메뉴얼（3）

How to Start Make Game

# 기초편

## 개발환경

**필수설치프로그램**：

* **Redis**：key-value데이터베이스，2.6.x이상버전
* **데이터베이스**：MSSQL 2005이상버전
* **Web서버**： iis 7.0이상버전
* **.NET런타임**：.Net Framework 4.5.1이상버전

**개발도구**

1. 2013이상 visual studio 사용권장
2. Python 스크립트 쓸려면 IronPython 2.6.1 RC1 for .Net 4.0RC플러그인 사용

## 패킷생성기

패킷생성도구는 개발자가 개발단계에있을 때 빠르게 클라와 서버간 주고받는 프로토콜 코드를 빠르게 제공한다. (생성환경운행시 다시는 필요하지 않다.(???) )。

통신프로토콜관련 오류를 수정하고 양쪽의연결조정시간을 줄여서 개발효율을 높여준다 ( 주: 개발환경운행시, 두번다시 필요하지 않다) , 또한 임의의 클라이언트가 http 나 socket 요청을 했을 때 테스트 서버action 프로토콜, 서버는 Console 을사용하여 socket 요청을하고, 웹을 이용해서 http 요청을 한다.

**데이터베이스 초기화**

일괄처처리방식 채용( 일괄처리방식으로 데이터베이스를 초기화한다.)

내용에 불만있으면 아래를 수정해서 써라

|  |
| --- |
| Set dbServer=. 서버의주소 mysql 은 localhost 라고 써야함 FAQ 에있음  set dbAcount=sa 디비관리계정  set dbPass=123 로그인암호  set gameuser=game\_user 게임서버가 사용할 계정  set gamepass=123 게임서버가 사용할 비밀번호  set dbpath=%cd%\Data 데이터베이스 저장위치 |

**웹사이트 설정**

1. 제어판->관리도구-> Internet 정보서비스관리기
2. 웹사이트에 contract.scutgame.com추가
3. 웹사이트 로컬디렉토리 경로를 ContractTools/Relase 로 변경
4. 사용어플리케이션풀을 프렘웍4.0과 클래식으로 설정
5. host 파일에 127.0.0.1 contract.scutgame.com 추가
6. web.config 파일을 열어서 디비연결문자열수정 (mysql 을 사용한다면 providerName 을 수정한다)

|  |
| --- |
| <?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> |

### 사용설명

파라메터유형추가: 요청파라메터와 응답파라메터

요청파라메터: 클라이언트가 제공하는 파라메터 정의, URL 파라메터표준. 형태: name=Jim&age=1;

응답파라메터: 게임서버가 클라이언트에게 응답하는 데이터스트림(2진수), 내려보내는 데이터형은 short,int,string,byte 기본형과 record, end 는 순환체의 시작과 끝을 나타내는 플래그이다.

예> 응답으로 내려보내는 학생 정보.

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

Id Name Age

1 Jim 20

2 Coco 18

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

격식정의

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

Record

Id Int

Name String

Age Age

End

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

## 분배센터Site（DirCenter）

DirCenter 는 분배서버이다., 게임의 정보와 게임서버의 정보를 관리하고 서버의접근경로를 제공한다:

1. Service.aspx 폐이지에 지정게임의 서버(군)정보를 클라이언트에 제공하고, 사용자는 서버를 골라진입하면 상응하는 게임서버에 방문한다. (Url 주소를 얻고, Http나 socket 방식으로 방문), 내용형식은 2진 바이트 스트림이다.
2. OfficialService.aspx 페이지는 공식게임과 서버정보를 제공하고, 랭킹이있으면 내용은 json 격식이다.
3. DirService.asmx 는 추가,변경,삭제와 같은 조작을 뒷단에 제공한다. 일반적으로 IIS 로 IP 방문권한을 설정해줄 필요가있다.

**사이트 설정**

1. 컨트롤패널-> 관리도구->인터넷서비스관리
2. 웹사이트우클릭 추가 dir.scutgame.com
3. Scut:\Source\DirCenter\release로실제디렉토리설정
4. 어플리케이션풀 프렘워크4.0과 클래식으로 설정
5. 호스트화일에 127.0.0.1 dir.scutgame.com 추가
6. web.config 파일을열어서 연결문 변경, Mysql 을 사용한다면 providerName 을 MySqlProvider 로 변경한다.

|  |
| --- |
| <?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> |

## Lua디버깅

Decoda 툴을 사용해서 Lua 코드를 디버깅한다, 먼저 Decoda를 연다. Project->setting 에서 아래처럼 설정한다.



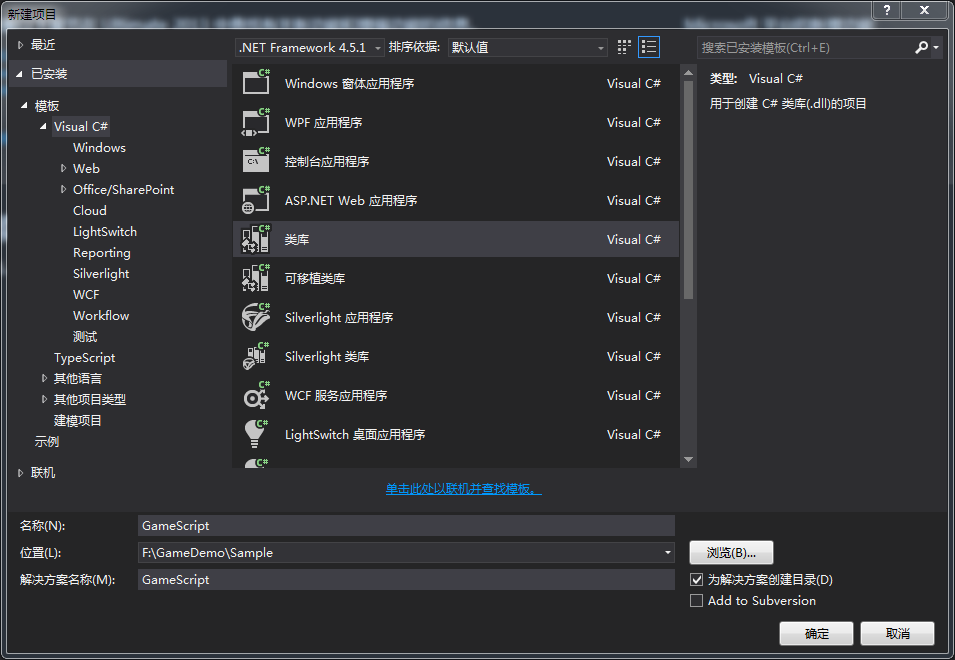
이어서 Debug->Start Debugging 을 선택해서 디버깅을 시작한다(아님F5); 만약 수평이면 Command 에 460 960 을 설정한다



