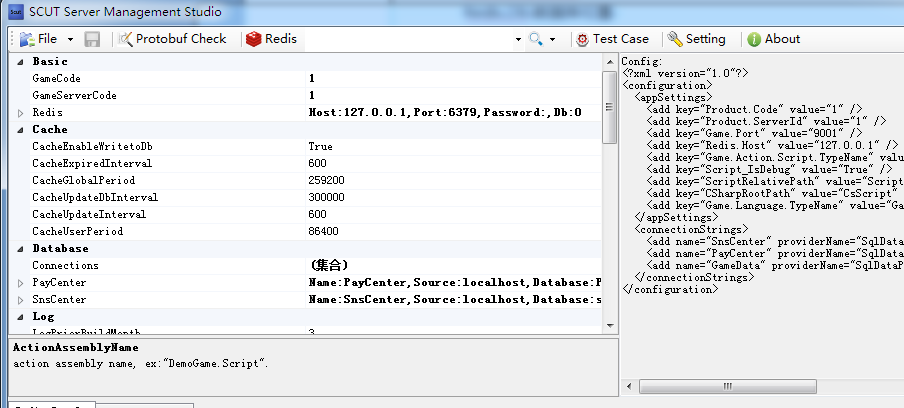
### 添加项目协议

打开协议生成器，增加“GameNotice”项目方案，并定义Action2001接口协议；如下：（参考[例子](#_使用说明)）



### 项目Config配置

1）使用ScutSMS工具打开GameServer.exe.config文件配置



|  |
| --- |
| <?xml version="1.0"?>  <configuration>  <appSettings>  <add key="Product.Code" value="1" />  <add key="Product.ServerId" value="1" />  <add key="Game.Port" value="9001" />  <add key="Redis.Host" value="127.0.0.1" />  <add key="Game.Action.Script.TypeName" value="GameServer.CsScript.Action.Action{0}" />  <add key="Script\_IsDebug" value="True" />  <add key="ScriptRelativePath" value="Script" />  <add key="CSharpRootPath" value="CsScript" />  <add key="Game.Language.TypeName" value="GameServer.CsScript.Locale.SimplifiedLanguage" />  </appSettings>  <connectionStrings>  <add name="SnsCenter" providerName="SqlDataProvider" connectionString="Data Source=localhost;Database=snscenter;Uid=sa;Pwd=123;Pooling=true;" />  <add name="PayCenter" providerName="SqlDataProvider" connectionString="Data Source=localhost;Database=PayDB;Uid=sa;Pwd=123;Pooling=true;" />  <add name="GameData" providerName="SqlDataProvider" connectionString="Data Source=localhost;Database=GameData;Uid=sa;Pwd=123;" />  </connectionStrings>  </configuration> |

如果没有GameData数据库，手动建立数据库，并授权登录帐号（game\_user）有创建和修改表，新增、修改、删除记录等权限

### 定义Model实体类

在项目中创建“Model”目录（存放实体表与数据库表的映射）；新建Notice.cs类，如下：

|  |
| --- |
| using System;  using ProtoBuf;  using ZyGames.Framework.Model;  namespace GameNotice.Model  {  /// <summary>  ///  /// </summary>  [Serializable, ProtoContract]  [EntityTable(CacheType.Entity, "GameData")]  public class Notice : ShareEntity  {  /// <summary>  /// </summary>  public Notice()  : base(false)  {  }  /// <summary>  /// </summary>  public Notice(int id)  : this()  {  Id = id;  }    [ProtoMember(1)]  [EntityField(true)]  public int Id { get; private set; }    [ProtoMember(2)]  [EntityField]  public string Title { get; set; }  /// <summary>  ///  /// </summary>  [ProtoMember(3)]  [EntityField]  public string Content { get; set; }  /// <summary>  ///  /// </summary>  [ProtoMember(4)]  [EntityField]  public DateTime CreateDate { get; set; }    }  } |

