Scut 입문초급튜토리얼

|  |  |  |
| --- | --- | --- |
| 文件状态：  [ ] 草稿  [ ] 正式发布  [ ] 正在修改 | 文件标识： |  |
| 当前版本： | V1.0 |
| 作 者： | Game.wu |
| 完成日期： |  |

目录 목록

[Scut입문초급튜토리얼 1](#_Toc396227433)

[Gavin.Wu 2014.6.26 1](#_Toc396227434)

[1 Scut소개 4](#_Toc396227435)

[1.1 레이어 아키텍쳐 4](#_Toc396227436)

[1.2 서버구조 4](#_Toc396227437)

[2 개발운영환경구축 4](#_Toc396227438)

[2.1 Window플랫폼 운영환경 4](#_Toc396227439)

[2.2 Linux플랫폼 운영환경 4](#_Toc396227440)

[3 HelloWorld프로그램개발에서 부터 4](#_Toc396227441)

[3.1 개발스크립트언어설정 4](#_Toc396227442)

[3.2 새로운 프로젝트 4](#_Toc396227443)

[3.3 생성 Action 프로토콜 4](#_Toc396227444)

[3.4 변경MainClass 클래스 4](#_Toc396227445)

[3.5 기동Server 4](#_Toc396227446)

[3.6 Client테스트 4](#_Toc396227447)

[3.7 Console퍼블리싱 4](#_Toc396227448)

[4 기초지식 4](#_Toc396227449)

[4.1 통신메시지구조 4](#_Toc396227450)

[4.2 사용자정의통신메시지구조 4](#_Toc396227451)

[4.3 단방향요청모형 4](#_Toc396227452)

[4.4 엔티티Model설계 4](#_Toc396227453)

[4.4.1 프라이빗모형데이터 4](#_Toc396227454)

[4.4.2 공유모형데이터 4](#_Toc396227455)

[4.4.3 숫자모형데이터 4](#_Toc396227456)

[4.4.4 로그모형데이터 4](#_Toc396227457)

[4.4.5 Model독립프로젝트 4](#_Toc396227458)

[4.5 엔티티와 엔티티관계 4](#_Toc396227459)

[4.5.1 1대1 관계 4](#_Toc396227460)

[4.5.2 1 대 다 의 리스트관계 4](#_Toc396227461)

[4.5.3 1 대 다의 사전 관계 4](#_Toc396227462)

[4.6 엔티티Model캐쉬 4](#_Toc396227463)

[4.6.1 Search Cache Entity 4](#_Toc396227464)

[4.6.2 Update Cache Entity 4](#_Toc396227465)

[4.6.3 Delete Cache Entity 4](#_Toc396227466)

[4.6.4 ManualUpdate Entity 4](#_Toc396227467)

[4.7 Memory Cache 4](#_Toc396227468)

[4.8 스크립트 암호화 인터페이스 4](#_Toc396227469)

[4.9 Redis 데이터 복구 4](#_Toc396227470)

[4.10 Event사건통지 4](#_Toc396227471)

[4.11 메시지푸쉬와 브로드캐스팅 4](#_Toc396227472)

[4.12 서버간의 통신 4](#_Toc396227473)

[4.13 정렬과 페이징 4](#_Toc396227474)

[4.14 Log일지트래킹 4](#_Toc396227475)

[5 분배서버센터 설정 4](#_Toc396227476)

[6 프로토콜생성기 4](#_Toc396227477)

[6.1 설치설명 4](#_Toc396227478)

[6.2 사용설명 4](#_Toc396227479)

[7 참고자료 4](#_Toc396227480)

# ****Scut소개****

Scut은 무료오픈소스게임서버엔진이다. AVG, SLGRPG, MMOG 등졸류의 인터넷게임을 만드는데 적합하다, HTTP 와 Socket 통신방식을 동시에 지원하고 윈도우, MAC, 리눅스 멀티플랫폼에서 동작한다. REDIS 메모리데이터베이스와 Micosoft MSSQL 과, MySQL 데이터베이스를 지원한다, 서버엔진프레임워크는 C# 으로 작성되었다. 게임로직층은 C#, 파이선, 루아등의 스크립트를 사용해서 개발을 진행할수 있으며, 스크립트의 동적반영을 지원한다. 클라이언트는 Cocos2D, Unity3D, FlashAir 사용해서 서버와 연결할수있다. 풍부한 미들웨어를 제공하고. 아주빠르게 당신의 게임을 개발할수 있다.

## ****계층구조****

主要包括：통신패킷층，Action제어층，미들웨어층，데이터캐쉬층，데이터엔티티층，데이터보존층。각층의 관계는 아래와 같다.：



* **통신프로토콜층**

서버는 HTTP와 Socket 두종류의 통신인터페이스를 지원한다

2진수통신패킷을 사용하여, 프로토콜생성도구를 통해 클라이언트와 서버간의 통신 패킷 Action 인터페이스를 설계할수 있다.

바이너리통신을 사용하며, 프로토콜생성기를 통해 툴이 서버와 클라이언트의 프로토콜을 설정해준다.

프로토콜생성툴은 서버와 클라이언트의 인터페이스 소스를 생성해준다

서버와클라이언트 패킷처리코드를 맞추는 시간을 절약해준다

또한 커스터마이징 통신 패킷도 지원하려면 IActionDispatcher를 구현해야한다。

커스터마이징 프로토콜은 IActionDispatcher 인터페이스를 구현해야한다.

* **Action동작층 (액션 동작층)**

로직층모듈에 속한다.

개발자가 구현해야한다.

C#,Python, Lua 중에 맘에드는 스크립트언어를 사용해서 개발한다.

모든종류의요청에 대응하는 Action과 그에따르는 데이터를 설계해야한다.

클라이언트가 데이터를 얻는법은 두가지가 있다.

하나는 클라이언트가 원하는데이터를 서버에 요청하는방식, 그렇지 않으면 서버가 클라이언트에 Action을 푸쉬하여 상응하는 데이터를 전달하는 방식이다.

아니면 서버가 상응하는 액션이나 상응하는 데이터를 푸쉬하는것이다. (Request-Response모델과 푸쉬모델을 말하는듯)

액션층이 사용하는 스크립트 우선순위: 파이선 -> 루아 -> C#

* **미들웨어 층**

제공하는 게임모듈미들웨어는 편하게 빠른개발을 할수 있게한다, 특별히 特别对繁琐的渠道登录和充值的SDK模块接入进入封装，

특히나 자질구레한 채널로그인이나 충전등을 위한 SDK 를 다루는 것을 간단한설정으로 접근할수 있다. 미들웨어 사용 문서를 참조하기 바란다

* **데이터 캐쉬층**

서버의 반응 속도를 올리기 위해서 처리가 필요한 엔티티데이터를 캐쉬에 넣어야한다,

디비에서 데이터를 가져오는 횟수가 감소한다，

동시에 캐쉬내에엔티티데이터가 변경된후,

엔진은 비동기방식(100ms의딜레이)으로 데이터베이스로 동기화할것이다.

캐쉬내의 데이터는 일정시간(설정가능) 사용되지 않을시,

엔진이 자동으로 메모리를 릴리즈 할것이다.

캐쉬의 종류는 글로벌공유와, 사유의 두종류가있다,

공유하는 캐쉬가 만약 메모리상에 없을시,

데이터베이스내의 모든데이터를 메모리로 로딩할수 있다;

반면에 사유형은 상응하는 키에의거해서 데이터의 일부분을 메모리로 로딩한다;

그래서 유저의 특유의 데이터(예:가방, 퀘스트, 인스턴스던전)은 사유데이터로정의한다，

랭킹과 같은 종류는 공유데이터로정의한다

* **데이터엔티티층**

데이터엔티티는 캐쉬에저장되는기본단위이다，

엔티티 직렬화를통해 영구저장을 처리한다

（현재 C# 스크립트만 지원된다，엔티티클래스는 비즈니스로직처리를 할수없다），

동시에 데이터베이스테이블구조정보를 정의한다

，표구조 정보를 통해 자동으로 표를 생성하고컬럼그리고 SQL 갱신문등을 만들어낸다

개발자는 데이터베이스표구조설계를 신경쓸 필요가 없다.

개발자는 데이터베이스 인터페이스설계를 신경쓸 필요가 없다.

