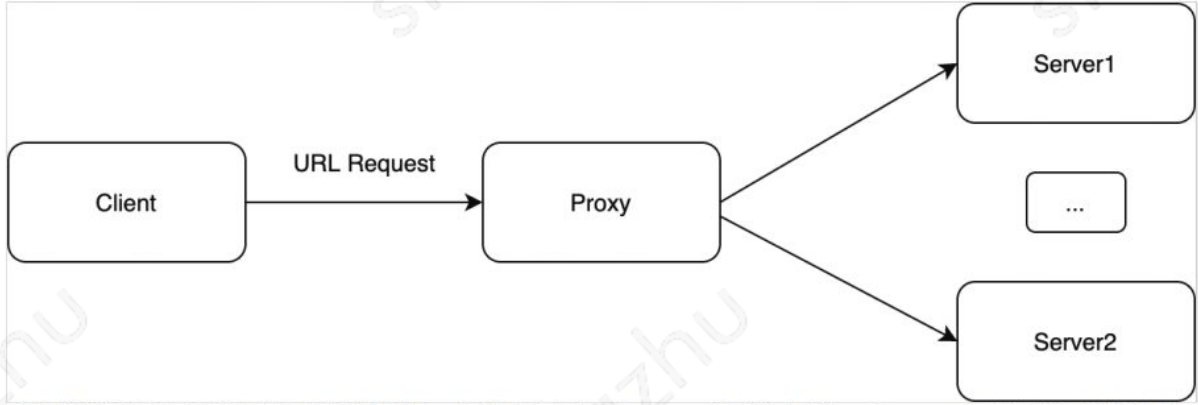
# 简单API网关接入系统个人工作总结

# 1.项目简介：



设计和实现一个接入网关。客户端的请求不直接到达服务器，会首先经过一个Proxy网关代理，由网关代理转发到服务器。通过这种模式，可以在网关上部署统一鉴权、恶意检测、流量调度、访问控制等各种能力。

以一个抽奖业务为例接入网关，每年圣诞晚会门票有限，所有采取抽奖形式确定员工是否有资格进入晚会现场，假定门票为四万张，需要实现一个抽票系统来公平的分配圣诞门票。

需要提供的业务能力：

* 公平随机分配门票
* '查询某员工是否拥有门票

考察点;

* 容灾回复能力：该系统具备一定的服务容灾和扩展能力，支持下游节点的腹胀剔除和水平扩展。
* 业务一致性：门票不能重复分配，查询结果必须准确

# 2.设计方案：

## 2.1为什么选择基于libevent开发网关与服务器

经过前期的查阅资料与导师的推荐，了解到nginx能通过简单的参数配置就可以实现一个高性能的网关，实现消息的转发与下游服务器的容灾与水平扩展，通过nginx的其他模块可以实现对客户端限流等基本功能。

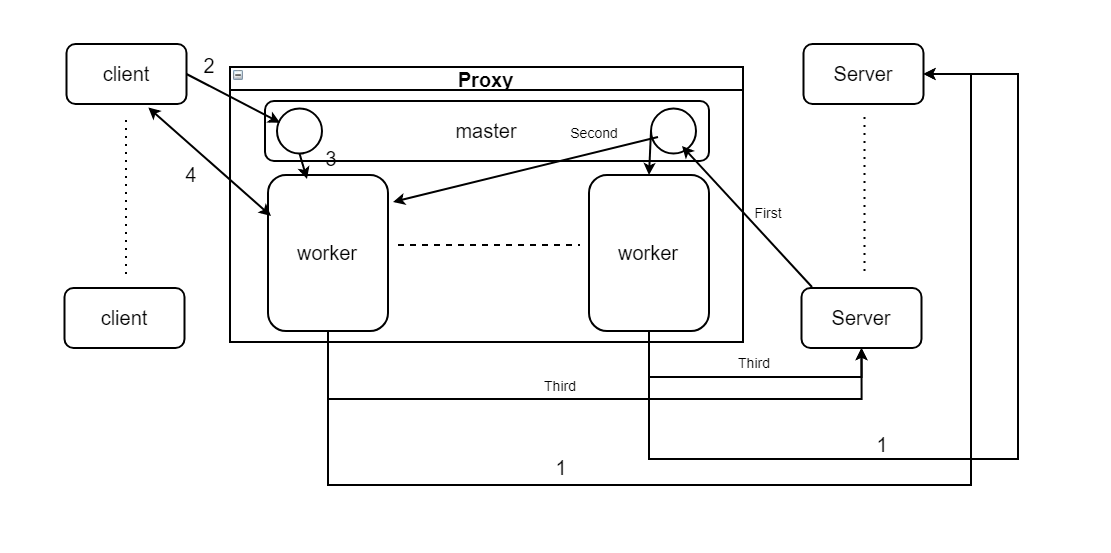
但是经小组讨论，觉得自己实现一个网关系统更有意义，故放弃了nginx，选择自己搭建网关与服务器。