### 使用Python脚本

在项目中创建“Script/PyScript/Action”目录结构，复制Python类库（Scut:\Release\PythonLib）到项目PyScript中；在Action目录下新建action2001.py文件（从协议生成器中Copy Py脚本代码），修改结果如下：

|  |
| --- |
| import clr, sys  import ReferenceLib  from action import \*  from ZyGames.Framework.Common import \*  from ZyGames.Framework.Cache.Generic import \*  from ZyGames.Framework.Game.Cache import \*  from ZyGames.Framework.Game.Service import \*  from GameNotice.Model import \*  class UrlParam(HttpParam):  def \_\_init\_\_(self):  HttpParam.\_\_init\_\_(self)  self.GameType = 0  self.ServerID = 0  self.PageIndex = 0  self.PageSize = 0  class ActionResult(DataResult):  def \_\_init\_\_(self):  DataResult.\_\_init\_\_(self)  self.PageCount = 0  self.DsItemCollect = None  def getUrlElement(httpGet, parent):  urlParam = UrlParam()  if httpGet.Contains("GameType")\  and httpGet.Contains("ServerID")\  and httpGet.Contains("PageIndex")\  and httpGet.Contains("PageSize"):  urlParam.GameType = httpGet.GetIntValue("GameType")  urlParam.ServerID = httpGet.GetIntValue("ServerID")  urlParam.PageIndex = httpGet.GetIntValue("PageIndex")  urlParam.PageSize = httpGet.GetIntValue("PageSize")  else:  urlParam.Result = False  return urlParam  def takeAction(urlParam, parent):  actionResult = ActionResult()  noticeList = ShareCacheStruct[Notice]().FindAll()  result = MathUtils.GetPaging[Notice](noticeList, urlParam.PageIndex, urlParam.PageSize)  if result:  actionResult.dsItemCollect = result[0]  actionResult.PageCount = result[1]  return actionResult  def buildPacket(writer, urlParam, actionResult):  writer.PushIntoStack(actionResult.PageCount)  writer.PushIntoStack(len(actionResult.dsItemCollect))  for info in actionResult.dsItemCollect:  dsItem = DataStruct()  dsItem.PushIntoStack(info.Title)  dsItem.PushIntoStack(info.Content)  dsItem.PushIntoStack(info.CreateDate.ToString("yyyy-MM-dd HH:mm:ss"))  writer.PushIntoStack(dsItem)  return True |

### 项目启动类

打开MainClass类

|  |
| --- |
| using System;  using GameNotice.Model;  using ZyGames.Framework.Cache.Generic;  using ZyGames.Framework.Game.Context;  using ZyGames.Framework.Game.Contract;  using ZyGames.Framework.Game.Runtime;  using ZyGames.Framework.Script;  namespace Game.Script  {  public class MainClass : GameSocketHost, IMainScript  {  protected override BaseUser GetUser(int userId)  {  return null;  }  protected override void OnStartAffer()  {  ActionFactory.SetActionIgnoreAuthorize(2001);  InitNotice();  }  protected override void OnServiceStop()  {  GameEnvironment.Stop();  }  private void InitNotice()  {  var cacheSet = new ShareCacheStruct<Notice>();  for (int i = 0; i < 5; i++)  {  int id = (int)cacheSet.GetNextNo();  Notice notice = new Notice(id);  notice.Title = "tile" + id;  notice.Content = "Content" + id;  notice.CreateDate = DateTime.Now;  cacheSet.Add(notice);  }  }  }  } |

