使用文档（3）

如何开始开发游戏

# 开发环境

**需要安装的软件**

1. VS2010开发工具
2. IronPython2.6.1 RC1 for .NET 4.0 RC插件

# 协议生成器站点

**站点部署**

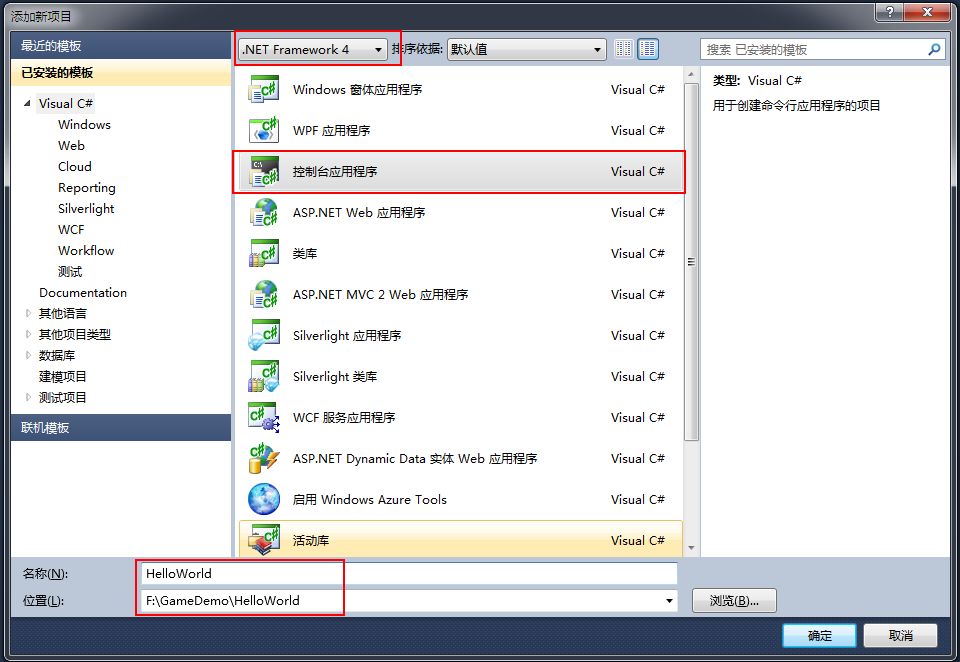
1. 打开控制面板->管理工具-> Internet 信息服务(IIS)管理器
2. 在网站结点右击，添加站点contract.scutgame.com
3. 将站点路径指向到Release\WebSite\ContractTools目录
4. 设置应用程序池“.NET Framework v2.0”和“集成”模式
5. 设置Host文件，添加配置“127.0.0.1 contract.scutgame.com”
6. 修改数据库连接，打开Web.config文件

|  |
| --- |
| <appSettings>  <!-- 连接字符串是否加密 -->  <add key="ConStringEncrypt" value="false"/>  <add key="ConnectionString" value="server=.;database=ContractDB;uid=sa;pwd=123"/>  </appSettings> |

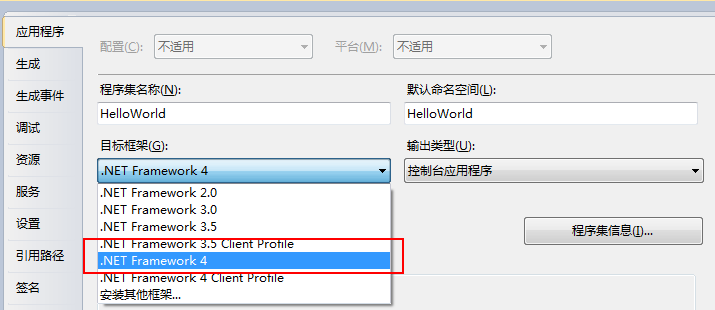
# HelloWorld程序

## 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“控制台应用程序”，选择“.Net Framework 4.0”后，输出项目名称（HelloWorld）及位置，再点击“确定”；如图：



接着在HelloWorld项目，右击点“属性”，在应用程序项设置目标框架为“.Net Framework 4”