Libevent 是一个用C语言编写的、轻量级的开源高性能事件通知库，主要有以下几个亮点：事件驱动（ event-driven），高性能;轻量级，专注于网络，支持 Windows、 Linux、 \*BSD 和 Mac Os；支持多种 I/O 多路复用技术， epoll、 poll、 dev/poll、 select 和 kqueue 等；支持 I/O，定时器和信号等事件；注册事件优先级。

Libevent是前期学习的一本书中了解到的，当时简单的体验了一下libevent中的事件注册，回调机制，觉得非常方便。并且觉得手写服务器直接使用libevent进行事件注册与回调，而不是直接处理epoll等io事件，会更加便捷，开发效率更高，所以最终确定了基于libevent事件库开发网关与服务器的整体路径。

## 2.2网关基础架构

网关基础架构选择的一主进程，多工作进程的模式。之所以选择多进程而不是多线程，是考虑到本项目中网关只有一个，而且需要进行消息的转发，从效率和安全的角度选用了多进程。



上图所示是网关的整体架构示意图。主进程监听两个端口：客户端监听端口，服务端监听端口；客户端监听端口暴露给客服端，是客户端的连接入口；服务器监听端口是暴露给服务器的端口，服务器可以通过该端口向网关注册自身。

该网关服务器具备以下特点：

* 单连接，异步
* 一主进程，多工作进程
* 工作进程与所有服务器都有连接
* 主进程负责监听客户端端口和服务端端口
* 客户端在工作进程之间轮询分配
* 同一工作进程中的客户端请求在不同服务器之间轮询分配