### 运行与调试

在Action2001.py文件中设置断点，按F5运行程序，结果如下：

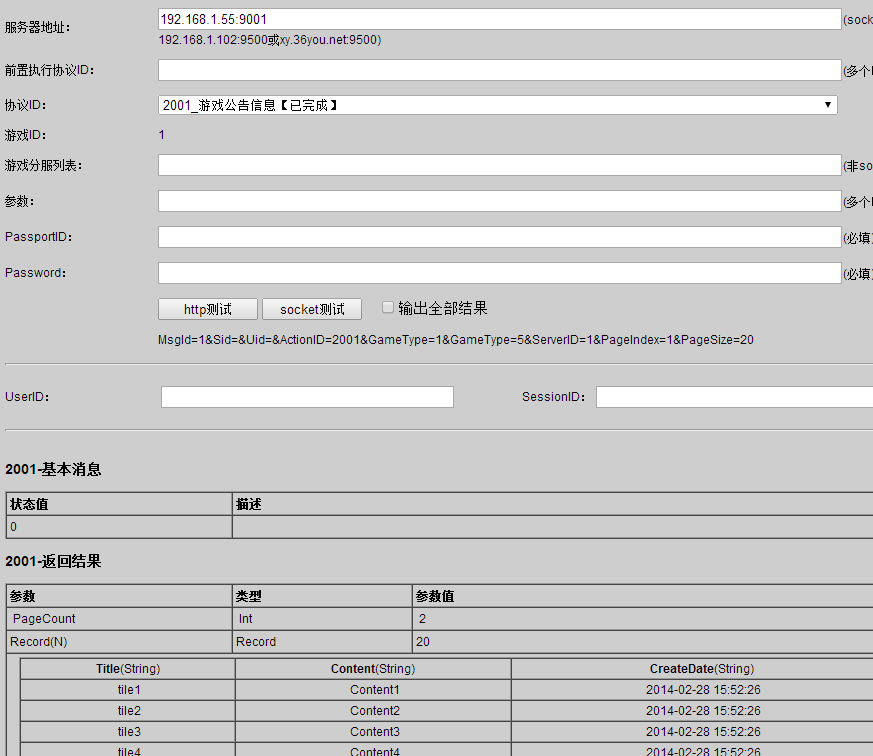


2）查看日志，Nlog.config设置项目Log输出日志（这里是"D:\NLog"目录下）；

3）打开协议生成器的“单元测试”功能，设置服务器地址：”192.168.1.104:9001“,协议ID选择2001，点击"socket测试"发送请求；

4）VS2010中会捕获请求定位到断点位置，再按F5继续运行

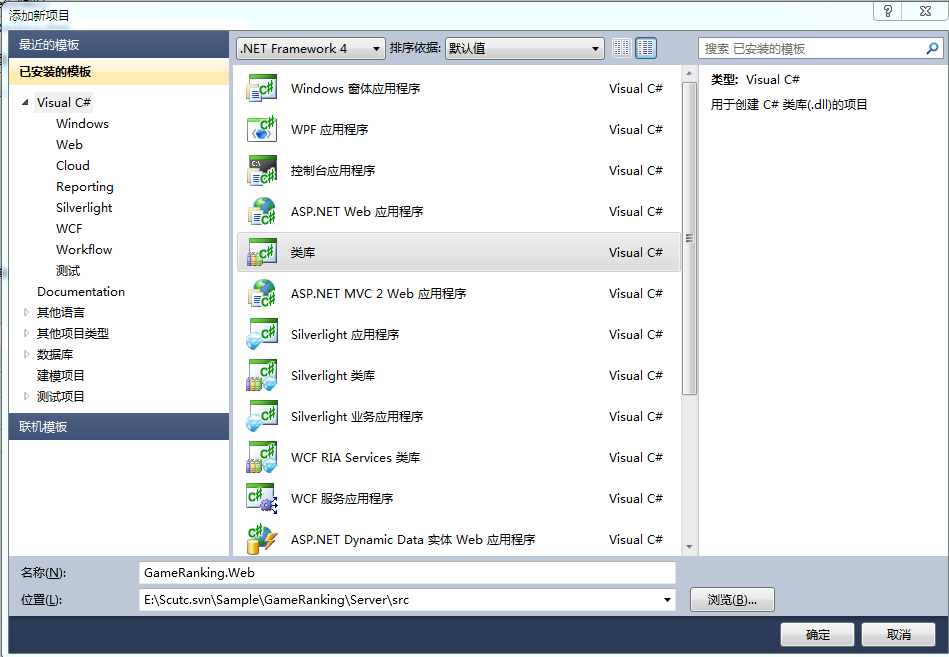
5）协议生成运行结果如下：



## 游戏排行榜程序（Web C#版本）

### 创建项目

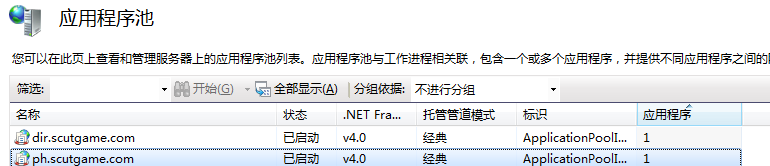
打开VS2010 在菜单上选择 文件 -> 新建 -> 项目；弹出“新建项目”对话窗口，在左则展开“Visual C#”，选择“类库”，输出项目名称（GameRanking.Web）及位置，再点击“确定”；如图：