데이터엔티티클래스 멤버만정의 하면된다

자신의 데이터실제클래스속성을 정의해야한다

（클래스이름과데이터베이스표이름은 일치되고, 필드명과 멤버변수는 일치된다）。

* **데이터지구층 Persistance**

游戏数据持久化是最重要、

게임 데이터 지구화는 최고로중요하다,최고로복잡하다,

그리고 제일문제가 발생하기 쉬운부분이다

프레임워크는 데이터베이스를 캡슐화해서 일기,쓰기,갱신,삭제,표생성,유지보수등관리를 지원한다.

우리는 Redis 와 MSSQL, MySQL 등 다수의 데이터베이스를 지원한다.

## **서버아키텍쳐**

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：

포함: 유저센터서버,분배서버, 게임서버, 레디스 서버, 데이터베이스서버, 서버네트워크 아키텍쳐그림



* 유저센터서버

유저센터서버는 유저의등록과 로그인,인증의 서비스를 담당한다

* 분배센터서버

분배센터 다수의게임서버상태를 관리한다, 명칭, 통신주소등의 정보, 서버리스트정보와 서버상태쿼리를 제공한다

* 게임서버

게임서버는 구체적인 게임로직을 제공하고, 동시에 많은사람이 함께 게임할수있게해준다. 한 개의 서버는 한 개의 Redis서버와 한 개의 데이터베이스와 연결되어있다..

* Redis서버

레디스 서버는 주로 유저의 게임정보를 보존한다. 월드서버가 리부팅후에도 유저들의 정보는 소실되지 않는다

* 데이터베이스서버

데이터베이스는 게임설정정보를 제공한다, 유저의조작로그정보, 또는 데이터분석보고를 제공한다

# 개발운영환경 설정

## Window플랫폼운영환경

필수설치소프트웨어

* .NET런타임

Net 프레임워크 4.5.1 버전이상. 개발환경은 비졀스튜디오 2013

추천플러그인: Re#Sharper

* 데이터베이스

MSSQL 2005 나 MySQL5.0 이상버젼

* Scut 설치본

Scut 의 정식사이트에서 최신설치본을 다운로드한다. 만약 32비트시스템이면 32비트를 다운받는다.

* Web서버

6.7.9.0以上版本将不支持IIS部署方式，提供Console的Web支持Http协议

6.7.9.0 이상버전부터 IIS Deploy를 지원하지 않는다. Console 의 Web 지원 HTTP 프로토콜을 제공한다

* 파이선 개발플러그인(옵션)

Scut 의 파이선 스크립트를사용해서 개발할시. IronPython2.6.1 RC1 for .NET 4.0 플러그인을 설치한다.

시스템환경변수추가 (매우중요함, 게임서버엔진이 로딩될 때 Python 설치된 lib 를 로딩할수 있다)

내컴퓨터 우클릭

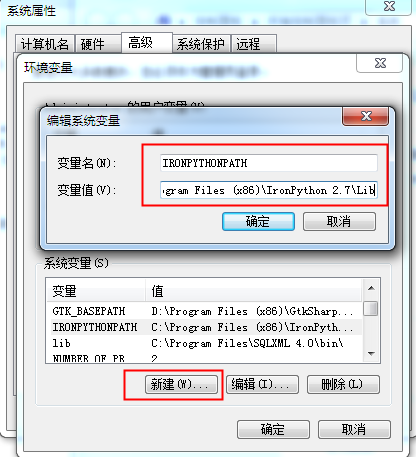
속성 클릭

고급시스템설정 선택

시스템속성윈도우 열기

고급선택에서 환경변수를 열고, IRONPYTHONPATH 변수를 추가

아래그림과 같이 설치디렉토리를 설정함



## Linux플랫폼운영환경

리눅스 VI 명령사용설명: Shitft + I 수정, ESC 수정나가기, W보존, Q 나가기

필요설치소프트웨어는다음과 같다

* Mono플랫폼

필요하다 Mono3.4.0이상버전，지원하다.Net Framework 4.5런타임라이브러리；설치순서는 아래같다：

1. 사용yum설치，먼저 시스템업데이트

|  |
| --- |
| #yum –y update |

1. **인스톨Mono필요한라이브러리**

|  |
| --- |
| #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. Mono필요한GDI+호환API의 라이브러리Libgdiplus

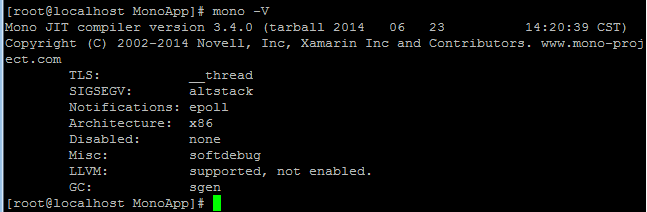
|  |
| --- |
| #cd /usr/local/src/  #wget http://download.mono-project.com/sources/libgdiplus/libgdiplus-2.10.tar.bz2  #tar -jxvf libgdiplus-2.10.tar.bz2  #cd libgdiplus-2.10  #./configure --prefix=/usr  #make  #make install |

1. 설치Mono3.4.0

|  |
| --- |
| #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 install |

1. 설치완성，테스트로아래처럼입력，mono버전정보가보이면 설치성공

|  |
| --- |
| #mono –V |

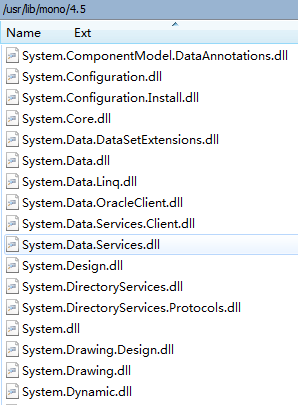


마지막으로건의 ldconfig 명령을 실행해봐라， 모두 mono를 설치한후 ldconfig 를 실행해봐라（ldconfig명령의 용도,주로 기본검색디렉토리(/lib 와 /usr/lib) 그리고 동적라이브러리, 설정파일/etc/ld.so.conf 내의 모든디렉토리에서 동적링크라이브러리를 검색해찾아낸다

(기본검색디렉토리(/lib 와 /usr/lib) 와 동적라이브러리 설정화일 /etc/ld.so.conf 내 모든디렉토리리스트에서, 검색해서 공유가능 동적라이브러리를 연결한다. 공유가능한 동적연결라이브러리를 찾는다(lib\*.so\*), 더나아가 동적으로 생성된 프로그램을 설치한다(ld.so) 모든 필요한 연결과 캐쉬파일. 캐쉬파일의 기본이름은 /etc/ld.so.cache 이다. 이파일은 이미 정렬된 동적연결라이브러리 이름의 리스트를 저장하고 있다)

1. **업그레이드.Net Framework 4.5e 몇개의DL찾을수없는문제，4.0디렉토리에서4.5디렉토리로 복사가필요하다；（CentOS명령은아래와같음）**

|  |
| --- |
| # mkdir /usr/lib/mono/4.5  # cp -rf /usr/local/src/mono-3.4.0/mcs/class/lib/net\_4\_5/\* /usr/lib/mono/4.5  # mono GameServer.exe |



Linux플랫폼 실행결과：



Dll의 검색경로，먼저현재디렉토리에서 찾음,다시Mono설치디렉토리에서 찾음，mono설치디렉토리의 4.5 디렉토리아래 이 DLL 이있는지 확인할필요가있다（없으면4.0디렉토리에서복사한다 ），만약GameServer실행시 여전히 dll을찾을수없다고나오면 Linux 를 재부팅하고 다시 시도해본다

* 데이터베이스

MySql5.0이상버전을 설치한다，yum사용해서 下载Mysql 다운로드받아설치는 다음과 같이 한다

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

계속해서 MySQL 을 설치합니다

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

설치완료후,root계정비밀번호를 설정합니다

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

/etc/my.cnf설정파일을 수정한다，데이터베이스의 문자열셋과 테이블명 대소문자를 설정합니다

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

저장한후 나가，Mysql연결을 테스트한다

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

* Redis데이터베이스

설치Redis최신버전，아래Redis2.6.16버전설치예이다

|  |
| --- |
| #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 |

기동Redis후 redis 프로세스가 있는지 확인한다.