## 콘솔프로젝트 게임 빌드

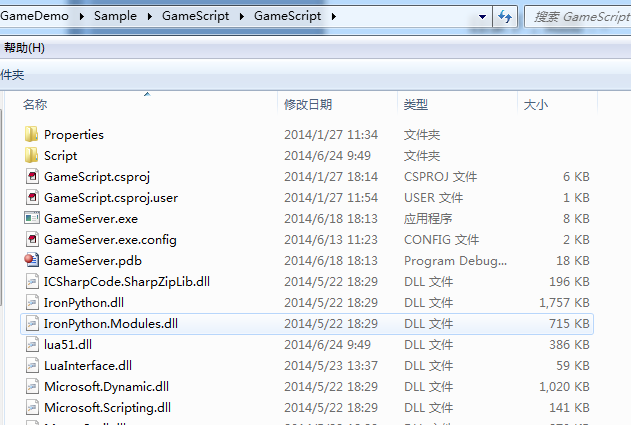
### 새프로젝트

VS2013 을 열고 클래스라이브러리 프로젝트를 새로 만든다. 그리고 게임이름을 넣는다. (예: GameScript)

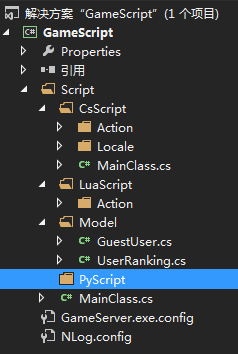


### 라이브러리 파일 복사

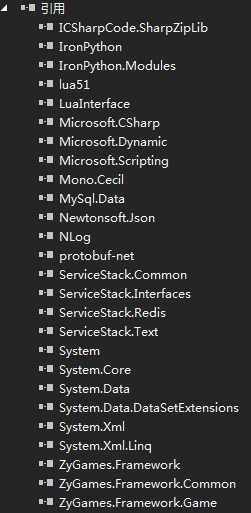
Scut.svn\Release\xx\Console 디렉토리(주:xx 최신버젼, 오피셜페이지나 GitHub 에서다운로드받을수있음, 32비트시스템은 console(x86) 디렉토리)복사해온다.



GameScript 프로젝트모든 파일 보기를 누른다， GameScript 프로젝트에서 빨간색 저버튼을눌러서 모든문서가 보이게 한후 GamServer.exe.config 와 Nlog.config 파일과 스크립트 폴더를 프로젝트에 포함한다. 아래그림처럼



응용레퍼런스를 추가한다. GameScript 디렉토리 및에 모든 Dll 파일을 전부 추가한다. 아래그림처럼

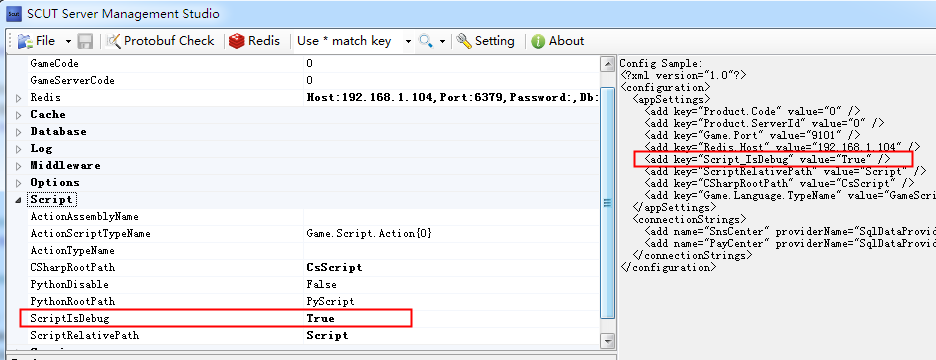


### 프로젝트 설정

GameScript 프로젝트 우클릭해서 속성을 선택, Debug 항목을 열어서 외부실행프로그램을 GameServer.exe 로 설정한다. 워킹디렉토리는 GameServer.exe파일의 위치로 설정한다. 아래그림처럼



이어서 Config 설정을 변경한다. 프로젝트의 ScutSms.exe 프로그램을 연다. ScriptIsDebug 속성을 True 로 설정하고 저장한다. 아래그림처럼

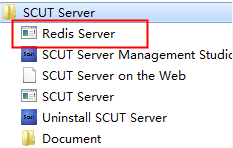


### 디버깅과 운영

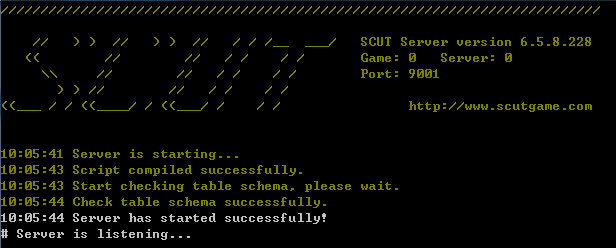
MainClass 클래스의 OnStartAfter 메서드중 BreakPoint 를 설정한다. F5 를 눌러 디버깅을 시작한다. 만약 “the redis server is not started” 오류가 표시되면. GameServer.exe.config 설정화일에 Redis 서버주소가 ping 이 통하는지 확인하고 redis 서버가 가동중인지 확인한다.



Scut 설치본에 RedisServer 를 메뉴에서 찾을수 있다. 32비트 시스템은 사이트에서 상응하는 x86 설치본을 다운받는다.



启动成功结果如下： 기동후 결과는 아래와 같을것이다.



## 웹프로젝트서버 빌드

### 새프로젝트만들기



vs2013 을 열고 ASP.NET 웹응용프로그램 프로젝트를 새로생성한다. 게임의 이름(예:GameWebScript) 를 입력하고 생성확인을 누른다.

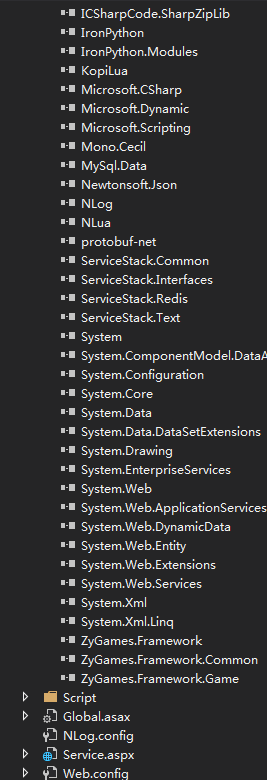
### Scut라이브러리화일복사

웹프로젝트를 설정할 때, Scut 은 이미 컴파일완료된 웹 프로그램껍데기를 제공한다. 새로생성한 클래스라이브러리 프로젝트는 컴파일로 문제점을 찾을수 있게한다. Scut 은 스스로 동적 스크립트 컴파일 프로그램이다. VS 프로젝트가 컴파일한 DLL 과 무관하다.

