10W QPS真刀实操__以及基于ZK+Netty手写分布式测试工具

参考链接

系统架构知识图谱（一张价值10w的系统架构知识图谱）

https://www.processon.com/view/link/60fb9421637689719d246739

秒杀系统的架构

https://www.processon.com/view/link/61148c2b1e08536191d8f92f

10W QPS真刀实操的要点

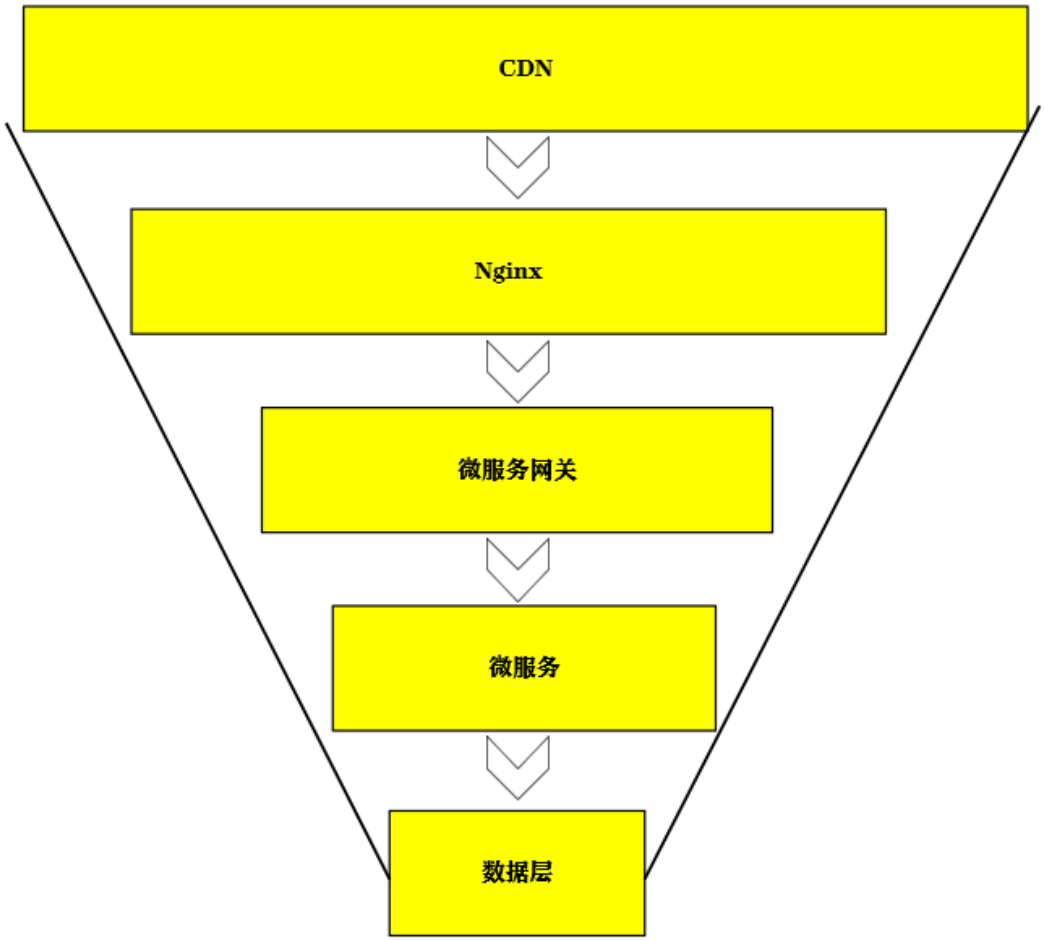
这个是一个复杂的工程

但是很有意义

漏斗模型中的请求分层过滤

漏斗型请求处理模型的核心策略，对请求进行 **分层过滤。**

而针对漏斗型请求处理模型（如秒杀场景）一 种核心策略，就是对请求进行分层过滤，从而过滤掉一些无效的请求。



案例：秒杀系统的分层过滤

比如,在秒杀系统中，请求分别经过 CDN、Nginx（商品详情）、微服务（如交seckill）和数据库 这几层，那么：

1 大部分数据和流量在用户浏览器或者 CDN 上获取，这一层可以拦截大部分静态资源的读取；

2、经过第二层 Nginx（商品详情）时，尽量得走 Nginx Cache，过滤一些可以直接访问Nginx缓存的请求；

经过第二层 Nginx（商品详情）时，也可以进行流控，还可以进行黑名单过滤，拦截掉一些无效的流量；

3、再到服务层，进入微服务网关时，可以做用户的授权检验，对系统做好保护和限流，这样数据量和请求就 进一步减少；

4、业务层，还可以进行数据的有效性、一致性过滤，这里又减少了一些流量。

这样就像漏斗一样，尽量把数据量和请求量一层一层地过滤和减少了。

分层过滤的核心思想是：在不同的层次尽可能地过滤掉无效请求，让“漏斗”最末端的才是有效 请求。而要达到这种效果，我们就必须对数据做分层的校验。

分层过滤的基本原则是：

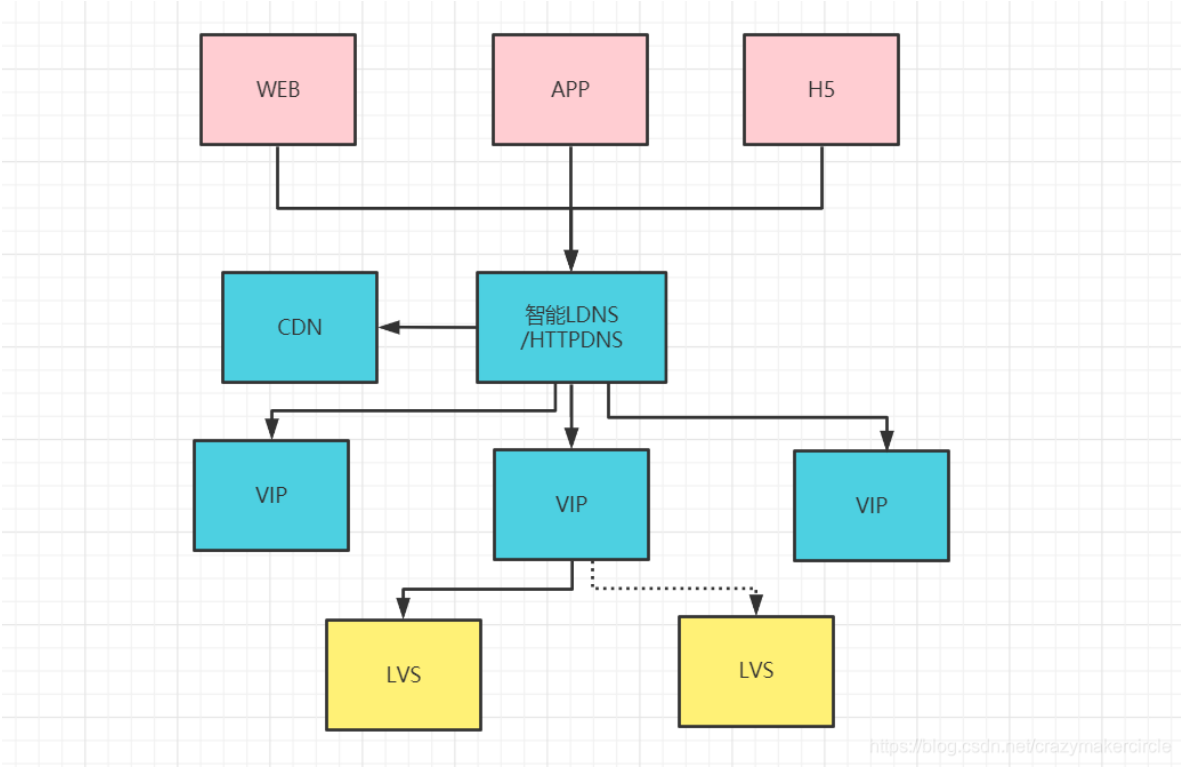
通过在不同的层次尽可能地过滤掉无效请求， 尽早处理掉请求。

- 通过CDN过滤掉大量的图片，静态资源的请求。
- 读请求尽量命中缓存，不要穿透到数据库；
- 尽量将动态读数据请求，命中在三级缓存,或者二级缓存，过滤掉无效的数据读；
- 对写入操作进行削峰，争取批量写入，提高写入的吞吐量；
- 分层限流：防止系统雪崩

接入层

流量分发、负载均衡

按照用户规模，流量规模（吞吐量规模），接入层的架构方案不一样



slb 负载均衡， gslb

LVS简介

LVS是Linux Virtual Server的简称，也就是Linux虚拟服务器, 由章文嵩博士发起的自由软件项目，它的官方站点是www.linuxvirtualserver.org。通过LVS提供的负载均衡技术和Linux操作系统实现一个高性能、高可用的服务器群集，它具有良好可靠性、可扩展性和可操作性。从而以低廉的成本实现最优的服务性能。LVS 是一个实现负载均衡集群的开源软件项目，LVS 架构从逻辑上可分为调度层、Server 集群层和共享存储。

负载均衡(LB) 集群的架构和原理很简单，就是当用户的请求过来时，会直接分发到 Director Server 上，然后它把用户的请求根据设置好的调度算法，智能均衡地分发到后端真正服务器 (real server) 上。为了避免不同机器上用户请求得到的数据不一样，需要用到了共享存储，这样保证所有用户请求的数据是一样的。

lvs 的优点?

抗负载能力强，因为 lvs 工作方式的逻辑是非常之简单，而且工作在网络 4 层仅做请求分发之用，没有流量，所以在效率上基本不需要太过考虑。
有完整的双机热备方案，当节点出现故障时，lvs 会自动判别，所以系统整体是非常稳定的。
基本上能支持所有应用，因为 lvs 工作在 4 层，所以它可以对几乎所有应用做负载均衡，包括 http、数据库、聊天室等等。

lvs 负载均衡机制

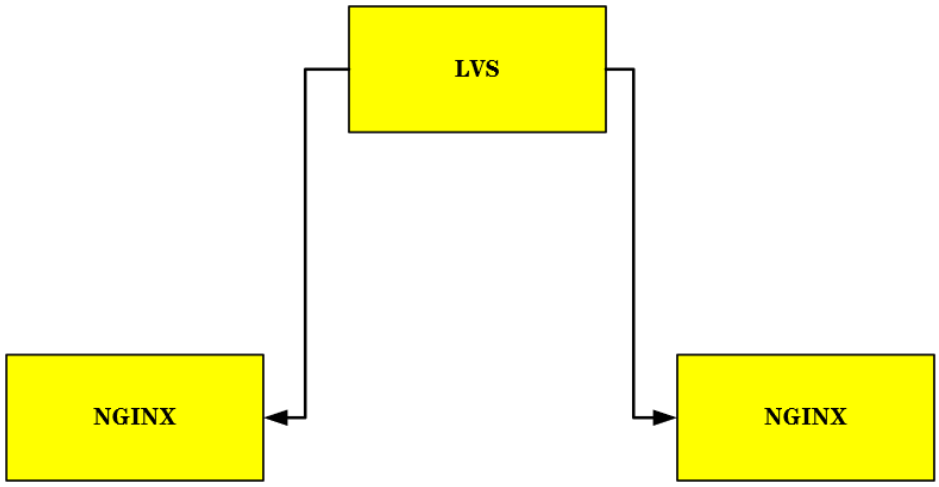
lvs 是四层负载均衡，也就是说建立在 OSI 模型的第四层——传输层之 上
传输层上有 TCP/UDP，lvs 支持 TCP/UDP 的负载均衡
因为 LVS 是四层负载均衡，因此它相对于其它高层负载均衡的解决办法， 比如 DNS 域名轮流解析、应用层负载的调度、客户端的调度等，它的效 率是非常高的
lvs 的转发可以通过修改 IP 地址实现（NAT 模式）
lvs 的转发还可以通过修改直接路由实现（DR 模式）

lvs 与 nginx 对比?

负载度　lvs 优于 nginx
稳定度　lvs 优于 nginx
服务器性能要求 lvs 优于 nginx
网络层数的效率 lvs 优于 nginx □ 网络七层：应用层、会话层、表示层、传输层、网络层、链路层、 物理层
功能多少　nginx 优于 lvs

