

## [Firefly 引擎][学习笔记一][已完结]带用户验证的聊天室二次...

### 前言:

早在群里看到大鸡蛋分享他们团队的 Firefly 引擎,但一直没有时间去仔细看看,恰好最近需要开发一个棋牌 Game,朋友推荐了很多引擎,参考了众多引擎后,觉得 Firefly 是一个轻量级易扩展的引擎(好吧,其实就是我懒,懒得去看成熟引擎的手册),项目也没太多时间让我去熟悉大型引擎,于是决定了用 Firefly。(很多人就问了:开源棋牌类 game 这么多,为什么非要自己开发一个,因为我懒,懒得去逐行读代码,我自己有成熟的开发方案及发布流程,为了以后的易维护易扩展原则,所以重新开发,在这里多嘴一句,不是所有的东西都是适合你的,最适合你的,是你自己的东西,但这不成为你不学习别人的优秀作品的理由。)

### 备注:

此文章仅作为本人记录以及其他群友参考所用

好了,言归正传。

## 一、系统环境以及开发者技能简介

### 系统环境:

Windows 7 32bit

Python 2.7.5

Mysql 5.1.28

Memcache for windows (没有官方版,只有个人版,不作为项目需要,只做开发没问题)

### 开发者技能简介:

2 年 Python 开发经验,熟悉 PyQt

## 二、安装过程

此记录略过, windows 下的一些常见错误官方教程里有提过,部分没有提过的错误谷歌也有资料,就不再多嘴

## 三、下载 Demo 并进行扩展

注:这个段落因为是持续性的,所以占楼编辑=====

1、下载教程中的开发 Demo 压缩包(含服务器端及 Socket 客户端)

2、配置 Config.json

3、引用&扩展 Config 配置

因为没有看到官方给出的全局配置的引用资料,所以查看了项目下的代码,发现其引入规则是读取 json 文件,并 load 之,所以在 APP 的 server.py 下加入以下代码:

```
1. import json
```

```
2. _config = json.load('../config.json','r'))
```

```
#然后采用_config.get 方法取回配置 dict 对象
```

复制代码

...

4、自定义模块封装以及学习说明

看了一下关于 memcache 和 DBUtils 数据库连接池的 Demo,没有封装好的数据库对象,那么只能自己封装一个数据库对象,已于 18 日晚简易封装完毕

19 日上图:

```

1  #-*- coding: utf8 -*-
2  __framework__ = 'Firefly v1.2.3'
3  __author__ = '一瞬间的错觉(Firefly)'
4  __secondary_author__ = 'Aeolus(QQ:251920948)'
5  __date__ = "$Date: 2013/11/17 $"
6  __license__ = "Python"
7
8  from firefly.server.globalobject import netserviceHandle
9  from firefly.server.globalobject import GlobalObject
10 from datetime import *
11 from firefly.dbentrust.dbpool import dbpool
12
13
14 import json,sys,os,time, hashlib
15
16 class ConnectionMysql:
17     '''连接数据库的类, 这个模块你可以自己扩展, 或是增加其他功能, 你懂的, 本想扩展更加能完善的模块, 但考虑到这是学习,
18     为了继续走通以后流程, 所以简单封装了一下, 在以后的笔记中, 逐步完善该模块'''
19     def __init__(self):
20         self.dbConfig = json.load(open('config.json', \
21             'r')).get('db')
22         dbpool.initPool(host = self.dbConfig['host'], \
23             user = self.dbConfig['user'], \
24             passwd = self.dbConfig['passwd'], \
25             port = self.dbConfig['port'], \
26             db = self.dbConfig['db'], \
27             char = self.dbConfig['charset'])
28         self.conn = dbpool.connection()
29         self.cursor = self.conn.cursor()
30
31     def getOne(self, sqlStr):
32         '''获取一条数据'''
33         try:
34             self.cursor.execute(sqlStr)
35             result = self.cursor.fetchone()
36             return result
37         except Exception, e:
38             print '[ ERROR ]:',e
39             return False
40
41     def getList(self, sqlStr):
42         '''获取多条数据'''
43         try:
44             self.cursor.execute(sqlStr)
45             result = self.cursor.fetchall()
46             return result
47         except Exception, e:
48             print '[ ERROR ]:',e
49             return False
50
51     def query(self, sqlStr):
52         '''insert和update和delete操作'''
53         if 'insert into' not in sqlStr.lower() and 'update' not in sqlStr.lower() and 'delete from' not in sqlStr.lower():
54             return False
55         try:
56             self.cursor.execute(sqlStr)
57         except Exception, e:
58             print '[ ERROR ]:',e
59             return False
60         if 'insert' in sqlStr.lower():
61             self.cursor.execute("SELECT @@IDENTITY AS id")
62             result = cur.fetchone()
63             return result[0]
64         return True
65
66     def close(self):
67         '''关闭数据库连接, 重构方法'''
68         self.cursor.close()
69         self.conn.close()
70         self.conn = False
71
72 dbObject = ConnectionMysql()
73 authenticationKey = '32921hjijklj1292h2111k20'
74 loginTemplist = []

```

#引入DBUtil在Firefly封装的模块  
#(补充说明: 当然, 你如果有已经封装好的DBUtils模块  
#也可以调用自己的, 比如那种分布式结构)  
#载入一些基础的模块

#载入配置文件

这里改成charset, 因为底层框架改动了。

#获取一条数据结果

#返回结果

#获取多条数据结果

#返回结果

#定义数据库对象全局变量  
#定义用户验证随机串  
#定义登陆用户客户端的缓存List对象

然后查看了引擎里的代码, 找到了断开连接请求的方法, 并应用到创建连接重构的方法中。18日主要是查看引擎的安装与使用文档

查看了客户端源码, 这里提一点, 就是数据封包的那段代码可能很多新手朋友不太理解, 这里涉及到python数据类型, struct的手册里面也有详细说明, 看到不少朋友提问为什么要截取17个字符, 你len一下协议头就知道为什么要截取17个字符了

协议头是哪里？

## 5、关于用户端验证的设计方案

采用Web中常用的Cookie验证(你知道Session或Memcache其实是要吃内存的, 当然, memcache的文件缓存例外, 不过没必要)

方案为: md5(userId + userName + 其他参数[如用户组等需要验证的] + SYSUSERKEY) = userKey

====19日晚回去贴上18日代码截图(据说有文字限制)。====

关于Server监听的方法暂不忙更新, 等晚上写完了再贴上来, 免得修改  
截止目前为止: 服务端与客户端通讯以及数据库通讯已可用, 不过有BUG

BUG 是建立连接之后, welcome 文字发送到 Speak\_1000 去了, 这个明晚再调整

需要注意的事: 由于官方给出的 Socket Demo 没有规定数据长度的方法, 所以我参考了一些文档之后, 采用了定结束符的方法, 当然, 如果数据过长的话, 遇到 Socket 阻塞则需要多次取出, 就要重写方法了, 这个在之后的笔记中给出, 目前 Demo 用的是 1024 个字节。

19 日晚更新=====

截止 23:42 为止, 程序客户端和服务端模块开发完毕, 效果如下图:

The screenshot displays three windows from a Windows XP environment:

- Top Left Window (cmd.exe - startmaster.py):** Shows server logs for a multi-threaded application. It includes timestamps, IP addresses, and messages like "Client 0 login out.", "Client 3 login in.", and "call method speak\_1000".
- Bottom Left Window (cmd.exe):** Shows the client's perspective. It displays a login attempt for user 'abc', a successful login message "欢迎登陆, 0<|\_|>0哈哈~", and a subsequent login attempt for user 'cba' which fails with the message "哈哈, 我测试下线你那边显示不.". It also shows a traceback error: "File 'C:\Users\Aeolus\Desktop\client.py', line 138, in <module>: pass".
- Right Window (Database Table Viewer):** Shows a table named 'gm\_login' with columns 'user\_id', 'user\_name', and 'password'. The table contains two rows: (1, 'abc', '123456') and (2, 'cba', '123456'). A red box highlights the table, and a red arrow points to it with a list of instructions: "1.建立数据库: gm\_login; 2.建立数据库中表lo\_login; 3.插入上面的2条数据."