Scut.svn\Release\xx버전\Lib와Web을 파일탐색기로 열고, VisualStudio 의 GameWebScript 우클릭을 눌러서 파일탐색기를 연다 ，모든 파일을 GameWebScript디렉토리로 복사한다.

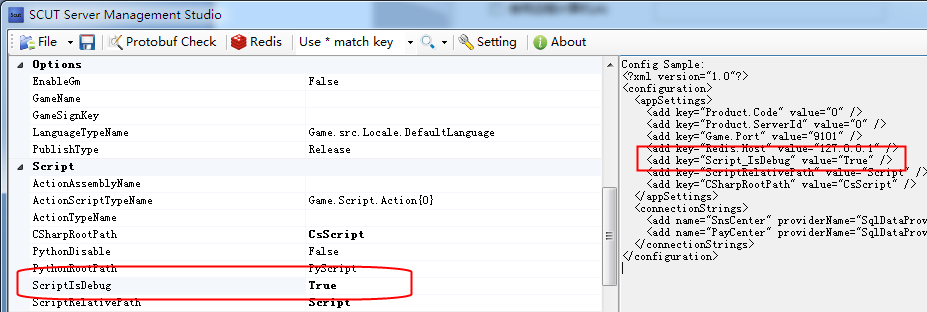
GameWebScript프로젝트에서 버튼을 눌러 모든 파일을 표시하도록 하고，Global，Service，Web.config그리고NLog.config문서와 Script디렉토리를 프로젝트에 포함한다； 모든 dll 을 프로젝트 참조에 추가한다.

아래그림처럼각파일들을 프로젝트에 추가해준다.



### 프로젝트 설정

Config 설정을 변경한다. ScutSms 프로그램을 열어 ScriptIsDebug 속성을 True 로 설정하고 보존한다. 아래그림처럼



### 디버깅과 운영

MainClass 클래스의 Start 메서드에 BreakPoint 를 설정한다

F5 를눌러서 디버깅을 시장한다

### 发布 (퍼블리싱)

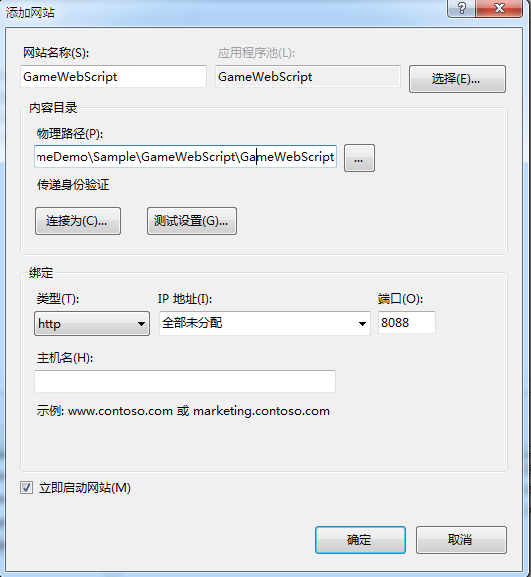
**右击项目，选择“发布”，以“文件系统”方式发布；**

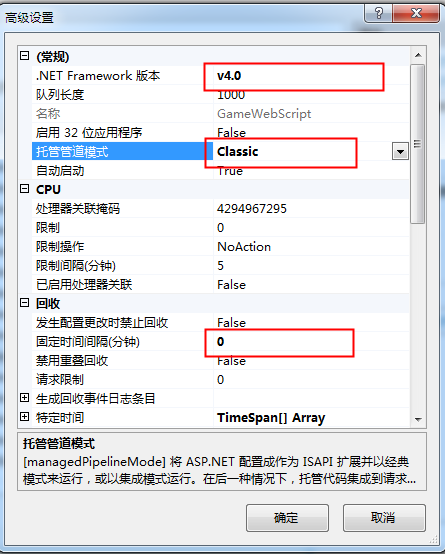
**프로젝트우클릭해서 퍼블리시를 선택하고 파일시스템 방식으로 퍼블리싱한다.**

**IIS建立站点 (IIS 사이트 설정)**

打开IIS管理器，建立“GameWebScript”站点，指向到VS发布的文件目录下，并设置应用程序池为4.0，如图：

IIS 관리기를 열고 GameWebScript 웹사이트를 생성한다. 실제디렉토리경로를 VS 가 퍼블리싱한 파일디렉토리로 설정한다. 어플리케이션프레임워크는 4.0으로 설정한다





## 自定义通讯结构（Unity3D接入）(커스텀 통신 구조)

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：통신 프로토콜은 커스텀정의가 가능하다. 바이트 스트림으로 전송하기에 데이터의 입출력 격식을 구현하면된다. 아래는 protobuf 를 사용해 오브젝트를 바이트로 시리얼하여 전송하는 예제이다.

바이트스트림의 규격은 아래와 같다.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 第一段（4位长度）  제일단 | 第二段  제2단 | 第三段  제삼단 |
| 字节流段 | Byte[L] | Byte[N] | Byte[M] |
| 说明 | headLength  헤드길이 | headBytes  헤드바이트 | 내용바이트 |

L代表头部信息字节数组长度的字节数据（int占4位） L 은 헤더의 내용의 크기에대한 크기이다.

N代表头部信息字节数组长度 N 은 헤더의 정보의 해더내용의 길이에대한 길이이다.

M代表内容序列化后的字节数组长度 내용의 시리얼화된 바이트어레이의 크기이다.

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

아래 구조는 오직 참고자료(6.3.7.5 버전에제공된기능), GitHub 상의 디렉토리 Sample\GameRanking\Server\src\GameRanking.HostU3D4\_5 이다.

### 定义通讯结构包模型 통신구조모형 정의

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：

GameRanking.Pack 클래스라이브러리를새로 생성한다. 임의프로토콜구조 Model 를 정의 한다. 요청패킷의헤더부 MessagePack 클래스로 정의하고 상응하는 응답패킷을 ResponsePack 클래스로 정의한다.

그리고 GameServer 모든 프로젝트 설정파일의 ScriptAsmReference 속성을 Game.Pack.dll 로 설정한다. MessagePack 클래스의 구현은아래와같다.

|  |
| --- |
| [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： ResponsePack 구현은 아래와같다,

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

业务协议结构 워킹프로토콜 구조

|  |
| --- |
| [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） Game.Pack.dll 클래스라이브러리를 Unity 클라이언트에 제공한다.