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

Redis클라이언트를 기동한다, 설치가 성공했는지 테스트한다

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

만약 "bar" 가표시된다면，정상동작이다

메모리분배정책설정, /etc/sysctl.config 파일을 수정한다

|  |
| --- |
| #vi /etc/sysctl.config  --아래내용을 추가한다  vm.overcommit\_memory=1 |

새설정을 반영한다

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

보충소개

|  |
| --- |
| 如果内存情况比较紧张的话，需要设定内核参数：  만약메모리상황이비교적타이트하다면, 커널파라메터를 변경할 필요가 있다.  > /proc/sys/vm/overcommit\_memory  여기말해보자 이설정의 의미를：/proc/sys/vm/overcommit\_memory  이파일은 커널의메모리할당정책을 명시한다，실제사용가능값은0、1、2 이다.  0：커널이 프로세스가 필요로하는 메모리가 충분한지 검사한다  만약 충분하면 메모리를 신청한다  보충；그렇지않으면，메모리신청실패한다，오류가 어플리케이션프로세스에 반환한다.  1：커널의 모든물리메모리할당을 허용한다，현재메모리상태가 어떠한지 상관하지않는다.。  2：물리메모리크기를 넘어 와 Swap 영역과 종합한 영역의 할당을 허용한다. |

방화벽 포트 개방

|  |
| --- |
| # vi /etc/sysconfig/iptables  --아래내용추가  -A INPUT -m state --state NEW -m tcp -p tcp --dport 6379 -j ACCEPT |

":wq"입력하여 저장하고 빠져나간다，방화벽을 재기동한다

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

다수의 Redis 를 설정예제, 한 개의 Redis 서버는 여러 개의 redis 인스턴스를 기동할수있다, 각각의 인스턴스는 한 개의 포트를 분배받는다, 동시에 각각의 Redis 인스턴스는 각각의 설정파일이 필요하다；분배6379(기본포트)，6380，63813개의 포트가 있다면：redis.conf、redis6380.conf、redis6381.conf 세개의 설정파일이 필요하다。

redis.conf설정데이터

|  |
| --- |
| #데몬으로 프로그램을 실행할지 설정  daemonize yes  #데몬으로 실행한다면， PID룰을 명시해야한다 기본은 /var/run/redis.pid 이다  pidfile redis.pid  #바인딩 메인서버 IP, 기본값은 127.0.0.1이다  #bind 127.0.0.1 #Redis 기본리스닝포트  port 6379  #클라이언트 클라이언트와 통산아 몇초가 없을경우，연결을끊어开 연결할때 기본값은 300 초기값은 300초이다  #로그기록등급, 4개의 선택 이다  有4个可选值，debug，verbose（默认值），notice，warning 일명  Debug, verbos,(기본값), 공지, 경고  loglevel verbose  #지정한 로그출력된 화일명이다，기본값은 stdout , /dev/null 로 설정해서 로그를 차단할수있다.  logfile stdout  #사용할 수 있는 데이터베이스수，기본값은 16이다. 기본데이터베이스는 0이다  databases 16  #디스크에 저장하는 책략  #Keys 데이터가 1개가 변경된후，900초후 디스크에 한번쓴다  save 900 1  # Keys데이터가 10개변경된후, 300초후 디스크에 한번쓴다  save 300 10  # keys1만개가 변경된후, 60초후 디스크에 한번쓴다  save 60 10000  #dump .rdb저장할 때 압축할지말지 결정  rdbcompression yes  #데이터베이스파일이름은, 기본값은 dump.rdb 이다  dbfilename dump.rdb  #데이터베이스 파일저장 디렉토리，기본값은./ 현재디렉토리이다  dir /var/lib/redis/  ########### Replication #####################  #Redis의 복제 설정  # slaveof <masterip> <masterport> 이설정이 슬레이브로 기능할시, 마스터서버의 아이피와 포트 설정  # masterauth <master-password> 종속서버로기능시, 마스터서버의 비밀번호 설정  #연결비밀번호  # requirepass foobared  #최대클라이언트 연결수, 기본값은 무제한  # maxclients 128  #최대 메모리사용량설정, 설정값에 도달한후, 레디스는 이미 유효기간이지나거나 곧유효기간이 다가올 키들을 삭제해본다. 이방법을 실행한후, 그래도 최대메모리사용량을 넘을경우 쓰기동작은 더 이상 동작하지 않는다.  # maxmemory <bytes>  #매번 갱신조작후 로그를남길지설정，켜지않으면，정전됫을시 일정기간내의 데이터를 잃어버린다.왜냐하면redis의 동기화는 위에 Save 조건에의해 동기화가이루어진다. 그래서 어떤데이터들은 일정시간동안 메모리에만 존재하게된다. 기본값은 no 이다  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

Scut使用Console的版本支持Http通讯方式（6.7.9.0以上版本）

* 설치Python2.7（옵션）

SCUT 을 사용해서 파이선으로 개발을 하려고 하면 설치가 필요하다，파이선을 설치할 때 디렉토리는 /usr/local/src 에 넣는다，아래처럼：

|  |
| --- |
| #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 |

시스템환경변수에 “IRONPYTHONPATH”를"/usr/local/src/Python-2.7.5/Lib" 설정한다，Putty 를 열어서 터미널에서 환경변수 문서를 변경한다

|  |
| --- |
| #vim /etc/profile  --아래의 내용을 추가한다  export IRONPYTHONPATH=/usr/local/src/Python-2.7.5/Lib |

저장후나간다，갱신해서 유효하게 한다

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

# 헬로월드프로그램시작

헬로월드 프로그램을 통해 C# Python, Lua등 스크립트를이용해서 로직을 처리해보자

개발전, 패킷생성기를 사용해서, 서버와 클라간의 통신 파라메터를 정의한다.패킷생성기는 자동으로 서버클라간 양쪽의 , 패킷처리 소스코드를 생성한다.

## 개발스크립트언어 설정

먼저 오피셜웹사이트에서 다운받은 SCUT 설치본이 필요하다. 설치완료후 설치디렉토리에서 ScutSMS 프로그램을 실행한다. 다음그림과 같다.



파이썬스크립트로 개발하려면, PythonDisable 속성을 false 로 설정해야한다

Lua 스크립트로 개발하려면 LuaDisable 속성을 false 로 설정해야한다

아니면 Notepad 로 직접 GameServer.exe.config 파일을 열어 아래와같이 설정하자

|  |
| --- |
| <?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>  <connectionStrings />  </configuration> |

파이선스크립트를 사용해서 개발하면 python\_disable 속성을 false 로 설정하고 Lua 스크립트를 사용해서 개발하면 Lua\_disable 을 false 로 설정한다. 내비두면 C# 스크립트로 개발하는것입니다.

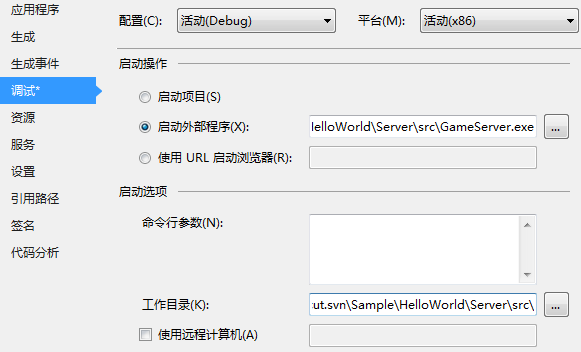
Script\_IsDebug 는 스크립트 디버깅가능한지 설정하는것이다. 만약 스크립트가 디버깅할수 없으면 이것을 설정해야한다( 루아스크립트는 디버깅할수없다. CPRINT 로 로그를 남겨라)

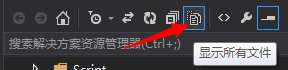
## 새프로젝트 만들기

1. VS2013 개발도구를 연다, Class Library 프로젝트를 새로 만든다. Net FrameWork 4.5.1 을 선택한다



1. SCUT 설치 디렉토리에서 모든화일을 새로만든 프로젝트 디렉토리로 복사한다
2. 헬로우월드 프로젝트 우클릭후 속성을 선택한후 디버깅항목을 열고 외부실행프로그램을 GameServer.exe 로 설정한다. 그리고 시작디렉토리를 HelloWorld 프로젝트가있는 위치로 설정한다. 아래그림처럼….