分析程序不足:







未做重复验证处理, 未做密码加密处理 (这两个都很好做, 所以就没做)

登录时, 不能向其余在线人员发送提示消息 (这个的解决办法想单独靠 Scket 就有点难, 或者说我知识还匮乏, 因为当前设计是通过 socket 登陆并验证, 然后转给 server, server 里也有验证规则, 不用每次都查数据库, 想了一下解决方案, 可以在客户端存一个当前在线人员列表的缓存, 服务端接收到其他客户端请求 speak 方法

时，每次都向客户端发送协议请求方 sessionno，如果缓存中存在，则继续执行，不存在则提示上线，这是根据目前的结构的方法，这种设计结构不太合理，因为是二次开发，所以有些东西，也就不想改了，可以实现功能没错，这种弊端就是连接了不一定判断为登陆，必须请求 speak 方法激活才算登陆，这也是我的不足，大家一起学习进步)

### 改动的一些说明：

分离 Login 与聊天 Server 监听，重写验证机制，分离数据库模块将其模块化，增加了 Model 模块，引用方法就是把 model 路径加入 sysPath 然后直接调用

	app	2013/11/19 23:36	文件
	model	2013/11/19 22:31	文件
	tool	2013/11/17 19:26	文件
	appmain.py	2013/11/17 23:56	Pytho
	config.json	2013/11/19 22:31	JSON
	startmaster.py	2013/11/17 19:26	Pytho

本地磁盘 (E:) > gameserver > testgame > chat_rooms > model		
共享 ▾ 刻录 新建文件夹		
名称 ▴	修改日期	类
 __init__.py	2013/11/19 21:05	Py
 __init__.pyc	2013/11/19 22:31	Co
 mysqlObj.py	2013/11/19 21:07	Py
 mysqlObj.pyc	2013/11/19 22:31	Co

代码压缩包下载：

本帖隐藏的内容



[testgame.zip](#) (14.49 KB, 下载次数: 376)

+++++此次笔记就告一段落，之后将进入进阶模式的开发。希望大家和我一起学习一起进步。

### [学习笔记二]预告：

既然 socket 没有问题了，那么我们反思一下 Socket 的适用范围，它的定义应该是一个持久化的长连接，那么我们用在什么地方最合适呢？对，就是用在游戏中的服务器端主动提醒和心跳，那么用户验证和上下线以及聊天，都可以通过心跳和队列的组合方法来请求不同的 Server Socket 来达到目的，没错，也许你猜到了，学习笔记二将开始逐步切入游戏项目，开始设计各个功能模块了。敬请期待

### [Firefly 引擎][学习笔记二][已完结]卡牌游戏开发模型的设计

在此补充一下 Socket 的验证机制：

socket 登陆验证。会采用 session 会话超时的机制

做心跳接口验证

保持一个长连接

也为推送消息提供



=====这段是一个以前同事给我的 JAVA Socket 验证机制的例子=====

socket 通信一般是找不到头这些的，要自定义封装通信消息类

如开源框架 netty,消息进出都有自定义加密和选择性压缩的

socket 不想 http 一样能找到某个方法，他就监听 ip 的某个端口

通过解析消息类中的主题方法，抽象接口实现的

消息接收：

```
@Override
public void messageReceived(ChannelHandlerContext ctx, MessageEvent e) throws Exception {
    Message message = (Message) e.getMessage();
    Channel channel = e.getChannel();
    Connection conn = channel == null ? null : (Connection) channel.getAttachment();

    System.out.println("Server read message : " + message);

    long sessionKey = message.getSessionKey();
    if(conn == null && sessionKey != 0) {
        conn = server.getConnection(sessionKey);
    }

    if(message.getSubject() != Message.Subject.AUTHORIZATION &&
        message.getSubject() != Message.Subject.REGISTER &&
        (sessionKey == 0 || conn == null)) {
        Message result = message.newResult();
        result.setPayload(new Registered());
        result.setResult(Message.Result.UNAUTHORIZED); // 没权限
        if(channel != null && channel.isConnected()) {
            try {
                // channel.write(result); //这个后面必须开启
            } catch (Exception e2) {
            }
        }
        // return; //这个后面必须开启
    }

    Message result = server.process(message);
```

其他图就上不了，主要是一个概念的普及

前言：

如果你有仔细分析学习笔记一中的代码，那么相信你对 python 以及 Firefly 已经有了一个大致认识，那么剩余的不认识的也就没关系了，官方提供了开发手册，所以就不逐步深入出笔记了，该有的，会有人发的。

开始准备工作：

【目标】

现在，我们的目标是一个卡牌游戏，我选择了传统的斗地主来进行开发。

【开发前的准备】

任何一款游戏都会有策划书的，策划书我们就免了，因为斗地主谁都会，而且也有前辈的开源作品，所以我们的准备工作是，清楚斗地主的游戏规则，和程序的实现方案。

于是，翻了翻互联网资料，找到一个参考资料：

<http://blog.csdn.net/cq361106306/article/details/7855582>

虽然是 JAVA 版的，但是原理都是一样的，函数也差不多一样的，不一样的只是内置函数的写法和语法规范

而已，要看懂肯定是没有问题的

### 【服务器架构】

首先，我们需要一个 LoginServer，这是毋庸置疑的事情

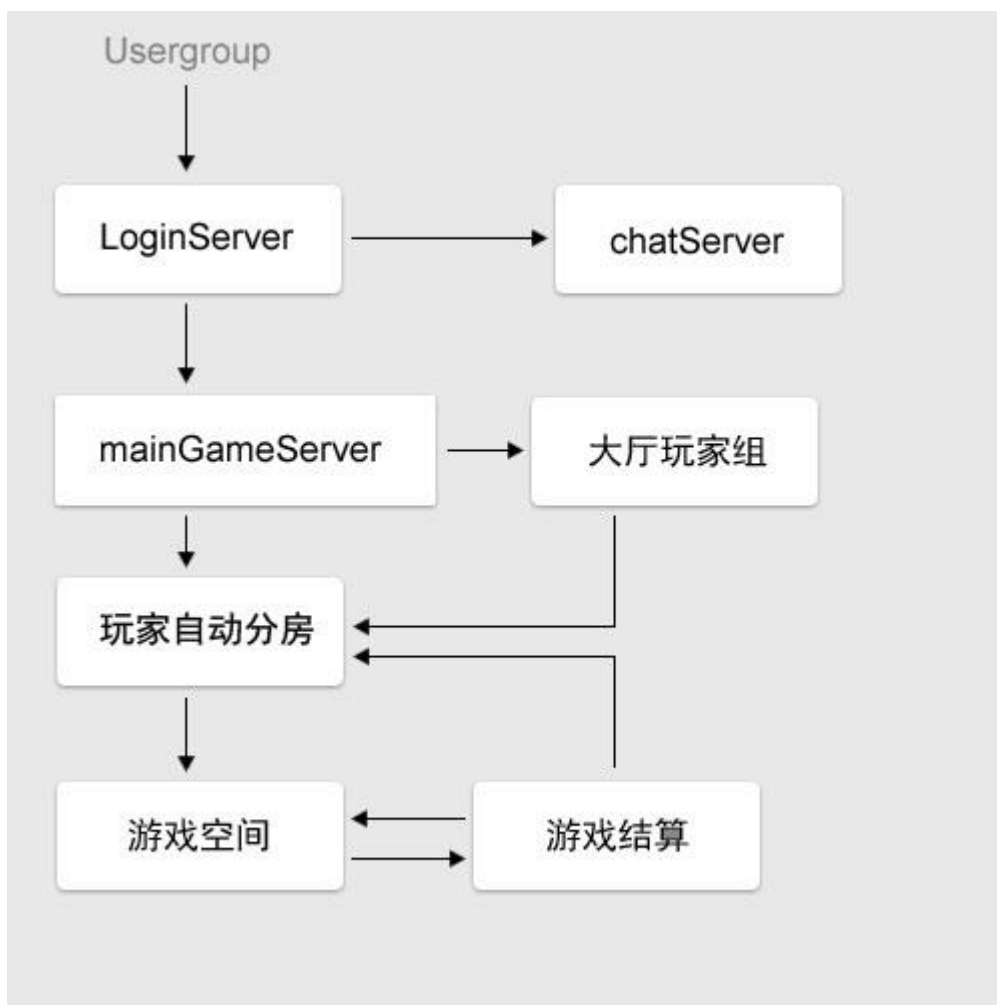
然后，我们需要一个游戏主服务器 mainGameServer

（也许，你会需要一个排行服务器）

最后，我们则需要一个聊天服务器 chatServer（既然是网游，没有聊天系统不等于单机么？）

本身这个游戏不复杂，所以我们就这样设定吧。

### 【服务器架构图】



题外话：不知道大家有没有做过私服，出名的如传奇，诛仙，QQ 西游私服。里面的结构大概也是如此，不说它到底是有用还是无用，存在即是合理。这点，在我还是小白的时候就深以为然，以前老听人家说 GS 报错，GS 未启动，感觉好牛 X 好高端的样子，后来渐渐明白，其实就是 GameServer。😏

