Scut入门初级教程

Gavin.Wu 2014.6.26

## 介绍

Scut是一款免费开源的游戏服务器引擎，适用于开发AVG、SLGRPG、MMOG等类型的网络游戏，同时支持Http和Socket协议通讯，支持Window、Mac和Linux多种平台部署，支持Redis内存数据库和Microsoft SQL、MySql数据库；服务器引擎框架基于C#编写，游戏逻辑层可以选择使用C#、Python和Lua多种脚本进行开发，支持热更新的方式部署；客户端可以使用Coscos2d、Unity3d、FlashAir与服务器引擎对接；提供了丰富的中间件，可以简单快捷的搭建您的游戏。

**Scut Server层次结构**

主要包括：通讯协议层，Action动作层，中间件层，数据缓存层，数据实体层，数据持久层。

层次关系如下：

* **通讯协议层**

服务器支持同时提供Http和Socket两种通讯接入方式；使用二进制的通讯协议，通过“协议生成器”工具设计客户端与服务器的通讯协议Action接口，它将生成客户端与服务器两端的接口通讯代码，减少通讯协议联调时间；另外也支持自定义的通讯协议，需要实现IActionDispatcher接口。

* **Action动作层**

它属于业务逻辑层模块，需要开发者实现，你可以选择喜欢的（C#、Python和Lua）脚本语言开发，需要为每种请求行为设计相应的Action响应数据，客户端获得数据有两种方式，一种由客户端主动请求获得响应数据，或者由服务器主动推送相应的Action响应数据。

* **中间件层**

提供的游戏模块中间件可以方便快速构建应用，特别对繁琐的渠道登录和充值的SDK模块接入进入封装，简单的配置即可完成接入，参考《中间件使用文档》。

* **数据缓存层**

为了提高服务器的响应速度，我们会将需要处理的实体数据放到缓存中，减少从数据库取数据的次数，同时缓存中的实体数据被改变后，引擎以异步的方式（100ms延迟时间）将数据同步到数据库中；当缓存中的数据一定时间（可配置）未被使用时，引擎会自动将内存释放掉。

缓存的类型分为共享全局的和私有的两种，共享的缓存如果在内存中不存在时，它会从数据库中加载所有的数据到内存中；而私有类型的只会根据相应的Key加载部分数据到内存中；因此玩家特有的数据（如：背包，任务，副本）定义成私有的，像排行榜类型的定义成共享的。

* **数据实体层**

数据实体是缓存存储的基本元素，通过对它进行序列化方式持久化存储（目前只支持C#脚本定义，实体类中不能有业务逻辑处理），并且定义有数据库表结构信息，通过表结构信息可以自动创建表、列，也及生成更新sql语句等功能，开发人员不必关注数据库结构设计，需要定义自己的数据实体类及属性（类名对应数据库的表名，属性对应字段）。

* **数据持久层**

游戏数据持久化是最重要、最复杂、也是最容易出错的部分，框架封装了对数据库数据的读取、更新、删除和表的创建维护等管理；我们支持Redis、MSSQL、Mysql等多种数据库的方式持久化。

**服务器架构**

包括：用户中心服务器，分服中心服务器，游戏私服服务器，Redis服务器，数据库服务器。

服务器的拓扑结构图：

* 用户中心服务器

用户中心负责用户注册，登录验证等服务。

* 分服中心服务器

分服中心管理多个游戏私服的状态、名称、通讯地址等信息，提供获取私服列表及私服状态查询服务。

* 游戏私服服务器

游戏私服提供具体的游戏业务逻辑服务，支持同时多人在线；一个私服对应一个Redis服务顺和一个数据库服务器。

* Redis服务器

Redis服务器主要负责存储玩家游戏数据信息，游戏私服重启后玩家的数据不会丢失。

* 数据库服务器

数据库提供游戏配置数据信息，玩家操作日志信息，或者提供报表分析数据。

# 搭建开发与运行环境

## Window平台运行环境

需要安装的软件如下：

* .NET运行库

[Net Framework 4.5.1](http://msdn.microsoft.com/zh-cn/library/5a4x27ek(v=vs.110).aspx)以上版本，开发环境需要安装[Microsoft Visual Studio 2013](http://www.cocoachina.com/bbs/read.php?tid=195353)