1. 솔루션뷰에서 참조를 우클릭해서 참조추가를 누른다.  튀어나온 윈도우에서 browse 를 선택에서 HelloWorld 디렉토리아래의 모든 \*.dll 파일을 선택한다. 확인을 누른다, dll 프로그램 추가가 완성되었다.
2. 솔루션 관리기에서 현재 프로젝트를 선택한다.모든 파일 표시를 클릭한다.  다시 Script 디렉토리의 모든 \*.config 파일을 선택한다. 우클릭하여 프로젝트에 포함을 선택한다. 파일은 이미 프로젝트에 포함되었다

## Action 프로토콜 생성

action 설계 요구사항은 아래와같다.：

클라이언트가 서버로 요구사항을 보낸다. 서버는 클라이언트에게 응답으로 Hello World 를 보낸다.

프로토콜 생성도구에서 클라이언트통신 파라메터를 설정한다. 아래쪽 STRING 형 에 Content 파라메터를 아래그림과 설정한다.



설정한 스크립트 언어 유형에따라 상응하는 Action 프로토콜이 생성된다. 아래부분은 C#, Python, Lua 스크립트 언어별로 분류되었다.

1. C# 스크립트

CsSscript/Action 디렉토리에 Action100.cs 인터페이스를 새로생성한다. 코드는 아래와 같다.

|  |
| --- |
| 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;  }  }  } |

만약 다른 C# 스크립트작업디렉토리정의가 필요할경우 CsScript 디렉토리 밑에 있어야한다(예: CsScript/BLL/)

1. 파이션 스크립트

PyScript/Action 디렉토리에 Action100.py 인터페이스를 생성한다. Release\PythonLib\Lib 폴더내의 모든 py 파일을 PyScript 디렉토리로 복사합니다. 소스코드는 아래와 같습니다.

|  |
| --- |
| """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 |

또다른 파이선 스크립트가 필요하다면 필수적으로 PyScript 디렉토리 밑에 만들어야한다 예) PyScript/BLL/디렉토리

1. Lua脚本 (루아 스크립트)

LuaScript/Action 디렉토리에 Action100.lua 인터페이스를 만듭니다. 소스는 아래와 같다( 주: 루아 스크립트는 디버깅이 안됨, CPrint 로 Log 를 찍을수 있음)

|  |
| --- |
| 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 |

CsSciprt 디렉토리에 ScriptProxy.cs 클래스를 새로만든다. Lua 가 호출할 C# 메서드들을 만든다.

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

## 메인클래스 수정

메인클래스 크래스는 스크립트의 입구 파일이다. 프로그램이 시작할 때 OnStartAfter 함수와 프로그램이 끝날 때 OnServiceStop 함수와 클라이언트가 요청했을 때 OnRequested 등의 메서드를 포함한다. 소스는 아래와같다

|  |
| --- |
| 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 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());  }  }  } |

## 서버기동

먼저 Redis서버 데이터베이스서버가 열렸는지 확인한다. 그다음 GameServer.exe 를 실행한다. 만약 Server has started successfully 라고 표시되면 서버기동이 완성된것이다. 만약 Error: the redis server is not started 가표시된다면 redisServer 가 기동되지 않은것이다. (설치본을 설치하면 찾을수 있다).. Server has started failed 라고 표시된다면. Log/Exception 디렉토리의 에러정보를 체크해봐야한다.(오류가 난 시작부분과 묘사와 비헤비어정보가 있다) 아래는 만약 snscenter 데이터베이스를 찾지 못할경우 아래와 같은 메시지가 표시된다. 이데이터베이스는 서비스분배센터 설정하는데 참고한다.

|  |
| --- |
| Time:2014/7/27 21:04:15-Trace>>Server failed to start error:System.Exception: Not connect to the database server "localhost" database "snscenter". |

## 기동 클라이언트 테스트

Sample/HelloWorld/Client 디렉토리에서 MainApp.exe 를 실행하고 아래그림의 Send 를 눌러서 Request 를발송한다

]

## Console 배포

배치명령어를 사용해서 release.bat 를 생성한다.

|  |
| --- |
| ROBOCOPY scr release \*.dll \*.exe \*.config \*.cs \*.py \*.lua /S /PURGE /XD bin obj log temp ScriptRuntimeDomain |

Release 폴더의 내용을 서버로 업로드한다. 그리고 GameServer.exe 를 기동한다. (Redis 먼저 켜고)

# Knowledge base

## 통신메시지구조

* **바이너리구조**

**요청 구조**

**HTTP 와 소켓 요청형식은 URL의 파라메터방식으로 전달된다(예: Name=xxx&id=2), 파라메터는 아래와 같다**

**.**

|  |  |
| --- | --- |
| 파라메터명 | 설명 |
| MsgId | 메시지 ID, 클라이언트가 매시지를 보낼때마다 증가한다 |
| ActionId | 액션ID, 서비스가 정의한 프로토콜ID |
| Sid | 세션ID, 서버가자동생성해서 클라이언트에 보내준아이디, 클라이언트가 요청을 할때마다 서버에 제공해야한다. 소켓을 사용할 때 접속이 끊어지면 재접속할 때 사용되며 동시에 여러곳에서 접속하는 것을 방지한다. |
| Uid | 유저아이디, 로그인후 신분을표시한다. |
| St | 시간값. 요청테스트할때 고정된 st사용함 |
| 확장파라메터 | 액션 프로토콜의 요청 파라메터의 정의 |
| 시그네쳐 | 암호화결과 |

**파라메터(q)：sgId=1&ActionId=1005&Sid=1000000000&Uid=9999&St=st&확장파라메터**

**확장파라메터：PageIndex=1&PageSize=20**

**서명：MD5(q+Key)，Key는 자체정의한 16이나32글자의 임의문자이다**

**파라메터데이터 (d)：(urlencode(q+”&sign=”+sign)중국어는encoding 될필요가 있기 때문에**

**HTTP 요청예시.** <http://localhost/Service.aspx?d=d>

**소켓요청 예시**

**Data packet(pack)：GetBytes("?d=" + d)，스트링은 byte[]이다.**

**데이터헤더(head)：GetBytes(pack.Length)**

**발송하는 데이터：CombineByte(head, pack)**

**출력 구조**

**바이너리 방식으로 클라이언트에게 전달할 때 byte, short, int, string 등기본데이터형을 byte 로 변환해서 보낸다. Byte 는 순서가 있고 string 을 제외한 다른형식들의 길이는 고정적이다. 길이에대한 코드를 따로 넣을 필요가 없다. String 의경우 특별히 길이에대한 코드를 넣는다.**

기본형에 대한 정의는 아래와 같다.

|  |  |  |
| --- | --- | --- |
| 파라메터타입 | 점유바이트 | 설명 |
| Byte | 1 | 바이트 |
| Int16(Short) | 2 | Short |
| Int32 | 4 | Int |
| String | Len(4) + Len(string) | 문자열 len 은 길이를 뜻한다 |
| Record | Count(4) + Len(Record) | 순환체 Count 는 열수를 표시한다. |
| End | 空!!!?? 리얼리? | 순환끝표식 |

**실례：**

**서버는 Byte,Short, Int,String 형의 필드와 한 개의 List 정보를 내려보낸다. 리스트는 Int 형인 두개의 컬럼을 포함한다. 프로토콜 파라메터 정의는 아래와같다.：**



**내려보내는 내용: 1 + 10 + 100 + Hello + Record(20,3) + Record(21,4) 이진수로 변환한후 내용은 다음과 같다**

**(주: 오렌지색배경은 라인번호를 기록한다, 파란색배경은 원래의 길이를 말한다. 하약색의 전경은 원래의 뎅이터를 말한다). 바이너리 내용은 기본형 + 레코드카운트 + 레코드1 + 레코드2 로 나뉜다.**

**첫단계(필드):파란색(5000)은 문자열내용의 길이를 말한다.**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **1** | **10** | **0** | **100** | **0** | **0** | **0** | **5** | **0** | **0** | **0** | **72** | **101** | **108** | **108** | **111** |

**단계2: 레코드 행수 (순환횟수)**

|  |  |  |  |
| --- | --- | --- | --- |
| **2** | **0** | **0** | **0** |

**레코드1항 (8계산공식, ID 4자리 + LV 4자리 더한값)**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **8** | **0** | **0** | **0** | **20** | **0** | **0** | **0** | **3** | **0** | **0** | **0** |