好了，言归正传，既然架构图有了，游戏方案也有了，那么下一步就是数据建模了，数据结构一定要做好，因为这跟你服务器的性能和扩展息息相关。

在这里我用的工具是 PowerDesigner 这款软件进行建模，方便又实用，易管理，易查询👍

首先我们确定几个核心的表

User

-----userMain 用户主表，存放用户账号，密码，余额等重要信息 表类型：InnoDB，这么重要的表，你肯定要支持事务回滚来着。

-----userBase 用户基础表, 存放一些只做查询用的字段, 如头像, 个人简介  
Log

-----loginLog 用户登陆日志

-----actionLog 用户操作日志

-----rfloat 用户资金流水

-----gameLog 游戏记录日志

Main //暂时只有两个, 因为业务逻辑暂时不忙管, 只管游戏

-----sysMsg 系统消息表

----- userMsg 用户消息表//暂时弃用, 因为还不涉及到用户短信, 至于为什么要和系统消息分开, 是因为消息表数据庞大

=====持续更新=====

度过了一个愉快的周末, 继续更新

贴上数据模型的代码以及数据库模型文件

用户主表		
user_id	int(11)	<pk>
用户账号	varchar(32)	
用户密码	varchar(64)	
密码随机码	varchar(8)	
用户余额	int(11)	
用户资金key	varchar(64)	
是否锁定	tinyint(1)	

用户附表		
user_id	int(11)	<pk>
用户昵称	varchar(64)	
邮箱	varchar(256)	
手机	varchar(16)	
QQ	varchar(16)	
密码提示问题1	varchar(256)	
回答1	varchar(256)	
密码提示问题2	varchar(256)	
回答2	varchar(256)	
证件号	varchar(20)	
证件图片冗余	text	
注册时间	int(11)	
注册时间冗余	date	
注册IP	varchar(15)	
最近登录时间	int(11)	
最近登录时间冗余	date	
最近登录IP	varchar(15)	

系统消息表		
sysmsg_id	int(11)	<pk>
from_userid	int(11)	
to_userid	int(11)	
消息类型	tinyint(1)	
标题	varchar(256)	
内容	text	
阅读状态	tinyint(1)	
发送时间冗余	date	
阅读时间冗余	date	

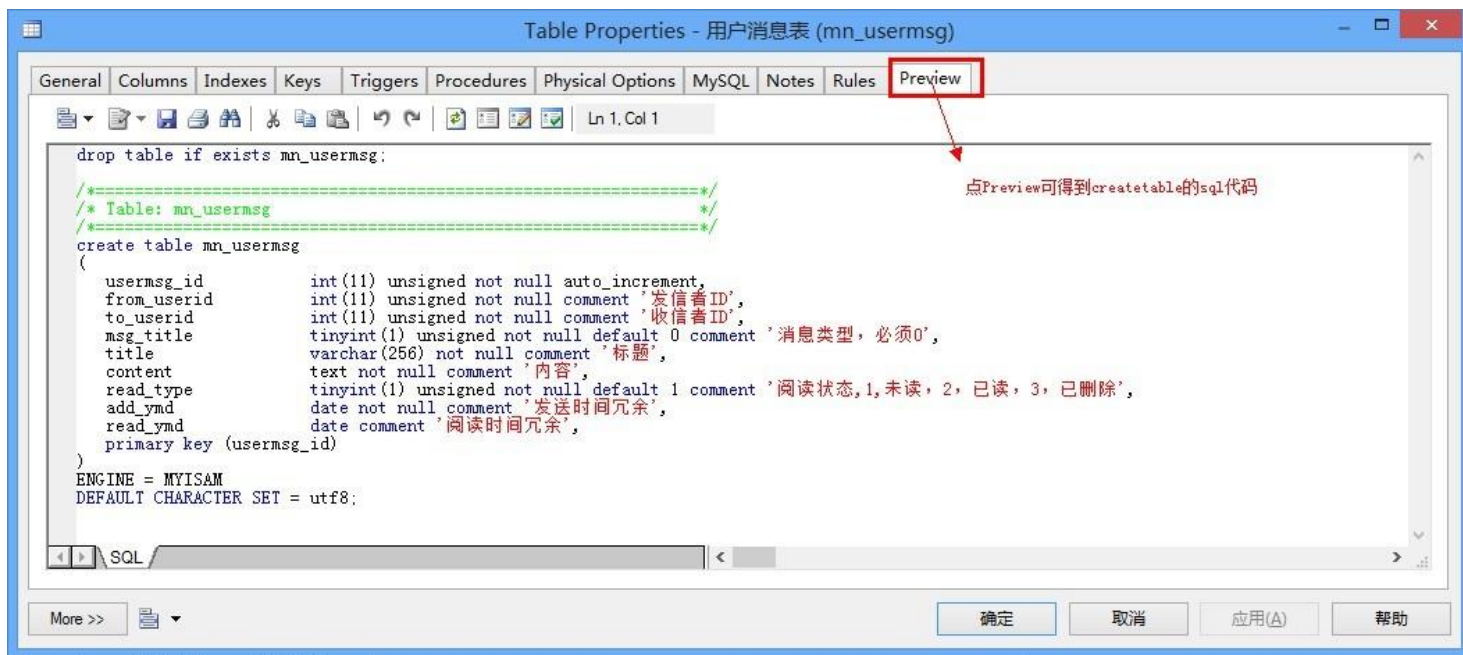
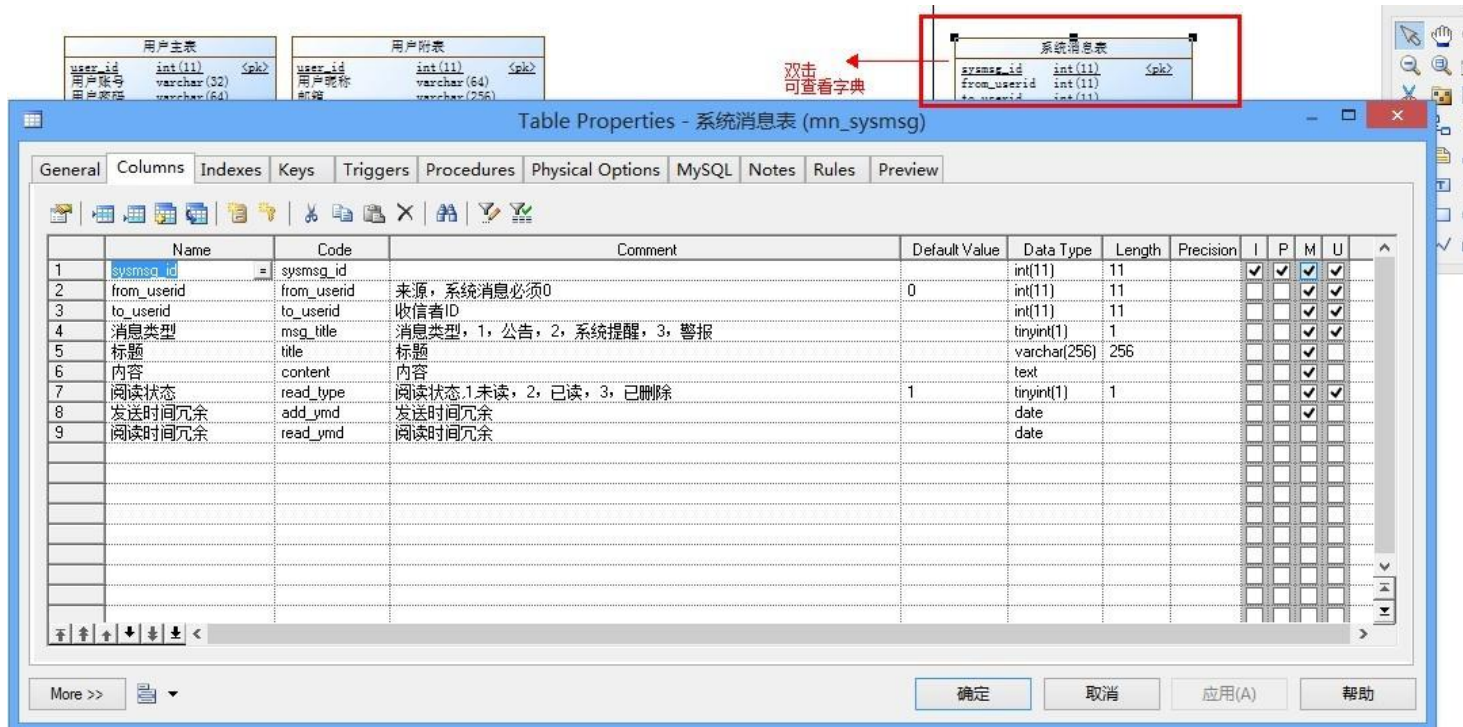
用户消息表		
usermsg_id	int(11)	<pk>
from_userid	int(11)	
to_userid	int(11)	
消息类型	tinyint(1)	
标题	varchar(256)	
内容	text	
阅读状态	tinyint(1)	
发送时间冗余	date	
阅读时间冗余	date	

