# 自我介绍

您好，我叫王必宇，25岁，来自海南。2018年6月毕业于位于四川西南科技大学的信息与计算科学专业，学校是普通一本，在校内我成绩应该还算不错。因为专业偏向数学和计算机理论嘛，所以主要在课上学到的就是数学，C语言，算法，网络，操作系统，数据库这样的基本功。

17年秋招进了北京的完美世界实习，毕业后在原项目组转正工作。在完美的这段时间主要是全栈写业务。这段时间主要是熟悉了C++在大型项目里边怎么用，在实战里写一些简单的，复杂的业务，也看明白了项目服务器用消息驱动的架构，看了服务器使用的MySQL + Redis存储机制。19年下半年因为想要往系统层面发展，加上公司996的制度，所以离开完美。

回到家的几个月学习在Linux系统编程和网络编程的知识。在20年1月入职了海南一家做Windows安全的公司，这家公司要求保密。大概就是在写红客工具，在这段时间主要在做Windows的系统编程。后面在转正的时候，觉得不太合适，因为公司里偏向保密制度，很多事让员工做就得做，没有什么沟通的余地。也就是沟通上存在一些差异，我就和公司协商离职。

然后在同年7月入职了武汉的和信创天，主要业务是云桌面。我主要参与两个项目：云桌面和信息采集项目。我主要的工作还是在写Linux系统编程相关的东西。目前这两个项目基本已经没什么事情了，公司向上提供的方向是研究X11，但是我对写界面这块意向不大，所以想出来了解一下新的机会。

目前应该是更擅长Windows，Linux的系统编程和网络编程，我自己也封装了类似OSProcess，OSPipe，OSFifo，OSTCPSocket和OSUDPSocket这样的类。我用在完美看到的框架，实现了一个线程消息队列和线程池开发框架，主要是在项目中进行模块解耦和避免加锁。用这套框架实现了信息采集项目，还有自己写的两个个人项目，一个是用epoll做网络模块写的远程文件管理客户端+服务器，一个是用libevent+MySQL+Redis写的游戏服务器架子。数据库这块，因为没有实际项目开发，只是写Java的时候学过MySQL和Oracle，Redis是前段时间自学的。

我自己比较热衷热衷于摸索源码级的实现原理。我的基本功应该还算比较扎实，也一直在做系统和网络这块的学习和开发。自己是想要往后端和内核级开发走，所以想要参与Go和C++的服务端开发，Windows，Linux的系统级开发。

说了比较多，谢谢您听完~

# 项目

## 云梦四时歌

### 装备系统

当时设计的是，数据集中在星魂背包和卡牌身上。卡牌身上会有6个装备槽，装备星魂时将数据从背包取出，放到卡牌身上。星魂有复杂的属性，通过读表获取。卡牌通过计算属性函数获得装备后的属性。

### 庭院

为每个玩家设计了一个庭院系统，其服务器机制庭院Routinue里面会存储所有玩家庭院的基本信息，通过玩家ID索引。单个庭院为副本场景，当有玩家访问一个庭院时，从庭院池取出一个庭院进行初始化，最后一个玩家离开后15分钟清理庭院副本。在副本内创建NPC，实现庭院生产、庭院留言的入口。

### 在线答题

在线活动，在每天固定时间触发，验证答案，答对后给予玩家奖励。在帮会Routinue有一个答题模块，心跳检测到某个时刻时，向聊天Routinue发送系统消息表示题目，玩家在帮会频道的发言会被转发到帮会Routinue，答题模块会检测当前答案是否正确，给玩家发奖励。

### 帮会战

在线活动，每周结算，每天会在固定时间触发。流程为：报名->玩家设置防守阵容->下午5点匹配进攻的帮会->下午7点开始，攻击方可以攻打防守方的阵容镜像->活动结束，结算。匹配机制为设置一个防守对手数组，从中随机一个即可（可以重复），每次战斗结束后，返回战斗结果到帮战Routinue计算得分。帮战Routinue主要存储报名的帮会，及其防守据点设置，及其防守人员卡牌镜像。