客户端和服务端都利用bufferevent为其分配缓存，提高消息的收发效率。

### 2.2.1 客户端与网关交互流程

1、 已经准备好的服务器开始运行，打开监听端口

2、 网关开始运行，工作进程读取配置文件，知道哪些服务器可用，与这些启动的服务器建立连接

3、 客户端connect网关的客户端监听端口

4、 主进程负责监听客户换监听端口，发现有客户端connect请求，轮询选择一个工作进程，通过管道通知其有新的客户端连接请求

5、 被通知的工作进程accept该客户端

6、 客户端向工作进程发送请求

7、 工作进程通过轮询的方式选择一个服务器发送该客户端请求，并在请求报文中标记该客户端

8、 服务器向网关返回报文，根据报文中客户端的标记找到对应客户端，向客户端返回服务器应答

整个流程有以下几个特点：

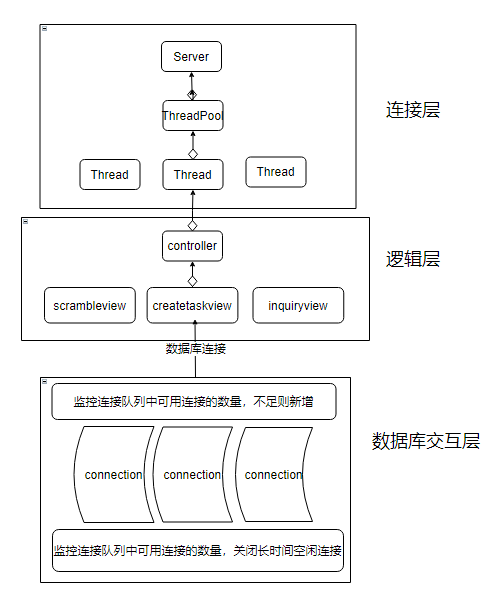
1、 工作进程与服务器之间是单连接

2、 客户端请求的转发与回复是异步的

3、 客户端连接请求可在不同工作进程之间轮询，做到客户端的平均分配

4、 客户端请求可在不同服务器之间轮询，做到客户端请求在不同服务器之间的轮询分配

## 2.3 服务端整体架构



服务端架构采用传统的三层设计：连接层，逻辑层，数据库交互层。数据库是一台mysql。

连接层的整体框架采用多线程，一个主线程和多个工作线程。整体io逻辑与网关类似，主要负责网关工作进程的接入和数据的读取。主线程负责监听端口，检测到有新客户连接则直接accept客户端，通过管道将套接字传递给工作线程。工作线程为其分配缓存，并注册读回调函数，实现消息的读取。

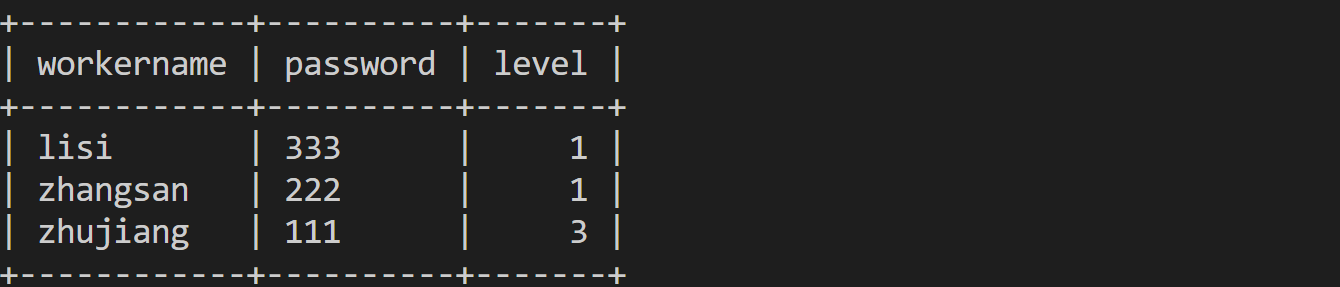
逻辑层主要通过虚基类实现了对不同业务的统一调用接口。通过继承虚基类，重写虚基类的消息处理接口，并利用map，将不同的消息类型映射到不同的消息处理类，最终实现了用一个统一的接口处理不同消息的效果。

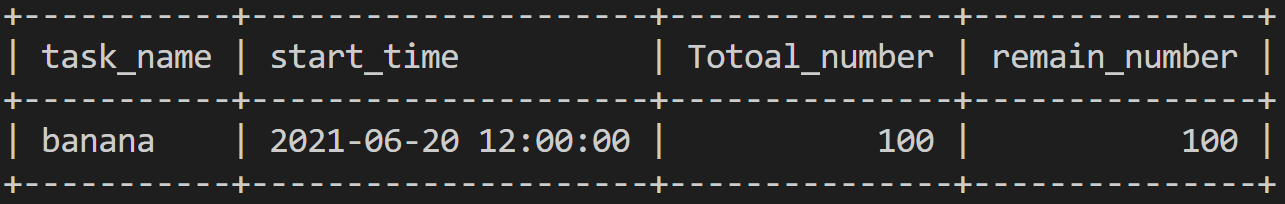
数据库交互层实现了一个数据库连接池，通过一个队列将所有数据库连接依次排序；利用锁与条件变量，实现数据库连接队列的维护。