用户登陆日志		
userlogin_id	int(11)	<pk>
user_id	int(11)	
登陆时间	int(11)	
登陆时间冗余	date	
登陆IP	varchar(15)	
登陆状态	tinyint(1)	
失败原因	varchar(256)	

用户操作日志		
useraction_id	int(11)	<pk>
user_id	int(11)	
操作的对象	varchar(256)	
修改的主键值	int(11)	
操作说明	varchar(256)	

用户资金流水		
rfloat_id	int(11)	<pk>
user_id	int(11)	
资金状态	tinyint(1)	
操作类型	tinyint(1)	
操作的金额	int(11)	
操作前的金额	int(11)	
操作时间	int(11)	
操作时间冗余	date	
操作说明	varchar(256)	

游戏记录日志		
gamelog_id	int(11)	<pk>
游戏参与者ID集合	varchar(35)	
winner ID集合	varchar(23)	
loser ID集合	varchar(23)	
倍数	int(11)	
底价	int(11)	
游戏时间	int(11)	
游戏时间冗余	date	



本帖隐藏的内容



[Database.zip](#) (13.85 KB, 下载次数: 97)

库建设完毕, 现在开始一些基本模块的封装, 在这里说一点, 不是所有的模块都需要封装, 看需求和维护而定, 我个人需要封装是因为我封装后能让我开发效率提高, 每个人情况不一样, 底子也不一样, 根据自身情况来。

在这里我把每个模块及其包含的内容列出来 (持续更新):



模块名	方法名	方法说明
用户基础模块 userModel	userLogin	用于用户登陆时的方法
	regUser	注册用户的方法
	loginCheck	检查用户登陆状态的方法
	heartCheck	用户心跳检测的方法
	fundsCheck	检查资金是否异常的方法
	loginUpdate	登陆时更新用户登陆记录并写日志的方法
	sendSysMsg	发送系统消息的方法
	sendUserMsg	发送用户消息的方法
	readSysMsg	读取系统消息并更新状态的方法
	readUserMsg	读取用户消息并更新状态的方法
	userTalk	用户在游戏中发言的方法 (这里我考虑是像 QQ 斗地主一样固定发言防止作弊，还是不限用户发言，个人觉得防作弊好一点)
日志模块 logModel	sysLog	系统日志的方法(保存 log 日志的方法)
	userLoginLog	用户写登陆日志的方法
	userActionLog	用户写操作日志的方法
	gameResultLog	游戏结果日志的方法
	rfloatLog	写资金流水的方法
游戏主模块 gameMainModel	showUserCount	在线用户统计的方法
	pushMessage	推送消息的方法
	joinGameQueue	加入游戏队列的方法
	gameMatching	匹配玩家并生成游戏空间,随机用户顺序的方法
	shufflingLicensing	洗牌发牌的方法，并整理排序
	grabLandlord	抢地主的方法
	landlordCards	地主牌分发及公示的方法
	showCards	出牌的方法
	doubledFunds	炸弹倍数翻倍计算的方法
	cardAlarm	报警的方法
	settleAccounts	结算并将用户清除队列的方法
系统模块 sysModel	mysqlObject	数据库模块，具体方法不再罗列，这里采取封装主从库
	memcacheEx	memcached 封装的模块，用于分布式集群

目前从代码上看，觉得没用使用memcache

好了，关于学习笔记二卡牌游戏开发模型的设计就告一段落了，学习笔记本三将开始对各个模块进行逐个封装。现在模型有了，开发思路就越发的清晰，当模块封装完毕之后，就是服务器端的开发了，我们一步步走，不一口气

吃成一个胖子，所以，学习笔记三将持续很长一段时间逐个更新，因为毕竟我只有晚上才有时间去鼓捣，所以耐心等待吧，我没有已有代码参考，不能复制粘贴，所以速度会稍微慢一点

另外，诚请一枚 AS 攻城狮帮我完成笔记，有意思的圣兽请联系我 QQ：251920948，跪谢🙏

[Firefly 引擎][学习笔记三][已完结]所需模块封装

学习笔记三导读：

笔记三主要就是各个模块的封装了，这里贴出各个模块一览表，封装完毕我就更新一个状态，并且补上模块说明

模块名	方法名	方法说明	封装状态
用户基础模块 userModel	userLogin	用于用户登陆时的方法	已封装
	regUser	注册用户的方法	已封装
	heartCheck	用户更新心跳时间的方法	已封装
	heartCheck	用户心跳检测的方法	已封装
	fundsCheck	检查资金是否异常的方法	暂不封装
	loginUpdate	登陆时更新用户登陆记录并写日志的方法	暂不封装
	sysMsg	发送系统消息的方法	已封装
	userTalk	用户在游戏中发言的方法 （这里我考虑是像 QQ 斗地主一样固定发言防止作弊，还是不限制用户发言，个人觉得防作弊好一点）	已封装
日志模块 logModel	sysLog	系统日志的方法（保存 log 日志的方法）	暂不封装
	userLoginLog	用户写登陆日志的方法	暂不封装
	userActionLog	用户写操作日志的方法	暂不封装
	gameResultLog	游戏结果日志的方法	暂不封装
	rfloatLog	写资金流水的方法	暂不封装
游戏主模块 gameMainModel	showUserCount	在线用户统计的方法	已封装
	pushMessage	推送消息的方法	取消封装
	joinGameQueue	加入游戏队列的方法	已封装
	gameMatching	匹配玩家并生成游戏空间,随机用户顺序的方法	已封装
	shufflingLicensing	洗牌发牌的方法，并整理排序	已封装
	grabLandlord	抢地主的方法	未封装
	landlordCards	地主牌分发及公示的方法	未封装
	showCards	出牌的方法	未封装
	doubledFunds	炸弹倍数翻倍计算的方法	未封装
	cardAlarm	报警的方法	未封装

	settleAccounts	结算并将用户清除队列的方法	未封装
系统模块	mysqlObject	数据库模块，具体方法不再罗列，这里采取封装主从库	已封装
sysModel	memcacheEx	memcached 封装的模块，用于分布式集群	已封装

每个模块中都有调用 Demo，可直接使用

## 更新记录：

后续更新

=====2013-12-12=====

- 1、封装完成洗牌发牌的方法
- 2、将所有储存用户数据的指针转储 memcached

=====2013-12-11=====

- 1、封装用户登录验证的方法并确定 socket 验证方案
- 2、封装心跳检查等方法
- 3、封装用户发言和系统发言方法
- 3、封装在线统计方法
- 4、封装加入队列方法
- 5、封装随机匹配玩家方法
- 6、封装扑克牌生成方法
- 7、封装洗牌的方法

先放出 userModel

=====2013-12-10=====

- 1、由于封装的那个数据库操作类及其不方便使用，而且功能不强大，我想，要做就做好，所以，狠心重新封装 mysqlDB 类，由于懒，所以参考 <http://my.oschina.net/zhongguanghu/blog/32422> 作者进行二次封装，新增错误代码及错误提示方法，新增多库支持，新增事物多库支持，并贴上示例代码：

```

1.      #s = MysqlObject()

2.      #print s.getAll('us', 'select * from us_user')

3.      #print s.getOne('us', 'select * from us_user')

4.      #print s.getMany('us', 'select * from us_user',4)

5.      #print s.insertOne('us', "insert into us_user (user_name, user_pass, pass_rand, balance,
funds_key, is_lock) values (%s, %s, %s, %s, %s, %s)", ['ddd33',
'1s','2sa',5,'3df',1],1)                                #参数 4 默认 1，返回新增的主键，为 0 时返回
修改的行数

6.      #如果是 innoDB 请 commit()

7.      #s.commit('us')

8.      #print s.getErrorMsg().decode('UTF-8').encode('gb2312')

9.      #print s.update('us', 'update us_user set user_name=%s where user_id=%s',['dddd34', 1])

10.     #s.commit('us')

11.     #print s.delete('us', 'delete from us_user where user_id=%s',[15])

12.     #s.commit('us')