## 云桌面

### 客户端代码重构

将各个职能的业务提取，封装，提供对外的访问接口，使其多处的调用保持一致。

### 版本兼容

使用策略模式，将有版本差异的部分提取成策略，根据当前版本，调用抽象策略的接口，实现版本与业务解耦。

### 客户端多进程间的通信框架

主客户端会开一个TCP端口，接受各个云应用进程的连接，通过自定义的Json结构完成数据交互。主客户端也会打开一个UDP端口，接受来自其他工具类应用的消息。

### uamhost虚拟通道

虚拟通道是RDP协议中的一种通讯机制，需要单独实现客户端-服务器，是一对一的通信。

uamhost通道服务于云主机端和管理端，负责将管理端的管理指令传输到云主机端执行。我在客户端接收管理端的指令，通过json自定义简单通讯协议，经由通道传输到云主机端执行指令。

### usbip虚拟通道

usbip通道服务于云主机端和客户端，客户端负责将云主机端要求的USB设备绑定，以及获取已经接入的usb设备等功能。客户端实现为虚拟通道+系统服务，系统服务监听一条UDP端口，接收外部传来的消息，执行需要root权限才能运行的规定操作。

## 信息采集

### 软硬件数据采集

通过WMI技术查询一般数据，使用Windows API补充WMI缺失的数据。

### 采集端代码重构

使用线程消息队列框架，将采集模块和网络模块分离：采集模块调用较为耗时，且Com在多线程下，会偶现异常，网络模块RocketMQ API调用会有阻塞情况。两者通过消息通信，互不干涉。实现了功能解耦。

### rocketMQ客户端源码中的异常

在测试中，发现客户端偶现崩溃，经验证是RocketMQ.dll的报错，于是进源码调试，找到了诸如返回栈string变量的c\_str()，变量判断宏为空等问题。处理后客户端正常。

### 热更新

采集客户端被系统服务通过CreateProcessAsUser唤起。服务在唤起采集端前，会向服务器询问最新版本数据。如果发现版本不一致，则会更新版本后再启动客户端。

## 系统函数和数据结构封装

### OSProcess,OSPipe,OSFifo,OSThread,FileManager

对Linux，Windows上的系统函数提取其功能相同的函数，通过预处理指令，对不同平台提供不同编译代码。

### 线程消息队列框架

目前实现为，为每个模块单开一条系统线程，并给每个线程绑定一条消息队列。每次线程心跳时，会将消息拷贝出来，逐条处理。保证了多线程间数据同步性。外部可以通过调用ThreadManager的函数，将消息丢到模块线程的消息队列。

未来打算将每个模块绑定的系统线程取消，将模块的初始化、心跳函数扔到线程池运行，执行完一次以后才能调用下一次。避免占用系统线程资源。

消息队列这块，今天看到一个无锁队列的实现，打算研究一下用上去~

### 线程池

通过std::thread实现，目前分为：任务调度，阻塞队列，管理线程，工作线程。

任务调度开放接口接收外部传入的std::function，并将其投放到阻塞队列的任务队列中。其实现互斥接口：获取忙碌线程数，获取空闲线程数，获取任务数，忙碌线程++，忙碌线程--等接口，供给管理线程调用。

阻塞队列通过std::mutex和两个std::condition\_variable实现，分别表示队列满、队列空的条件变量。当队列空时，工作线程等待条件变量唤醒。当队列满时，插入队列的调用阻塞，等待队列任务数减少后继续执行。

管理线程实现了简单的负载均衡，根据忙碌线程数、空闲线程数、等待任务数提高或者减少工作线程数量。当忙碌时新增工作线程；在空闲时通过条件变量的notify\_all唤醒工作线程，线程醒来后发现需要退出，则工作线程直接退出。

工作线程实现为消费者模型，等待队列空的条件变量唤醒。唤醒后调用任务队列中的std::function完成外部输入的函数调用。

### ConnectUser

在消息头部封装int32\_t类型的消息长度、消息ID，消息包来时，将数据放到缓冲区，若缓冲区大小满足消息包头+消息长度，则调用虚函数将消息传递到子类处理。

### IOMemoryStream

内存流：内部维护一个缓冲区，将变量的内存表示写入到缓冲区尾部，读出数据后通过movemove，将数据移动缓冲区到头部。

### BitSet

实现为4字节BitSet，通过 |= 1<< nbit实现设置位为1。通过&=(~(1 << nbit))设置位为0（异或后除了nbit位为0，其余位全位1），做位与后该位被设置位0，其他位保持原样。