**레코드행 2항**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **8** | **0** | **0** | **0** | **21** | **0** | **0** | **0** | **4** | **0** | **0** | **0** |

Data Packet 형식: **헤더 + Len(data) + data**

**헤더의 필드 요소**

|  |  |  |
| --- | --- | --- |
| 원형 | 내용 | 설명 |
| Int32 | 0 | 에러코드 |
| Int32 | 1 | 요청메시지의 아이디, 요청의상응하는 것을 판단한다. Push 방식은 고정된 0을 사용한다. |
| String | "" | errorInfo |
| Int32 | 1001 | Action을 처리할 때 Action 층의아이디. |
| String | "st" | 시간값. 인증할때는 고정된 st 를 사용한다? |

**Head = 0 + 1 + "" + 1001 + "st"**

**메시지 바디의 규격**

|  |  |  |
| --- | --- | --- |
| 통신형식 | 응답형식(규격) | 설명 |
| Http | Len(Pack) + Pack | Http메시지처리와관계없다，Len 부분만 정의한다  그냥 길이만 정의해라.  길이가 60이라면  Ex：60 + Pack |
| Socket | Len1(Pack) + Len2(Pack) +Pack | Len1은 패키징된 패킷을 처리하기위함이다.  Ex: 60 + 60 + Pack |
| Socket Gzip | Len1(Pack) + Gzip(Len2(Pack) +Pack) | 10k보다 큰패킷일 때 자동으로 GZip 압축 전송이 기동된다.  Ex: 60 + Gzip( 60 + Pack )  Gzip압축여부헤더 4바이트(16진수)：1F 8B 08 00 |

**Socket이 GZip압축을 사용하기시작하면 아래와 같이 된다. （16진수，smsniff툴로패킷을잡음）：**



## ****자체정의 통신메시지 구조****

**메시지요청**

**요청메시지는 필수 제공 파라메터 필수로제공해야하는 파라메터만 제공하면.메시지구조를 당신마음대로 정의할수 있다. 암호화전송을 해도되고.아니면 오직 헤더부분만으로 인증을 할수도 있고 물론 이런것들은 모두 스스로 구현해야한다. 필수제공 파라메터는 아래와 같다.**

|  |  |
| --- | --- |
| **파라메터** | **설명** |
| **MsgId** | **유저의 요청 인덱스 매요청시 증가됨** |
| **SessionId** | **유저의 세션아이디. 로그인성공시 클라에 세션아이디를 보낸다** |
| **ActionId** | **요청한일을 처리할 Action 번호..** |
| **UserId** | **유저의 신분아이디** |

**자체정의통신메시지를 구현하려면 IActionDispatcher 인터페이스를 구현해야한다 샘플의 게임랭킹을 참고하라**

* **자신의 ActionDispatcher 의 정의**

**IActionDispatcher인터페이스를 구현한 CustomActionDispatcher 클래스를 정의한다，그중TryDecodePackage메서드는 Socket、Http（Web）와HttpListener（Scut이구현한）요청 처리로 나뉘어진다，메서드는 해당통신의 메시지패킷의 헤더분석한다，헤더정보는 MsgId、SessionId、ActionId、UserId이 4개의 정보를 포함해야한다，소켓통신만 사용한다면 그중에 한 개의 메서드만 구현하여 사용하면된다; 메시지내용은 액션층에서 해석된다；**

* **(비즈니스)워킹층에서 패킹과 언패킹**

**정의된 액션 프로토콜중(예:Action1001), GetUrlElement 메서드와 BuildResponsePack 메서드 가 워킹층에서 팩과 언패킹의 조작을하고있다**

|  |
| --- |
| **public override bool GetUrlElement()**  **{**  **byte[] data = (byte[])actionGetter.GetMessage();**  **if (data.Length > 0)**  **{**  **requestPack = ProtoBufUtils.Deserialize<Request1001Pack>(data);**  **return true;**  **}**  **return false;**  **}**  **protected override byte[] BuildResponsePack()**  **{**  **return ProtoBufUtils.Serialize(responsePack);**  **}** |

**메인클래스 클래스에서 커스텀프로토콜을 사용한다고 설정한다.**

|  |
| --- |
| **public MainClass()**  **{**  **GameEnvironment.Setting.ActionDispatcher = new CustomActionDispatcher();**  **}** |

## 단방향요청모형

**단방향프로토콜. 요청을 받은후 응답을 하지 않는다. Actionxx 클래스중 IsNotRespond 속성을 true 로 설정한다.**

## 엔티티Model설계

게임의 실제데이터설계는 주요하게 네개의 형식으로나눌수있다: 유저의 프라이빗데이터, 유저의 공용데이터, 설정데이터, 로그데이터.

게임서버의 엔티티데이터 저장구조는 다음과같이 분류된다. ：엔티티，딕셔너리등；메모리컨테이너가 딕셔너리모형이다， Entity의저장형식은（key：entitykey，value：entity），Dictionary의저장형식은（key: persionId，value：Dictionary[key：entitykey，vallue：entity]）; Dictionary의 격식의 주된 목적은 유저데이터의 그룹분류이다.， Entity가 전부이다.

**Model리플레션 데이터테이블명의 규칙**，TableNameFormat속성을 설정하여 테이블명 격식을 정의할수 있다

예：로그의 테이블명은 “년월일”의 격식이 필요하다(Log\_201408\_ModelName），

[EntityTable(TableNameFormat="Log\_$date[yyyyMMdd]\_{0}")]

(주："Log\_"는 고정된 문자열，“$date[yyyyMMdd]"는 년월일의 표현식，"{0}"은 Model 의 명칭；그리고 1년중 몇번째 주인지표시하는 "$week"이다，예：Log\_$date[yyyyMM]$week\_{0})

모델 매핑데이터표이름생성의 규칙. 설정TableNameFormat 속성

모델 데이터 테이블의 이름을 반영해서 규칙을 생성한다. TableNameFormat 속성을 자체정의형식을지원하도록 설정한다.

예. 일기의 표이름은 년월일 형식으로 만든다 (Log\_201408\_ModelName)

[EntityTable(TableNameFormat="Log\_$date[yyyyMMdd]\_{0}")]

주: “Log\_” 는 고정된문자, “$data[yyyMMdd]” 는 일자의 표현식, “{0}” 은 모델의 이름. 그리고 일년중 몇번째 주인지 표시하는 $week 예: Log\_$date[yyyyMM]$week\_{0})

### 사유모형데이터

파라메터CacheType.Dictionay 종류설정: 설정엔티티는 유저사유데이터이다. 모든유저가 이데이터에 접근할수 있다. 그리고 엔티티클래스는 BaseEntity 클래스를 상속해야한다. 유저퀘스트 실례는 아래와 같다( 주: ConnData 는Config 설정한데이터베이스연결문자열의 Name 값이다)

|  |
| --- |
| <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; }  }  } |

### 공유모형데이터

CacheType.Entity 유형을 설정한다. 설정엔티티는 글로벌하게 읽고쓸수있다. 엔티티형은 shareEntity 클래스를 상속받아야한다. Ranking 예제이다.

|  |
| --- |
| 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;  }  }  } |

### 숫자모형데이터

설정파라메터AccessLevel.ReadOnly를 설정한다，엔티티는 읽기전용이다, 엔티티클래스는 ShareEntity 를상속한다；예> 지도정보실예：

|  |
| --- |
| 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;  }  }  } |

### 로그모형데이터

설정파라메터AccessLevel.WriteOnly유형,이엔티티유형은 LogEntity클래스를 상속한다；예> 유저의 로그인 로그 실예：

.

|  |
| --- |
| namespace Demo.Model  {  [Serializable, ProtoContract]  [EntityTable(AccessLevel.WriteOnly, "ConnLog")]  public class UserLoginLog: LogEntity  {  public UserLoginLog ()  {  }  //配置自增列 Auto increment  [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;  }  }  } |

### Model독립프로젝트

1）Model을 스크립트방식으로 사용할시 Script/Model 디렉토리 아래에 넣어주기만 하면된다.

2）Model을 독립프로젝트로 정적 DLL 생성으로 사용할시，프로젝트화일의 설정이 필요하다, 수동으로 모델에 이벤트를 주입해야한다. NotePad 로 ModelProject.csproj 파일의 맨아래 다음과 같은 설정을 추가한다..