```

```

13.         '''开启事务,涉及到几个库就要开启几个前缀'''
14.         #s.begin('us')
15.         #s.begin('lo')
16.         '''执行代码操作,用 try 执行, 捕获异常, 如果执行数据库返回值为 False, 那么就创建异常'''
17.         #.....
18.         #try:
19.         #         .....如果出错了, 可以自定义错误
20.         #         raise Exception('err')  #抛出异常
21.         #         .....如果没有出现异常, 则提交
22.         #         s.commit('us')
23.         #         s.commit('lo')
24.         #         return True                                     #根据需要附上返回值
25.         #except Exception, e:
26.         #         .....返回自定义错误
27.         #         s.rollback('us')                                #回滚
28.         #         s.rollback('lo')

```

复制代码

2、由于大鸡蛋没封装 cursorclass、autocommit 等参数和方法,所以我没办法取返回数据的 key 值以及设置事务模式,很是无奈,好了,话不多说,先贴上更新的 sysModel,此次更新后, userModel 貌似就不能用了,必须改,好在我没有上传,哈哈

3、重新封装用户注册类,其实我有 PHP 版的注册登录验证机制,但是为了学习嘛,就在这里封装了。

=====2013-12-3=====

申明:没有在日记中贴太多的代码,是因为:

第一、代码太多,贴不过来,压缩包都有,无需画蛇添足

第二、上面有各模块的命名及功能说明,对照文件中的各个模块可以很清晰的看到代码的设计思路以及流程走向,这些,说起来太麻烦,部分代码也体现不出来,索性要么初学者直接调用,要么进阶者逐行读代码理解,甚至可以自己修改封装。也算设置一个门槛,因为在我看来,这些都是摸索路程,我给你们提供一个方向,你们自行去走适合自己的路。同时,这也是我自己参考的笔记,所以,就不贴太多代码了,只是一些需要说明的,我会贴出来

=====2013-12-2=====

因前几日事情繁多,故暂停更新,将于本日起重新更新

1、修正 sysModel 中的部分导致异常的 BUG

2、修改 query 方法抛出异常时的返回值为-1 (原为 False,由于外键返回的 IDENTITY 为 0,会与 False 冲突,故如此改动)

3、新增事务模拟函数,支持 InnoDB, MyISAM

4、新增事务回滚函数 uncommit() #目前暂只支持 insert 回滚,后续加上

sysModel 暂时存在问题,主要是事务模块,故暂不同步文件压缩包到本贴

5、完成用户注册模块

=====2013-11-26=====

1、重新封装 MemcacheEx 模块,集成了 memcache for firefly 自带的功能,如 get\_multi,并新增 hostname 参数

2、合并 MemcacheEx 与 MysqlObject 为 sysModel 模块

3、MysqlObject 支持多库,config.json 增加多库配置

4、创建数据库表,上传数据库备份的 SQL 文件

5、MysqlObject 扩展了方法调用参数,分离 sql 语句与用户参数组合成 tuple,用 execute 占位符的方式避免 sql 注入



由于我时间不多，所以没从 webpy 的数据库层里面抠代码出来自己改个 sqlbuilder，  
这里感谢 **Lany 大神** 的指导

```
1. #mysql
2. obj = MysqlObject()
3. print obj.getOne('us', ('select * from us_user', []))          #无参数情况下
4. print obj.getOne('us',('select * from us_user where user_name=?', ['ddd33']))
5. #有参数情况下，这里用?还是%s 根据你 mysqldb 默认格式来定，我的默认格式是 format，所以我用%s
```

复制代码

...

6、更新依赖关系，userModel 等其他模块依赖 sysModel

=====2013-11-25=====

1、mysqlObject 封装在学习笔记一基础上分离长连接模式，使 mysql 连接方式分离为主从库连接模式

2、memcacheEx 封装原有 memcache 并加入分布式集群的调用方式

附件已上传

本帖隐藏的内容



[dataSql.zip](#) (1.91 KB, 下载次数: 140)



[config.zip](#) (1.18 KB, 下载次数: 122)

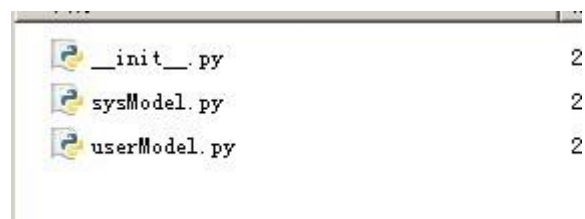
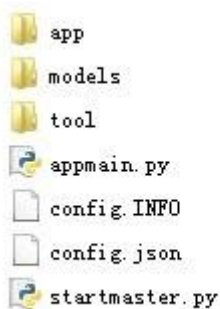


[sysModel.zip](#) (3.2 KB, 下载次数:

120)



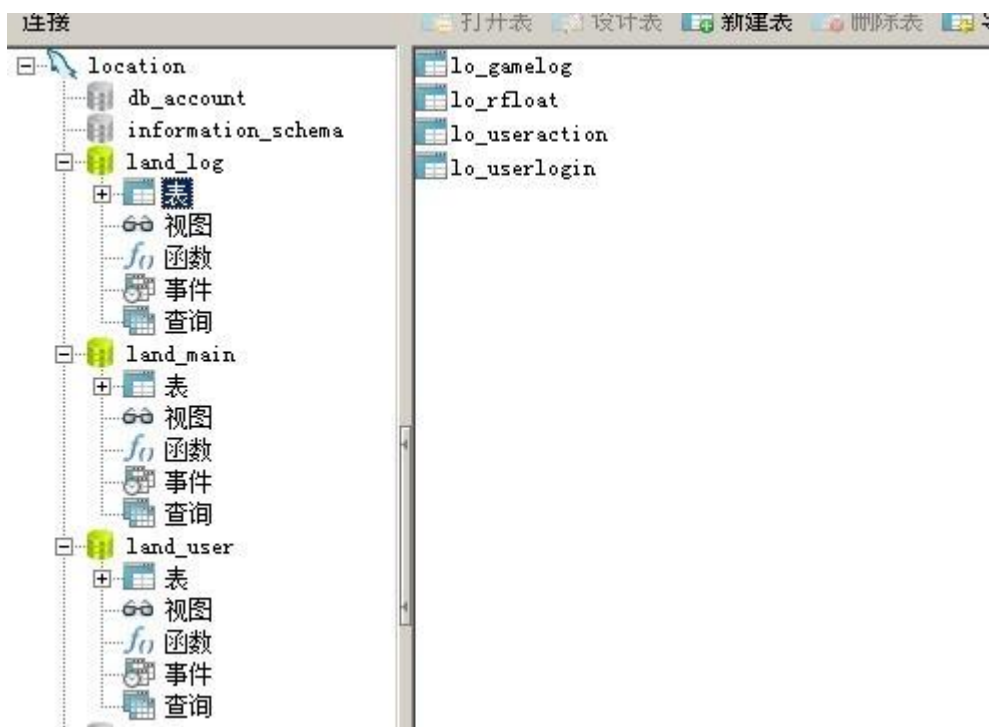
[userModel.zip](#) (2.32 KB, 下载次数: 126)



```

{
  "master":{"rootport":9999,"webport":9998},
  "servers":{
    "testserver":{"netport":1000,"rootport":20001,"name":"gate","app":"app.apptest"}
  },
  "db":{
    "us":{
      "main":{
        //user库数据库配置
        //主数据库，数据库用户最好别用root，你懂的，这里没分多库自动分配，代码可以自己写
        "host":"127.0.0.1",
        "user":"root",
        "passwd":"123456",
        "port":3306,
        "db":"land_user",
        "charset":"utf8"
      },
      "query":{
        //从数据库
        "host":"127.0.0.1",
        "user":"root",
        "passwd":"123456",
        "port":3306,
        "db":"land_user",
        "charset":"utf8"
      }
    },
    "lo":{
      //log库数据库配置
      "main":{
        "host":"127.0.0.1",
        "user":"root",
        "passwd":"123456",
        "port":3306,
        "db":"land_log",
        "charset":"utf8"
      },
      "query":{
        "host":"127.0.0.1",
        "user":"root",
        "passwd":"123456",
        "port":3306,
        "db":"land_log",
        "charset":"utf8"
      }
    },
    "mn":{
      //main库数据库配置
      "main":{

```



```
E:\gameserver\Landlords>E:\gameserver\Landlords\models\l.py
<<1L, 'dddd34', 'BB140A5AA6EE9B0AAA0C3575A8E91458', 'QE47wCK1', 0L, '6B0DC0CC141D80839CDCB9B33F0E9920', 1>, <2L, 'ddd33', '1s', '2sa', 5L, '3df', 1>, <14L, 'ddd33', '1s', '2sa', 5L, '3df', 1>, <15L, 'ddd33', '1s', '2sa', 5L, '3df', 1>, <16L, 'ddd33', '1s', '2sa', 5L, '3df', 1>, <17L, 'ddd33', '1s', '2sa', 5L, '3df', 1>
>
<1L, 'dddd34', 'BB140A5AA6EE9B0AAA0C3575A8E91458', 'QE47wCK1', 0L, '6B0DC0CC141D80839CDCB9B33F0E9920', 1>
<<1L, 'dddd34', 'BB140A5AA6EE9B0AAA0C3575A8E91458', 'QE47wCK1', 0L, '6B0DC0CC141D80839CDCB9B33F0E9920', 1>, <2L, 'ddd33', '1s', '2sa', 5L, '3df', 1>, <14L, 'ddd33', '1s', '2sa', 5L, '3df', 1>, <15L, 'ddd33', '1s', '2sa', 5L, '3df', 1>>
0
0
1
```