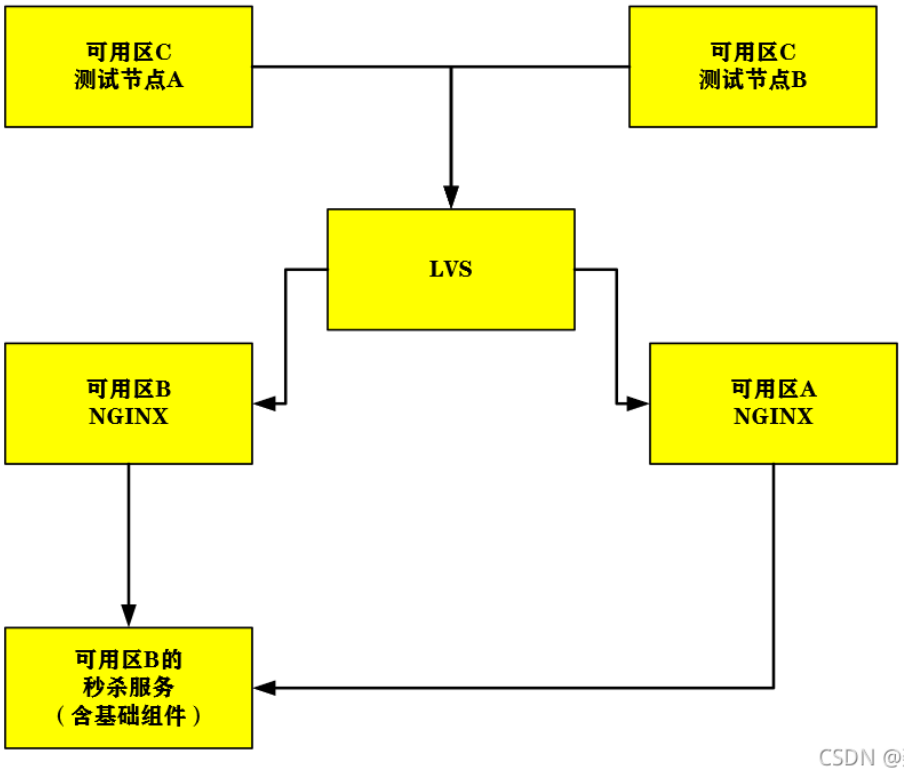
10W QPS实操的系统架构

10W QPS实操的接入层逻辑架构



10W QPS实操的物理架构

阿里云购买5个8核、32g的ecs服务器，来进行实操，具体的架构如下。



阿里云机器购买

购买虚拟机

三个可用区，四个虚拟机

a、b区各一个，作为接入主机

c区两个，做为 服务层的主机，和测试主机

登录&注册

> 登录阿里云

百度搜索阿里云，进入阿里云官网（访问）。如下图所示。

在上图的页面的右上角，找到“登录”按钮，点击以后，会弹出如下登录界面：

在这个登录界面中，输入用户名和密码。

选择付费类型

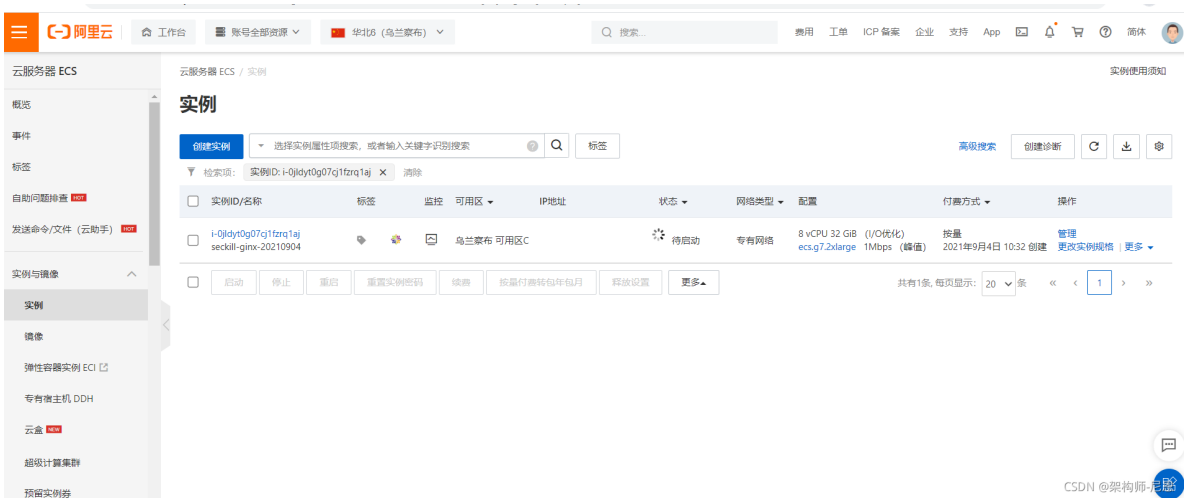
选择系统类型

选择网络类型

选择输入登录密码和节点名称

ecs详情

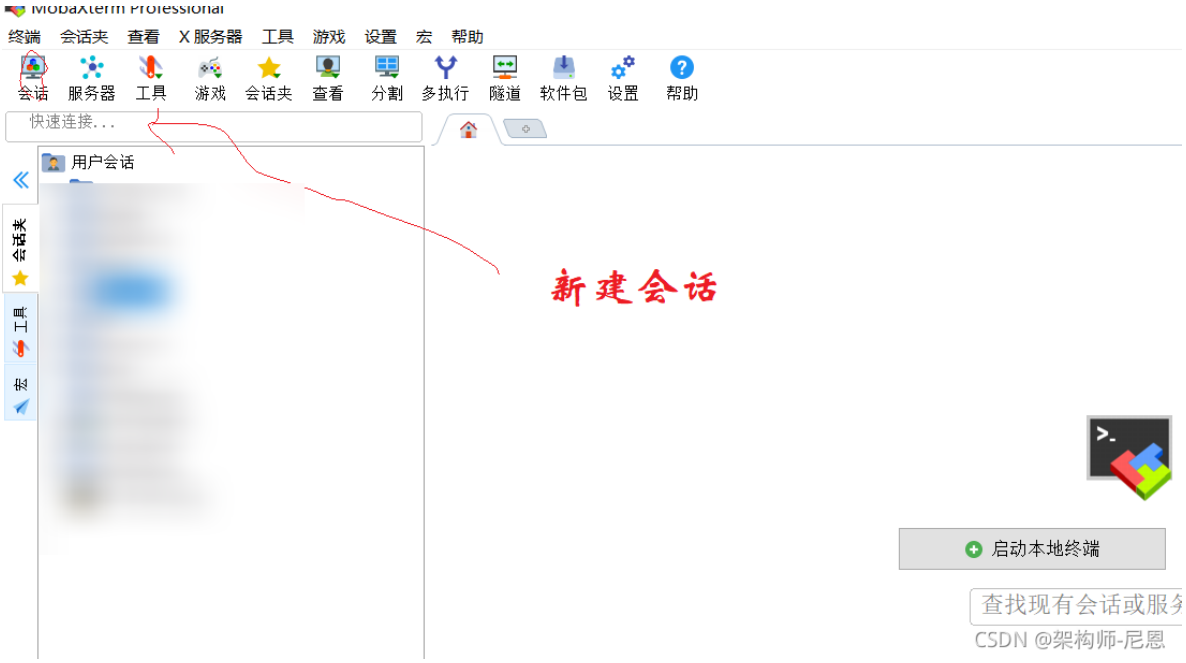
控制台详情



云主机的登录与配置

云主机的登录

新建ssh会话

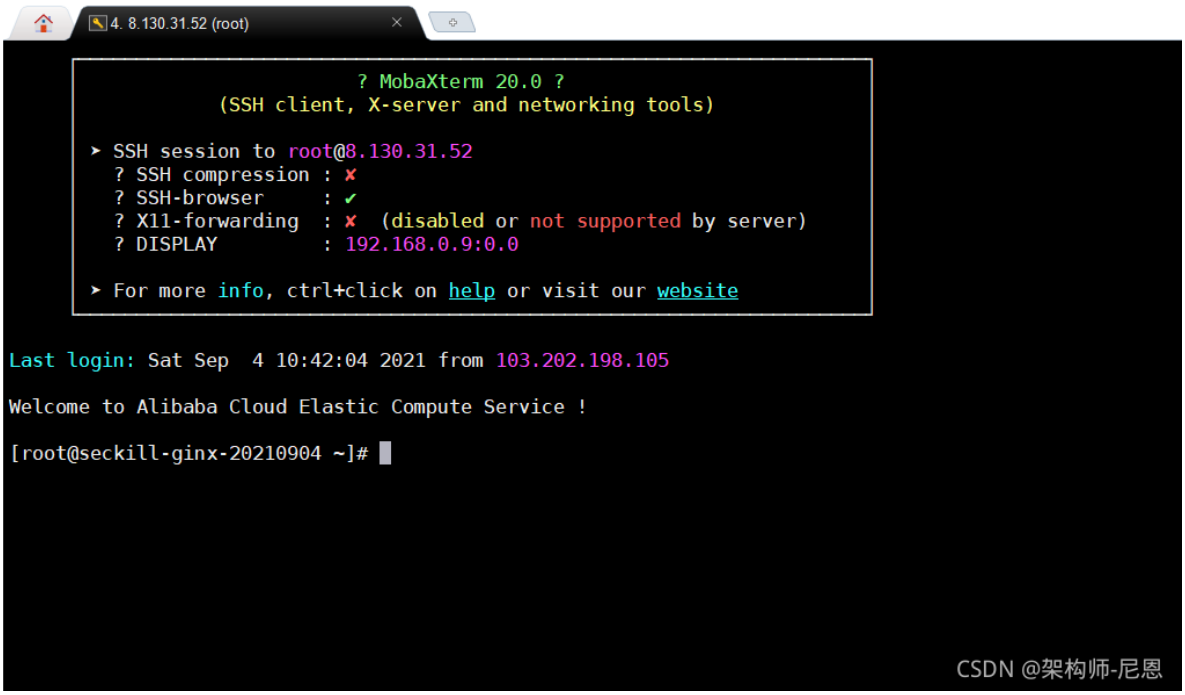


输入ecs的IP

从阿里云控制台去复制一下ip，注意，是公网ip

输入密码后，登录成功

输入购买的时候设置的密码



查看核心数量和内存大小

查看CPU

```
grep 'core id' /proc/cpuinfo | sort -u | wc -l
```

查看内存

```
[root@seckill-ginx-20210904 ~]# grep 'core id' /proc/cpuinfo | sort -u | wc -l
4
[root@seckill-ginx-20210904 ~]# free -h
              total        used        free      shared  buff/cache   available
Mem:            30G        375M         30G        312K        352M         30G
Swap:            0B          0B          0B
```

文件句柄数的配置

查看最大句柄数限制

```
#查看单个进程最大句柄数

ulimit -n


cat /etc/security/limits.conf


#查看系统打开句柄最大数量


cat /proc/sys/fs/file-max
```

结果:

```
[root@seckill-ginx-20210904 ~]# cat /etc/security/limits.conf
# /etc/security/limits.conf
#
#This file sets the resource limits for the users logged in via PAM.
#It does not affect resource limits of the system services.
#
#Also note that configuration files in /etc/security/limits.d directory,
#which are read in alphabetical order, override the settings in this
#file in case the domain is the same or more specific.
#That means for example that setting a limit for wildcard domain here
```

```
#can be overriden with a wildcard setting in a config file in the
#subdirectory, but a user specific setting here can be overriden only
#with a user specific setting in the subdirectory.
#
#Each line describes a limit for a user in the form:
#
#<domain>        <type>  <item>  <value>
#
#Where:
#<domain> can be:
#        - a user name
#        - a group name, with @group syntax
#        - the wildcard *, for default entry
#        - the wildcard %, can be also used with %group syntax,
#                 for maxlogin limit
#
#<type> can have the two values:
#        - "soft" for enforcing the soft limits
#        - "hard" for enforcing hard limits
#
#<item> can be one of the following:
#        - core - limits the core file size (KB)
#        - data - max data size (KB)
#        - fsize - maximum filesize (KB)
#        - memlock - max locked-in-memory address space (KB)
#        - nofile - max number of open file descriptors
#        - rss - max resident set size (KB)
#        - stack - max stack size (KB)
#        - cpu - max CPU time (MIN)
#        - nproc - max number of processes
#        - as - address space limit (KB)
#        - maxlogins - max number of logins for this user
#        - maxsyslogins - max number of logins on the system
#        - priority - the priority to run user process with
#        - locks - max number of file locks the user can hold
#        - sigpending - max number of pending signals
#        - msgqueue - max memory used by POSIX message queues (bytes)
#        - nice - max nice priority allowed to raise to values: [-20, 19]
#        - rtprio - max realtime priority
#
#<domain>      <type>  <item>         <value>
#

#*               soft    core            0
#*               hard    rss             10000
#@student        hard    nproc           20
#@faculty        soft    nproc           20
#@faculty        hard    nproc           50
#ftp             hard    nproc           0
#@student        -       maxlogins       4

# End of file
root soft nofile 65535
root hard nofile 65535
* soft nofile 65535
* hard nofile 65535

[root@seckill-ginx-20210904 ~]#  cat /proc/sys/fs/file-max
```

```
3195223
```

修改最大句柄数限制

cat /proc/sys/fs/file-max

```
#追加
cat >> /etc/security/limits.conf <<EOF
* soft nproc 65535
* hard nproc 65535
* hard nofile 1200000
* soft nofile 1200000
EOF

#编辑
vi /etc/security/limits.conf

#追加
cat >> /etc/sysctl.conf <<EOF
fs.nr_open=2000000
fs.file-max=10000000
EOF
#编辑
vi /etc/sysctl.conf

#生效
sysctl -p


cat /etc/security/limits.conf

cat /proc/sys/fs/file-max
```

为啥要调整最大文件句柄

我们常常会遇到类似“Socket/File: Can’t open so many files”，“无法打开更多进程”，或是coredump过大等问题，这些都可以设置资源限制来解决。

局部文件句柄数和全局文件句柄数的详解

增加局部文件句柄数

- 临时修改当前局部文件句柄数：

```
ulimit -n 10000
```

这是临时设置，系统重启后设置会丢失

- 永久修改当前局部文件句柄数：修改文件/etc/security/limits.conf，

```
#编辑
vim /etc/security/limits.conf
#查看
cat /etc/security/limits.conf
```

- 文件末尾加上：

```
 * hard nofile 1200000
 * soft nofile 1200000
```

- 修改文件后需要重启机器，才能生效

修改后的效果

```
#*               soft    core            0
#*               hard    rss             10000
#@student        hard    nproc           20
#@faculty        soft    nproc           20
#@faculty        hard    nproc           50
#ftp             hard    nproc           0
#@student        -       maxlogins       4
* soft nproc 65535
* hard nproc 65535
* soft nofile 6553500
* hard nofile 6553500
# End of file
```

soft nproc： 单个用户可用的最大进程数量(超过会警告);
hard nproc：单个用户可用的最大进程数量(超过会报错);
soft nofile ： 可打开的文件描述符的最大数(超过会警告);
hard nofile： 可打开的文件描述符的最大数(超过会报错);

局部文件句柄数上限配置

可通过修改/ect/sysct.conf 修改 fs.nr_open 值，sysctl -p生效

```
//查看

cat /etc/sysctl.conf

//修改

vim /etc/sysctl.conf

fs.nr_open=150000
```

生效的shell指令

```
sysctl -p
```

新的问题： 局部的不能大过全局的限制

还有一个Linux kernel能分配的全局文件句柄数限制

全局句柄数限制的配置

全局句柄数限制，字面上看file-max，file-max 确实像是对应最大文件数，而在Linux内核文档中它们两的解释是：
file-max：

```
The value in file-max denotes the maximum number of file-
handles that the Linux kernel will allocate. When you get lots
of error messages about running out of file handles, you might
want to increase this limit
```

查看全局文件句柄的限制

局部的不能大过全局的限制，查看我所有进程能够打开的最大文件数是多少

```
[root@cdh1 ~]# cat /proc/sys/fs/file-max
185213

[root@cdh1 ~]#
```

增加全局文件句柄数

- 查看：cat /etc/sysctl.conf
- 配置文件 /etc/sysctl.conf

  vim /etc/sysctl.conf
- 在文件末尾加上：fs.file-max=1000000

  ```
  fs.file-max = 1500000
  ```
- 使配置文件生效：sysctl -p