* 数据库

Microsoft SQL2005或MySql5.0以上版本

* Scut安装包

在[Scut官方网站](http://scutgame.com/)首页下载最新安装包，如果是32位系统则在下载区选择32位的安装包下载。

* Web服务器（可选）

Scut使用Http通讯方式时需要安装IIS 7.0以上版本Web服务器，在“控制面板”->“程序和功能”->“打开或关闭Window功能”窗口中勾选“Internet信息服务”进行安装；安装成功后，选择根目录的功能视图，检查“ISAPI和CGI限制”，设置“ASP.NET v4.0”为允许。如图：





示例使用Http通讯方式的Sample有：GameRanking、Koudai等。

* Python开发插件（可选）

使用Scut的Python脚本开发时，安装IronPython2.6.1 RC1 for .NET 4.0插件，[点击下载](http://www.scutgame.com/download/IronPython2.6.1%20RC1%20for%20.NET%204.0%20RC.msi)

## Linux平台****运行环境****

**Linux VI命令使用说明：Shift + I 修改，Esc 退出修改， :w 保存, :q 退出**

需要安装的软件如下：

* Mono平台

需要Mono3.4.0以上版本的，支持.Net Framework 4.5运行库；安装步骤如下：

1. 使用yum安装，先**更新系统**

|  |
| --- |
| #yum –y update |

1. **安装Mono需要的库**

|  |
| --- |
| #yum -y install gcc gcc-c++ bison pkgconfig glib2-devel gettext make libpng-devel libjpeg-devel libtiff-devel libexif-devel giflib-devel libX11-devel freetype-devel fontconfig-devel cairo-devel |

1. 安装Mono需要的GDI+兼容API的库Libgdiplus

|  |
| --- |
| #cd /usr/local/src/  #wget http://download.mono-project.com/sources/libgdiplus/libgdiplus-2.10.tar.bz2  #tar -jxvf libgdiplus-2.10.tar.bz2  #cd libgdiplus-2.10  #./configure --prefix=/usr  #make  #make instal |

1. 安装Mono3.4.0

|  |
| --- |
| #cd /usr/local/src/  #wget http://download.mono-project.com/sources/mono/mono-3.4.0.tar.bz2  #tar -jxvf mono-3.4.0.tar.bz2  #cd mono-3.4.0  #./configure --prefix=/usr  #make  #make instal |

1. 安装完成，测试输入如下，有mono版本信息则说明安装成功

|  |
| --- |
| #mono -V |

最后建议你运行下命令ldconfig，建议大家在安装mono后，都 ldconfig一下 。（ldconfig命令的用途,主要是在默认搜寻目录(/lib和/usr/lib)以及动态库配置文件/etc/ld.so.conf内所列的目录下,搜索出可共享的动态链接库(格式如前介绍,lib\*.so\*),进而创建出动态装入程序(ld.so)所需的连接和缓存文件.缓存文件默认为/etc/ld.so.cache,此文件保存已排好序的动态链接库名字列表. ）。

* 数据库

安装MySql5.0以上版本，使用yum下载Mysql安装包如下：

|  |
| --- |
| # yes | yum install mysql  # yes | yum install mysql-server |

接着开始安装MySql

|  |
| --- |
| # service mysqld start |

安装完成后，设置root帐号密码

|  |
| --- |
| # /usr/bin/mysqladmin -u root password '123' |

修改/etc/my.cnf配置，设置数据库编码格式及大小写规则

|  |
| --- |
| # vim /etc/my.cnf  以下my.cnf内容  [mysqld]  lower\_case\_table\_names=1  [mysql]  default-character-set = gbk |