GameServer.exe.config文件如下配置： GameServer.exe.config 파일을아래처럼 설정한다.

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

### 实现IActionDispatcher分发器接口

IactionDispatcher 구현

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

Action Dispatcher 관리클래스를 구현한다.,바이트데이터를 디코딩하는 TryDecodePackage 메서드를 구현한다. 워킹인터페이스는 아래와 같다.

|  |
| --- |
| 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 CustomActionDispatcher 사용시작

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

### 定义Action协议接口 Action Protocol 인터페이스 정의

定义BaseStruct子类，实现父亲GetUrlElement、TakeAction、WriteResponse方法，如排行榜列表1001接口下： BaseStruct 의 서브클래스를 정의한다. 부모클래스의 GetUrlElement 와 TakeAction 과 WriteResponse 메서드를 구현한다. 만약 1001 번 랭킹 인터페이스는 아래와 같다

|  |
| --- |
| 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 모델 엔티티클래스 설계

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； 게임의 실제데이터설계는 주요하게 네개의 종류로 나뉜다. 유저의 프라이빗데이터. 유저의 공유데이터, 설정숫자데이터와 로그데이터이다.

### 玩家私有数据 유저프라이빗데이터

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）

CacheType.Dictionary 클래스 파라메터를설정한다. : 프라이빗데이터와 모든 사용자가 접근할수있는데이터. 엔티티클래스는 BaseEntity 클래스를 상속받아야한다. 유저의 퀘스트 실례는 아래와같다. (주: ConnData 는 Config 설정의 데이터베이스 연결 이름이다)

|  |
| --- |
| <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 (유저공용데이터)

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：

CacheType.Entity 형식을 설정하고. 엔티티를 global하게 읽고쓸수있습니다. 엔티티는 ShareEntity 클래스를 상속받아야한다. Ranking 실례는 아래와같습니다

|  |
| --- |
| 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 (숫자데이터설정 (csv))

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：

AccessLevel.ReadOnly를 설정합니다, 엔티티는 오직 읽을수만있습니다 엔티티클래스는 ShareEntity 클래스를 상속합니다. 지도정보실례가 아래와같습니다.

|  |
| --- |
| 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 (로그데이터)

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：

AccessLevel.WriteOnly 형식을 설정, 엔티티는 LogEntity 클래스를 상속해야한다. 유저의 로그인로그 실례는 아래와같다

|  |
| --- |
| 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 조합형구조

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；

모든클래스는 EntityChangeEvent 클래스를 상속해야한다. 동시에 Serializable 과 ProtoContract 확장속성을 표시해야한다. 멤버와 속성 속성은 ProtoMember 확장속성이 표시되있어야한다.

**EntityChangeEvent子类结构 EntityChangeEvent 서브클래스구조**

**父亲与子类是一对一的关系 부모와 자식은 일대일 관계이다.**

|  |
| --- |
| 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 CacheList 링크드리스트 구조**

**父亲与子类是一对多的关系 부모와 자식은 일대 다 관계**

|  |
| --- |
| 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 CacheDictionary (사전구조)**

**父亲与子类是一对多的关系 부모와 자식은 일대다 관계**

|  |
| --- |
| 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 사용 CacheStruct 캐쉬

CacheStruct缓存分为：ShareCacheStruct和PersonalCacheStruct类；

CacheStruct 캐쉬는나뉜다: ShareCacheStruct 와 PersonalCacheStruct 클래스로

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；

ShareCacheStruct 클래스: 글로벌 캐쉬구조 형식, Redis 메모리나 DB 에서 추가한 모든유저의데이터메모리상체류

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；

PersonalCacheStruct 클래스: 유저프라이빗한구조의 형식, 일반적으로 UserID 를 주키로 레디스에서 DB 로 로딩된 레디스의 데이터들이다.

### 查找缓存实体 (캐쉬에서 엔티티찾기)

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

ShareCacheStrcuts 검색방법은 Find, FindKey, FindAll 메서드( 만약 공백이라면 Redis 나 DB 로부터 자동로딩), 게다가 Forech 메서드로딩안됨, 오직 모든 유저의 데이터캐쉬로부터 검색함

|  |
| --- |
| //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 검색메서드는 TryFind, TryFindKey, TryFindAll, Find, FindKey, FindAll 메서드 포함함 (공백이라면 Redis나 DB로부터 자동로딩), 그리고 FindGlobal, Foreach 메서드로드할수없다 모든유저데이터캐쉬에서 찾을수 있다.

|  |
| --- |
| //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 (갱신 캐쉬엔티티)

|  |
| --- |
| //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 (캐쉬실체삭제)

|  |
| --- |
| //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);  } |

## 使用Lua脚本 (루아스크립트 사용)

Demo例子可以参考“游戏公告程序”데모 예제 게임공지프로그램참고

### 开启Lua脚本支持配置 Lua 기동 지원설정

打开项目里的ScutSMS.exe程序，设置启用Lua脚本配置



或在GameServer.exe.config配置文件中增加

아니면 GameServer.exe.config 를 수정한다

|  |
| --- |
| <appSettings>  <add key="Lua\_Disable" value="False" />  </appSettings> |

### 注册Lua调用.Net的方法 Lua 등록 호출닷넷메서드

打开脚本启动类MainClass.cs文件，在OnStartAffer注册Lua调用的.Net的方法，如下

스크립트기동클래스 MainClass.cs 파일을 연다, OnStartAffer 에 닷넷메서드를 호출하는 루아를 등록한다

|  |
| --- |
| //Lua调用函数代理类  public class LuaFuncProxy  {  private static LuaFuncProxy instance = new LuaFuncProxy();  public static LuaFuncProxy GetIntance()  {  return instance;  }  private LuaFuncProxy()  {  }  /// <summary>  /// 获取Url参数  /// </summary>  /// <param name="actionGetter"></param>  /// <param name="name"></param>  /// <returns></returns>  public string GetActionParam(ActionGetter actionGetter, string name)  {  return actionGetter != null ? actionGetter.GetString(name) : "";  }  /// <summary>  ///获取排行榜数据的方法，Lua不支持泛型  /// </summary>  /// <returns></returns>  public UserRanking[] GetUserRankingList()  {  var cache = new ShareCacheStruct<UserRanking>();  var rankingList = cache.FindAll(false);  rankingList = MathUtils.QuickSort<UserRanking>(rankingList, compareTo);  rankingList = rankingList.GetPaging(PageIndex, PageSize, out PageCount);  return rankingList.ToArray();  }  private int compareTo(UserRanking x, UserRanking y)  {  int result = y.Score - x.Score;  if (result == 0)  {  result = y.UserID - x.UserID;  }  return result;  }  } |