总体的效果

```
cat /etc/sysctl.conf
fs.nr_open=2000000
fs.file-max=10000000
```

file-max是所有进程最大的文件数

nr_open是单个进程可分配的最大文件数

特别注意：

> 局部文件句柄数一定不要超过全局文件句柄数！！！
>
> 把局部文件句柄数设置超过了全局文件句柄数，可能导致无法开机后无法登陆的现象！！！

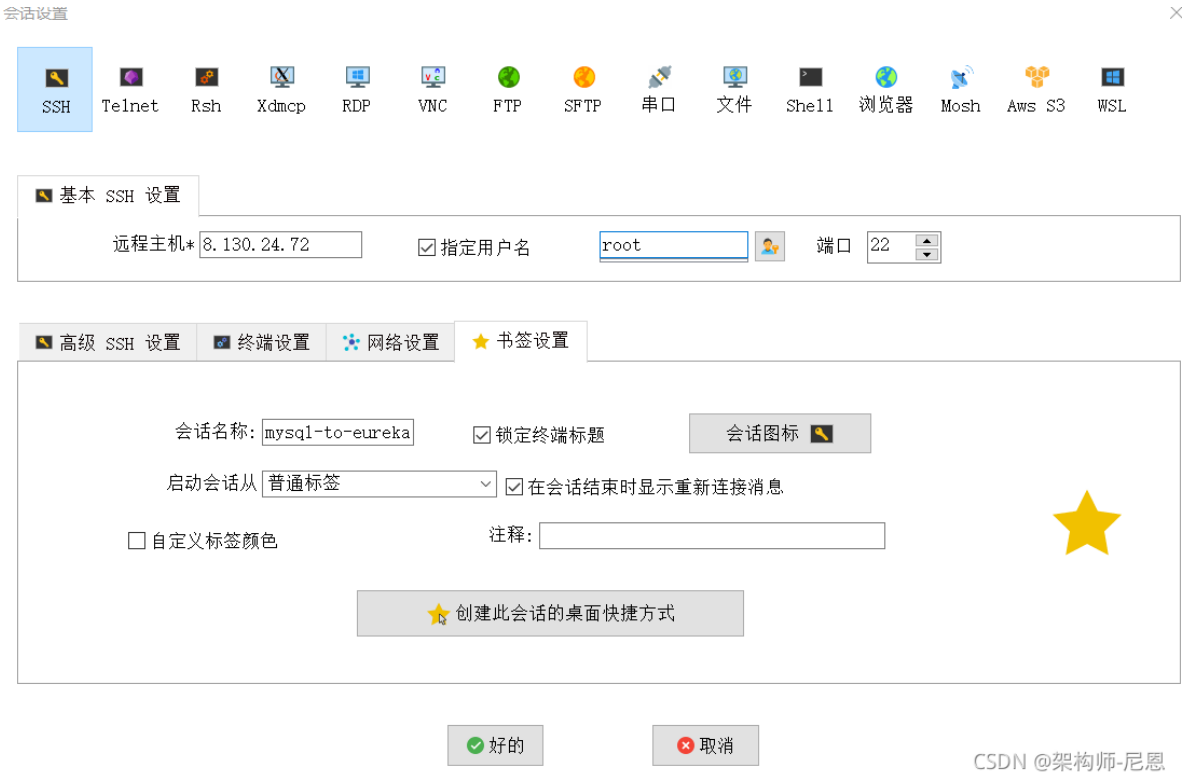
借助shell脚本__极速安装基础组件

mysql的安装

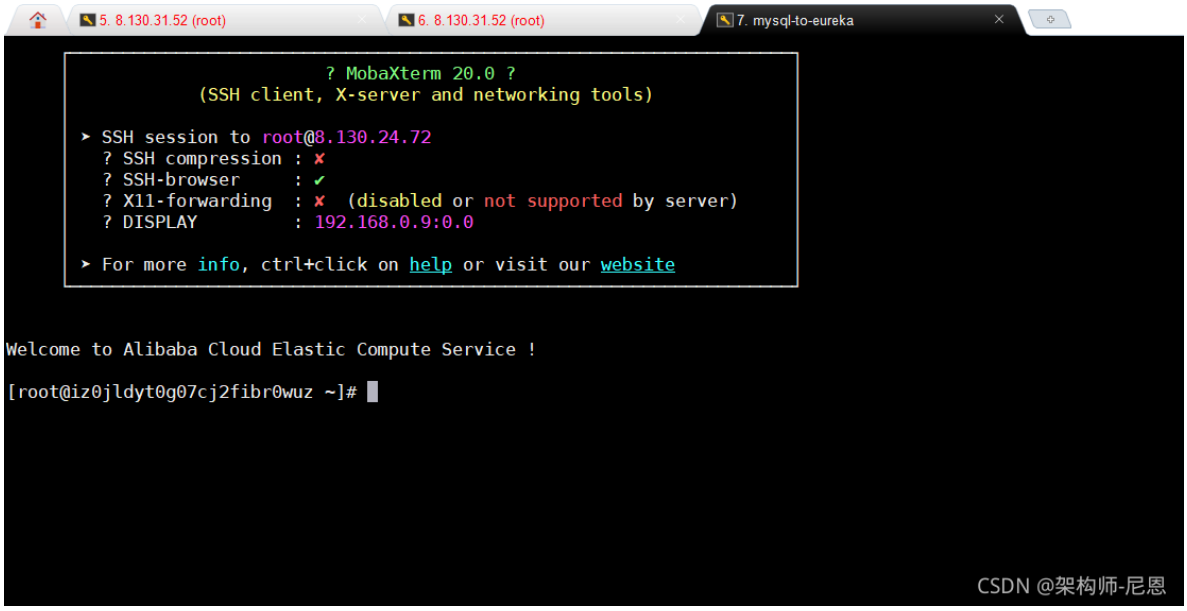
连接一个虚拟机

在可用区c上找一个实例， 改名为，mysql-to-eureka，并且启动

连接服务器



使用密码登录



一键安装Mysql

参考链接的入口地址

https://www.cnblogs.com/crazymakercircle/p/9904544.html

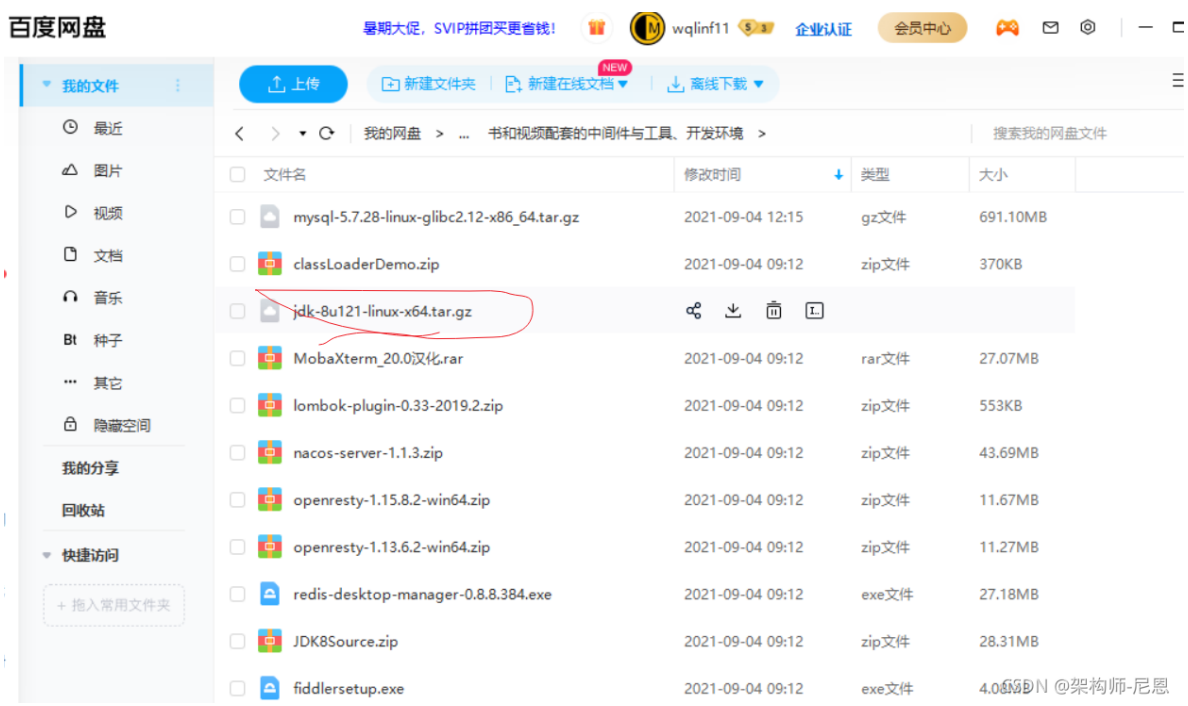
创建数据库

```
mysql -uroot -p'wTJJJJZZZ@SSDDC'
```

```
CREATE DATABASE store DEFAULT CHARACTER SET utf8 COLLATE utf8_general_ci;


root@localhost [(none)]>show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| store              |
| sys                |
+--------------------+


GRANT ALL PRIVILEGES ON store.* TO "root" @'%'  IDENTIFIED BY 'WTJJJJZZZ@SSDDC';


flush privileges;
```

一键安装jdk

可以从百度网盘下载jdk1.8U212



上传到 /work

然后执行下面的脚本

```
mkdir -p  /usr/local/java
tar -zxvf /work/jdk-8u121-linux-x64.tar.gz -C /usr/local/java


cat  >> /etc/profile <<EOF
export JAVA_HOME=/usr/local/java/jdk1.8.0_121
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
export PATH=$JAVA_HOME/bin:$PATH
EOF

 source /etc/profile
```

```
[root@izOjldytOgO7cj2fibrOwuz jdk1.8.0_121]# java -version
java version "1.8.0_121"
Java(TM) SE Runtime Environment (build 1.8.0_121-b13)
Java HotSpot(TM) 64-Bit Server VM (build 25.121-b13, mixed mode)
```

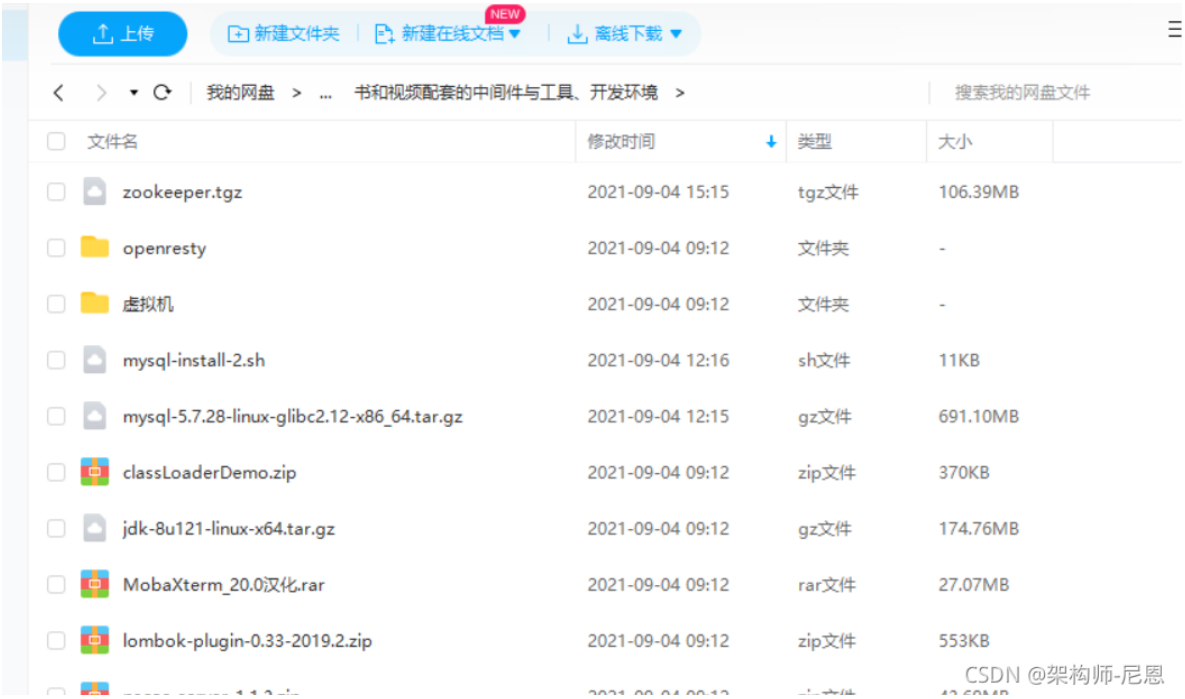
一键安装zk的安装

伪装的集群模式

参考链接的入口地址

https://www.cnblogs.com/crazymakercircle/p/9904544.html

测试环境的一键安装包



上传一键安装包，开始安装

```
[root@ work]# cd /work

[root@ work]# tar -zxvf zookeeper.tgz
```

开机启动脚本：

```
[root@ work]# /usr/bin/su  - root  -c
"/work/zookeeper/zookeeper_01/bin/zkServer.sh start"
ZooKeeper JMX enabled by default
Using config: /work/zookeeper/zookeeper_01/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED
[root@ work]# /usr/bin/su  - root  -c
"/work/zookeeper/zookeeper_02/bin/zkServer.sh start"
ZooKeeper JMX enabled by default
Using config: /work/zookeeper/zookeeper_02/bin/../conf/zoo.cfg
Starting zookeeper ... STARTED

[root@ work]# jps
```

```
6672 QuorumPeerMain
26672 QuorumPeerMain
```

查看状态

```
[root@ work]# /usr/bin/su  - root  -c
"/work/zookeeper/zookeeper_01/bin/zkServer.sh status"
ZooKeeper JMX enabled by default
Using config: /work/zookeeper/zookeeper_01/bin/../conf/zoo.cfg
Mode: follower

[root@ work]# /usr/bin/su  - root  -c
"/work/zookeeper/zookeeper_02/bin/zkServer.sh status"
ZooKeeper JMX enabled by default
Using config: /work/zookeeper/zookeeper_02/bin/../conf/zoo.cfg
Mode: leader
```

redis安装

参考链接的入口地址

https://www.cnblogs.com/crazymakercircle/p/9904544.html

启动

```
/usr/local/redis/bin/redis-server  /usr/local/redis/redis.conf
```

测试是否可以访问

```
/usr/local/redis/bin/redis-cli -h 127.0.0.1 -p 6379 -a "123456"

 /usr/local/redis/bin/redis-cli -h cdh1 -p 6379 -a "123456"
```

```
[root@izOjldytOgO7cj2fibrOwuz bin]#  /usr/local/redis/bin/redis-cli -h cdh1 -p
6379 -a "123456"
Warning: Using a password with '-a' or '-u' option on the command line interface
may not be safe.
cdh1:6379> quit
```

rocketmq的安装

参考链接的入口地址

https://www.cnblogs.com/crazymakercircle/p/9904544.html

Linux openresty 安装	Linux openresty 安装
【必须】Linux Redis 安装（带视频）	Linux Redis 安装（带视频）
	redis集群架构知识（史上最强，面试必看）
【必须】Linux Zookeeper 安装（带视频）	Linux Zookeeper 安装, 带视频
	ZooKeeper 客户端: GUI+命令行（史上最全）
Windows Redis 安装（带视频）	Windows Redis 安装（带视频）
RabbitMQ 离线安装（带视频）	RabbitMQ 离线安装（带视频）
RocketMQ 原理 - 部署 - 入门	RocketMQ 原理 - 部署 - 入门 （图解）
ElasticSearch 安装, 带视频	ElasticSearch 安装, 带视频
Nacos 安装（带视频）	Nacos 安装（带视频）
【必须】Eureka	Eureka 入门，带视频
【必须】springcloud Config 入门，带视频	springcloud Config 入门，带视频
【必须】SpringCloud 脚手架打包与启动	SpringCloud脚手架打包与启动
Linux 自启动 假死自启动 定时自启	Linux 自启动 假死启动

启动：

```
nohup mqnamesrv -n 172.26.9.107 &

nohup sh mqbroker -n 172.26.9.107:9876 autoCreateTopicEnable=true -c
/usr/local/rocketmq/rocketmq-all-4.4.0-bin-release/conf/broker.conf &


mqbroker -m
```

部署服务层

配置环境变量

cat /etc/profile

```
export SCAFFOLD_DB_HOST=cdh1
export SCAFFOLD_DB_USER=root
export SCAFFOLD_DB_PSW=wTJJJJZZZ@SSDDC
export SCAFFOLD_REDIS_HOST=cdh1
export SCAFFOLD_REDIS_PSW=123456
export SCAFFOLD_EUREKA_ZONE_HOSTS=http://cdh1:7777/eureka/
```

```
export SCAFFOLD_ZOOKEEPER_HOSTS=cdh1:2181
export LC_ALL=en_US.UTF-8
export ROCKETMQ_HOME=/usr/local/rocketmq/rocketmq-all-4.4.0-bin-release
export JAVA_HOME=/usr/local/java/jdk1.8.0_121
export
CLASSPATH=.:/usr/local/java/jdk1.8.0_121/lib/dt.jar:/usr/local/java/jdk1.8.0_121
/lib/tools.jar
export PATH=/usr/local/rocketmq/rocketmq-all-4.4.0-bin-
release/bin:/usr/local/java/jdk1.8.0_121/bin:/usr/bin:/usr/local/mysql/bin:/root
/bin

export SCAFFOLD_PREFERRED_NETWORKS=172.26.9.
export SCAFFOLD_DB_PWD=wTJJJJZZZ@SSDDC
export SCAFFOLD_ROCKETMQ_HOSTS=172.26.9.107:9876
```

cat /etc/hosts

```
[root@izOjldytOgO7cj2fibrOwuz tester-node-1.0-SNAPSHOT]# cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6
172.26.9.107 cdh1
192.168.56.122 cdh2
192.168.56.123 cdh3
```

注册中心与配置中心

eureka

springcloud config

```
[root@izOjldytOgO7cj2fibrOwuz ~]# sh /work/cloud-eureka-1.0-
SNAPSHOT/bin/deploy.sh start
PORT:7777
JVM:-server -Xms64m -Xmx256m
nohup: appending output to ‘nohup.out’
cloud-eureka-1.0-SNAPSHOT.jar is running,pid is 3292
[root@izOjldytOgO7cj2fibrOwuz ~]# sh /work/cloud-config-1.0-
SNAPSHOT/bin/deploy.sh start
PORT:7788
JVM:-server -Xms64m -Xmx256m
cloud-config-1.0-SNAPSHOT.jar is running,pid is 3536
```

网关zuul

```
sh /work/cloud-zuul-1.0-SNAPSHOT/bin/deploy.sh start
```

库存服务与秒杀服务

stockprovider

seckillprovider

```
sh /work/stock-provider-1.0-SNAPSHOT/bin/deploy.sh start

 sh /work/seckill-provider-1.0-SNAPSHOT/bin/deploy.sh start

tail -f /work/logs/stock-provider-1.0-SNAPSHOT/output.log
```

开通网关的端口，可以访问swagger页面

秒杀库存操作

http://8.130.31.52:8080/stock-provider/swagger-ui.html#

部署接入层

另外找一台机器，

Nginx的安装

参考链接的入口地址

https://www.cnblogs.com/crazymakercircle/p/9904544.html

修改主机名称

```
cat  >> /etc/hosts <<EOF
172.26.9.107 cdh1
172.26.9.105 cdh2
192.168.56.123 cdh3
EOF
```

上传前端工程：

> https://gitee.com/crazymaker/LuaDemoProject.git

> mkdir -p /vagrant/LuaDemoProject/src/proxy_temp/

查看ng的work进程的执行用户组

```
[root@seckill-ginx-20210904 ~]#  ps aux | grep "nginx: worker process" | awk
'{print $1}'
nobody
nobody
nobody
nobody
nobody
nobody
nobody
nobody
root
```

> chown -R root:root /vagrant/LuaDemoProject

```
chown -R  nobody:nobody  /vagrant/LuaDemoProject
```

> sh /vagrant/LuaDemoProject/sh/linux/openresty-restart.sh
>
> curl http://localhost:8080/echo