## ****Lib引用****

|  |  |
| --- | --- |
| ****项目**** | **引用路径** |
| **HostServer** | **Lib\NLog.dll**  **Lib\protobuf-net.dll**  **Lib\** **ServiceStack.Common.dll**  **Lib\ServiceStack.Redis.dll**  **Lib\ZyGames.Framework.Common.dll**  **Lib\ZyGames.Framework.dll**  **Lib\ZyGames.Framework.Game.dll**  **Lib\Microsoft.Dynamic.dll**  **Lib\Microsoft.Scripting.dll** |

**设置Microsoft.Dynamic.dll和Microsoft.Scripting.dll属性“复制本地”为true**

## ****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：



## ****使用C#脚本****

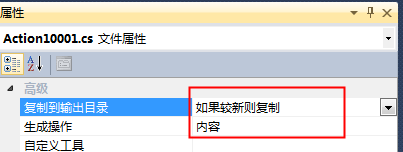
在项目“HelloWorld”下，新建“Script/Action”目录存放C#脚本文件；并设置脚本的解析命名空间，在App.config配置文件中appSettings结点下增加如下

|  |
| --- |
| <appSettings>  <!--配置  Game.Port:监听端口  Product.Code:产品代码  Product.ServerId:产品游服代码  Redis.Host:Redis主机地址  Redis.Port:Redis主机端口  Redis.Password:密码  Redis.Db:数据库位置  Game.Action.Script.TypeName： 脚本定义的命名空间类型表达式  -->  <add key="Game.Port" value="9001" />  <add key="Product.Code" value="1"/>  <add key="Product.ServerId" value="1"/>  <add key="Redis.Host" value="192.168.1.104" />  <add key="Redis.Db" value="0" />  <add key="Game.Action.Script.TypeName" value="HelloWorld.Script.Action.Action{0}" />  </appSettings> |

接着可以在Script/Action目录下新建Action1001.cs脚本文件,内容如下：

|  |
| --- |
| using ZyGames.Framework.Game.Service;  namespace HelloWorld.Script.Action  {  public class Action1001 : BaseStruct  {  public Action1001(HttpGet httpGet)  : base(1001, httpGet)  {  }  public override bool GetUrlElement()  {  return true;  }  public override void BuildPacket()  {  this.PushIntoStack("Hello World");  }  public override bool TakeAction()  {  return true;  }  }  } |

右击Action1001.cs文件设置属性“生成操作”：“编译”改为“内容”，并设置“如果较新复制”；如图：



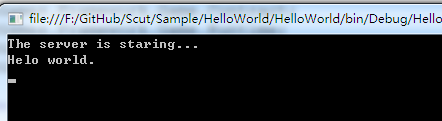
## ****项目启动类****

打开Program文件，并将Program类继承至GameSocketHost基类，实现父类方法；接着在Main方法中增加启动代码，如图：

|  |
| --- |
| class Program : GameSocketHost  {  static void Main(string[] args)  {  new Program().Start();  }    protected override void OnConnectCompleted(object sender, ConnectionEventArgs e)  {  Console.WriteLine("Client:{0} connect to server.", e.Socket.RemoteEndPoint);  }  protected override void OnRequested(HttpGet httpGet, IGameResponse response)  {  Console.WriteLine("Request data:{0}", httpGet.ParamString);  ActionFactory.Request(httpGet, response, null);  }  protected override void OnStartAffer()  {  try  {  //时间间隔更新库  int cacheInterval = 600;  GameEnvironment.Start(cacheInterval, () =>{  return true;  });  Console.WriteLine("The server is staring...");  Console.WriteLine("Helo world.");  }  catch (Exception ex)  {  Console.WriteLine("error:{0}", ex.Message);  }  }  protected override void OnServiceStop()  {  GameEnvironment.Stop();  Console.WriteLine("The server is stoped");  }  } |