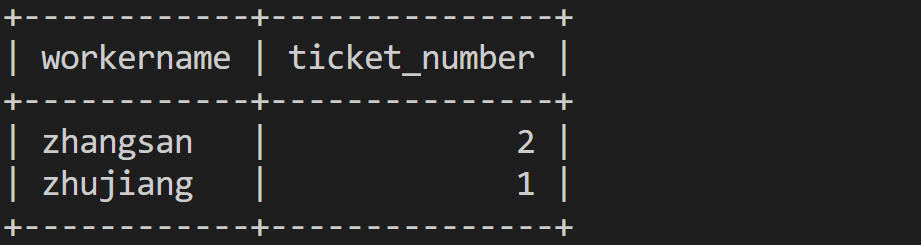
### 2.3.1数据库库表设计

为了实现抢票任务，共有三种表：

* 用户信息表
* 抢票任务信息表
* 某个抢票任务的用户信息记录表







用户信息表用于记录用户名，密码，用户等级等身份信息；抢票任务信息表用于记录抢票任务名，开始时间，总的库存，剩余库存信息；某个抢票任务的用户信息记录表用于记录成功抢到票的用户，及其票的编号

### 2.3.2业务逻辑设计

留下三个业务接口：查票、抢票、新建抢票任务

#### 2.3.2.1 查票

查询流程如下：

* 检查该用户身份
* 检查对应活动信息表中是否有该用户的信息

#### 2.3.2.2 抢票

抢票流程如下：

* 身份验证
* 判断是否达到活动开始时间
* 开启事务
* 检查该用户是否已经抢到票，检查过程出错或已经抢到票则结束事务
* 在抢票信息表中给该抢票任务上行锁，上锁失败则结束事务
* 检查该抢票任务的剩余库存是否大于零，若等于零或检查过程出错则结束事务
* 使该抢票任务的剩余库存减一，操作失败则回滚
* 在该抢票任务对应的用户信息表中为该用户添加一条记录，操作失败则回滚
* 提交事务。操作失败则回滚

抢票业务需要保持数据一致性，通过对任务库存加锁的方式保证了票不会超发。

#### 2.3.2.3 新建抢票活动

新建抢票活动流程如下：

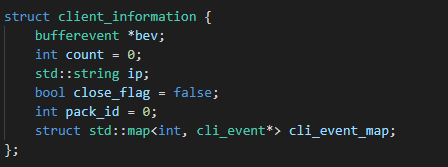
* 通过用户信息表检查该用户的权限等级，若等级不够则返回
* 开启事务
* 在抢票任务信息表中新增任务记录，失败则回滚
* 新建该抢票任务的用户信息表，失败则回滚
* 提交事务，失败则回滚

# 3网关新增功能：用户限流

为了避免恶意用户发送请求的速率过快，需要网关具有客户端限流的功能，即限制客户端在一定时间内的请求次数。

## 3.1客户端状态描述结构

为了描述客户端状态，使用如下客户端描述结构



Bev: 客户端的bufferevent，用于客户端的消息读写

Count: 用于记录客户端在一段时间内的发包次数

Ip: 用于记录该客户端的ip地址

Close\_flag用于标识是否关闭该客户端

## 3.2 客户端限流实现策略

在客户端的读回调函数中，每成功读取一个报文，将其count+1

当客户端的count达到某个设定值时，认为该客户端发包频率过快，是恶意用户，将其close\_flag位设置位true

工作进程添加定时事件，定时重置每个客户端的count, 并将close\_flag位为真的客户端关闭，并将其ip通过消息队列上传至主进程；

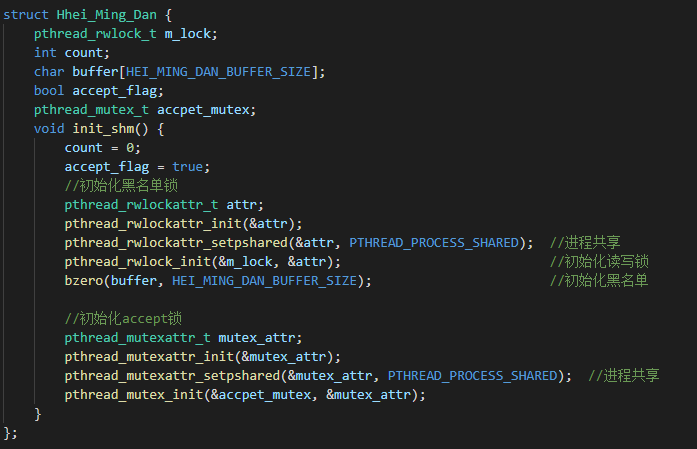
主进程接收到恶意用户IP，将其添加到黑名单中；每个工作进程在接收新的客户端时会检查该客户端的ip是否在黑名单中，如果在则拒绝连接