特意说说配置文件

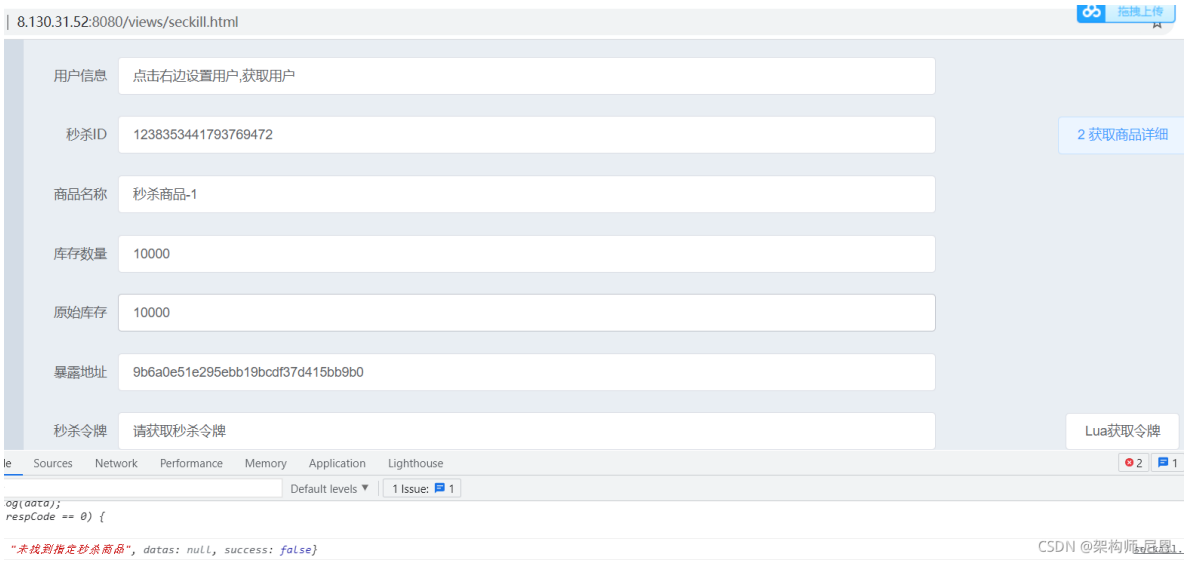
接入层秒杀

http://8.130.31.52:8080/views/seckill.html

获取商品详情

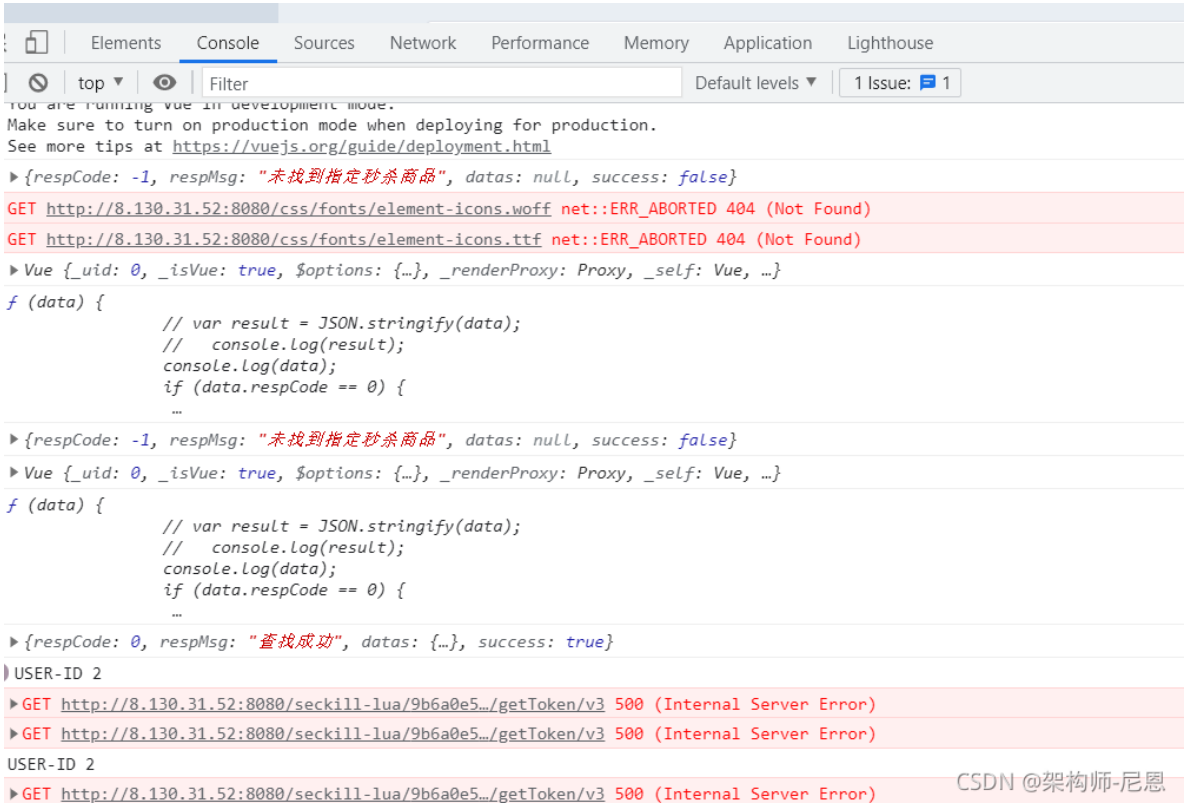
输入秒杀商品id

1238353441793769472



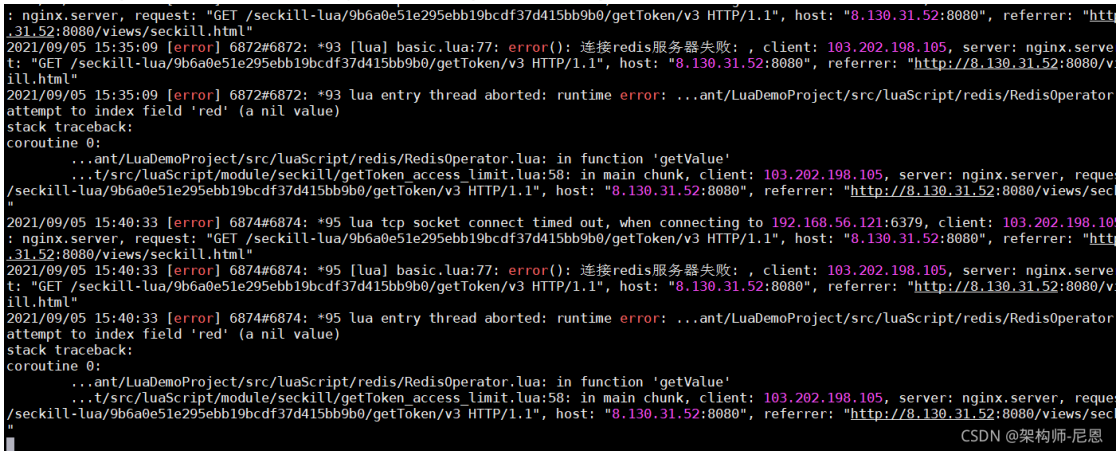
获取秒杀令牌

有错误，看日志



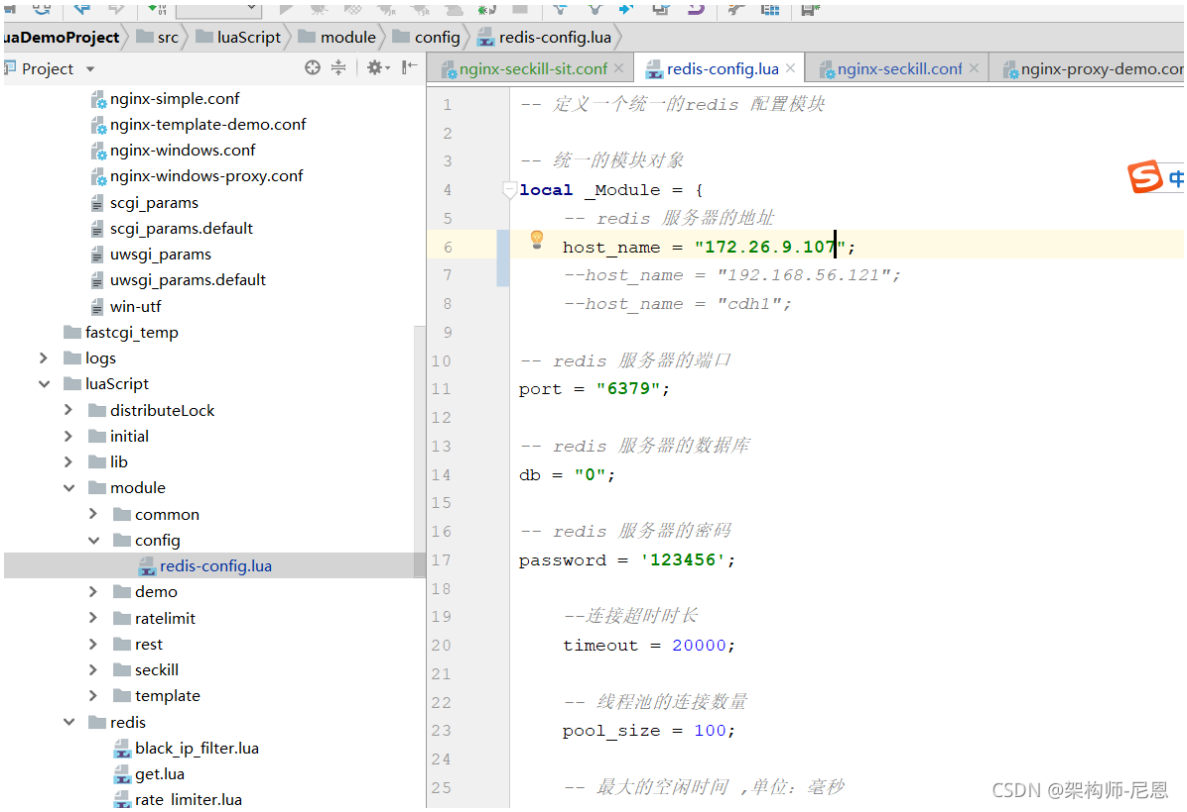
有错误，看日志

> tail -f /vagrant/LuaDemoProject/src/logs/error.log



找到ip：172.26.9.107

修改配置文件



上传并且，重启ng

云主机使用的问题

ssh不能登录

可能是sshd无法启动呢，然后输入这两行命令：

```
可采取以下两步解决
    chown -R root.root /var/empty/sshd
    chmod 744 /var/empty/sshd
    systemctl start sshd.service
    systemctl status sshd.service

netstat -ntlp | grep 22

    就可以解决上述的问题。

     如果还提示错误，查看empty的权限是否被修改
```

修改配置文件

按照实际情况，增加连接池的数量， 关闭日志

```
#user  nobody;
#worker_processes  1;
worker_processes  8;  #这个根据硬件有多少核CPU而定
```

```
#开发环境
#error_log  logs/error.log  debug;
#生产环境
error_log  logs/error.log;


pid     logs/nginx.pid;


events {
  worker_connections  1024000;
}


http {
  default_type 'text/html';
  charset utf-8;

  log_format  main  '$remote_addr - $remote_user [$time_local] "$request" '
  '$status $body_bytes_sent "$http_referer" '
  '"$http_user_agent" "$http_x_forwarded_for"';

  #access_log  logs/access_main.log  main;
  access_log off;   #日志功能要关闭
  #sendfile: 设置为on表示启动高效传输文件的模式。
  # sendfile可以让Nginx在传输文件时直接在磁盘和tcp socket之间传输数据。
  # 开启这个参数后可以让数据不用经过用户buffer。表示开启零复制，建议生产环境使用
  sendfile        off;
  #sendfile        on;
  #tcp_nopush     on;

  #keepalive_timeout  0;
  keepalive_timeout  65;

  #gzip  on;
  gzip off;
  #gzip最小长度 一般设置成1K就行，小于1K的就不压缩了 不然会越压越大
  gzip_min_length 1024;
  #gzip压缩比，1 压缩比最小处理速度最快，9 压缩比最大但处理最慢（传输快但比较消耗cpu）。
  gzip_comp_level 1;
  #匹配MIME类型进行压缩
  gzip_types text/plain application/json application/javascript application/x-
javascript text/css application/xml text/javascript application/x-httpd-php
image/jpeg image/gif image/png application/vnd.ms-fontobject font/ttf
font/opentype font/x-woff image/svg+xml  font/woff;
  gzip_vary on;
  gzip_disable "MSIE [1-6]\.";
  #设置系统获取几个单位的缓存用于存储gzip的压缩结 果数据流。 例如 4 4k 代表以4k为单位，按照原始数据大小以4k为单位的4倍申请内存。 4 8k 代表以8k为单位，按照原始数据大小以8k为单位的4倍申请内存。
  #如果没有设置，默认值是申请跟原始数据相同大小的内存空间去存储gzip压缩结果。
  gzip_buffers 4 8k;
  #gzip默认最低支持1.1现在改成最低支持1.0 近代浏览器基本不设置
  gzip_http_version 1.0;
  #Nginx作为反向代理的时候启用，开启或者关闭后端服务器返回的结果，匹配的前提是后端服务器必须要返回包含"Via"的 header头。
  #off - 关闭所有的代理结果数据的压缩
  #expired - 启用压缩，如果header头中包含 "Expires" 头信息
```

```
  #no-cache - 启用压缩，如果header头中包含 "Cache-Control:no-cache" 头信息
  #no-store - 启用压缩，如果header头中包含 "Cache-Control:no-store" 头信息
  #private - 启用压缩，如果header头中包含 "Cache-Control:private" 头信息
  #no_last_modified - 启用压缩,如果header头中不包含 "Last-Modified" 头信息
  #no_etag - 启用压缩 ,如果header头中不包含 "ETag" 头信息
  #auth - 启用压缩 , 如果header头中包含 "Authorization" 头信息
  #any - 无条件启用压缩
  gzip_proxied expired no-cache no-store private auth;


  #指定缓存信息
  lua_shared_dict ngx_cache 128m;
  #指定缓存信息
  lua_shared_dict seckill_cache 128m;
  #保证只有一个线程去访问redis或是mysql-lock for cache
  lua_shared_dict cache_lock 100k;
  #lua扩展加载

  # for linux
  # lua_package_path
"./?.lua;/vagrant/LuaDemoProject/src/?.lua;/usr/local/ZeroBraneStudio-
1.80/?/?.lua;/usr/local/ZeroBraneStudio-1.80/?.lua;;";
  # lua_package_cpath "/usr/local/ZeroBraneStudio-1.80/bin/clibs/?.so;;";
  lua_package_path
"./?.lua;/vagrant/LuaDemoProject/src/?.lua;/vagrant/LuaDemoProject/vendor/templa
te/?.lua;/vagrant/LuaDemoProject/src/?/?.lua;/usr/local/openresty/lualib/?/?.lua
;/usr/local/openresty/lualib/?.lua;;";
  lua_package_cpath
"/usr/local/openresty/lualib/?/?.so;/usr/local/openresty/lualib/?.so;;";

  # for windows
  # lua_package_path
"./?.lua;C:/dev/refer/LuaDemoProject/src/vendor/jwt/?.lua;C:/dev/refer/LuaDemoPr
oject/src/?.lua;E:/tool/ZeroBraneStudio-
1.80/lualibs/?/?.lua;E:/tool/ZeroBraneStudio-
1.80/lualibs/?.lua;E:/tool/openresty-1.13.6.2-win32/lualib/?.lua;;";
  # lua_package_cpath "E:/tool/ZeroBraneStudio-
1.80/bin/clibs/?.dll;E:/tool/openresty-1.13.6.2-win32/lualib/?.dll;;";


  # 初始化项目
  init_by_lua_file luaScript/initial/loading_config.lua;

  #调试模式（即关闭lua脚本缓存)
  lua_code_cache on;
  #lua_code_cache off;

  #内部网关的代理，内部网关带有 token 认证
  upstream zuul {
    # idea 开发环境
    #    server 192.168.56.121:7799;
    # centos 自验证环境
    server "cdh1:8888";
    keepalive 1000;
  }

  underscores_in_headers on;
```

```
  limit_req_zone  $arg_sku_id  zone=skuzone:10m      rate=6r/m;
  limit_req_zone  $http_user_id  zone=userzone:10m      rate=6r/m;
  limit_req_zone  $binary_remote_addr  zone=perip:10m      rate=6r/m;
  limit_req_zone  $server_name        zone=perserver:1m   rate=10r/s;
  limit_req_zone  $server_name        zone=seckill_server:1m   rate=50000r/s;

  server {
    listen       80;
    server_name  admin.nginx.server;
    default_type 'text/html';
    charset utf-8;


    limit_req  zone=perip;
    limit_req  zone=perserver;

    #管理控制台的web页面
    location  / {
      if ($request_method = 'OPTIONS') {
        add_header Access-Control-Max-Age 600000;
        add_header Access-Control-Allow-Origin *;
        add_header Access-Control-Allow-Credentials true;
        add_header Access-Control-Allow-Methods 'GET, POST, OPTIONS';
        add_header 'Access-Control-Allow-Headers' 'DNT, X-Mx-ReqToken, Keep-
Alive, User-Agent, X-Requested-With, If-Modified-Since, Cache-Control, Content-
Type, token';
        return 204;
      }
      # IDEA 管理控制台的web页面
      proxy_pass http://192.168.56.121:8066/ ;
      # proxy_pass http://zuul;
    }
  }


  server {
    listen       80 default;
    server_name  nginx.server *.nginx.server;
    default_type 'text/html';
    charset utf-8;

    # 转发 zuul
    location / {
      if ($request_method = 'OPTIONS') {
        add_header Access-Control-Max-Age 600000;
        add_header Access-Control-Allow-Origin *;
        add_header Access-Control-Allow-Credentials true;
        add_header Access-Control-Allow-Methods 'GET, POST, OPTIONS';
        add_header 'Access-Control-Allow-Headers' 'DNT, X-Mx-ReqToken, Keep-
Alive, User-Agent, X-Requested-With, If-Modified-Since, Cache-Control, Content-
Type, token';
        return 204;
      }

      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forward-For $proxy_add_x_forwarded_for;
      proxy_set_header Host $http_host;
      proxy_set_header X-Nginx-Proxy true;
```

```
    proxy_pass http://zuul;
  }

  # 开发调试： 用户服务
  location  ^~ /uaa-provider/ {
    # idea 开发环境
    #  proxy_pass http://192.168.56.121:7702/;
    # centos 自验证环境
    proxy_pass http://192.168.56.121:7702/uaa-provider/ ;
  }

  # 开发调试： 秒杀服务
  location  ^~ /seckill-provider/ {
    proxy_pass http://192.168.56.121:7701/seckill-provider/ ;
  }



  # 开发调试： 管理服务
  location  ^~ /backend-provider/ {
    proxy_set_header X-Real-IP $remote_addr;
    proxy_set_header X-Forward-For $proxy_add_x_forwarded_for;
    proxy_set_header Host $http_host;
    proxy_set_header backend  'true'; # 给后台的请求加上自定义头，方便session区分
    proxy_set_header X-Nginx-Proxy true;
    # 指向微服务
    proxy_pass http://192.168.56.121:6600/backend-provider/ ;

    #指向gate way
    # proxy_pass http://zuul;
  }

  # 反向代理：秒杀web页面
  location  ^~ /seckill-web/ {
    proxy_pass http://192.168.56.121:6601/seckill-web/ ;
  }

  # Nginx+lua 秒杀：获取秒杀 token
  location = /seckill-provider/api/seckill/redis/token/v2 {
    default_type 'application/json';
    charset utf-8;
    # 限流的 lua 脚本
    access_by_lua_file luaScript/module/seckill/getToken_access_limit.lua;
    # 获取秒杀token lua 脚本
    content_by_lua_file luaScript/module/seckill/getToken.lua;
  }


  #  ratelimit by sku id
  location  = /ratelimit/sku {
    limit_req  zone=skuzone;
    echo "正常的响应";
  }

  #  ratelimit by user id
  location  = /ratelimit/demo {
    limit_req  zone=userzone;
    echo "正常的响应";
```

```
    }


    location = /50x.html{
      echo "限流后的降级内容";
    }

    error_page 502 503 =200 /50x.html;

    #  访问的路径 http://cdh1/ratelimit/demo2?seckillSkuId=3
    #  ratelimit by sku id
    location  = /ratelimit/demo2 {
      # 限流的 lua 脚本
      access_by_lua_file luaScript/module/seckill/getToken_access_limit.lua;

      echo "正常的响应";
    }




  }

  server {
    listen       8080 default;
    server_name  nginx.server *.nginx.server ;
    limit_req  zone=seckill_server;


    #第一个lua脚本 hellword
    location /helloworld {
      default_type 'text/html';
      charset utf-8;
      content_by_lua_file luaScript/module/demo/helloworld.lua;
    }

    location / {         # 自动匹配到(htm|html)格式
      ## 开发阶段，配置页面不缓存html和htm结尾的文件
      add_header Cache-Control "private, no-store, no-cache, must-revalidate,
proxy-revalidate";
      index index.html;
      root /vagrant/LuaDemoProject/src/www/static; #服务器路径
      default_type 'text/html';
    }

    location ~ .*\.(htm|html)$ {         # 自动匹配到(htm|html)格式
      ## 开发阶段，配置页面不缓存html和htm结尾的文件
      add_header Cache-Control "private, no-store, no-cache, must-revalidate,
proxy-revalidate";
      root /vagrant/LuaDemoProject/src/www/static; #服务器路径
      default_type 'text/html';
    }


    location ~ .*\.(js|script)$ {         # 自动匹配到(jpg|gif|png)格式
      add_header Cache-Control "private, no-store, no-cache, must-revalidate,
proxy-revalidate";
      root /vagrant/LuaDemoProject/src/www/static; #服务器路径
```

```
      default_type 'application/javascript';
    }




    location ~ .*\.(css)$ {         # 自动匹配到(css)格式
      root /vagrant/LuaDemoProject/src/www/static; #服务器路径
      default_type 'text/css';
    }


    # 开发调试：  库存服务
    location  ^~ /stock-provider/ {
      proxy_pass http://zuul/stock-provider/ ;
    }

    # 开发调试：  秒杀服务
    location  ^~ /seckill-provider/ {
      #      proxy_pass http://localhost:7701/seckill-provider/ ;
      #指向gate way
      proxy_pass http://zuul/seckill-provider/;
    }


    # Nginx+lua ：获取 商品信息
    location = /stock-lua/gooddetail {
      default_type 'application/json';
      charset utf-8;
      # 限流的 lua 脚本
      #access_by_lua_file luaScript/module/seckill/getToken_access_limit.lua;
      # 获取秒杀token lua 脚本
      content_by_lua_file luaScript/module/seckill/good_detail.lua;
    }


    # Nginx+lua 秒杀：获取秒杀 token
    location  ~ /seckill-lua/(.*)/getToken/v3 {
      default_type 'application/json';
      charset utf-8;
      set $skuId $1;
      limit_req  zone=userzone;

      # 限流的 lua 脚本
      access_by_lua_file luaScript/module/seckill/getToken_access_limit.lua;
      # 获取秒杀token lua 脚本
      content_by_lua_file luaScript/module/seckill/getToken_v3.lua;
    }

  }

}
```