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

그리고config설정화일에 Game.Entity.AssemblyName의 값을 추가한다

|  |
| --- |
| <add name="Game.Entity.AssemblyName" value="**ModelProject.dll**" |

ILSpy 도구로 역어셈블해서 Model의 Notice 클래스의 속성이 변경된 것을 찾을수 있다.



## 엔티티와 엔티티 관계

### 1:1 관계

**모든 엔티티는 EntityChangeEvent 클래스를 상속해야한다. 또한 Serializable 과 ProtoContract 확장속성 표식을 해야하고 멤버와 속성멤버는 ProtoMember 확장속성 표식을 해야한다.**

부친과 자식의 1대1관계

|  |
| --- |
| 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;  }  }  } |

### 일대다 의 리스트 관계

부친과자식클래스는 1대다관계

|  |
| --- |
| 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;  }  }  } |

### 일대다 사전 관계

부친과자식클래스는 1대다 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)  {  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;  }  }  } |

## 엔티티모델캐쉬

CacheStruct캐쉬는 다음과 같이 분류한다.：ShareCacheStruct와PersonalCacheStruct클래스，（주：ConfigCacheSet와GameDataCacheSet는 더 이상사용하지 않는다 ）；

ShareCacheStruct클래스：글로벌의 캐쉬구조모형，레디스메모리나 DB 에서 로딩한 모든유저의 데이터가 남아있는 메모리；

PersonalCacheStruct클래스：유저사유의구조모형，일반적으로 UserId 가 주키로 사용되고 오직 레디스메모리나 데이터베이스에서 로딩된 자신에게 속해있는 데이터이다.

### 캐쉬엔티티찾기

ShareCacheStructs찾는메서드는 다음과같다：Find、FindKey、FindAll메서드（없으면자동으로 레디스나 데이터베이스에서 로딩한다 ），반면Foreach메서드는 로딩되지 않는다. 오직 캐쉬메모리데이터에서만 검색한다

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

PersonalCacheStruct찾는메서드는 다음과 같다.：TryFind、TryFindKey、TryFindAll、Find、FindKey、FindAll메서드（없으면 자동으로 레디스나 데이터베이스에서 로딩한다. ），반면 FindGlobal、Foreach메서드는 로딩하지 않는다. 오직 캐쉬메모리에서만 검색한다.

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

### 캐쉬엔티티갱신

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

### 캐쉬엔티티삭제

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

### 수동갱신엔티티

DataSyncQueueManager클래스는 데이터베이스나 레디스데이터를 업데이트할수 있는 Send 와 SendToDB 메서드를 제공한다, SendToDB 메서드는 오직 DB 만 갱신한다. 엔티티삭제는 onDelete 메서드를 사용하면 된다.

|  |
| --- |
| var user = new GuestUser();  user.UserId = 1380001;  user.NickName = "test";  user.PassportId = "z1000";  user.RetailId = "0000";  user.OnDelete();  DataSyncQueueManager.SendToDb(user); |

## Memory캐쉬

MemoryCacheStruct일종의 오직 메모리에만 존재하는 데이터 모형을 제공한다，데이터베이스난 레디스와 동기화가 필요하지 않다, 수동으로 데이터베이스의 내용을 메모리에 로딩할수 있다.，패키징 추가삭제수정검색등의 동작을 메모리방식으로 수행한다.；이모형을 사용하려면 MemoryEntity 클래스를 상속 구현해야한다.

용례

|  |
| --- |
| public class MyEntityMemory : MemoryCacheStruct<MyEntity>  {  protected override bool InitCache()  {  //load data from db  var dbProvider = DbConnectionProvider.FindFirst().Value;  var command = dbProvider.CreateCommandStruct("table", CommandMode.Inquiry);  command.Filter = dbProvider.CreateCommandFilter();  command.Filter.Condition = string.Format("{0} and {1}",  dbProvider.FormatFilterParam("fromDate", ">="),  dbProvider.FormatFilterParam("toDate", "<"));  command.Filter.AddParam("fromDate", DateTime.Now.AddDays(-1));  command.Filter.AddParam("toDate", DateTime.Now);  command.Parser();  using (var read = dbProvider.ExecuteReader(CommandType.Text, command.Sql, command.Parameters))  {  while (read.Read())  {  var t = new MyEntity();  t.Code = read["Code"].ToString();  t.Name = read["Name"].ToString();  this.AddOrUpdate(t.Code, t);  }  }  return true;  }  }  public class MyEntity : MemoryEntity  {  public string Code { get; set; }  public string Name { get; set; }  }  //调用  var cacheSet = new MyEntityMemory();  var result = cacheSet.FindAll(t => t.Code == "1001"); |

## 스크립트 암호/복호인터페이스

Script 디렉토리밑 스크립트는 암호화한후 퍼블리슁한다. 암호화엔진제공이필요하다. 스크립트암호화기능은 스스로개발한다. Scut 은 암호화복호화를 위해 ScutSecurity 클래스라이브러리를 제공한다. 스크립트암호화 설정은 아래와 같다.

|  |
| --- |
| <!--스크립트복호화설정-->  <add key="Game.Script.DecodeFunc.TypeName" value="ScutSecurity.ScriptDes,ScutSecurity"/> |

## Redis데이터 복구

Scut은 로딩데이터는 데이터베이스데이터를 로딩한 것이 아니다 DB 데이터와 레디스와 동기화되지 않은 상황이 있을수 있기 때문에 우리가 사용하는 레디스는 아주 믿을만한방식으로 데이터를 보존한다. 만약 임시로 데이터베이스의 데이터로 레디스의 데이터를 복구하고 싶다면 아래의 방법을 사용할수 있다.

|  |
| --- |
| var cache = new ShareCacheStruct<UserRank>();  SchemaTable schema = EntitySchemaSet.Get<UserRank>();  var provider = DbConnectionProvider.CreateDbProvider(schema);  var filter = new DbDataFilter(0);  filter.Condition = provider.FormatFilterParam("userid");  filter.Parameters.Add("userid", userid);  cache.TryRecoverFromDb(filter)；  //all  cache.TryRecoverFromDb(new DbDataFilter(0))； |

## Event사건통지

비동기로직을 처리할 필요가 있을 때 아니면 타임아웃을제어해야하는 기능을 실행할 때 Event 사건통지 모형을 사용할수 있다.

|  |
| --- |
| var handle = new NotifyEventArgs();//异步 비동기  handle.Callback += args =>  {  //这里处理异步业务 비동기처리를 여기서해  };  EventNotifier.Put(handle);  var handle = new TimeoutNotifyEventArgs(new TimeSpan(0, 0, 0, 30));//超时事件30s 30초가넘어가면  //handle.Target = UserData;  handle.Callback += args =>  {  //这里处理超时业务 시간넘어간처리 여기서해  }; |

자체정의이벤트는 NotifyEventArgs클래스를 상속한다.

## 메시지푸쉬와 브로드캐스팅

；서버가주동적으로 클라이언트에 푸쉬하거나 브로드캐스팅한다. GameSession 클래스가 제공하는 정적메서드Get 과 GetAll 사용해서 푸쉬할 유저를 가져온다. 다시 GameSession을 통해 오브젝트의 PostSend 메서드로 2진수메시지를 발송한다.

。ActionFactory 클래스는 Action층의패킷을 제공한다. SendAsyncAction 메서드를 사용해서 특정유저에게 서로다른 Action 처리결과를 푸쉬할수 있다. BroadcastAction 메서드는 특정유저에게 똑 같은 Action 처리결과를 푸쉬할수 있다.