保存并退出，测试Mysql连接

|  |
| --- |
| #mysql -u root -p123  mysql> show databases; |

* Redis数据库

安装Redis最新版本，以Redis2.6.16版本安装为例

|  |
| --- |
| #mkdir /usr/local/soft/  #cd /usr/local/soft/  #wget http://download.redis.io/releases/redis-2.6.16.tar.gz  #tar zxvf redis-2.6.16.tar.gz  #cd redis-2.6.16  #make |

启动Redis，并查看进程是否有一个叫“src/redis-server”的进程

|  |
| --- |
| # src/redis-server  #ps aux | grep redis |

启动Redis客户端，测试安装是否成功

|  |
| --- |
| #src/redis-cli  #set foo bar  #get foo |

如果显示正确的"bar"，说明运行正常

配置内存分配策略，修改/etc/sysctl.config文件

|  |
| --- |
| #vi /etc/sysctl.config  --添加以下是内容  vm.overcommit\_memory=1 |

刷新配置生效

|  |
| --- |
| # sysctl vm.overcommit\_memory=1 |

补充介绍

|  |
| --- |
| 如果内存情况比较紧张的话，需要设定内核参数：  > /proc/sys/vm/overcommit\_memory  这里说一下这个配置的含义：/proc/sys/vm/overcommit\_memory  该文件指定了内核针对内存分配的策略，其值可以是0、1、2。  0：表示内核将检查是否有足够的可用内存供应用进程使用；如果有足够的可用内存，内存申请允许；否则，内存申请失败，并把错误返回给应用进程。  1：表示内核允许分配所有的物理内存，而不管当前的内存状态如何。  2：表示内核允许分配超过所有物理内存和交换空间总和的内存 |

开放防火墙端口

|  |
| --- |
| # vi /etc/sysconfig/iptables  --以下增加内容  -A INPUT -m state --state NEW -m tcp -p tcp --dport 6379 -j ACCEPT |

输入":wq"保存退出，并重启防火墙

|  |
| --- |
| #/etc/init.d/iptables restart |

配置启动多个Redis实例，一台Redis服务器可以启动多个Redis实例，为每个实例分配一个端口，同时需要为每个实例分配相应的Redis配置文件；分配6379(默认端口)，6380，6381三个端口为例，配置文件为：redis.conf、redis6380.conf、redis6381.conf。

redis.conf配置参数

|  |
| --- |
| #是否作为守护进程运行  daemonize yes  #如以后台进程运行，则需指定一个pid，默认为/var/run/redis.pid  pidfile redis.pid  #绑定主机IP，默认值为127.0.0.1  #bind 127.0.0.1  #Redis默认监听端口  port 6379  #客户端闲置多少秒后，断开连接，默认为300（秒）  timeout 300  #日志记录等级，有4个可选值，debug，verbose（默认值），notice，warning  loglevel verbose  #指定日志输出的文件名，默认值为stdout，也可设为/dev/null屏蔽日志  logfile stdout  #可用数据库数，默认值为16，默认数据库为0  databases 16  #保存数据到disk的策略  #当有一条Keys数据被改变是，900秒刷新到disk一次  save 900 1  #当有10条Keys数据被改变时，300秒刷新到disk一次  save 300 10  #当有1w条keys数据被改变时，60秒刷新到disk一次  save 60 10000  #当dump .rdb数据库的时候是否压缩数据对象  rdbcompression yes  #本地数据库文件名，默认值为dump.rdb  dbfilename dump.rdb  #本地数据库存放路径，默认值为 ./  dir /var/lib/redis/  ########### Replication #####################  #Redis的复制配置  # slaveof <masterip> <masterport> 当本机为从服务时，设置主服务的IP及端口  # masterauth <master-password> 当本机为从服务时，设置主服务的连接密码  #连接密码  # requirepass foobared  #最大客户端连接数，默认不限制  # maxclients 128  #最大内存使用设置，达到最大内存设置后，Redis会先尝试清除已到期或即将到期的Key，当此方法处理后，任到达最大内存设置，将无法再进行写入操作。  # maxmemory <bytes>  #是否在每次更新操作后进行日志记录，如果不开启，可能会在断电时导致一段时间内的数据丢失。因为redis本身同步数据文件是按上面save条件来同步的，所以有的数据会在一段时间内只存在于内存中。默认值为no  appendonly no  #更新日志文件名，默认值为appendonly.aof  #appendfilename  #更新日志条件，共有3个可选值。no表示等操作系统进行数据缓存同步到磁盘，always表示每次更新操作后手动调用fsync()将数据写到磁盘，everysec表示每秒同步一次（默认值）。  # appendfsync always  appendfsync everysec  # appendfsync no  ################ VIRTUAL MEMORY ###########  #是否开启VM功能，默认值为no  vm-enabled no  # vm-enabled yes  #虚拟内存文件路径，默认值为/tmp/redis.swap，不可多个Redis实例共享  vm-swap-file /tmp/redis.swap  #将所有大于vm-max-memory的数据存入虚拟内存,无论vm-max-memory设置多小,所有索引数据都是内存存储的 (Redis的索引数据就是keys),也就是说,当vm-max-memory设置为0的时候,其实是所有value都存在于磁盘。默认值为0。  vm-max-memory 0  vm-page-size 32  vm-pages 134217728  vm-max-threads 4  ############# ADVANCED CONFIG ###############  glueoutputbuf yes  hash-max-zipmap-entries 64  hash-max-zipmap-value 512  #是否重置Hash表  activerehashing yes |