更改启动脚本

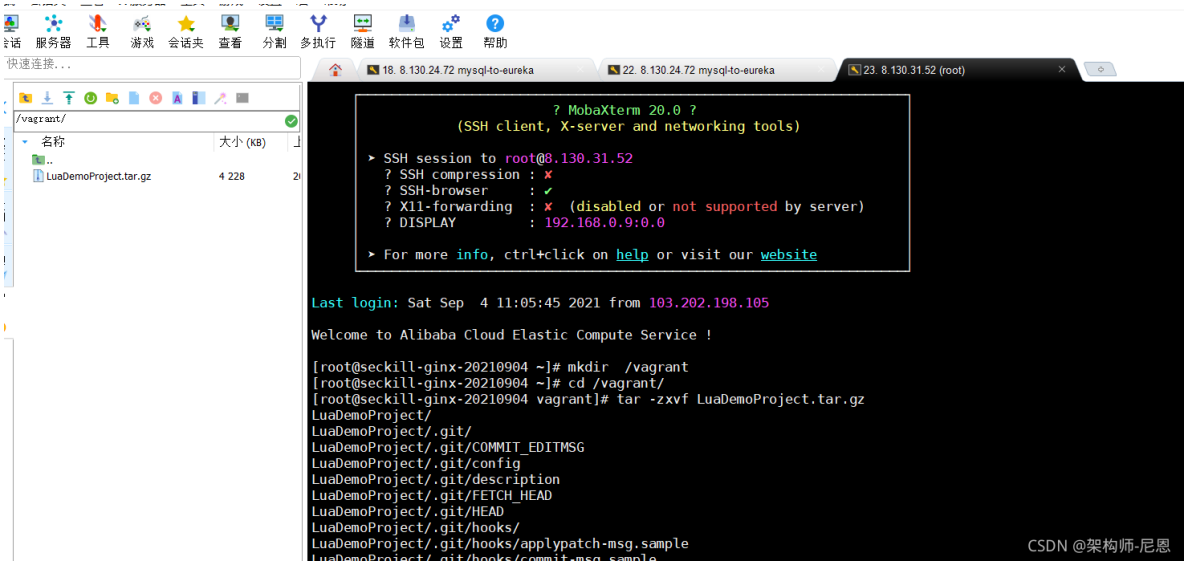
```
# PROJECT_CONF="nginx-seckill.conf"
PROJECT_CONF="nginx-seckill-sit.conf"
```

打包

```
MINGW64 /e/virtual/workcluster
$ tar -zcvf LuaDemoProject.tar.gz LuaDemoProject
```

创建目录并且启动

```
[root@seckill-ginx-20210904 ~]# mkdir  /vagrant
[root@seckill-ginx-20210904 ~]# cd /vagrant/
[root@seckill-ginx-20210904 vagrant]# tar -zxvf LuaDemoProject.tar.gz
```



增加域名解析，然后启动

```
[root@seckill-ginx-20210904 vagrant]# cat  >> /etc/hosts <<EOF
> 172.26.9.107 cdh1
> 192.168.56.122 cdh2
> 192.168.56.123 cdh3
> EOF
[root@seckill-ginx-20210904 vagrant]# sh
/vagrant/LuaDemoProject/sh/linux/openresty-restart.sh
shell dir is: /vagrant/LuaDemoProject/sh/linux
openrestry/nginx is not running!
OPENRESTRY_PATH:/usr/local/openresty
PROJECT_PATH:/vagrant/LuaDemoProject/src
openrestry/nginx starting succeeded!
pid is 3161 3162 3163 3164 3165 3166 3167 3168 3169
```

```
cat  >> /etc/hosts <<EOF
172.26.9.107 cdh1
172.26.9.105 cdh2
192.168.56.123 cdh3
EOF
```

sh /vagrant/LuaDemoProject/sh/linux/openresty-restart.sh

测试一下访问的端口：

```
[root@seckill-ginx-20210904 vagrant]# curl http://cdh2:8080/echo
echo


[root@seckill-ginx-20210904 ~]# curl http://cdh2:8080/echo -X POST
```

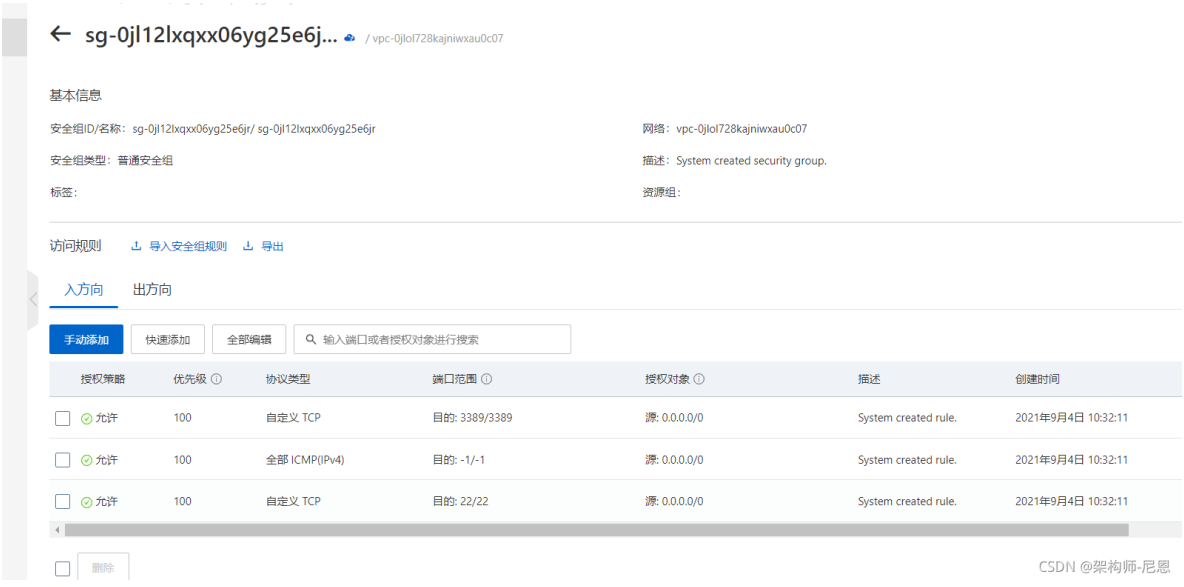
开放端口，然后访问

开放端口

实例详情

安全组

配置规则



端口范围

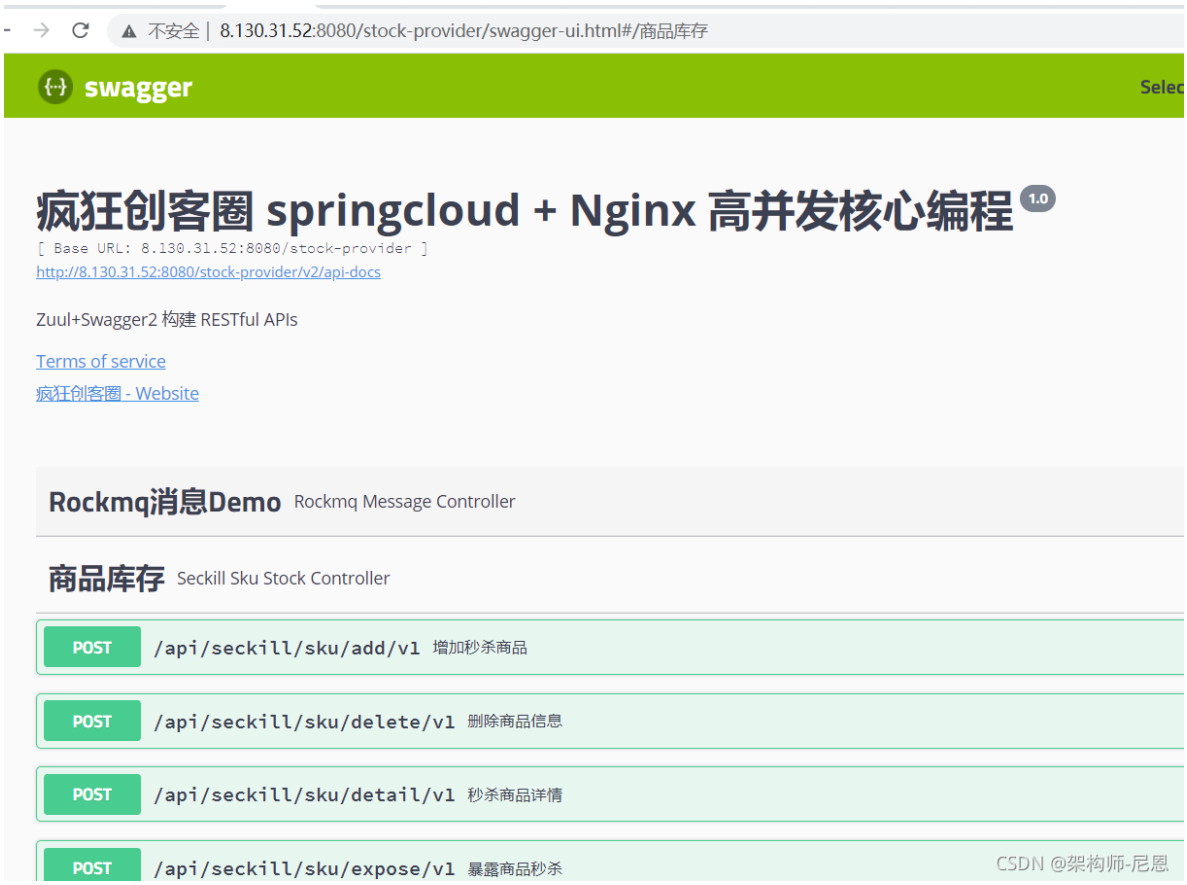
然后访问

```
curl http://8.130.31.52:8080/echo
```