### 红黑树

通过2-3树的表示而来，红连接表示3节点，黑连接表示2节点。红黑树需保证：

红连接均为左连接（3节点的特性），若左黑右红：左旋。

没有任何一个节点和两条红连接相连，若左连接和左孩子的左连接都红：右旋。

如果两个孩子都是红连接，则将两条连接变黑，和父节点的连接变红。（向3节点插入，将4节点分解为两个2节点，中间值拿到父节点）

插入时，如果底部为2节点（黑），如果是左插，则直接变成3节点（变红）；如果是右插，则先变成3节点，但右连接为红，则需要右旋。

左旋：h和h的右孩子交换位置，新根为h的右孩子，新根的左连接指向h。

右旋：h和h的左孩子交换位置，新根为h的左孩子，新根的右连接指向h。

### 哈希表

哈希值：std::Hash<T>计算值的哈希值，通过hash%数组length得到在数组中的索引。

哈希冲突：数组的元素是链表，当hash相同时，将值尾插到链表中。

## 远程文件管理

### epoll ET

epoll适用于连接的fd多，但活跃的fd不多。其监控队列使用红黑树，就绪队列使用双链表。ET模式只会在消息来时提醒进程一次。

使用struct epoll\_event的泛型参数实现事件机制，其指向一个结构体指针。将回调函数std::function、事件信息等封装成事件结构体，通过调用回调函数std::function，实现事件的处理。

### 线程消息队列

将服务器拆分为网络模块：调用epoll处理数据抵达，粘包拆包，发送包；用户模块：组织一个map，处理消息包具体逻辑；内部指令模块：和QT显示进程建立fifo，接收显示进程传来的指令，将客户端数据发送到显示进程。

### fifo

有名管道的封装，通过CreateNamedPipe/mkfifo,ConnectNamedPipe/open，WaitNamedPipe/open，CreateFile/open调用。实现两个无血缘关系的进程通信。

## 游戏服务器框架

### 线程消息队列

将服务器拆分为网络模块：调用libevent处理数据抵达，粘包拆包，发送包，调用ConnectUser的子类Player处理消息包：将注册、登录消息直接发到db模块，其他消息发往User模块；用户模块：组织一个map，处理登录后玩家的具体逻辑；DB模块：使用Redis+MySQL架构，处理数据的存储。数据层次上分为：内存数据，redis数据，mysql数据。

### libevent

调用libevent托管网络模块，在收到libevent的回调后，将消息包转发到Net模块，由Net模块处理客户端连接、客户端消息收发，总体实现类似epoll ET。

### Redis + MySQL

数据层次上分为：内存数据，redis数据，mysql数据。MySQL用户表存储为：用户名，密码，使用protobuf::\_DBUser序列化后的二进制块——使用MEDIUMBLOB类型存储，方便于在加新的成员时不影响MySQL结构，以及玩家上次活跃时间。在DB模块启动时：

select UserName,Password,LastLogin from user where LastLogin > %ld

查找所有上次登录时间在一天内的玩家，将他们除了\_DBUser以外的数据读到Redis。

Redis使用hash类型，用户名做key，”PassWord”,”LastLogin”等字段做field，将活跃玩家的常用数据插入到Redis。

[HMSET key field1 value1 [field2 value2 ]](https://www.runoob.com/redis/hashes-hmset.html)

HMSET username PassWord %s LastLogin %ld

#### 读取

当需要获取未在线玩家的数据时：如登录、获取Rank分数、获取庭院、获取上一次登录时间，因为不在线所以内存数据查找失败，则到Redis查找玩家数据，如果Redis命中则返回；如果Redis中也未能查询到（非活跃玩家），则从MySQL读取。

实际登录时候：

首先验证用户名密码，

HMGET key field1 [field2]

HMGET username PassWord

尝试获取玩家用户名和密码，若该玩家不活跃，则获取失败。通过MySQL读取：

select password from user where username = '%s'

即可验证玩家用户名密码正确性。

验证通过后，将玩家视为活跃玩家，加入到Redis缓存：

HMSET username PassWord %s LastLogin %ld

从MySQL读取玩家的\_DBUser数据：

select DBUser from user where UserName = %s

将登录玩家的\_DBUser数据块读出来，通过protobuf反序列化，常驻在内存，成为内存数据。在玩家登出时，将\_DBUser回写到数据库。

#### 写入

在修改玩家数据时，需要同步三层数据。内存数据更新，其他模块每过一段时间，以及服务器关闭时，将发消息到DB模块，由DB模块将新数据直接写入到MySQL：

update user set (PassWord=%s,LastLogin=%ld) where UserName = %s

Redis调用：

HMSET username PassWord %s LastLogin %ld

将玩家最新数据同步到Redis缓存。