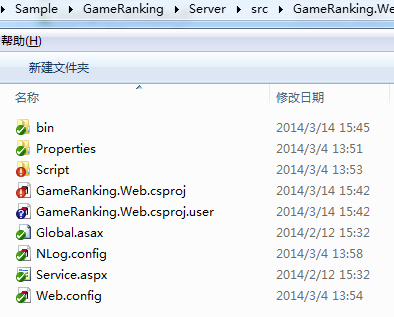
接着把Scut.svn\Release\xx\Web目录(注:xx最新版本)下的所有文件（包括Script目录）Copy到项目”GameRanking“目录下；

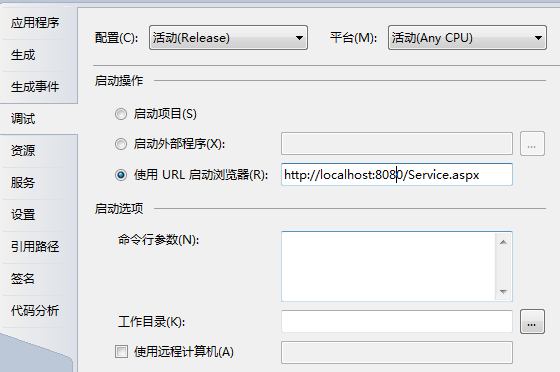
在IIS中配置“ph.scutgame.com”站点，应用程序池设置为NET4.0经典模式，访问站点如：“http://ph.scutgame.com/Service.aspx”

（在C:\Windows\System32\drivers\etc\hosts要增加ph.scutgame.com 127.0.0.1）



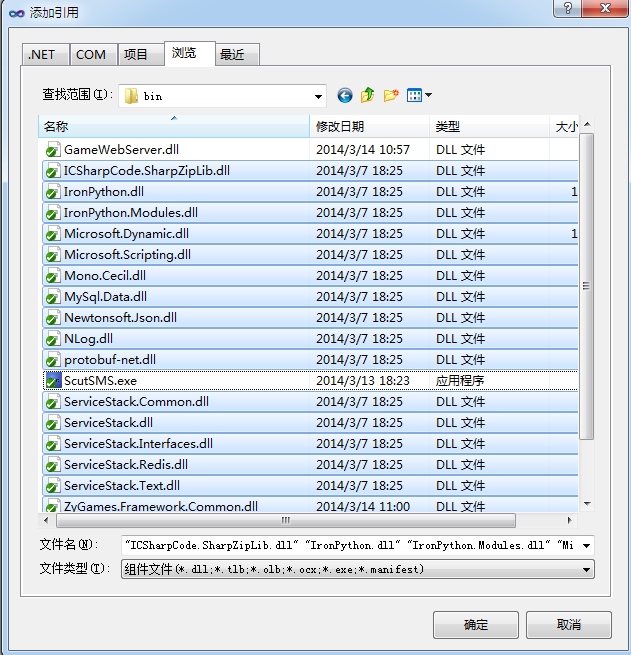
在GameRanking项目，右击点“属性”，在“调试”栏设置启动操作为“使用URL启动浏览器” URL：”http://ph.scutgame.com/Service.aspx“，如图：