MainClass类注册函数

MainClass 클래스 함수등록

|  |
| --- |
| protected override void OnStartAffer()  {  var luaProxy = LuaFuncProxy.GetIntance();  Type getterType = luaProxy.GetType();  ScriptEngines.LuaRegister("**ReadStringParam**", luaProxy, getterType.GetMethod("GetActionParam"));  } |

### 新建Lua的Action脚本 새로만들기 루아의 액션 스크립트

在LuaScript\Action目录下新建Action1001.lua脚本,调用ScutReaderReadString方法，如下：

LuaScript\Action 디렉토리에 Action1001.lua 스크립트를 새로만든다. ScutReaderReadString 메서드를 호출하는 다음과 같다.

|  |
| --- |
| Action1001 ={}  function Action1001:getUrlElement(httpGet, parent)  local urlParam = {}  urlParam.Result = true  urlParam.PageIndex = **ReadStringParam (httpGet, " PageIndex")**  return urlParam  end  function Action1001:takeAction(urlParam, parent)  local actionResult = {}  actionResult.Result = true  return actionResult  end  function Action1001:buildPacket(writer, urlParam, actionResult)  return true  end |

# 示例篇 (실례편)

## HelloWorld程序（Console C#版本） 헬로월드 콘솔씨샵버젼

Demo位置：Scut.svn\Sample\HelloWorld 데모위치 Scut.svn/Sample/HelloWorld

### 创建项目 프로젝트생성

打开VS2013 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 4.5.1”后，输出项目名称（HelloWorld）及位置，点击“确定”创建项目；

VS2013 을 열어 메뉴상 파일->새로만들기->프로젝트선택한다.

프로젝트새로만들기 대화창이 뜬다, VisualC# 을 고른후 클래스라이브러리를 선택하고 .Net Framework 4.5.1 을 선택하고 프로젝트명(HelloWorld)를 입력하고 확인을 눌러 프로젝트를 생성한다.

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“启动外部程序”和“工作目录”路径，接着增加项目dll引用；（详情参考“[搭建游戏Console项目](#_搭建游戏Console项目)”）；

그리고 Release/xxx/Console 디렉토리(xx는 버전번호) 아래의 모든 파일(Script 디렉토리포함)을 HelloWorld 디렉토리 및에 복사한다. HelloWorld 프로젝트에 우클릭해서 속성을.. 디버깅에서 외부실행프로그램과 워킹디렉토리를 설정한다. 그리고 DLL Reference 를 추가한다 (콘솔프로젝트만들기를 참고한다)

### ****添加项目协议 프로젝트 프로토콜 추가****

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)）

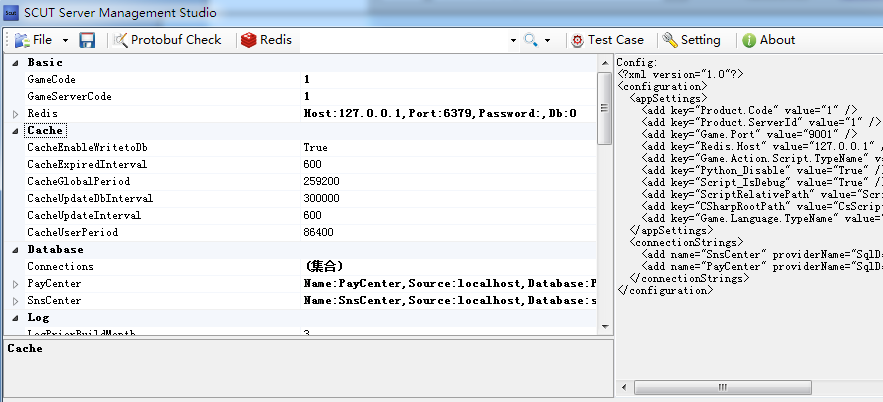
프로토콜생성기를 연다. HelloWorld 프로젝트를 추가한다. Action1001인터페이스프로토콜을 정의한다(예제참고)



### ****使用C#脚本 C# 스크립트 사용****

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：

프로젝트 HelloWorld 아래에 Script/CsScript/Action 디렉토리를 생성한다. 이디렉토리에 C# 파일을 저장한다 그리고 스크립트의 네임스페이스를 설정한다. ScutSMS 툴을 열어서 GameServer.exe.config 를아래와같이 설정한다



|  |
| --- |
| <?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="LuaRootPath" value="LuaScript"/>  <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--Lua-- C#，禁用Python脚本配置"Python\_Disable"为“True”，禁用Lua脚本配置“Lua\_Disable”为True）

주: 스크립트사용우선도는 파이선-루아-씨샵이다, 파이선사용하지 않으려면 Python\_disable 을 True로 Lua를 사용하지 않으려면 Lua\_Disable 을 True 로한다)

接着可以在Script/Action目录下新建脚本文件如下：

이어서 Script/Action 디렉토리에 아래와 같은 스크립트를 새로 생성한다

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 프로젝트 기동형****

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：

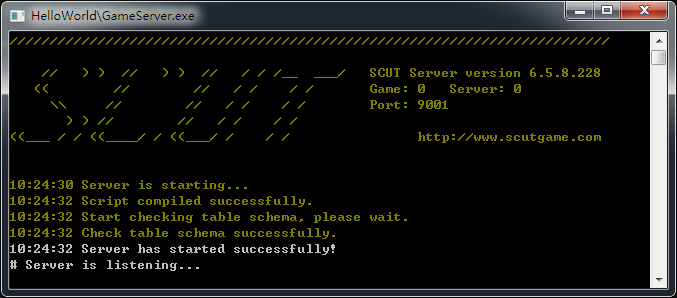
MainClass.cs 파일을 연다, OnStartAfter 메서드는 프로젝트기동후 처리하는로직 OnServiceStop 은 프로젝트끝날 때 처리되는 로직이다.

아래그림처럼.

|  |
| --- |
| 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 (운영과 디버깅)

1）在Action1001类中设置断点，按F5运行程序，结果如下：Action1001 클래스에 BreakPoint 를 추가하고 F5 를 눌러 프로그램을 실행하면 결과는 아래와 같다.



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

만약 디버깅이 안되면 Script\_isDebug속성이 true 로 되있는지 확인하라.

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

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； 프로토콜생성기의 단위테스트기능을 연다, “socket 테스트”를 눌러 request 를 발송한다.

4）VS中会捕获请求定位到断点位置，再按F5继续运行 VS 에서 BreakPoint 에 멈출수있다.,다시 F5 를 눌러서 계속 실행한다.