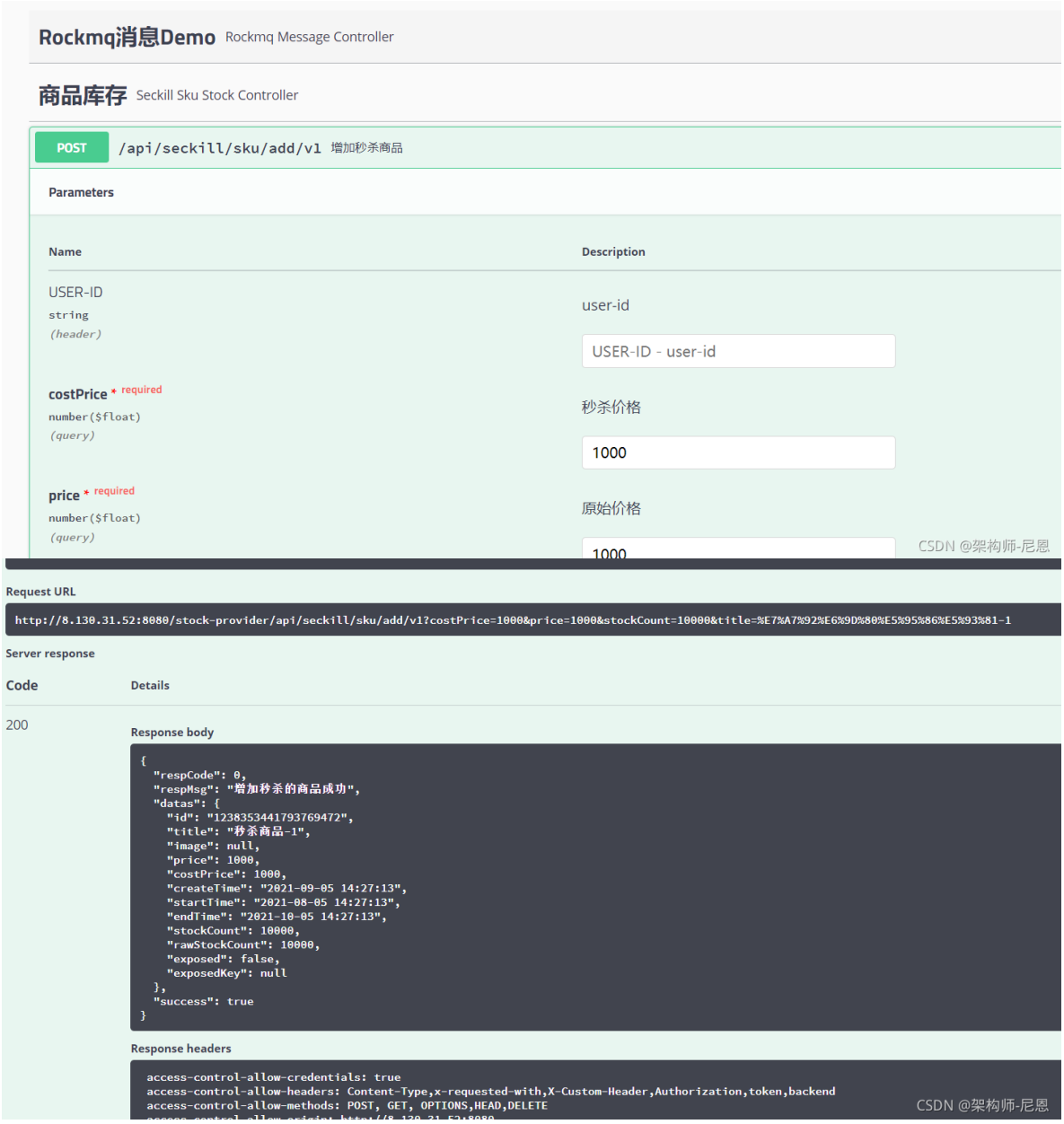
浏览器访问库存服务：

```
http://8.130.31.52:8080/stock-provider/swagger-ui.html

http://8.130.31.52:8080/stock-provider/swagger-ui.html
```

秒杀商品添加



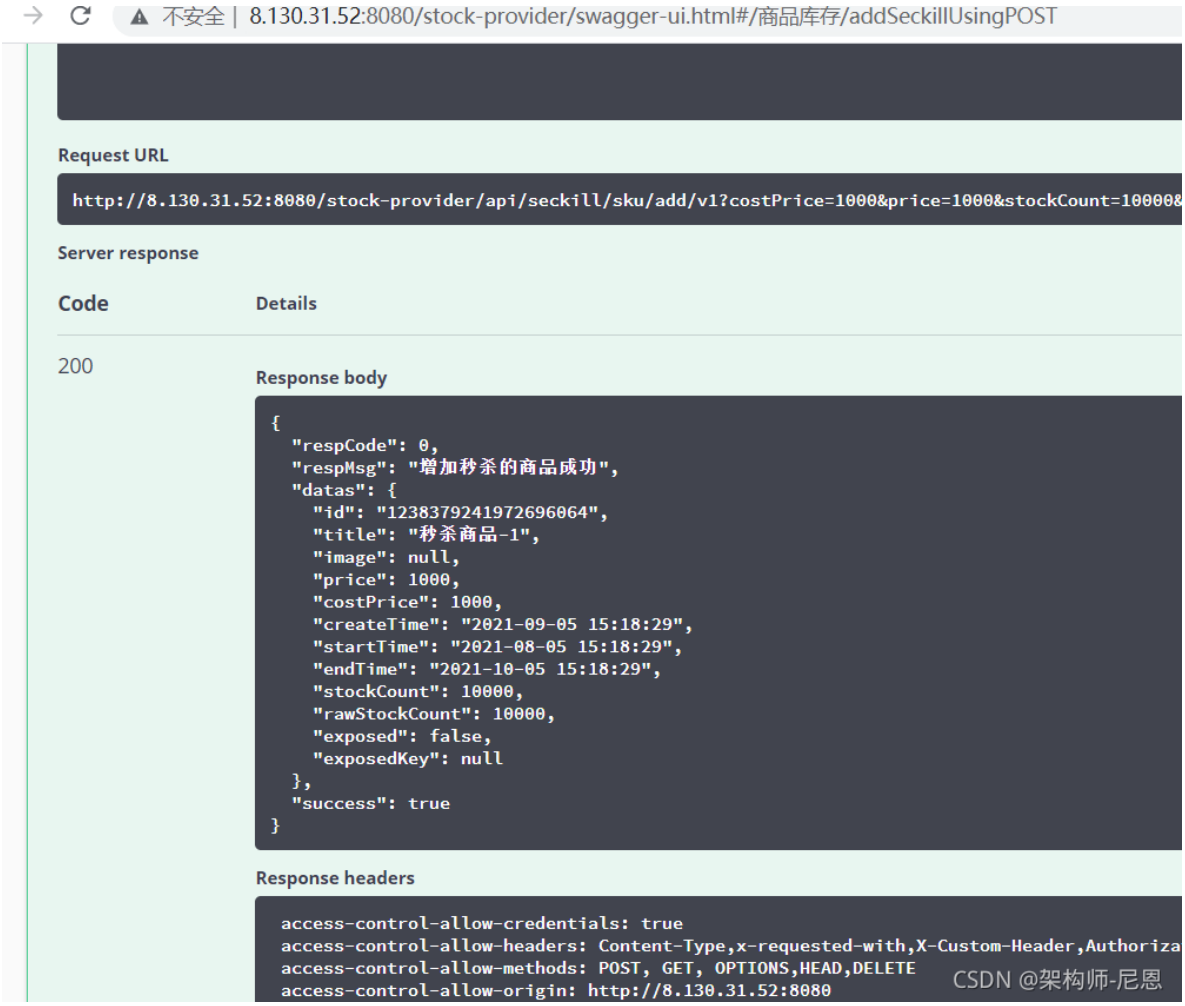
增加秒杀的商品



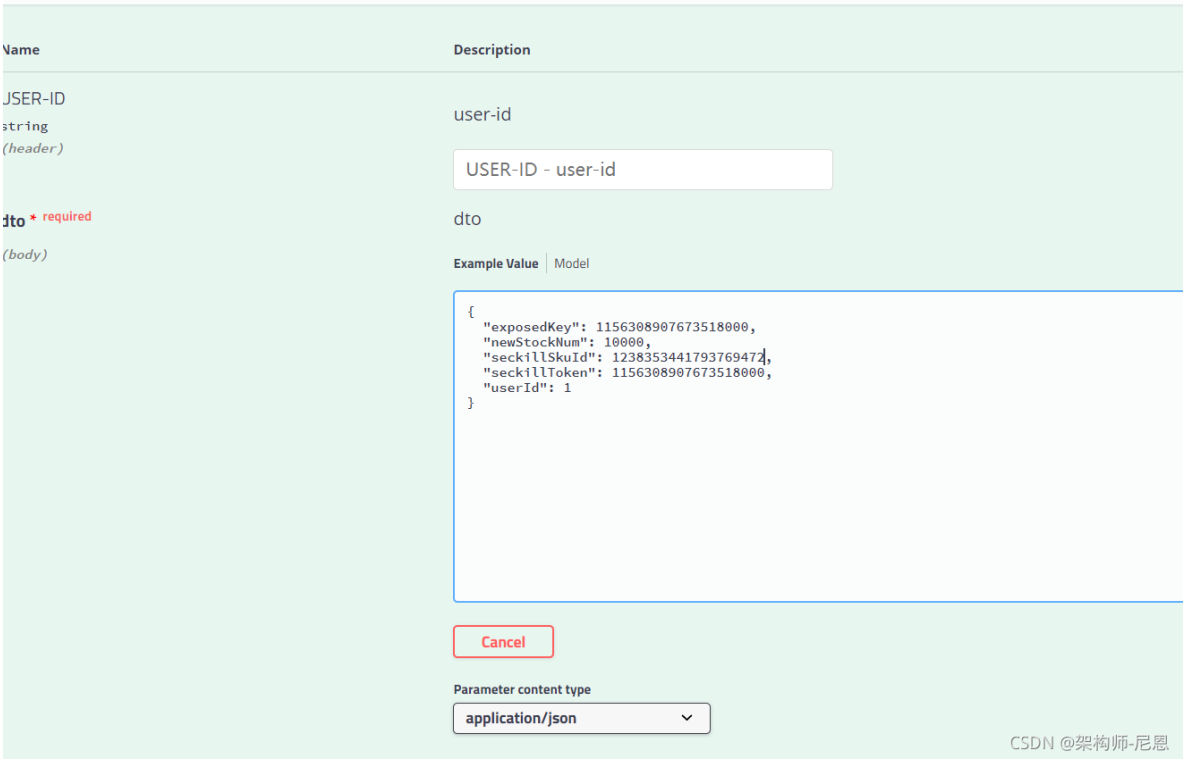
id: 1238353441793769472

添加成功，记住id

> 1238353441793769472

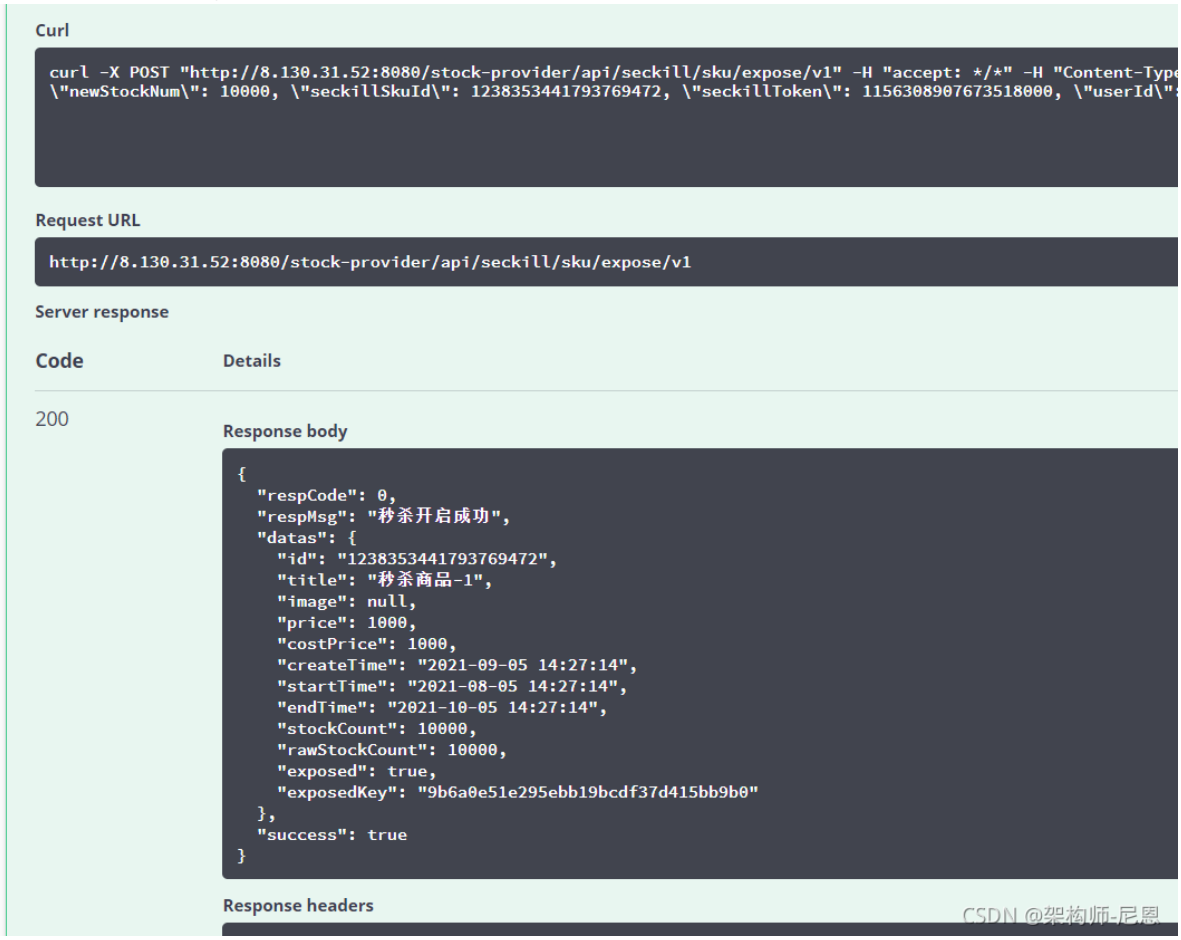

不安全 | 8.130.31.52:8080/stock-provider/swagger-ui.html#/商品库存/addSeckillUsingPOST
Request URL
http://8.130.31.52:8080/stock-provider/api/seckill/sku/add/v1?costPrice=1000&price=1000&stockCount=10000&
Server response
Code
Details
200
Response body
{
"respCode": 0,
"respMsg": "增加秒杀的商品成功",
"datas": {
"id": "1238379241972696064",
"title": "秒杀商品-1",
"image": null,
"price": 1000,
"costPrice": 1000,
"createTime": "2021-09-05 15:18:29",
"startTime": "2021-08-05 15:18:29",
"endTime": "2021-10-05 15:18:29",
"stockCount": 10000,
"rawStockCount": 10000,
"exposed": false,
"exposedKey": null
},
"success": true
}
Response headers
access-control-allow-credentials: true
access-control-allow-headers: Content-Type,x-requested-with,X-Custom-Header,Authoriza
access-control-allow-methods: POST, GET, OPTIONS,HEAD,DELETE
access-control-allow-origin: http://8.130.31.52:8080
CSDN @架构师-尼恩

秒杀暴露


Name
Description
USER-ID
string
(header)
user-id
USER-ID - user-id
dto * required
(body)
dto
Example Value
Model
{
"exposedKey": 1156308907673518000,
"newStockNum": 10000,
"seckillSkuId": 1238353441793769472,
"seckillToken": 1156308907673518000,
"userId": 1
}
Cancel
Parameter content type
application/json
CSDN @架构师-尼恩