## 3.3 黑名单共享

如果不同工作进程持有不同的黑名单，则因为客户端轮询分配的原因，恶意用户被1个工作进程标记之后，还可以连接其他工作进程。为了解决这个问题，需要主进程管理一个共享的黑名单。

因此主进程需要开辟一块共享内存，用于记录恶意ip，工作进程从该共享内存中读，来判断客户是不是恶意用户，这一产生了一个一写多读的问题，可以使用读写锁来控制资源的互斥访问。

### 3.3.1共享内存结构



黑名单管理主要涉及读写锁m\_lock, ip数量count, 恶意ip保存位置buffer.

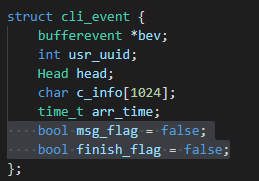
每个恶意ip在buffer中分配20字节，count用于记录buffer中已经添加了多少恶意ip。

主进程收到工作进程传递的恶意ip后将其添加到buffer中，并定时清空黑名单。

# 4 网关新增功能：客户请求缓存与超时重发

网关在转发客户请求时，如果不对客户请求进行缓存，只是将其转发给服务端，则在服务端发生故障时，部分客户请求会丢失。为了解决该问题需要在网关中缓存客户请求，并定时检查请求是否超时，超时则重发请求。

## 4.1客户端消息结构



Bev: 客户端消息收发缓存

Usr\_uuid：客户端标记

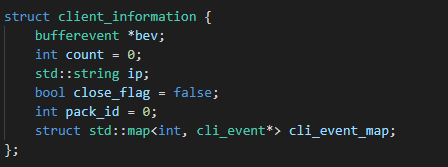
Head 消息头

C\_info 消息体

Arr\_time 消息转发时间

Finish\_flag 用于标识该消息是否收到回应

## 4.2 客户端描述结构



Cli\_event\_map用于缓存该客户端的未回应报文

## 4.3 缓存客户端消息策略

利用内存池未客户消息分配地址，避免内存的频繁分配与释放。新的客户端消息到来，利用内存池为其分配地址，将该消息的指针分别存入客户自身的cli\_event\_map, 和进程中一个总的消息队列queue。

存入客户自身的cli\_event\_map的目的是在收到服务器返回报文时，能够通过客户端标识和消息标识找到该消息，并将其finsih\_flag设置位true, 标识已经收到该消息的回应报文，同时在cli\_event\_map中删除该消息记录。

存入消息队列的目的时避免每次超时检测都要遍历所有客户的所有消息。通过队列按照转发顺序将所有消息依次排序，则不用每次将所有消息都遍历一遍。

设置定时任务，检查消息队列中的消息，从队列头部开始遍历，直到某个消息未超时，则停止遍历，即只处理超时的消息。

在超时的消息中，将客户端已经退出的消息、finsh\_flag为true的消息依次释放，否则重新转发该消息。

# 5实现过程中遇到的问题以及解决策略

## 5.1 网关主进程多次转发新客户到达通知

网关接收新的客户端采用的方式是主进程监听，谭厚通过管道通过某个工作进程，工作进程accept客户端的方式。

但是在测试时发现，一个新客户端connect，主进程可能会通知多个工作进程。

分析原因在于：主进程通知工作进程到工作进程accept新的客户端之间存在时间差。主进程可能在这个时间差内发现还有客户端在等待connect，于是又通知了另外一个工作进程。造成多个工作进程去accept同一个客户端。

解决策略：标志位+锁

主进程写管道之后就将标志位置为false，工作进程accept新客户端之后将标志位改为true；主进程只有在标记位位true的情况下才能写管道公职工作进程。

该方法使得只有在工作进程接收了新客户端之后，主进程才能通知下一个工作进程，而且由于对标志位的修改需要加锁，使得接收新客户端的效率不高。

# 6系统测试

今晚整体测试

之前自己测试了一下，一万个用户抢50张票，能够保证数据的一致性，不会超发。

# 7个人总结