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： 프로토콜생성기에 결과가 돌아온다( 상태값이 0이아니면 반응에 문제가 발생한것이다. 프로젝트의 로그를 찾아본다)



## 游戏公告程序（Lua和Python版本） 공지프로그램

Demo位置：Scut.svn\Sample\GameNotice 데모위치

### 创建项目 프로젝트생성

打开VS，新建一个控制台项目命名为GameNotice类库，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 “启动外部程序”路径和工作目录，接着增加项目dll引用；（详情参考“[搭建游戏Console项目](#_搭建游戏Console项目)”）；

VS를 연다, 콘솔타입의 프로젝트를 GameNotice 클래스라이브러리로 생성한다. 이어서 Scut.svn/Release/xx/Console 디렉토리의 모든 파일을 프로젝트 GameNotice 디렉토리로 복사한다;

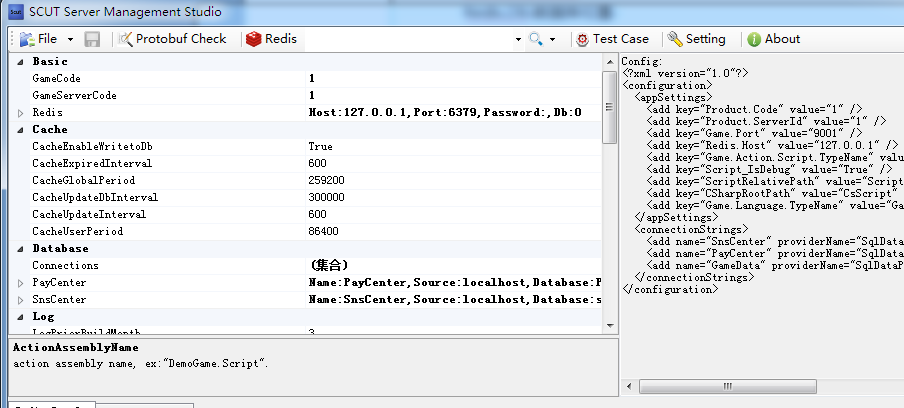
### 添加项目协议 (프로젝트프로토콜 추가)

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)） 프로토콜 생성기연다, GameNotice 프로젝트솔루션을 추가한다. 그리고 Action2001 인터페이스 프로토콜을 아래처럼 정의한다.



### 项目Config配置 프로젝트 Config 설정

1）使用ScutSMS工具打开GameServer.exe.config文件配置 ScutSMS 도구로 GameServer.exe.config 파일을 열어 설정한다.



|  |
| --- |
| <?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="GameData" providerName="SqlDataProvider" connectionString="Data Source=localhost;Database=GameData;Uid=sa;Pwd=123;" />  </connectionStrings>  </configuration> |

2）导入数据库，创建GameData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；打开Sample\GameNotice\db目录.bat文件，修改红色的配置值，双击直接运行bat文件导入数据库

Import 데이터베이스, GameData 데이터베이스를 생성, game\_user 에 로그인권한과 생성 변경 추가 삭제 권한을 부여한다. Sample/GameNotice/db 디렉토리의 ..bat 를 열어서 빨간색으로 되어있는 설정값들을 변경한다.

Bat 파일을 더블클릭해서 데이터베이스 Import 를 진행한다.

|  |
| --- |
| set dbServer=. 数据库服务器 데이터베이스서버  set dbAcount=sa  set dbPass=123 可创建数据库的SA密码 디비생성가능한 SA 비번  set gameuser=game\_user 游戏中使用的数据库帐号 게임이 사용하는 디비아이디  set gamepass=123 游戏中使用的数据库密码 게임이사용하는 디비암호  set dbpath=%cd%\Data 默认是数据库放在当前目录下 기본값은 현재디렉토리이다. |

### 定义Model实体类 모델 엔티티클래스정의

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：

프로젝트에 모델디렉토리를 생성한다. ( 엔티티표와 데이터베이스표가반영될곳); Notice.cs 클래스를 아래처럼 생성한다.

|  |
| --- |
| 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Lua脚本

在项目中创建“Script/LuaScript/Action”目录结构，在Action目录下新建Action2001.lua文件，代码如下：

|  |
| --- |
| Action2001 ={}  function Action2001:getUrlElement(httpGet, parent)  local urlParam = {}  urlParam.Result = true  urlParam.GameType = ReadNumberParam(httpGet, "GameType")  urlParam.ServerID = ReadNumberParam(httpGet, "ServerID")  urlParam.PageIndex = ReadNumberParam(httpGet, "PageIndex")  urlParam.PageSize = ReadNumberParam(httpGet, "PageSize")  return urlParam  end  function Action2001:takeAction(urlParam, parent)  local actionResult = {}  actionResult.PageCount = 0  actionResult.ItemCount = 0  actionResult.dsItemCollect = nil  actionResult.Result = true  local result = GetNoticeCache(urlParam.PageIndex, urlParam.PageSize)  if result == nil then  actionResult.Result = false  return actionResult  end  actionResult.PageCount = result[0]  actionResult.dsItemCollect = result[1]  actionResult.ItemCount = actionResult.dsItemCollect.Count  return actionResult  end  function Action2001:buildPacket(writer, urlParam, actionResult)  PushIntoStack(writer, actionResult.PageCount)  PushIntoStack(writer, actionResult.ItemCount)  local len = actionResult.ItemCount  for i=1,len,1 do  local info = actionResult.dsItemCollect[i-1]  dsItem = CreateDataStruct()  PushIntoStack(dsItem, info.Title)  PushIntoStack(dsItem, info.Content)  PushIntoStack(dsItem, FormatDateString(info.CreateDate, "yyyy-MM-dd HH:mm:ss"))  PushIntoStack(writer, dsItem)  end  return true  end |