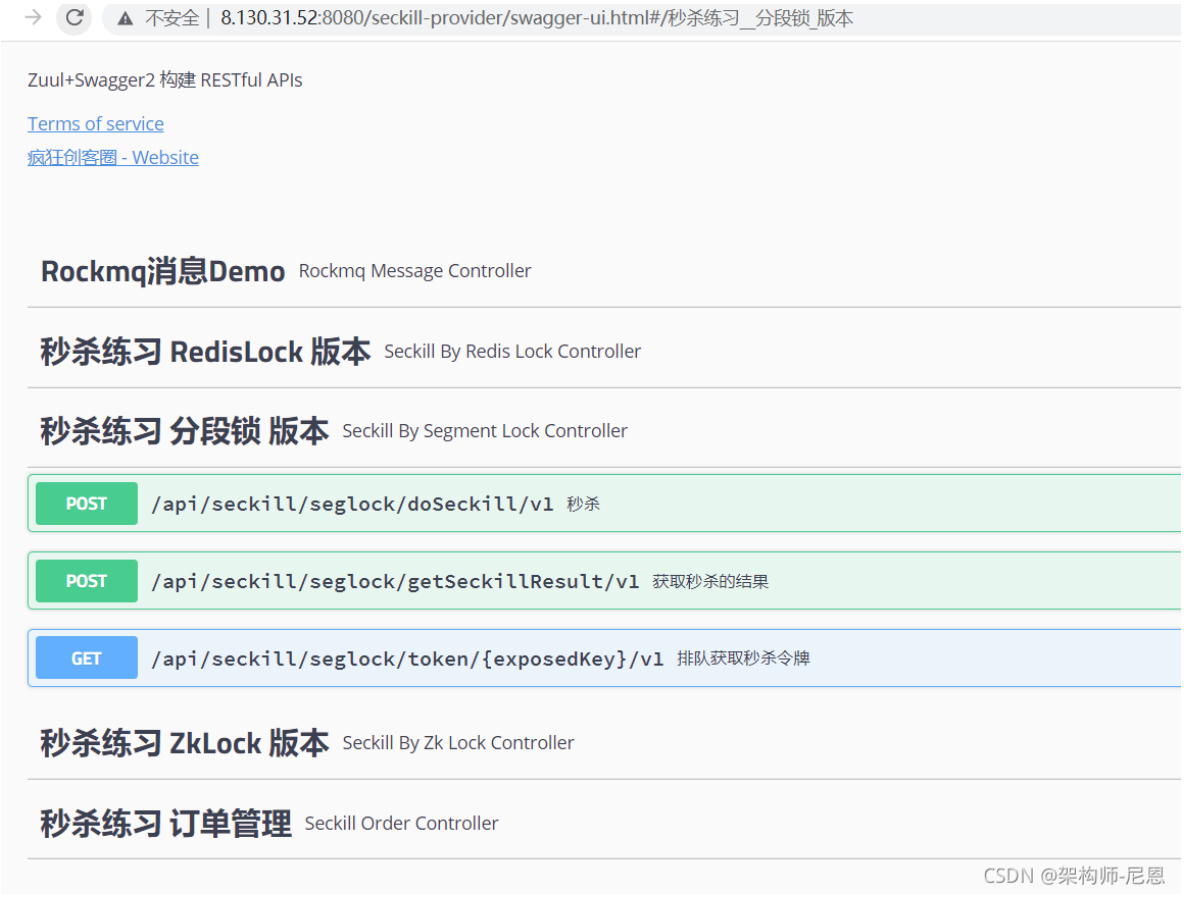
暴露成功

记住 秒杀的暴露key： 9b6a0e51e295ebb19bcdf37d415bb9b0

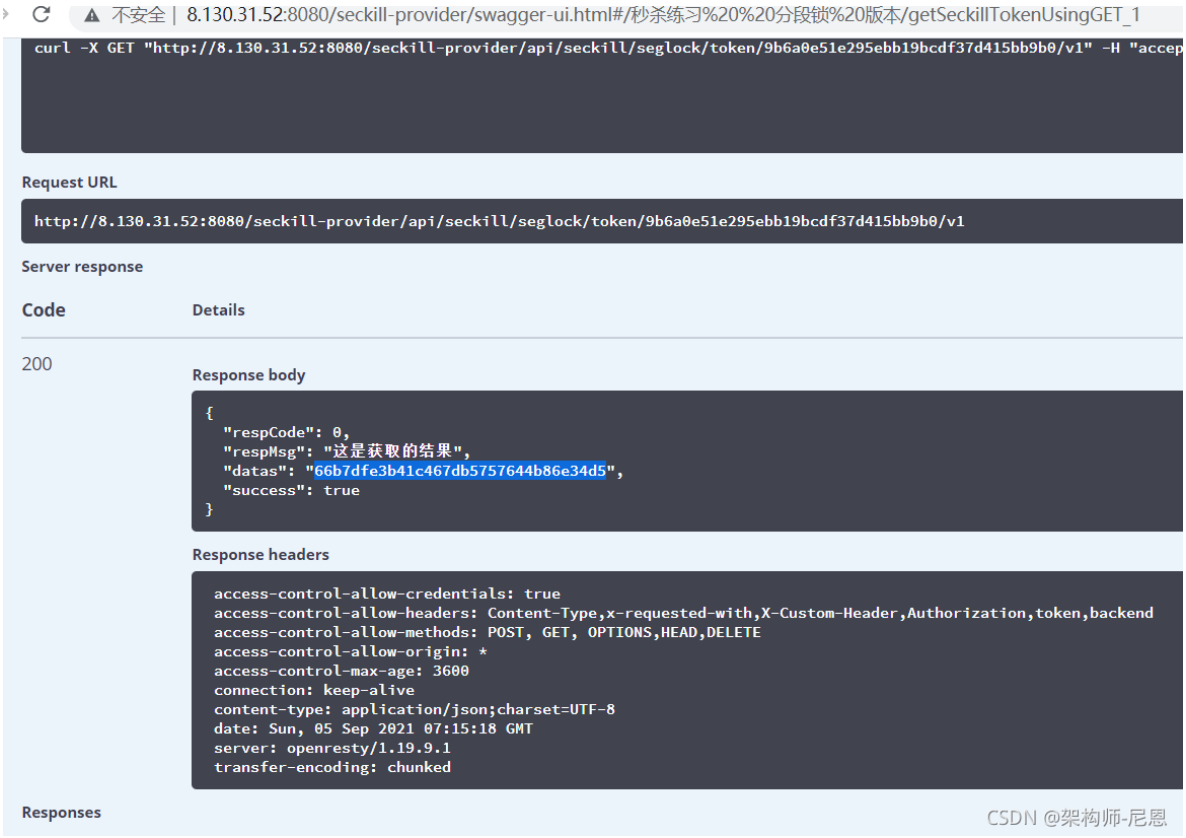


获取秒杀令牌

http://8.130.31.52:8080/seckill-provider/swagger-ui.html#/



获取令牌成功



高并发场景slb的使用实操

SLB概念

负载均衡（Server Load Balancer）是将访问流量根据转发策略分发到后端多台云服务器（Elastic Compute Service，简称 ECS）的流量分发控制服务。
负载均衡服务通过设置虚拟服务地址，将位于同一地域的多台ECS实例虚拟成一个高性能、高可用的应用服务池；再根据应用指定的方式，将来自客户端的网络请求分发到云服务器池中。负载均衡服务是ECS面向多机方案的一个配套服务，需要同ECS结合使用。
负载均衡服务会检查云服务器池中ECS实例的健康状态，自动隔离异常状态的ECS实例，从而解决了单台ECS实例的单点问题，提高了应用的整体服务能力。

在标准的负载均衡功能之外，负载均衡服务还具备TCP与HTTP抗DDoS攻击的特性，增强了应用服务的防护能力。

参考文档

负载均衡（阿里云帮助与文档）：

https://developer.aliyun.com/article/697165

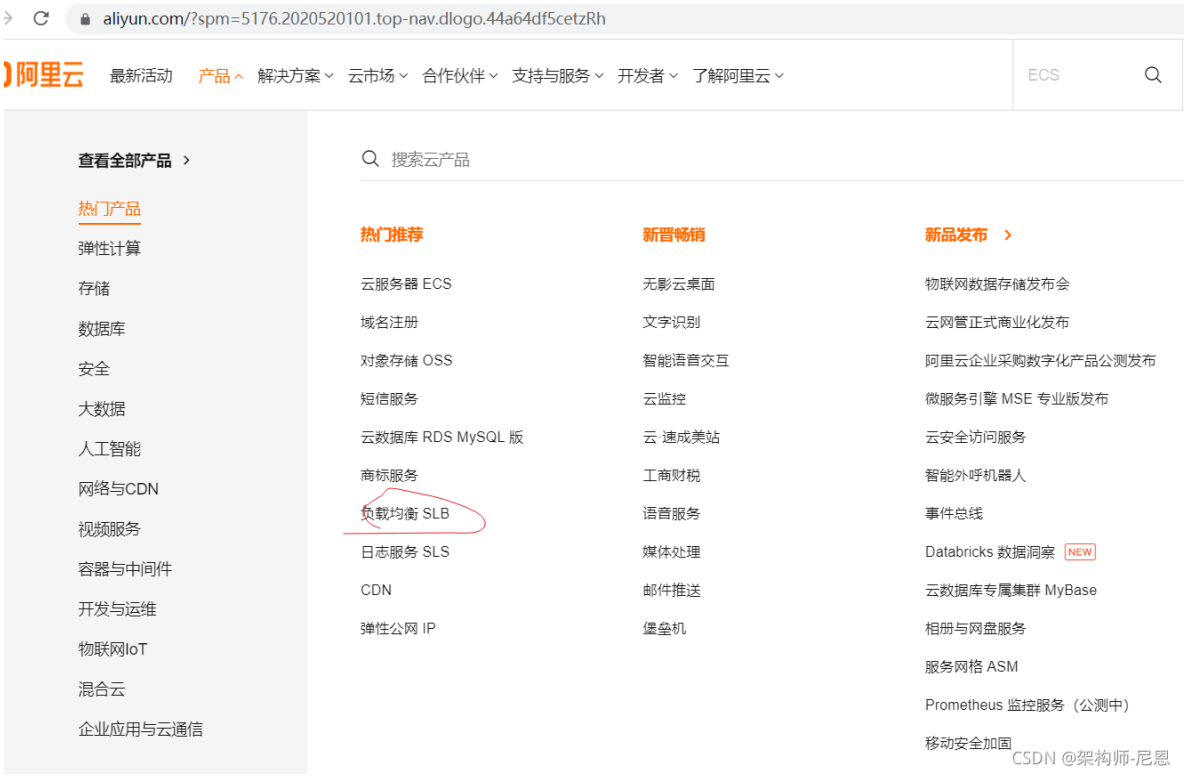
https://help.aliyun.com/document_detail/86451.html?source=5176.11533457&userCode=ffsbbyn0

负载均衡（SLB）使用最佳实践

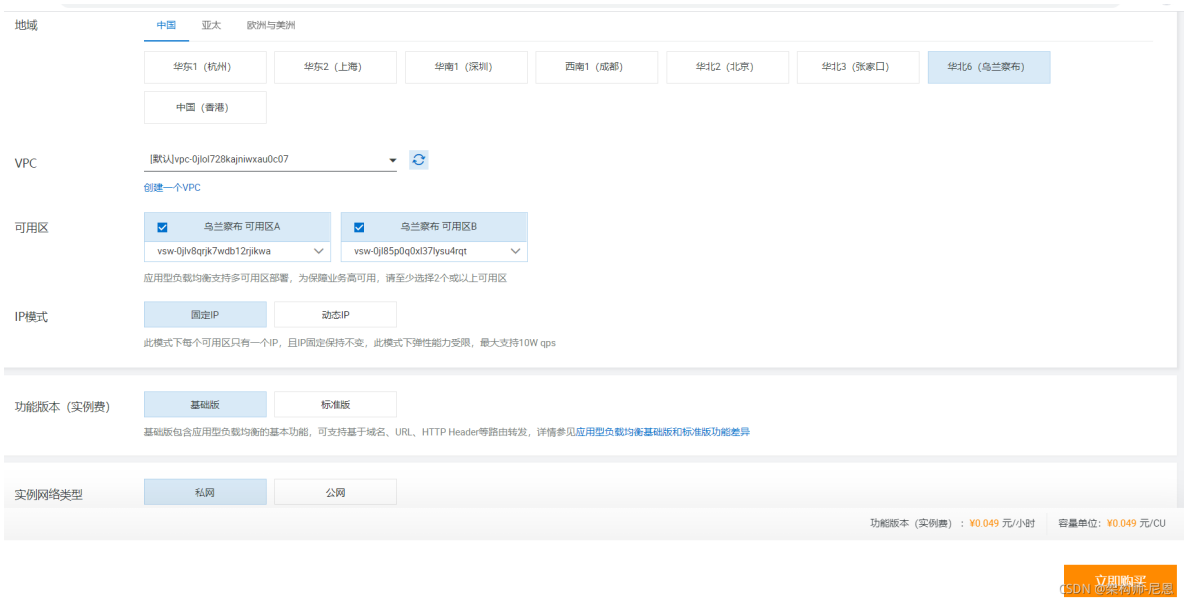
https://developer.aliyun.com/article/80055

一个slb至少需要两个可用区的机器

购买slb的服务



选择可用区



slb配置

三个部分组成

负载均衡服务由负载均衡实例、监听和后端服务器三个部分组成。

负载均衡实例 （Server Load Balancer Instance）

如果您想使用负载均衡服务，必须先创建一个负载均衡实例。一个负载均衡实例可以添加多个监听和后端服务器。

监听 （Listener）

在使用负载均衡服务前，您必须为负载均衡实例添加一个监听，指定监听规则和转发策略，并配置健康检查。
针对不同的需求，您可以单独配置四层（TCP/UDP）或七层（HTTP/HTTPS）监听。

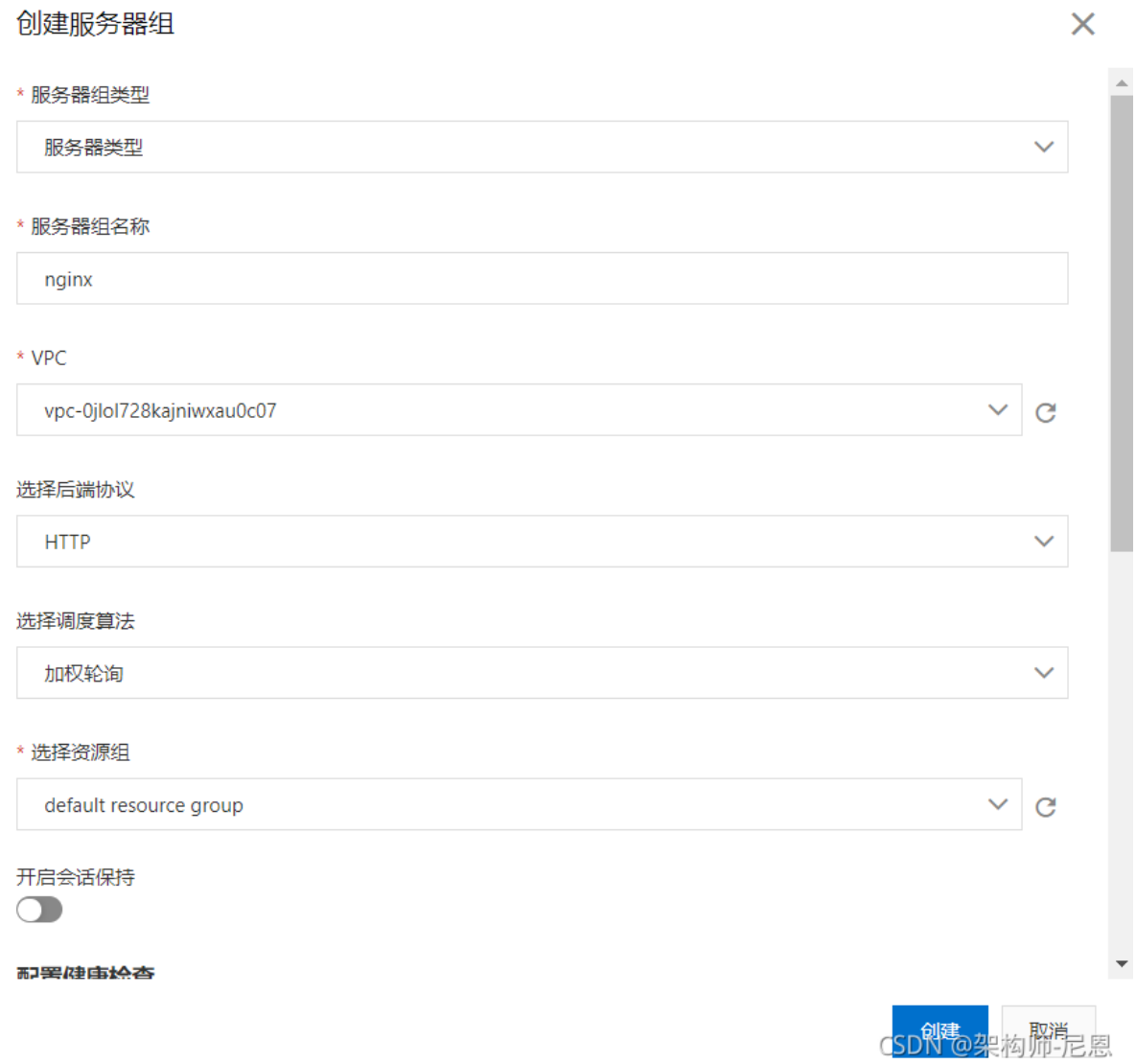
后端服务器（Backend Server）

一组接收前端请求的ECS实例。您可以单独添加ECS实例到服务器池，也可以通过虚拟服务器组或主备服务器组来批量添加和管理。
默认后端服务器是在实例维度上维护的，即负载均衡实例下的所有监听都只能够将请求转发到相同ECS实例的相同端口上。虚拟服务器组功能实现了监听维度的转发。您可以针对不同的监听创建不同的虚拟服务器组，即负载均衡实例中的不同监听可以将请求转发到不同端口的后端服务器上。
此外，七层负载均衡服务支持域名、URL转发策略，可以将来自不同域名或者URL的请求转发给不同的后端服务器处理。