[本主题由 Cissy 于 2016-1-8 17:31 删除回复](#)

本帖最后由 ddd33 于 2013-12-27 11:02 编辑

字节已满

=====13.12.19=====

1、封装了一个 json 文件读写类到 sysModel 中，用于一些文件数据的保存，如游戏房间数据

2、开始编写游戏主逻辑

=====13.12.27=====

提供牌型计算的函数：

```
1. def checkPukeType(pukeList):
2.     '''牌型判断'''
3.     #开始进行判断
4.     pukeLen = len(pukeList)
5.     #首先判断牌型是否合法
6.     for x in pukeList:
7.         if pukeData.has_key(x)==False:
8.             return False
9.     #再判断牌组中是否有重复
10.    pukeLenCheck = sorted(set(pukeList),key=pukeList.index)
11.    if len(pukeLenCheck)!=pukeLen:
12.        return False
13.    #再判断牌组中是否存在已出的牌，即玩家手中不存在的牌，这里的判断放到出牌逻辑，该函数只做牌型判断
14.
15.    #定义牌型
16.    c = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,0]
17.    countList = countValue(pukeList)
18.    if countList==False:
19.        return False
```

```
20.     #定义连续字符串
21.     isLian = '123456789101112'
22.     if pukeLen<5:
23.         #先判断小于 5 的
24.         if pukeLen==1:
25.             return c[0]#单
26.         if pukeLen==2 and len(countList)==1:
27.             return c[1]#对
28.         if pukeLen==2 and pukeData[pukeList[0]]>=14 and pukeData[pukeList[1]]>=14:
29.             return c[2]#王炸
30.         if pukeLen==3 and len(countList)==1:
31.             return c[3]#三不带
32.         if pukeLen==4 and len(countList)==2:
33.             return c[4]#三带一
34.         if pukeLen==4 and len(countList)==1:
35.             return c[5]#炸弹
36.     elif pukeLen>=5:
37.         #再判断大于等于 5 的
38.         if pukeLen==len(countList):
39.             d1 = []
40.             for x in countList:
41.                 d1.append(str(x))
42.             d1 = ''.join(d1)
43.             if d1 in isLian:
44.                 return c[6]#顺子
45.         if pukeLen==len(countList)*2 and len(countList)>=3:
46.             isL,d1 = 1,[]
47.             for x in countList:
48.                 #判断是否连续
49.                 d1.append(str(x))
50.                 if countList[x]!=2:
51.                     isL = 0
52.                     break
53.             d1 = ''.join(d1)
54.             if isL==1 and d1 in isLian:
```



```

55.         return c[7]#连对
56.     if pukeLen==5 and len(countList)==2:#四带 1
57.         for x in countList:
58.             if countList[x]==4:
59.                 return c[8]#四带 1
60.     if pukeLen==6:#四带对或四带 2
61.         if len(countList)==2:
62.             for x in countList:
63.                 if countList[x]==4:
64.                     return c[9]#四带对
65.         if len(countList)==3:
66.             for x in countList:
67.                 if countList[x]==4:
68.                     return c[10]#四带 2
69.     if pukeLen%5==0 and len(countList)==(pukeLen/5*2):#三带 2 或飞机 1
70.         if pukeLen==5:
71.             dt,dd = 0,0          #三张#两张
72.             for x in countList:
73.                 if countList[x]==3:
74.                     dt+=1
75.                 if countList[x]==2:
76.                     dd+=1
77.             if dt==1 and dd==1:
78.                 return c[11]#三带二
79.         else:#大于 5 张的
80.             d1,dt,dd = [],0,0
81.             for x in countList:
82.                 if countList[x]==3:
83.                     d1.append(str(x))
84.                     dt+=1
85.                 if countList[x]==2:
86.                     dd+=1
87.             if dt==pukeLen/5 and dd==pukeLen/5:#第一层判断，判断数量是否正确
88.                 if ''.join(d1) in isLian:#第二层判断，判断是否连

```

```

89.                 return c[12]#飞机 1
90.                 if pukeLen%4==0 and len(countList)==(pukeLen/4*2) and pukeLen>4:
91.                     #飞机二
92.                     dl,dt,ds = [],0,0
93.                     for x in countList:
94.                         if countList[x]==3:
95.                             dl.append(str(x))
96.                             dt+=1
97.                         if countList[x]==1:
98.                             ds+=1
99.                     if dt==pukeLen/4 and ds==pukeLen/4:
100.                        if ''.join(dl) in isLian:#第二层判断，判断是否连续          3-A
101.                            return c[13]#飞机 2,带单的飞机
102.                return c[14]

```

复制代码

## [Firefly 引擎][学习笔记四][已完结]服务器端与客户端的通讯

前言:

学习笔记三是模块封装，这个在持续开发中会不断更新， 因为写出来不一定是正确和最好用的，由于 1000 字限制，后续更新在 22#

测试我就决定直接和客户端连起来测试，更直观一点，当然，这得根据实际情况来决定

由于秀才做手术去了，所以我这边请了一个游戏公司的 AS 程序员来配合我开发游戏

更新记录

=====2013.2.7=====

斗地主已经完成，包括服务端以及客户端通讯，

晚上贴上源码和客户端源码。

=====2013.1.17=====

终于要看到曙光了不容易啊，满满的都是泪，斗地主游戏终于快结束了，大家给我点力气给我点动力🤖

出牌验证逻辑已经写完，现在就剩更新数据和加上定时器验证，以及报警和游戏结束的一些工作了，总算要结束了，满满的都是泪啊!!

客户端已经做到抢地主那部分了，现在好像牌显示有问题，有大小王 P1,P2 就显示不出资源，不过胜利就在前方，不是么？

截至目前，牌的显示问题以解决。居然是数据大小写问题。。。。我定义的是大写的 P1,P2。。。。。

```
def outPuke(pid, puke,userPrefix):
    data = mysqlObj.getOne('mn', 'select '+userPrefix+'p,f_u,s_u,t_u from mn_room where spend
    if data[0]=='':
        return {'s':-3,'m':'该游戏已结束'}
    db_puke = data[0].split(',')
    if len(db_puke)<len(puke):
        return {'s':-4,'m':'数据出现错误, 用户被强制退出登录, 并扣除金钱处罚'}
    for x in range(0,len(db_puke)):
        for y in puke:
            if db_puke[x]==y:
                del db_puke[x]
    up_puke = ','.join(db_puke)#修改牌组信息, 数据库移除
    #数据移到下一个用户
    #判断是否报警或为空
    if len(db_puke)<2!=len(db_puke)!=0:
        #报警
        pass
    if len(db_puke)==0:
        #游戏结束
        pass

def showPuke(pid, puke,pukeData):
    puke = puke.split(',')
    if len(puke)>0:
        puke = list(set(puke))
    for x in puke:
        if x not in pukeData:
            #如果不在牌组中
            return {'s':-1,'m':'非法数据'}
```



=====2013.1.10=====

开始封装一些其他的功能

- 1、离开游戏队列
- 2、用户信息拉取

=====2013.1.9=====

客户端 UI 已经七七八八了, 已发截图我看了

- 1、开始将用户心跳加入清除客户端连接事件，60 秒后清除未心跳的客户端
- 2、编写了加入游戏事件，用户点加入游戏之后，会根据游戏类型，自动进入匹配队列
- 3、自动生成房间功能已完成，实现了自动发牌，地主牌的功能，用消息推送给客户端

#### =====2013.1.7=====

- 1、重构服务器结构，采用事务分离的机构，实现类似 MVC 的结构，具体可参考暗黑等服务端，提高维护效率
- 2、组建定时器服务器，实现动态的数据推送以及客户端操作
- 3、创建游戏队列，构造游戏房间，并向客户端发送消息
- 4、发现并解决了 firefly 异步中初学者常遇见的问题，具体请参考帖子 <http://bbs.9miao.com/thread-45195-1-1.html>，也是我发的
- 5、与客户端完成通讯，数据已走通，客户端发牌逻辑已实现，目前发牌舞台是空白的 flash 舞台，明天客户端开始做 UI
- 6、完善用户登录验证以及缓存服务
- 7、学习笔记三中的大部分非基础模块已重写，改动最大的就是 userModel 了，到时候完成的时候把这些模块贴到这上面来