### ****Lib引用****

添加引用当前项目目录下的dll文件，如下；



### ****添加项目协议****

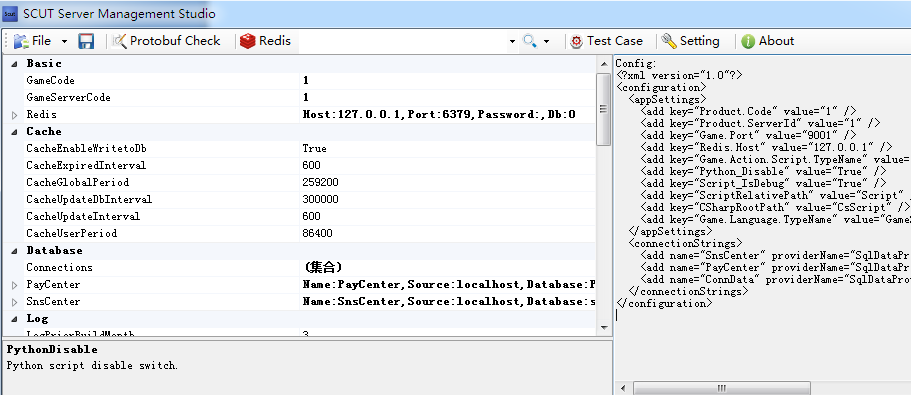
打开协议生成器，定义1000，1001接口协议；如下：（参考[例子](#_使用说明)）





### 项目Config配置

1）使用ScutSMS工具打开Web.config文件



|  |
| --- |
| <?xml version="1.0"?>  <configuration>  <appSettings>  <add key="Product.Code" value="1" />  <add key="Product.ServerId" value="1" />  <add key="Game.Port" value="9001" />  <add key="Redis.Host" value="127.0.0.1" />  <add key="Game.Action.Script.TypeName" value="GameServer.CsScript.Action.Action{0}" />  <add key="Python\_Disable" value="True" />  <add key="Script\_IsDebug" value="True" />  <add key="ScriptRelativePath" value="Script" />  <add key="CSharpRootPath" value="CsScript" />  <add key="Game.Language.TypeName" value="GameServer.CsScript.Locale.SimplifiedLanguage" />  </appSettings>  <connectionStrings>  <add name="SnsCenter" providerName="SqlDataProvider" connectionString="Data Source=localhost;Database=snscenter;Uid=sa;Pwd=123;Pooling=true;" />  <add name="PayCenter" providerName="SqlDataProvider" connectionString="Data Source=localhost;Database=PayDB;Uid=sa;Pwd=123;Pooling=true;" />  <add name="ConnData" providerName="SqlDataProvider" connectionString="Data Source=localhost;Database=PHData;Uid=sa;Pwd=123;" />  </connectionStrings>  </configuration> |

如果没有PHData数据库，手动建立数据库，并授权登录帐号（game\_user）有创建和修改表，新增、修改、删除记录等权限。

### 定义Model实体类

在项目中创建“Model”目录（存放实体表与数据库表的映射）；新建UserRanking.cs类，如下：

|  |
| --- |
| using System;  using ProtoBuf;  using ZyGames.Framework.Model;  namespace GameServer.Model  {  /// <summary>  ///  /// </summary>  [Serializable, ProtoContract]  [EntityTable(CacheType.Entity, "ConnData")]  public class UserRanking : ShareEntity  {  public UserRanking()  : base(false)  {  CreateDate = DateTime.Now;  }  [ProtoMember(1)]  [EntityField(true)]  public int UserID  {  get;  set;  }  [ProtoMember(2)]  [EntityField]  public string UserName  {  get;  set;  }  [ProtoMember(3)]  [EntityField]  public int Score  {  get;  set;  }  [ProtoMember(4)]  [EntityField]  public DateTime CreateDate  {  get;  set;  }  protected override int GetIdentityId()  {  return UserID;  }  }  } |