创建redis6380.conf配置文件，并修改配置，redis6381配置文件仿之

|  |
| --- |
| #cp redis.conf redis6380.conf  #vi redis6380.conf  --修改以下是内容  pidfile /var/run/redis/redis\_6380.pid  port 6380  logfile /var/log/redis/redis\_6380.log  dbfilename dump\_6380.rdb |

测试启动6380，6381实例

|  |
| --- |
| #redis-server /usr/local/redis/redis6380.conf  #redis-server /usr/local/redis/redis6381.conf  #ps aux | grep redis |

配置开机启动（黄色底色需要相应修改）

|  |
| --- |
| ###########################  PATH=/usr/local/soft/redis-2.6.16:/usr/local/bin:/sbin:/usr/bin:/bin    REDISPORT=6380  EXEC=/usr/local/soft/redis-2.6.16/src/redis-server  REDIS\_CLI=/usr/local/soft/redis-2.6.16/src/redis-cli    PIDFILE=/var/run/redis6380.pid  CONF="/usr/local/soft/redis-2.6.16/redis6380.conf"    case "$1" in  start)  if [ -f $PIDFILE ]  then  echo "$PIDFILE exists, process is already running or crashed"  else  echo "Starting Redis server..."  $EXEC $CONF  fi  if [ "$?"="0" ]    then    echo "Redis is running..."  fi    ;;  stop)  if [ ! -f $PIDFILE ]  then  echo "$PIDFILE does not exist, process is not running"  else  PID=$(cat $PIDFILE)  echo "Redis is stopping ..."  $REDIS\_CLI -p $REDISPORT SHUTDOWN  while [ -x ${PIDFILE} ]  do  echo "Waiting for Redis to shutdown ..."  sleep 1  done  echo "Redis stopped"  fi  ;;  restart|force-reload)  ${0} stop  ${0} start  ;;  \*)  echo "Usage: /etc/init.d/redis6380 {start|stop|restart|force-reload}" >&2  exit 1  esac  ############################## |

修改rc.local文件增加启动项

|  |
| --- |
| #vi /etc/rc.d/rc.local  --增加以下内容  bash /etc/init.d/redis start  bash /etc/init.d/redis6380 start  bash /etc/init.d/redis6381 start |

* Web服务器（可选）

Scut使用Http通讯方式的需要安装最新版本，这里安装Jexus5.4为例

|  |
| --- |
| #cd /usr/local/soft/  #wget http://www.linuxdot.net/down/jexus-5.4.tar.gz  #tar -zvxf jexus-5.4.tar.gz  #cd jexus-5.4  #sudo ./install |