#### =====2013.12.23=====

非常抱歉啊，ui 都做好了，现在 flash 跨域与服务器通讯有问题，所以一直卡在这里，由于我不清楚 flash 与服务端的 socket 安全策略，所以在恶补知识中。

所以没有更新，服务端包都封的七七八八了，客户端通讯搞不定很是纠结，所以在这里给大家道个歉。

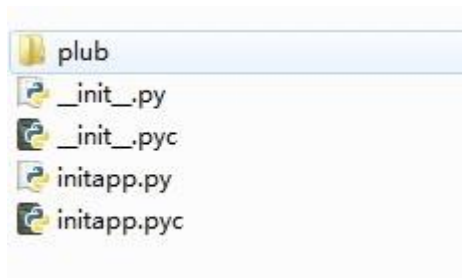
试过了 crossdomain.xml，也试过了 843，也试过了建立连接时直接返回 xml，但结果都是非法数据包，协议头是没有问题的，PB 是可以正常通讯，在 web 中就不行。如果哪位大神知道问题，跪求指点

Socket 安全策略已经搞定，待会会把 843 的代码发送上来，简化版的

开始进行登录验证通讯

gate	2014/1/7 21:01	文件夹	
net	2014/1/7 21:24	文件夹	
timer	2014/1/7 20:50	文件夹	
_init_.py	2013/11/25 19:35	Python File	0 KB
_init_.pyc	2013/12/13 20:28	Compiled Python...	1 KB
cross.py	2014/1/7 20:08	Python File	1 KB
cross.pyc	2014/1/7 21:30	Compiled Python...	1 KB
gateServer.py	2014/1/7 21:01	Python File	1 KB
gateServer.pyc	2014/1/7 21:30	Compiled Python...	1 KB
netServer.py	2014/1/7 21:02	Python File	1 KB
netServer.pyc	2014/1/7 21:30	Compiled Python...	1 KB
timerServer.py	2014/1/7 20:48	Python File	1 KB
timerServer.pyc	2014/1/7 21:30	Compiled Python...	1 KB





名称	修改日期
__init__.py	2014/1/7 21:0
__init__.pyc	2014/1/7 21:2
console.py	2014/1/7 21:1
console.pyc	2014/1/7 21:2
login.py	2014/1/7 21:1
login.pyc	2014/1/7 21:2
methodcallback.py	2014/1/7 21:2
methodcallback.pyc	2014/1/7 21:2
setheart.py	2014/1/7 21:1
setheart.pyc	2014/1/7 21:2

```
ady
2014-01-07 21:30:54+0800 [Broker,client] call method remote_connect on service[s
ingle]
2014-01-07 21:30:54+0800 [Broker,client] Starting factory <twisted.spread.pb.PBC
lientFactory instance at 0x0000000002FF7C88>
2014-01-07 21:30:54+0800 [BilateralBroker,0,127.0.0.1] call method timerconnecti
on_1000 on service[parallel]
2014-01-07 21:30:54+0800 [BilateralBroker,0,127.0.0.1] call method timerconnecti
on_999 on service[parallel]
2014-01-07 21:30:54+0800 [BilateralBroker,0,127.0.0.1] ***** 心跳定时服务器连接
成功 *****
2014-01-07 21:30:54+0800 [BilateralBroker,0,127.0.0.1] ***** 房间定时服务器连接
成功 *****
2014-01-07 21:30:54+0800 [BilateralBroker,0,127.0.0.1] call method timerconnecti
on_998 on service[parallel]
2014-01-07 21:30:54+0800 [BilateralBroker,0,127.0.0.1] call method timerconnecti
on_997 on service[parallel]
2014-01-07 21:30:54+0800 [BilateralBroker,0,127.0.0.1] ***** 抢地主定时服务器连
接成功 *****
2014-01-07 21:30:54+0800 [BilateralBroker,0,127.0.0.1] node [net] takeProxy read
y
2014-01-07 21:30:54+0800 [BilateralBroker,0,127.0.0.1] ***** 游戏定时服务器连接
成功 *****
```

## [学习笔记战果][斗地主]服务端+数据库+AS 客户端

本帖最后由 ddd33 于 2014-2-13 00:49 编辑

服务端版本:

Firefly1.3.1 (暂不支持最新版本)

Python 2.7

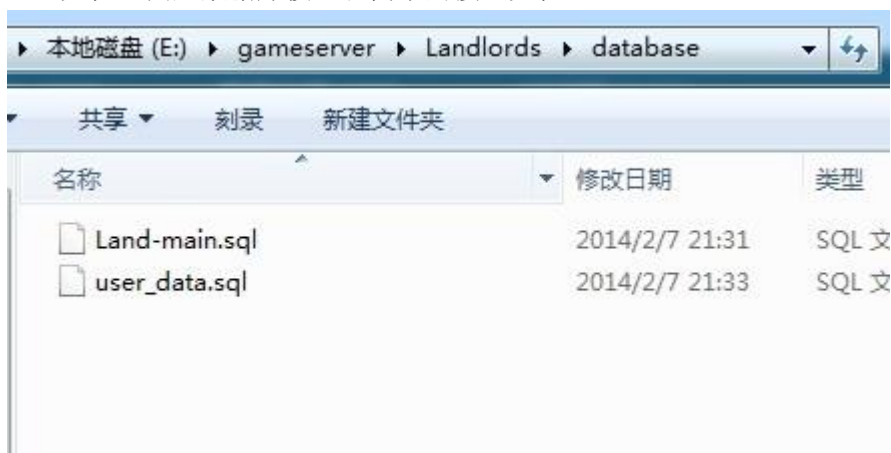
Mysql5.1

客户端是基于 AS 的, 网页版你们懂的, 我不太懂就不过多介绍了

发上来的这个版本是能够正常游戏的, 据客户端说, 结算信息显示没做, 因为他没看到我 API 文档里面有返回结算信息。你自己加上就可以了, 登录验证规则也需要配合自己的规则修改, 至于换皮啊, 程序优化啊, 那就是你们要做的事情了, 我准备用这个做一个产品出来, 里面的资源神马的, 都很简陋, 能正常的玩游戏, 心跳超时或客户端断开自动弹出并扣款。



Doc 文档里面是数据库模型和简单的接口文档



database 里面是数据库

其他的我就不过多介绍了，共两个文件，客户端一个，服务端一个

当然，BUG 肯定会有，就需要你们帮我找了，我也断断续续找了好多，游戏开发哪能没有 BUG 呢，如果找到，请联系我。

重庆-ddd33(Aeolus)(251920948) 9miao2000 人群

才从 WEB 编程转过来，很多思维逻辑都是 web 方面的，程序逻辑可能有很多问题，BUG 也常有，但验证规则做的很严格。

#### BUG 记录:

- 1、客户端回应说结算的时候有时候服务端会报错，原因为数据库中的用户缓存信息没获取到，返回了一个 bool 值，导致部分用户收取不到结算信息
- 2、客户端反应出牌时，对王不能炸 4 个（这里我检查了下源码，做了判断的，晚上回家再和客户端调试调试）
- 3、结算信息 winner 和 loser 的数据与客户端不匹配，（这里我估计是由于服务端报错哪里带来的连锁反应）

#### BUG 更新记录:

更新已知 BUG，修正服务端，服务端版本号更改为 1.0.2

本帖隐藏的内容



[PokerServer 1.0.2.zip](#) (131.54 KB, 下载次数: 325)



[PokerClient.zip](#) (6.34 MB, 下载次数: 509)

[Gfirefly 引擎][学习笔记五]架构模型学习以及迅速游戏开发

由于项目时间性缘故，加上 firefly 初创，资料不完善，故终止了研究。  
时隔接近一年，又是由于项目的需要，开始考虑采用一款服务器引擎来加速开发效率。

最先的想法是，自己写一套类似引擎的框架，方案如下：

方案一：SocketServer StreamRequestHandler

优势：是自动处理监听某一个客户端连接后新开线程的处理业务逻辑就行了

弱势：由于项目需求决定了一个客户端可能需要并发多个业务逻辑，在服务器不支持分布式的情况下，单进程多线程可能存在一定隐患

方案二：greenevent

直接不考虑，不能满足项目需求

方案二：Twisted

优势：单线程，事件驱动

正准备开始写的时候，朋友推荐了一款 LUA&C++的引擎[skynet]，看了一下 github 上面的一些资料 and 介绍，开始考虑，是否用 skynet，正巧，发现大鸡蛋在群里分享 gfirefly，想了一下，这次项目时间相对充裕一些，故而姑且看看 gfirefly，虽然没有开发手册，但好在有开发完毕的产品，就可以节省很多时间来学习如何使用 gfirefly。