注册Lua使用的.net方法，定义ScriptProxy类负责注册方法，代码如下：

|  |
| --- |
| using System;  using System.Collections;  using GameNotice.Model;  using ZyGames.Framework.Cache.Generic;  using ZyGames.Framework.Common;  using ZyGames.Framework.Game.Service;  using ZyGames.Framework.Script;  namespace Game.Script  {  public class ScriptProxy  {  public static void Load(string type, string[] files)  {  try  {  if (".cs".Equals(type))  {  //注册Lua调用C#方法  RegistMethodd();  }  }  catch (Exception ex)  {  }  }  public static void RegistMethodd()  {  ScriptProxy proxy = new ScriptProxy();  ScriptEngines.LuaRegisterObject(proxy);  }  [LuaMethod("ContainsParam")]  public bool ContainsParam(ActionGetter actionGetter, string name)  {  return actionGetter != null && actionGetter.Contains(name);  }  [LuaMethod("ReadStringParam")]  public string ReadStringParam(ActionGetter actionGetter, string name)  {  return actionGetter == null ? null : actionGetter.GetStringValue(name);  }  [LuaMethod("ReadNumberParam")]  public int ReadNumberParam(ActionGetter actionGetter, string name)  {  return actionGetter == null ? -1 : actionGetter.GetIntValue(name);  }  [LuaMethod("PushIntoStack")]  public void PushIntoStack(DataStruct writer, object value)  {  if (value is DataStruct)  {  writer.PushIntoStack((DataStruct)value);  }  else if (value is int)  {  writer.PushIntoStack((int)value);  }  else if (value is double)  {  writer.PushIntoStack(value.ToInt());  }  else if (value is short)  {  writer.PushIntoStack((short)value);  }  else  {  writer.PushIntoStack(value.ToString());  }  }  [LuaMethod("PushLenIntoStack")]  public void PushLenIntoStack(DataStruct writer, object value)  {  var list = value as IList;  writer.PushIntoStack(list == null ? 0 : list.Count);  }  [LuaMethod("CreateDataStruct")]  public DataStruct CreateDataStruct()  {  return new DataStruct();  }  [LuaMethod("FormatDateString")]  public string FormatDateString(DateTime value, string formart)  {  return value.ToString(formart);  }  [LuaMethod("GetNoticeCache")]  public object GetNoticeCache(int pageIndex, int pageSize)  {  var noticeList = new ShareCacheStruct<Notice>().FindAll();  if (pageIndex == 0 && pageSize == 0)  {  return noticeList;  }  int pageCount;  int recordCount;  noticeList = MathUtils.GetPaging<Notice>(noticeList, pageIndex, pageSize, out pageCount, out recordCount);  return new object[] { pageCount, noticeList };  }  }  } |

在MainClass类中调用注册类

|  |
| --- |
| protected override void OnStartAffer()  {  ScriptEngines.OnLoaded += ScriptProxy.Load;  ScriptProxy.RegistMethodd();  } |

### 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 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 |

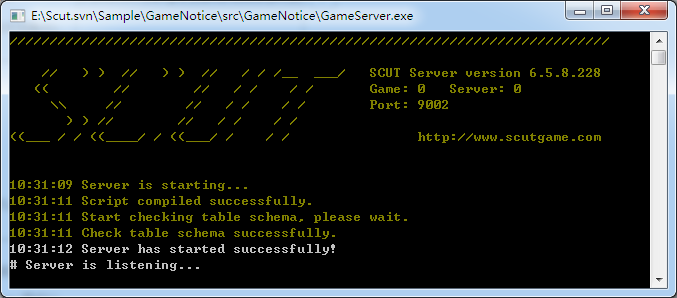
### 项目启动类 프로젝트 기동클래스

打开MainClass类 MainClass 클래스를 연다

|  |
| --- |
| 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 (운영과 테스트)

在Action2001.py文件中设置断点，按F5运行程序，结果如下： Action2001.py 파일중 BreakPoint 를 설정한다. F5 를 눌러 프로그램을 실행한 결과는 아래와같다.



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

로그를 확인하면 Nlog.config 설정 프로젝트 로그 익스포트로그 (여기선 d:\NLog 디렉토리 및)

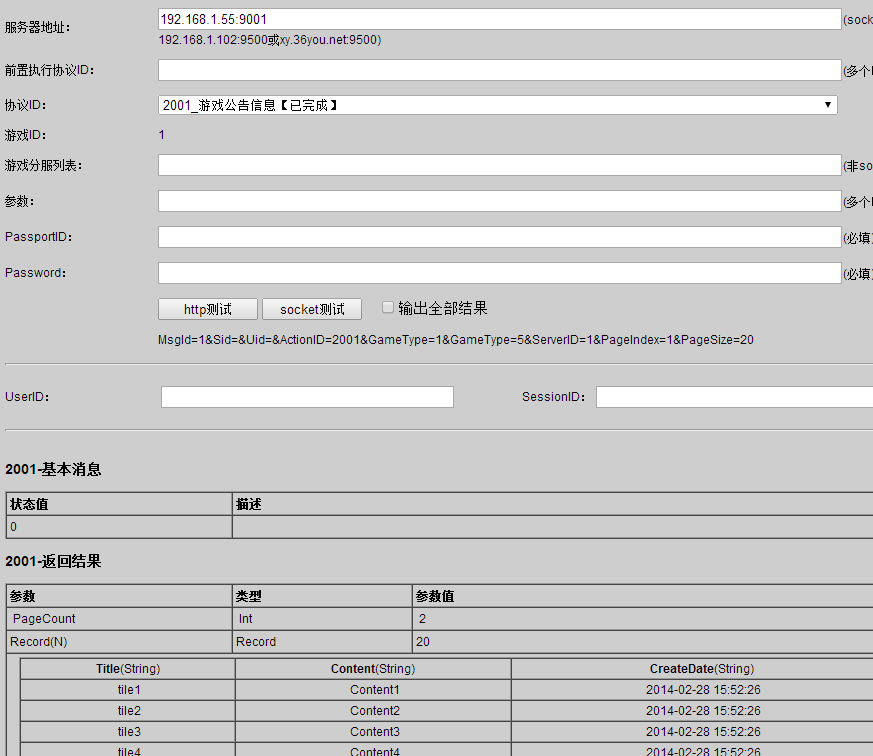
3）打开协议生成器的“单元测试”功能，设置服务器地址：”192.168.1.104:9001“,协议ID选择2001，点击"socket测试"发送请求；

프로토콜 생성기를 열고. 유닛테스트 기능을 연다. 서버의 주소를 192.168.1.104.9001 설정, 프로토콜ID w는 2001번을선택한다. “Socket 테스트” 를눌러서 요청을 발송한다.

4）VS中会捕获请求定位到断点位置，再按F5继续运行

VS 에서 Breakpoint 를잡아낼것이다. F5 를 다시눌러 계속 진행하자

5）协议生成运行结果如下： 프로토콜생성운행결과는 아래와같다.