安装完成，在执行install脚本时，jexus默认是安装到/usr/jexus/中，如果想安装到其他目录，可以在执行该脚本时增加一个目录的参数（比如 sudu install /usr/local/jexus/）。

启动jexus，检查是否安装成功

|  |
| --- |
| # cd /usr/jexus  #sudo ./jws start  #cat log/jws.log |

设置开机启动

|  |
| --- |
| #vim /etc/init.d/rc.local  --添加以下内容  /usr/jexus/jws start |

* 安装Python2.7（可选）

使用Scut的Python脚本开发时需要安装，安装Python时将定位到/usr/local/src目录下，如下：

|  |
| --- |
| #cd /usr/local/src  #wget http://www.python.org/ftp/python/2.7.5/Python-2.7.5.tar.bz2  #tar jxvf Python-2.7.5.tar.bz2  #cd Python-2.7.5  #./configure  #make  #make install |

设置系统环境变量“IRONPYTHONPATH”为"/usr/local/src/Python-2.7.5/Lib"，打开Putty进入终端中打开环境变量文件

|  |
| --- |
| #vim /etc/profile  --添加以下内容  export IRONPYTHONPATH=/usr/local/src/Python-2.7.5/Lib |

保存退出，刷新生效

|  |
| --- |
| #source /etc/profile |

# 第一个HelloWorld程序

通过HelloWorld程序，让我们了解如何使用C#、Python、Lua等脚本语言开发自己的游戏逻辑业务。

* 配置开发脚本语言

首先需要从官网下载Scut安装包，安装完成后，找到安装目录（C:\Program Files (x86)\SCUTServer\Console），打开ScutSMS.exe程序，如图：

需要使用Python脚本开发，则设置“PythonDisable”属性为“False”；Lua脚本开发则设置“LuaDisable”属性为“False”；默认则使用C#脚本。

或者用记事本直接打开“GameServer.exe.config”文件，配置如下：

|  |
| --- |
| <?xml version="1.0"?>  <configuration>  <appSettings>  <!--基本配置  Port:监听端口  Code:产品代码  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="Product.Code" value="0"/>  <add key="Product.ServerId" value="0"/>  <add key="Game.Port" value="9001"/>  <add key="Redis.Host" value="127.0.0.1"/>  <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"/>  <add key="Game.Action.Script.TypeName" value="GameServer.CsScript.Action.Action{0}"/>  <add key="Python\_Disable" value="True"/>  <add key="Lua\_Disable" value="True"/>  <add key="Script\_IsDebug" value="True"/>  </appSettings>  </configuration> |

使用Python脚本开发，则添加配置“Python\_Disable”属性为“False”；Lua脚本开发则设置“Lua\_Disable”属性为“False”；默认则使用C#脚本。“Script\_IsDebug”是脚本可调试配置，如果脚本不能调试需要设置它（Lua脚本不能调试，使用CPrint方法打Log）。

* 新建项目

1. 打开VS2013开发工具，新建一个“类库”项目，如图：
2. 将安装目录（C:\Program Files (x86)\SCUTServer\Console）的所有文件复制到新建立的项目目录（.\HelloWorld\HelloWorld）下；
3. 配置程序启动，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项目的当前相对目录，如图：
4. 先在开始菜单，选择SCUTServer项下启动RedisServer程序，再按F5运行调试程序，控制台打出“Server has started successfully”说明启动成功，否则启动失败（失败需要查看Log\Exception目录下的出错信息）；
5. 在VS2013项目解决方案视图中，添加HelloWorld项目引用，选择项目目录下的所有\*.dll文件；
6. 包含文件，在“解决方案管理器”显示所有文件，将Script目录与\*.config文件包括到项目中；