용례

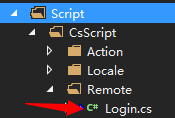
|  |
| --- |
| //GameSession클래스  byte[] data = Encoding.UTF8.GetBytes("Helo world");  var list = GameSession.GetAll().Take(10).ToList();  foreach (var session in list)  {  session.PostSend(data, 0, data.Length);  }  //ActionFactory클래스  var cacheSet = new PersonalCacheStruct<GameUser>();  var userList = cacheSet.FindGlobal(t => true);  //Client에게 브로드케스팅Action은1001의 패킷  ActionFactory.BroadcastAction(1001, userList, null, null);  var param = new Parameters();  param["Name"] = "Jim";  ActionFactory.SendAsyncAction(userList, 1002, param, actionGetter =>  {  Console.WriteLine("Send {0} action {1} ok", actionGetter.GetUserId(), actionGetter.GetActionId());  }); |

1001패킷을 브로드캐스팅 푸쉬한다면 Action1001클래스설정에 접근통제레벨을 설정할 필요가 없다.

|  |
| --- |
| class Action1001  {  protected override bool IgnoreActionId{ get{return false;}}  } |

## 서버간 통신

서버간통신은 Http 와 Socket프로토콜을 지원한다. 서버에서 RemoteClient 가 발송하는 Http 나 Socket 요청을 생성한다. 또다른 또다른 서버는 Remove 의 요청을 받도록 한다. Remote 요청의 분발은 로그인에서 처리된다. 분발규칙은 클래스명.메서드명 이다 아래그림을 보자 (Login.Check 이다)



Remote모형의 스크립트 Login코드는 아래와 같다

|  |
| --- |
| namespace Game.Script.Remote  {  public class Login : RemoteStruct  {  public Login(ActionGetter paramGetter, MessageStructure response)  : base(paramGetter, response)  {  }  protected override bool Check()  {  //较验请求 요청검증  return true;  }  protected override void TakeRemote()  {  var session = ParamGetter.GetSession();  if (session == null)  {  return;  }  Console.WriteLine("login {0} msgId:{1}", session.ProxyId, ParamGetter.GetMsgId());  var e = new NotifyEventArgs();  e.Callback += s =>  {  Console.WriteLine("Push message");  if (!session.IsSocket)  {  return;  }  MessageStructure ms = new MessageStructure();  ms.PushIntoStack("Hello..");  ms.WriteBuffer(new MessageHead());  var data = ms.PopBuffer();  session.SendAsync(data, 0, data.Length);  };  EventNotifier.Put(e);  }  protected override void BuildPacket()  {  }  }  } |

서버의 Remote 요청 발송은 ,RemoteService 클래스에서 먼저 프록시를 생성하고 Call 메서드를 호출한다..

호출하는 실례는 아래와 같다..

|  |
| --- |
| private static RemoteService \_httpRemote;  private static RemoteService \_tcpRemote;  private void Init()  {  \_httpRemote = RemoteService.CreateHttpProxy("proxy1", "http://127.0.0.1/service.aspx");  //tcp 하트비트인터벌은 30초  \_tcpRemote = RemoteService.CreateTcpProxy("proxy2", "127.0.0.1", 9001, 30 \* 1000);  tcpRemote.PushedHandle += OnPushCallback;  }  private void OnPushCallback(object sender, RemoteEventArgs e)  {  byte[] data = e.Data;  //여기서 TCP 형식의 Push 분류로직을 처리한다  }  private static void Test()  {  httpRemote .Call("**Login**", param, p =>  {  Console.WriteLine("msgid:{0} {1}ms",p.MsgId, MathUtils.DiffDate(p.SendTime).TotalMilliseconds);  });  tcpRemote.Call("**Login**", param, p =>  {  Console.WriteLine("msgid:{0} {1}ms",p.MsgId, MathUtils.DiffDate(p.SendTime).TotalMilliseconds);  });  } |

## 랭킹과 페이징

리스트에대한 소팅과 페이징 알고리즘을 제공한다. MathUtils 클래스나 List 오브젝트의 확장메서드로 InsertSort, QuickSort, GetPaging 를 제공한다

용례

InsertSort메서드

|  |
| --- |
| using ZyGames.Framework.Common;  var list = new List<int>();  for (int i = 0; i < 10; i++)  {  int item = RandomUtils.GetRandom();  list.InsertSort(item, OnComparision);  }  Console.WriteLine(string.Join(",",list));  private int OnComparision(int x, int y)  {  return x - y;  } |

QuickSort메서드

|  |
| --- |
| var list = new List<int>();  for (int i = 0; i < 10; i++)  {  int item = RandomUtils.GetRandom();  list.Add(item);  }  //list.QuickSort();  list.QuickSort(OnComparision);  Console.WriteLine(string.Join(",", list));  private int OnComparision(int x, int y)  {  return y - x;//내림차순  } |

GetPaging메서드 첫번재페이지 매페이지는 20개항

|  |
| --- |
| int pageCount;  int recordCount;  var pageList = list.GetPaging(1, 20, out pageCount,out recordCount); |

## Log트래킹

服务日志跟踪是必不可少的部分，它可以帮助你分析程序的运行状态和异常信息；使用Nlog组件打印日志，提供了TraceLog的封装类操作；划分以下目录：서버로그트래킹은 무시할 수 없는 부분이다. 로그는 서버운영상태와 이상상태정보를 분석하는데 도움을 준다. NLog 사용해서 로그를 출력한다.

Info：정식서버운행의 정보 적다. 성능에 영향이 적다

Debug：서버디버깅트래킹정보，테스트단계에서 버그문제를 트래킹하기위해사용함

Exception：서버이상정보

Fatal：서버다운급의 문제정보

Warn：서버경고등급의 정보

Client：클라이언트가 업로드한피드백, 크래쉬 정보등

SqlError：서버뒷단 큐처리데이터를 데이터베이스에 동기화할 때 이상정보

Complement：기타정보，Client，SqlError은 이타입의 하위 타입이다

용례

|  |
| --- |
| using ZyGames.Framework.Common.Log;  TraceLog.ReleaseWrite("Log info-{0}...", "hello");  TraceLog.ReleaseWriteDebug("Log debug...");  TraceLog.WriteError("Log error...");  TraceLog.ReleaseWriteFatal("Log fatal...");  TraceLog.WriteWarn("Log warn...");  TraceLog.WriteTo("Client", "Log client...");  TraceLog.WriteSqlError("Log sql error...");  TraceLog.WriteComplement("Client", "Log complement..."); |

# 분배센터

분배센터는 게임정보와게임서버(군)정보를 관리하고 아래 인터페이스를 제공한다.

1. Service.aspx 페이지는 특정게임의서버를 얻어와서 클라이언트에게 제공한다. 유저는 먼저 서버하나를 골라 진입한후 상응하는 게임서버에 접근한다(url 주소를 얻어내고 Http 나 Socket 방식억세스) 내용형식은 이진수이다.
2. OfficialService.asp 웹페이지는 게임과 게임서버정보와 게임랭킹을 공식웹사이트에 제공한다.내용은 Json 형식이다.
3. DirService.asmx 는 WebService 서버방식 뒷단 관리로 사용한다(추가,수정,삭제)등 작업. 일반적으로 IIS 설정 IP 권한으로 접근을 제어한다

**Import Database**

“Source\DirCenter\db\MsSql\必备库创建一键运行.bat”파일을 연다，필요한부분을 변경한다

|  |
| --- |
| set dbServer=.  set dbAcount=sa  set dbPass=123  set gameuser=game\_user  set gamepass=123  set dbpath=%cd%\Data |

더블클릭“必备库创建一键运行.bat”파일，그대로 배치명령을 실행한다.，분배서버데이터베이스를 Import 한다.

**웹사이트**

1. 컨트롤패널->관리도구-> 인터넷서비스관리기
2. 웹사이트에서 우클릭，웹사이트추가
3. 웹사이트 디렉토리설정Scut:\Source\DirCenter\release目录
4. 어플리케이션풀설정 “Framework v4.0”와 “클래식”방식
5. Host파일설정，설정을 추가한다 “127.0.0.1 dir.scutgame.com”

데이터베이스연결 수정，Web.config파일을연다. （MySqld이라면 providerName=“MySqlDataProvider”로 설정한다）

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

# 프로토콜생성기

## 설치설명

프로토콜생성툴, 개발자가 개발단계에서 빠르게 클라이언트와 서버의 통신 프로토콜코드를 생성을제공한다. 개발효율을 높여준다. (주: 개발환경운영시, 다시 필요하지 않다(???) 또한 클라이언트의 http나 소켓요청을 시뮬레이션해볼수있다. 서버의 액션 프로토콜을. 서버는 콘솔프로그램을 이용해서 소켓요청을 웹 프로그램으로 http 요청을 테스트할수 있다.