进入正题，还是先看看 config.json 来分析服务器配置：

各种端口配置和数据库配置以及 memcache 配置服务先放在一边，memcache 直接放弃，准备使用 redis 或者 MongodB 代替。个人想法是业务逻辑的数据直接存 nosql 或 cache 里面，核心数据再操作 mysql 降低数据库压力。

master 先放一边，因为管理什么的后面再说。

重点：servers，按照黑暗世界的配置是如下节点：

gate：消息分发服务器，根据[gfirefly 深入解读...]一文中的解释，gate 也是其它节点(net 节点等)的根节点，所以需要 rootport 等待其他子节点的连接，我们就将 gate 理解成为消息分发处理服务器，它的作用就是将用户发送的消息或者服务器自身的消息分发到对应的子节点上，用于处理各个不同的业务。

dbfront：这个节点主要是将表映射到 memcache 中（[gfirefly 深入解读...]一文），由于我们不需要 memcache 的，所以略过把，只需要知道是什么功能就好了。

net：net 是 client 端连接的节点，所以 netport 也就是 net 监听的端口，client 将向 netport 这个端口发送数据。重点在 remoteport，所谓的 remoteport 其实就是定义它的父节点，父节点可以有多个，所以是数组。我们看到父节点是 10000 端口，是 gate 监听的端口，所以说 gate 是 net 的父节点（[gfirefly 深入解读...]一文），理解起来很简单，就是 net 是 gate 的子节点，客户端发送数据到 net，net 将数据提交给 gate 父节点然后分发出去

game1：game1 节点并不需要监听其它端口，所以它没有定义自己的 rootport（[gfirefly 深入解读...]一文），他主要是处理游戏主业务逻辑，所以不需要监听 rootport。

admin：不说了，管理节点（略过）。

于是，结合自己业务逻辑的需要，我将其转化成适合自己的图形模型，方便更清晰的熟悉需要的架构

并且，参考暗黑世界，我将 config.json 文件中的 Server 修改，如下：

```
1. "servers":{
2.     "net":{"netport":1000,"name":"gate","remoteport":[{"rootport":20001,"rootname":"gate"
3.     }],"app":"app.NetServer"},
4.     "gate":{"rootport":20001,"name":"gate","app":"app.GateServer"},
5.     "game":{"remoteport":[{"rootport":20001,"rootname":"gate"}],"name":"game","app":"app.
6.     GameServer"},
7.     "room":{"remoteport":[{"rootport":20001,"rootname":"gate"}],"name":"room","app":"app.
8.     RoomServer","db":true}
9. }
```

复制代码

配置完成后，简单的做一个说明：

net 节点的唯一作用，监听客户端的请求， gate 作为分发服务器和登陆验证， game 作为游戏主服务器，进行游戏业务处理， room 为空间(房间，因为项目原因，原来的 Zone 改变为 Room)服务器，作用是为游戏业务提供一个类似计划任务推动游戏进程。

下面来说 Gfirefly 的内部通讯机制（参考暗黑世界和[gfirefly 深入解读...]）。

@netserviceHandle net 节点装饰

@rootserviceHandle root 节点装饰

@remoteserviceHandle 这个装饰是用于连接到 root 上

这 3 个句柄就是整个 gfirefly 的核心句柄

处理顺序是：net->root->childnode

net->root:

1. 有返回值: GlobalObject().remote['root 节点名'].callRemote("方法名",\*args,\*\*kw)
2. 无返回值: GlobalObject().remote['root 节点名'].callRemoteNotForResult(同上)

复制代码

root->childnode:

1. 有返回值: GlobalObject().root.callChild(节点名,\*args,\*\*kw) #eg.: callChild(节点名,命令号,参数)
2. 无返回值: GlobalObject().root.callChildNotForResult(同上)

复制代码



根据以上了解，再参考暗黑世界，我已经明白如何进行游戏的一个快速开发了，在想了一下，在验证用户是否登录时候，需要判断用户是否登录合法，需要写一个 `base` 类来进行验证，再参考了暗黑世界，发现他是构建了一个 `localserviceHandle` 的本地服务句柄(`app\gate\gateservice.py`)，于是我开始找基础类(`app\net\netapp.py`)，在这里面，暗黑构建了一个 `NetCommandService` 类继承了 `CommandService` 传值给 `netservice`，并重构了 `netserviceHandle` 为 `netservice` 注册，后来才发现，这个基础类不存在，所有的验证，写在了每个控制器里面，也就是这一段：

```
1.     player = PlayersManager().getPlayerByID(characterId)
2.     if not player or not player.CheckClient(dynamicId):
3.         return {'result':False,'message':u""}
```

复制代码

获取了一个用户实例，如果用户实例不存在或用户帐号与客户端 ID 不匹配，则直接返回 `False`

总之，知道了几个核心的知识点后，我就可以进行游戏的开发了，暗黑给我的启发很大

OK，继续下一步，我们在登录成功后，需要创建一个用户数据，我参考了暗黑世界，查看了 `gate\core\user` 和 `gate\core\usermanager`，发现他是用的 `Singleton` 设计模式即单例模式，单例模式的资料可见参考文献，仔细的查看了 `usermanager` 的代码，发现完全可以直接拿过来使用，它包含的方法如下：

```
1.  addUser #添加用户
2.  getUserByDynamicId #根据客户端的动态 ID 获取 user 实例
3.  getUserByUsername #根据用户名获取用户信息
4.  getUserByID #根据 ID 获取用户信息
5.  dropUser #处理用户下线
6.  dropUserByID #根据用户 ID 处理用户下线
7.
```

复制代码

用户管理实际上就是创建一个用户单例，将用户数据存储到里面，并绑定客户端 ID，那么如何与数据库通讯呢？我们查看 `gate\core\user`

`user` 核心库里面，我们看到了引用了 `dbuser`，转而看看 `share\dbopear\dbuser.py`，里面就是用户与数据库通讯的方法，而 `usermanager` 里面是需要一个用户类，那么 `user` 就是创建一个用户类 `usermanager` 可以直接使用，但 `user` 类，则需要自己根据自己的需求进行修改，我根据我的需要，对构造函数修改如下：

```
1.
2.     def __init__(self, name,password,dynamicId = -1):
3.         '''
4.         @param id: int 用户的 id
5.         @param name: str 用户的名称
6.         @param password: str 用户的密码
7.         @param money: float 用户的资金
8.         @param dynamicId: str 登录时客户端的动态 ID
9.         @param isEffective: bool 是否是有效的
10.        '''
```

```
11.         self.id = 0
12.         self.name = name
13.         self.password = password
14.         self.money = money
15.         self.dynamicId = dynamicId
16.         self.isEffective = True
17.         self.initUser()
```

复制代码

并对方法进行修改：

```
1.  initUser #初始化用户类
2.  getNickName #获取帐号名
3.  CheckEffective #检测账号是否有效
4.  checkClient #检测客户端 ID 是否匹配
5.  setDynamicId #设置客户端动态 ID
6.  setMoney #更新用户资金
7.  getDynamicId #获取用户动态 ID
8.  disconnectClient #断开连接时的操作
```

复制代码

修改完 user 类后，我们再根据业务需要，去修改 share\dbopear\dbuser 里面的数据库连接方法

OK，根据我的需要，删除了所有关于角色的操作，并新增了用户资金更新的操作，以及独立的密码算法 dbuser 完成

现在更新 gfirefly 的两处问题（结合暗黑 G 版）

一、pushObject 消息推送方法 msg 参数不能直接用 unicode 会报错，原因是 pack 方法粘包的问题，用 json.dumps 转一下就行了

二、gate 节点中的 pushObject 方法由于加入了 root 修饰符，所以不能直接 import，需要自己新定义一个 pushObject 方法，你懂的

目前开发进度：

已参考暗黑做好登录验证，暗黑的设计模式很值得学习，其实就是一个消息转发，在转发的时候判断是否登录，仔细看看暗黑代码就明白了

已封装好 game 服务器和 gamemanager，下一步等待客户端接入的节点转发，这点还不是太清楚，继续看看暗黑

参考文献：

[\[gfirefly 深入解析\]--总体架构及 demo 讲解](#)

[gevent-firefly 版本《暗黑世界》](#)

[《暗黑世界》教程文档\(用于参考其 API 学习\)](#)

[单例模式（Singleton）的 6 种实现](#)

[深入浅出单实例 Singleton 设计模式](#)