* 创建Action协议

需求：客户端向服务器发送请求，服务器接收到请求向客户端影应“Hello World”；

根据配置的开发脚本语言类型，创建相应的Action协议，以下分为C#、Python、Lua脚本语言协议。

1. C#脚本

在目录（CsScript/Action）下新建Action100.cs接口，代码如下：

|  |
| --- |
| using ZyGames.Framework.Game.Contract;  using ZyGames.Framework.Game.Service;  namespace GameServer.CsScript.Action  {  /// <summary>  /// 100\_Hello World  /// </summary>  public class Action100 : BaseStruct  {  private string content;  public Action100(HttpGet httpGet) : base(100, httpGet)  {  }  public override void BuildPacket()  {  this.PushIntoStack(content);  }  public override bool GetUrlElement()  {  return true;  }  public override bool TakeAction()  {  content = "Hello World for C#!";  return true;  }  }  } |

1. Python脚本

在目录（PyScript/Action）下新建Action100.py接口，并将目录（Release\PythonLib\Lib）的所有py文件复制到项目PyScript目录下，代码如下：

|  |
| --- |
| """100\_Hello World"""  import ReferenceLib  from action import \*  from ZyGames.Framework.Cache.Generic import \*  class UrlParam(HttpParam):  def \_\_init\_\_(self):  HttpParam.\_\_init\_\_(self)  class ActionResult(DataResult):  def \_\_init\_\_(self):  DataResult.\_\_init\_\_(self)  self.\_content = ''  def getUrlElement(httpGet, parent):  urlParam = UrlParam()  urlParam.Result = True  return urlParam  def takeAction(urlParam, parent):  actionResult = ActionResult()  actionResult.\_content = 'Hello World for Python!'  return actionResult  def buildPacket(writer, urlParam, actionResult):  writer.PushIntoStack(actionResult.\_content)  return True |

1. Lua脚本

在目录（LuaScript/Action）下新建Action100.lua接口，代码如下：(注：Lua脚本不能调试，可以使用CPrint打印Log跟踪)

|  |
| --- |
| Action100 ={}  function Action100:getUrlElement(httpGet, parent)  local urlParam = {}  urlParam.Result = true  return urlParam  end  function Action100:takeAction(urlParam, parent)  CPrint("use lua do action...")  local actionResult = {}  actionResult.Result = true  actionResult.\_content = "Hello World for Lua!"  return actionResult  end  function Action100:buildPacket(writer, urlParam, actionResult)  PushIntoStack(writer, actionResult.\_content)  return true  end |

在CsScript目录下新建ScriptProxy.cs类，注册Lua调用C#的方法

|  |
| --- |
| using System;  using System.Collections;  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)  {  Console.WriteLine(ex.Message);  }  }  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);  }  }  } |

* 修改MainClass类

MainClass类是脚本入口文件，包括程序启动OnStartAffer，程序结束OnServiceStop，客户请求OnRequested等方法，代码如下：

|  |
| --- |
| using System;  using ZyGames.Framework.Cache.Generic;  using ZyGames.Framework.Game.Context;  using ZyGames.Framework.Game.Contract;  using ZyGames.Framework.Game.Runtime;  using ZyGames.Framework.Game.Service;  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()  {  ActionFactory.SetActionIgnoreAuthorize(100);  //Lua script regist method  ScriptEngines.OnLoaded += ScriptProxy.Load;  ScriptProxy.RegistMethodd();  }  protected override void OnServiceStop()  {  GameEnvironment.Stop();  }  protected override void OnRequested(ActionGetter actionGetter, BaseGameResponse response)  {  Console.WriteLine("Client {0} request action {1}", actionGetter.GetSessionId(), actionGetter.GetActionId());  }  }  } |

# 发布部署

## Web发布

## Console发布

使用批处理命令部署，

# 基础知识

## 通讯消息结构

## Config配置

## MainClass文件

## 设计Action接口

## 语言包

## 设计实体

## 实体缓存

## Memory缓存

## 排序分页

## Log日志跟踪

# 参考资料

Scut官网开发文档和论坛

官网地址：http://www.scutgame.com

Github代码库：https://github.com/scutgame/scut

OSChina代码库：https://git.oschina.net/scutgame/Scut

博客园地址：http://www.cnblogs.com/scut/

技术交流QQ群：138266675