## 运行与调试

1）在Action1001类中设置断点，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：

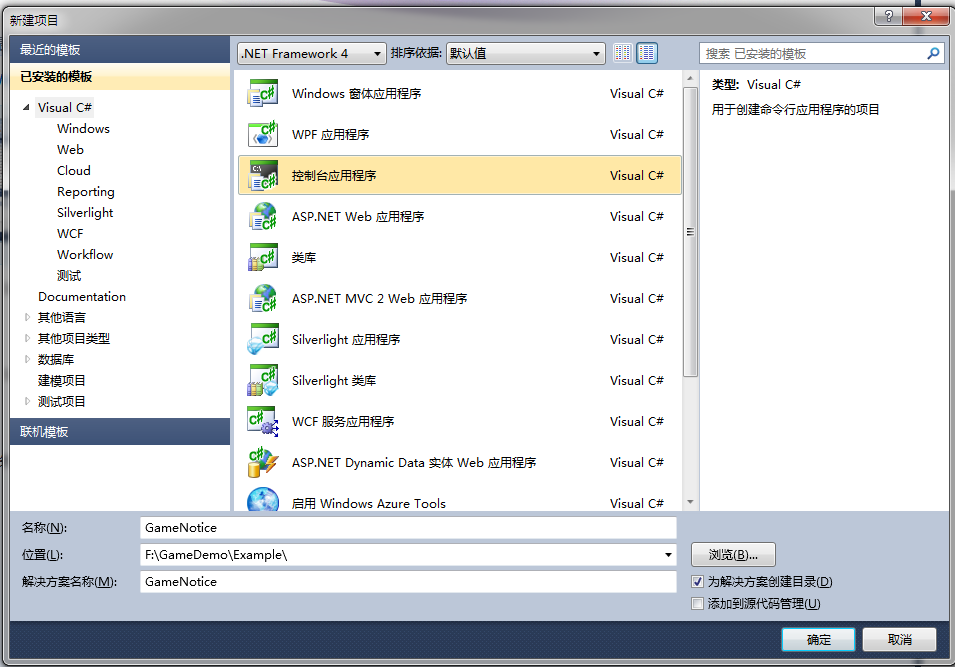


注：Hello项目未使用数据库，会提示“CheckTableSchema Temp\_EntityHistory error”异常可忽略，此表是Redis内存备份数据；

# 游戏公告程序

## 创建项目

打开VS2010，新建一个控制台项目命名为GameNotice和GameNotice.Model类库，并设置项目属性的目标框架为Net Framework 4.0；如图：



## Lib引用

|  |  |
| --- | --- |
| ****项目**** | **引用路径** |
| **GameNotice.Model** | **Lib\protobuf-net.dll**  **Lib\Mono.Cecil.dll**  **Lib\ZyGames.Framework.Common.dll**  **Lib\ZyGames.Framework.dll** |
| **GameNotice** | **Lib\Newtonsoft.Json.dll**  **Lib\NLog.dll**  **Lib\protobuf-net.dll**  **Lib\MySql.Data.dll**  **Lib\ServiceStack.Common.dll**  **Lib\ServiceStack.Redis.dll**  **Lib\ZyGames.Framework.Common.dll**  **Lib\ZyGames.Framework.dll**  **Lib\ZyGames.Framework.Game.dll**  **GameNotice.Model**  **//以下Dll需要检查”复制本地“属性为True**  **Lib\IronPython.dll**  **Lib\IronPython.Modules.dll**  **Lib\Microsoft.Dynamic.dll**  **Lib\Microsoft.Scripting.dll** |
|  |  |

（注：检查**Python的DLL组件引用属性面版， ”复制本地“属性为True**）

## 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：

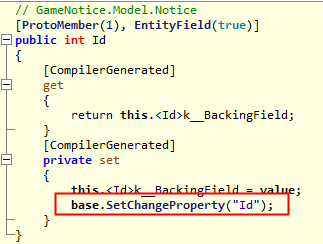


## 项目Config配置

1）修改GameNotice.Model项目csproj文件，用记事本打开GameNotice.Model.csproj文件，在结尾增加如下配置

|  |
| --- |
| <Project>  ... ...  <UsingTask TaskName="ZyGames.Framework.Common.Build.WeavingEntityTask" AssemblyFile="bin\$(Configuration)\ZyGames.Framework.Common.dll" />  <Target Name="AfterBuild">  <WeavingEntityTask SolutionDir=".\\bin\$(Configuration)" FilePattern=" GameNotice.Model.dll " />  </Target>  ... ...  </Project> |

(小提示：使用ILSpy工具反编译可以查看Model.dll的Notice类属性会被修改)