## 游戏排行榜程序（Web C#版本） 게임랭킹프로그램

Demo位置：Scut.svn\Sample\GameRanking 데모위치

### 创建项目 프로젝트 생성

打开VS 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”；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，接着增加项目dll引用；

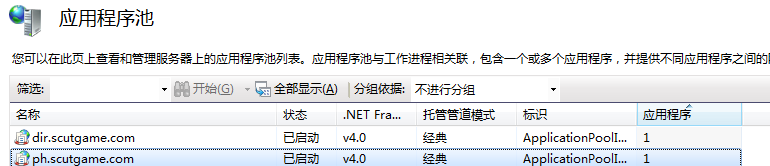
VS 연다, 메뉴판에서 파일->새로만들기->프로젝트 선택; 프로젝트새로만들기 대화창 나타난다. VisualC# 선택, 클래스라이브러리선택. 프로젝트명칭 (GameRanking.Web) 입력, 확인선택. 이어서 Scut.svn/Release/xx/Web 디렉토리의 모든 파일을 프로젝트 GameRanking 디렉토리밑으로 복사한다.

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>

IIS 에서 웹사이트 ph.scutgame.com 을 추가한다. 프렘워크는 NET4.0으로 클래식방식으로 추가한다. 방문주소는 <http://ph.scutgame.com/service.aspx> 로 한다.

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

Host 파일에 엔트리 추가한다



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“；（详情参考“[搭建游戏Web项目](#_搭建游戏Web项目)”）；

GameRanking 프로젝트에 우클릭->속성->디버깅에 시작조작을 사용, 브라우져사용으로 설정URL 을 <http://ph.scutgame.com/Service.aspx> 로 설정한다. 자세한내용은 웹프로젝트 게임만들기를 참고한다

### ****添加项目协议 프로젝트프로토콜 추가****

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)）

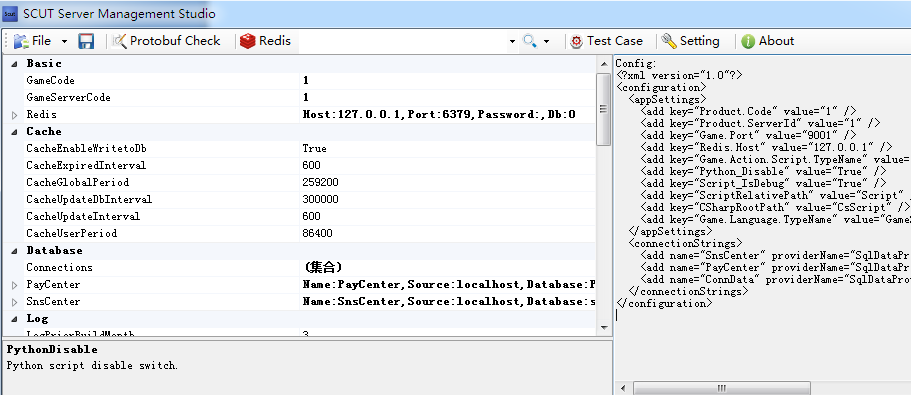
프로토콜생성기열고, 1000,1001 인터페이스프로토콜을 아래처럼 정의한다 아래예제를 참고





### 项目Config配置 (프로젝트Config 설정)

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="ConnData" providerName="SqlDataProvider" connectionString="Data Source=localhost;Database=PHData;Uid=sa;Pwd=123;" />  </connectionStrings>  </configuration> |

2）导入数据库，创建GameData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；打开Sample\GameRanking\Server\db目录.bat文件，修改红色的配置值，双击直接运行bat文件导入数据库

GameData 데이터베이스를 생성하고. Game\_user 에 로그인권한, 표생성권한, 추가 변경,삭제등의 권한을 부여한다. Sample/gameranking/server/db 의 .bat 파일을 연다. 빨간색으로 된 값을 변경한다. Bat 파일을 더블클릭해서 데이터베이스에 Import 한다.

|  |
| --- |
| set dbServer=. 数据库服务器  set dbAcount=sa  set dbPass=123 可创建数据库的SA密码  set gameuser=game\_user 游戏中使用的数据库帐号  set gamepass=123 游戏中使用的数据库密码  set dbpath=%cd%\Data 默认是数据库放在当前目录下 |

### 定义Model实体类 모델 엔티티형 정의

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：

프로젝트에서 Model 디렉토리를생성( 엔티티표와 디비표 매핑을 넣을곳) USerRanking.cs 클래스를 생성한다 아래처럼

|  |
| --- |
| 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 C#스크립트사용****

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；

GameRanking.Web 프로젝트에서 Script/CsScript/Action 디렉토리에 C# 스크립트 파일을 넣는다. 이어서 Action 디렉토리에 Action1000.cs 와 Action 1001.cs 스크립트파일을넣는다. 프로토콜생성기에서 자동생성된 C# 스크립트 파일을 복사해서~~

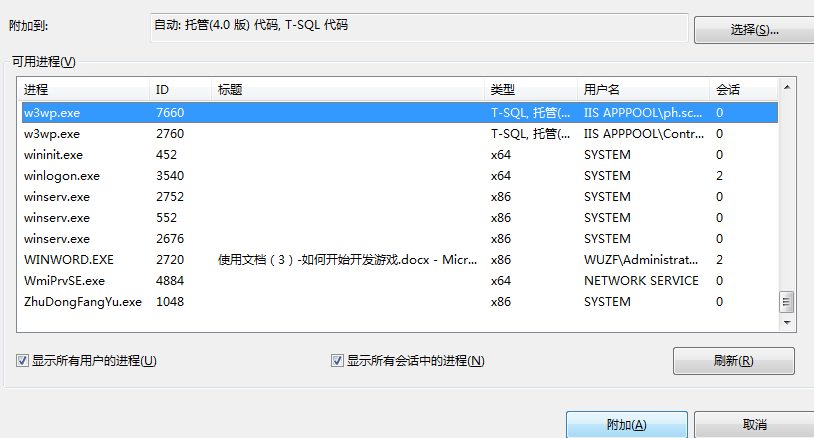
### ****项目启动类 프로젝트 기동클래스****

打开MainClass.cs文件，如： MainClass 클래스를 연다.

|  |
| --- |
| 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 (운영과 디버깅)

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； action1001.cs 에서 클래스내에 브렉포인트를 건다. 그리고 페이지 기동으로 설정하고 F5 를 눌러 프로젝트를 실행한다. IE 윈도우주소가 튀어나온다 <http://ph.scutgame.com/service.aspx>; 메뉴에서 디버깅을 선택하고, attach to process 를 선택하고 ph.scutgame.com 의 W3Wp.exe 에 Attach 한다.



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

3）打开客户端（Sample\GameRanking\Client）目录，修改lua\testScene.lua文件的url地址，클라이언트디렉토리를 열어서 lua/testScene.lua 파일에서 url 주소를 변경한다.

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

4）VS中会捕获请求定位到断点位置 VS 에서 Break Point 가 걸릴것이다/

5）运行结果如下： 아래처럼 돌아간다.