**데이터 베이스초기화**

배치명령사용생성 (SCUT디렉토리에 일괄처리화일이있다: Scut.svn\Source\Tools\ContractTools\db\mssql\必备库创建一键运行.bat) sa id를 사용해서 데이터베이스를 생성하고 필요한 파라메터를 설정한다 (MSSQL 용)

|  |
| --- |
| set dbServer=. 데이터베이스서버주소  set dbAcount=sa 계정（DB생성권한을가지고있는）  set dbPass=123 암호  set gameuser=game\_user 게임서버에서 사용계정  set gamepass=123 게임서버에서 사용계정의 암호  set dbpath=%cd%\Data 데이터베이스화일저장위치（디폴트는 현재위치） |

Mysql 을 사용하는 사람은. Db/mysql/importdb.sql 을 사용해서 수동으로 진행한다

**웹사이트 설정**

1. 컨트롤패널->관리도구-> 인터넷정보서비스관리기를 연다.
2. 웹사이트 노드에서 우클릭해서 사이트를 추가한다 (contract.scutgame.com)
3. 실제경로를 Scut:\Source\Tools\ContractTools\release 설정한다.
4. 어플리케이션풀을 Framework v4.0 설정하고 클래식방식으로 변경
5. C:\Windows\System32\drivers\etc\Host파일에다음항목을 추가한다  
   “127.0.0.1 contract.scutgame.com”
6. Web.config 파일을 열어 데이터베이스 연결을 변경한다.(MySql 을 사용한다면 providerName = MySqlDataProvider 이다)

|  |
| --- |
| <?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

웹사이트 <http://contract.scutgame.com> 을 열면 아래화면처럼표시된다.

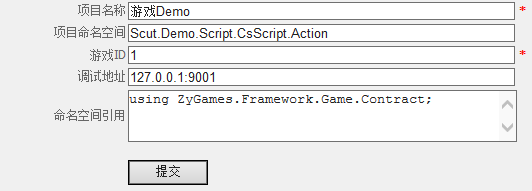


편집공구란



**프로젝트 관리**

게임프로젝트를 추가할 때 씨샵을사용해서 개발하면 프로젝트 네임스페이스와 네임스페이스 using 명령을 설정할수있다. 해당설정은C# 코드 생성할 때 참조된다. 게임ID 는 INT 여야한다. 1번부터 시작할수있다. 디버깅주소는 프로토콜 디버깅할 때 사용한다. http:// 를 앞에 붙이면 http 방식으로 요청한다. 아래는예시이다.



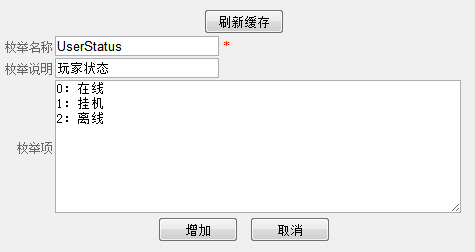
**버전관리**

게임개발중, 기획은 끊임없이 기능요구를 추가하고 여러 개의 버전이 동시에 진행될수 있다. 그래서 프로토콜인터페이스또한 버전별로 나뉠수 있다. 1.0 버전에서부터 시작될수 있다. 아래그림처럼.



**Enum 관리**

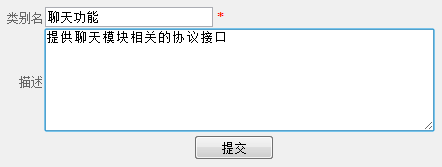
프로토콜 파라메터 설정할 때 상태등의 파라메터 등의 일종의 종류의 아이디를 설정할때가 있다., 서버는 종류의 ID의 설명이 필요하다. 도구의 오른쪽화면에 ENUM 값을 볼수 있다.





**카테고리 관리**

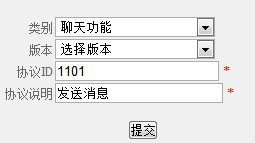
모든기능은 다수의 프로토콜 인터페이스를 가질수 있다. 프로토콜 인터페이스의 기능 분류해놓으면 검색할 때 매우 편리하다.



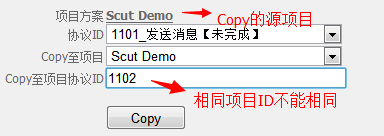
**프로토콜관리**

工具栏显示为“新增协议”、“编辑协议”；游戏客户的UI划分为：显示数据、按钮操作、触摸事件等，定义协议时可以按照显示数据的部分设计一个协议ID，按钮操作的设计一个协议ID；操作事件类的可能返回的结果类似，可以合并到一个协议，请求以Type参数区分；如图：

도구막대에 “프로토콜추가”, “프로토콜편집” 표시되있다; 게임클라이언트 UI는 다음과 같이 나뉜다: 데이터표시, 버튼컨트롤, 터치이벤트; 프로토콜정의할때 표시되는 숫자의 부분으로 한 개의 패킷 ID 를 정의할수있다. 버튼콘트롤용으로 한 개의 패킷ID, 조작이벤트유형의 결과리턴을 한 개의 패킷으로 합칠수 있다. 요청에 Type 파라메터를 추가해서 이하 예시그림:

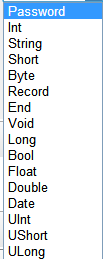


도구막대의 “copy 프로토콜” 기능, 프로토콜 파라메터가 같거나 대부분 같을경우 복사기능을 사용해서 현존하는 프로토콜을 복사하는 기능이다. 현재 프로젝트나 다른 프로젝트로도 복사할수 있다.

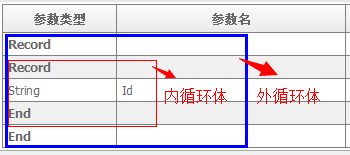


**프로토콜 파라메터 추가**

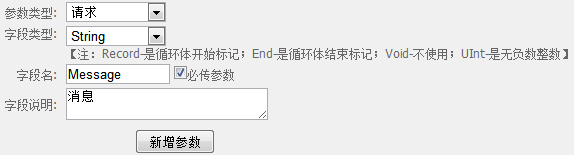
SCUT은 바이너리프로토콜을 지원한다, 아래의 형식만 지원한다



그중 Password 형식은 발송될 때 DES 로 암호화된다. Void 형은 이항목이 임시로 사용되지 않음을 나타낸다. ,Record 와 End 형은 Pair되어야한다. Loop 의 시작과 끝을 표시한다. 이범위안에 파라메터는 순환발송된다 순환체 내에 순환체가 포함될수 있다. 다중행의 리스트 데이터를 클라이언트에게 보내기 위해서 순환체가 자주 사용될것이다.



파라메터형태가 요청일 때 Record,End 형식의 표지를 사용하지않는다. 꼭필요한 파라메터를 설정할수도 있다.(설정하지 않은 파라메터는 생략가능)



커스터마이징 프로토콜 옵션을 사용하면 프로토콜ID설정만필요하다, 다른파라메터를 추가할필요가없다. 여기의 파라메터는 오직 Scut 의 이진수패킷에서 사용하기 위함이다; 이데이터로 클라이언트와 서버의 소스코드를 템플릿을 바탕으로 생성한다.

**단위 테스트**

HTTP 요청 주소는 <http://127.0.0.1/Service.aspx> 같이 설정한다.

소켓 주소 설정은 127.0.0.1:9001처럼 설정된다.

발송을 눌렀을 때 자동으로 요청방식을 판단한다.

요청한 파라메터는 요청리스트에 설정된 기본값일수 있다.

아니면 테스트할 때 파라메터란에 수동으로 기입한 값일수있다.

커스텀 프로토콜은 프로토콜 테스트 기능을 사용할수 없다.

이하참고그림:



주：

Msgid는 단일클라이언트의 매번요청의 유니크한 번호이다，일련번호는자동으로증가하는방식이다, 서버는 상응하는 요청의 MsgID 를 응답에 담아서 돌려준다.；Uid는 유저의 유니크일련번호이다，로그인성공후 서버가 발급해준다, 미로그인시 0을넣도록한다, Sid 는 유저의 세션ID 이다, 유저가 연결에 성공한후 한 개의 GameSession 오브젝트를 생성할수있다. 서버가 Sid 를 내려준다,연결이 끊어질경우 SID 를통해 GameSession 의 부할시키는 메커니즘을 구현할수 있다.

# 참고자료

Scut 공식개발문서와 토론

정식웹사이트：http://www.scutgame.com

Github소스코드창고：https://github.com/scutgame/scut

OSChina소스코드창고：https://git.oschina.net/scutgame/Scut

블로그주소：http://www.cnblogs.com/scut/

기술교류QQ단체방：138266675