2）appSettings结点配置

|  |
| --- |
| <appSettings>  <!--必须配置  Game.Port:监听端口  Product.Code:产品代码  Product.ServerId:产品游服代码  Redis.Host:Redis主机地址  Redis.Port:Redis主机端口  Redis.Password:密码  Redis.Db:数据库位置  -->  <add key="Game.Port" value="9001" />  <add key="Product.Code" value="1"/>  <add key="Product.ServerId" value="1"/>  <add key="Redis.Host" value="192.168.1.104" />  <add key="Redis.Db" value="0" />    <!--Python脚本配置  Python\_IsDebug：是否开启调试功能  -->  <add key="Python\_IsDebug" value="true"/>  </appSettings> |

2）数据库连接配置，设置" providerName"属性切换MSSql与MySql连接

|  |
| --- |
| <connectionStrings>  <!--数据库连接配置  name: 项目Model实体类配置的ConnectKey  providerName: 数据连接驱动提供类型(MYSQL: MySqlDataProvider, MSMQ: SqlDataProvider),默认是MSMQ  connectionString: 数据库连接字符串  -->  <add name="GameData" providerName="SqlDataProvider" connectionString="Data Source=.;Initial Catalog=GameData;Uid=sa;Pwd=123;"/>  </connectionStrings> |

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：

|  |
| --- |
| 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“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  from action import \*  clr.AddReference('ZyGames.Framework')  clr.AddReference('ZyGames.Framework.Common')  clr.AddReference('ZyGames.Framework.Game')  clr.AddReference('GameNotice.Model')  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 |

接着设置PyScript目录下的所有py文件属性生成时“如果较新则复制”；

## 项目启动类

增加GameHostApp类，继承至GameSocketHost基类，在Main方法中调用

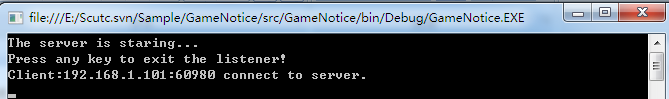
|  |
| --- |
| // Program.cs  static void Main(string[] args)  {  try  {  GameHostApp.Current.Start();  }  catch (Exception ex)  {  Console.WriteLine(ex.Message);  TraceLog.WriteError("HostServer error:{0}", ex);  }  finally  {  Console.WriteLine("Press any key to exit the listener!");  Console.ReadKey();  GameHostApp.Current.Stop();  }  } |

增加GameHostApp类中的OnRequested、OnStartAffer等方法处理代码：

|  |
| --- |
| // GameHostApp.cs  class GameHostApp : GameSocketHost  {  private static GameHostApp instance;  static GameHostApp()  {  instance = new GameHostApp();  }  private GameHostApp()  {  }  public static GameHostApp Current  {  get { return instance; }  }  protected override void OnConnectCompleted(object sender, ConnectionEventArgs e)  {  Console.WriteLine("Client:{0} connect to server.", e.Socket.RemoteEndPoint);  }  protected override void OnRequested(HttpGet httpGet, IGameResponse response)  {  try  {  ActionFactory.Request(httpGet, response, null);  }  catch (Exception ex)  {  Console.WriteLine("{0}", ex.Message);  }  }  protected override void OnStartAffer()  {  try  {  //时间间隔更新库  int cacheInterval = 600;  GameEnvironment.Start(cacheInterval, () => {  ActionFactory.SetActionIgnoreAuthorize(2001, 404);  InitNotice();  return true;  });  Console.WriteLine("The server is staring...");  }  catch (Exception ex)  {  TraceLog.WriteError("The server start error:{0}",ex);  }  }  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

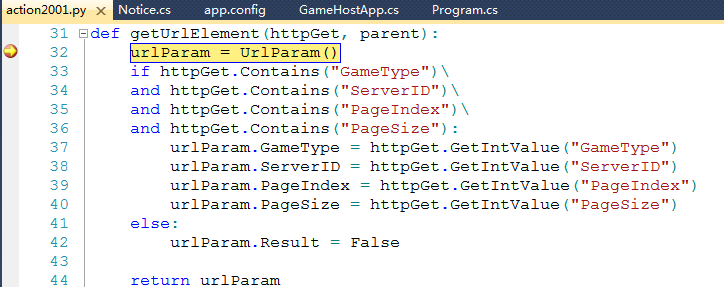
在Action2001.py文件中设置断点，按F5运行程序，结果如下：



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）协议生成器返回结果（状态值为不0，响应出错，在项目LOG是查看），如下：