GuestUser.cs类，如下：

|  |
| --- |
| using System;  using ProtoBuf;  using ZyGames.Framework.Game.Context;  using ZyGames.Framework.Model;  namespace GameServer.Model  {  [Serializable, ProtoContract]  [EntityTable("ConnData")]  public class GameUser : BaseUser  {  [ProtoMember(1)]  [EntityField(true)]  public int UserId { get; set; }  [ProtoMember(2)]  [EntityField]  public String NickName  {  get;  set;  }  [ProtoMember(3)]  [EntityField]  public String PassportId  {  get;  set;  }  [ProtoMember(4)]  [EntityField]  public String RetailId  {  get;  set;  }  public string SId { get; set; }  protected override int GetIdentityId()  {  return UserId;  }  public override string GetSessionId()  {  return SId;  }  public override int GetUserId()  {  return UserId;  }  public override string GetNickName()  {  return NickName;  }  public override string GetPassportId()  {  return PassportId;  }  public override string GetRetailId()  {  return RetailId;  }  public override bool IsFengJinStatus  {  get { return false; }  }  public override DateTime OnlineDate  {  get;  set;  }  }  } |

### ****使用C#脚本****

在项目“GameRanking.Web”下，新建“Script/CsScript/Action”目录存放C#脚本文件，接着可以Action目录下新建Action1000.cs和Action1001.cs脚本文件，从协议生成器中Copy 自动生成的“C#”脚本；

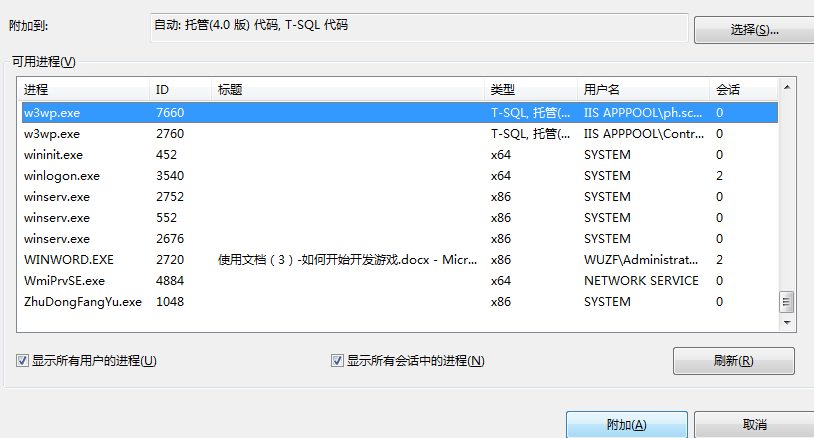
### ****项目启动类****

打开MainClass.cs文件，如：

|  |
| --- |
| using ZyGames.Framework.Game.Contract;  using ZyGames.Framework.Script;  namespace Game.Script  {  public class MainClass : IMainScript  {  public void Start(string[] args)  {  ActionFactory.SetActionIgnoreAuthorize(1000, 1001);  }  public void Stop()  {  }  }  } |

### 运行与调试

1）在Action1001.cs类中设置断点，并设置为启动页，按F5运行程序，弹出IE窗口地址" http://ph.scutgame.com/service.aspx "；在菜单“调试”选择“附加到进程”查找到ph.scutgame.com的W3Wp.exe进程附加；



2）查看日志，Nlog.config设置项目Log输出日志（这里是"D:\NLog"目录下）；

3）打开客户端（Sample\GameRanking\Client）目录，修改lua\testScene.lua文件的url地址，

ScutDataLogic.CNetWriter:setUrl("http://ph.scutgame.com/service.aspx")

4）VS2010中会捕获请求定位到断点位置

5）运行结果如下：