负载均衡实例配置

配置端口监听

配置服务器组



配置审核

后端服务器

启动可用区A和B上的ng

连接上两个ecs， 安装ng，复制vagrant目录过去，然后启动

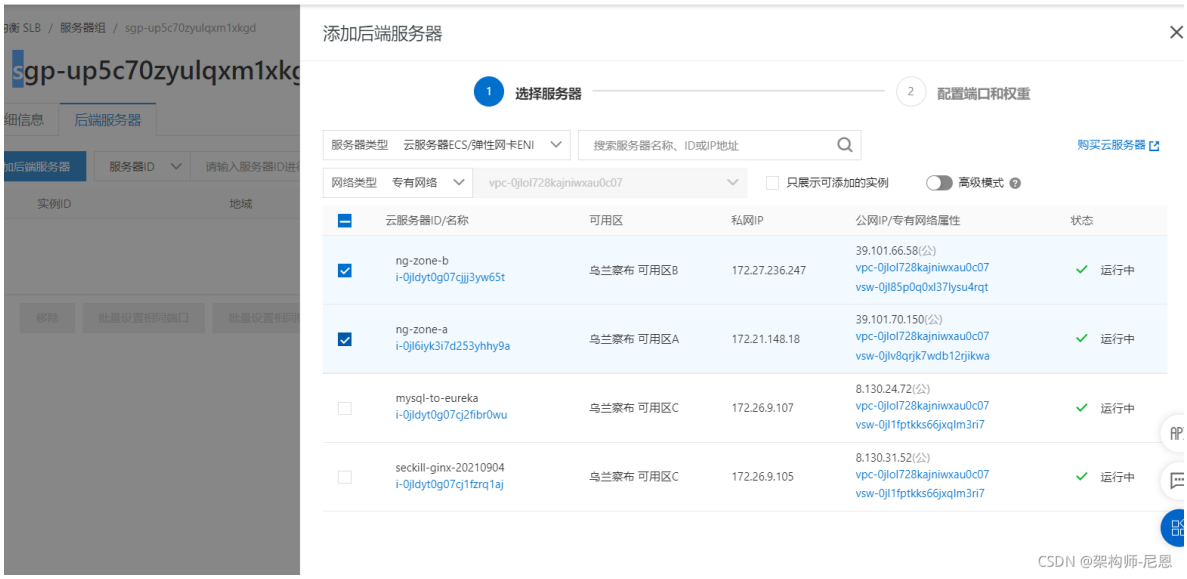
```
mkdir -p  /vagrant
```

```
scp -r /vagrant/  root@172.27.236.247:/
scp -r /vagrant/  root@172.21.148.18:/


sh /vagrant/LuaDemoProject/sh/linux/openresty-restart.sh


curl http://localhost:8080/echo
```

slb服务器组管理

添加后端服务器

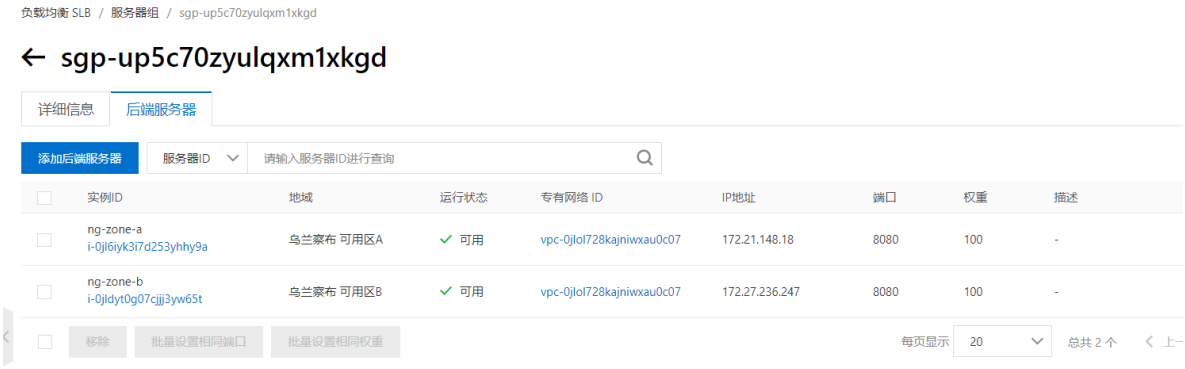
选择两接入层服务器



填写端口

提交成功

服务器组内的服务



测试

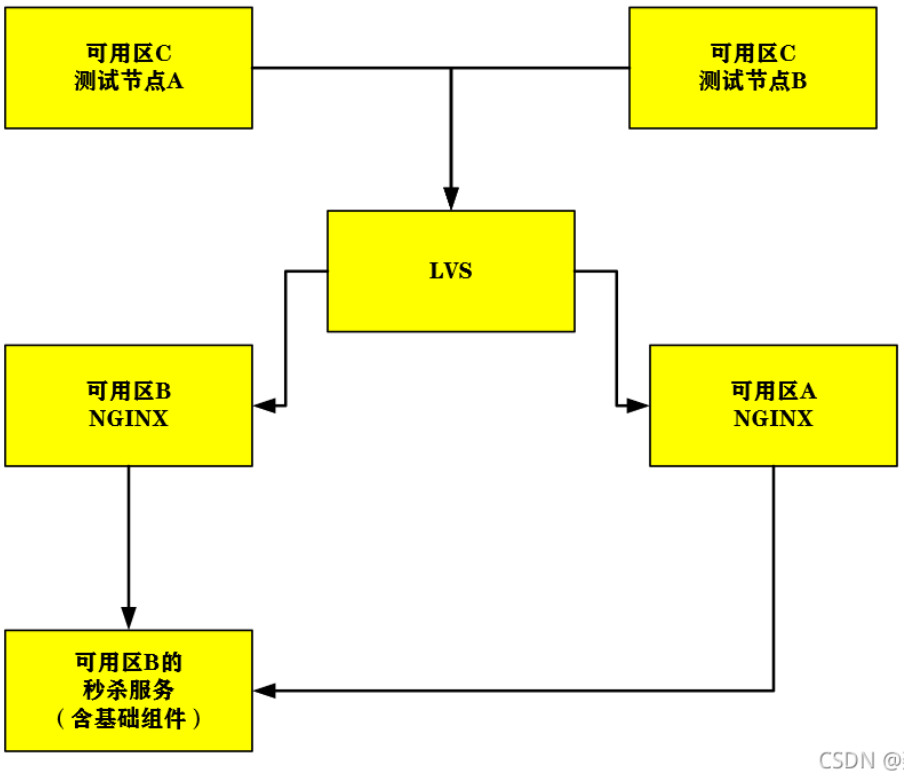
```
[root@nginx-zone-a vagrant]# ping alb-jvoaudpfrhgx7l3d1t.internal.cn-
wulanchabu.alb.aliyuncs.com
PING alb-jvoaudpfrhgx7l3d1t.internal.cn-wulanchabu.alb.aliyuncs.com
(172.21.148.19) 56(84) bytes of data.
64 bytes from 172.21.148.19 (172.21.148.19): icmp_seq=1 ttl=102 time=0.191 ms
64 bytes from 172.21.148.19 (172.21.148.19): icmp_seq=2 ttl=102 time=0.126 ms
64 bytes from 172.21.148.19 (172.21.148.19): icmp_seq=3 ttl=102 time=0.099 ms
64 bytes from 172.21.148.19 (172.21.148.19): icmp_seq=4 ttl=102 time=0.097 ms
64 bytes from 172.21.148.19 (172.21.148.19): icmp_seq=5 ttl=102 time=0.100 ms
64 bytes from 172.21.148.19 (172.21.148.19): icmp_seq=6 ttl=102 time=0.099 ms
64 bytes from 172.21.148.19 (172.21.148.19): icmp_seq=7 ttl=102 time=0.095 ms
^C
--- alb-jvoaudpfrhgx7l3d1t.internal.cn-wulanchabu.alb.aliyuncs.com ping
statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6000ms
rtt min/avg/max/mdev = 0.095/0.115/0.191/0.033 ms
```

测试接口

curl http://alb-jvoaudpfrhgx7l3d1t.internal.cn-wulanchabu.alb.aliyuncs.com:8080/echo

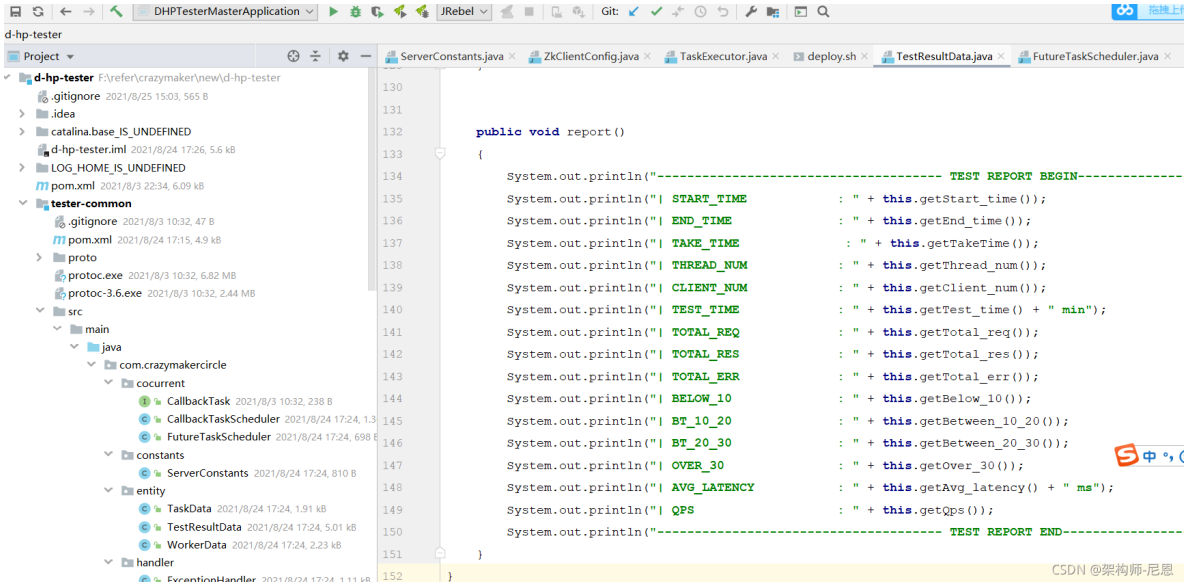
```
[root@nginx-zone-a vagrant]# curl http://alb-jvoaudpfrhgx7l3d1t.internal.cn-
wulanchabu.alb.aliyuncs.com:8080/echo
echo
[root@nginx-zone-a vagrant]#
```

10WQPS压力测试实操



工具的源码路径

https://gitee.com/crazymaker/d-hp-tester.git



本地验证一下 压力测试的配置文件

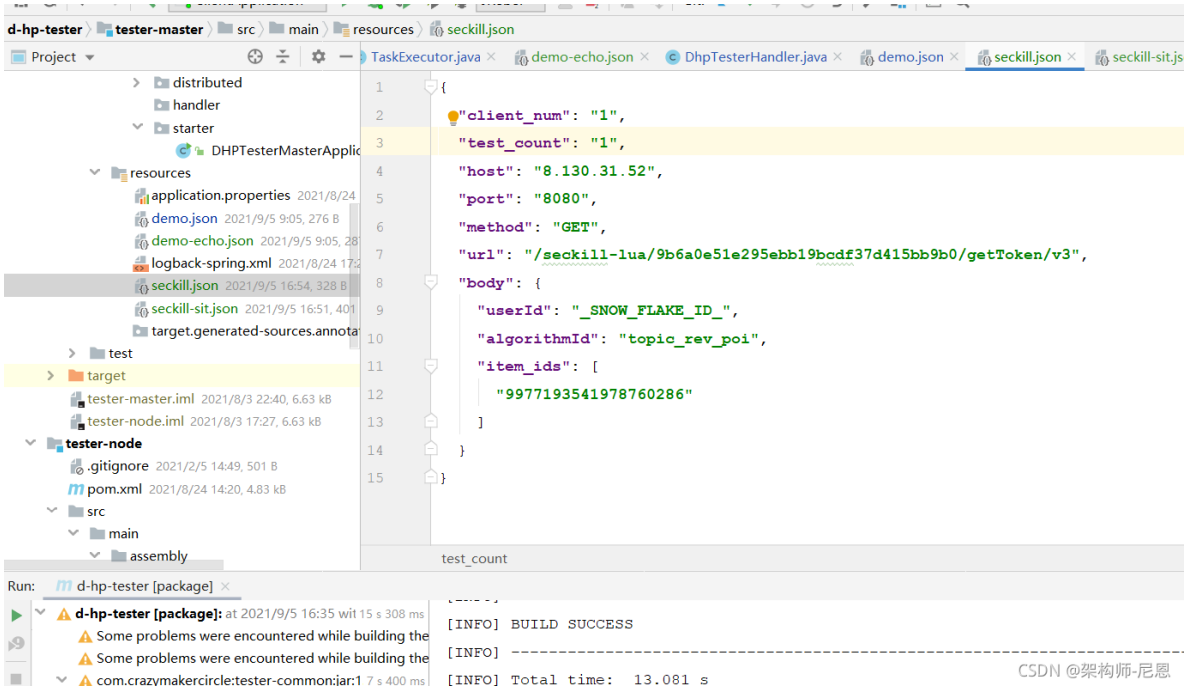
修改高并发的用例配置文件

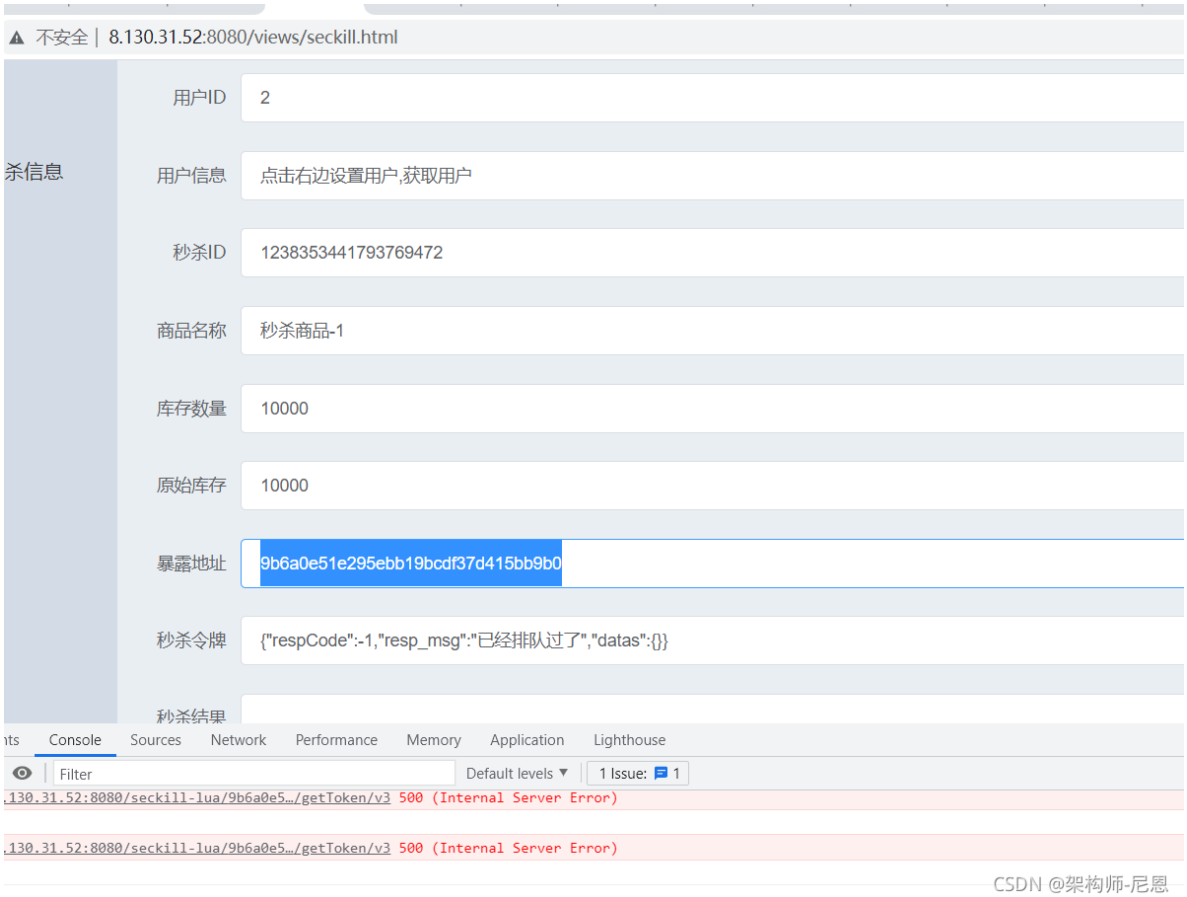
，修改主机、端口、url

改一下连接数 1,测试次数1

修改ip，这里用公网ip（8.130.31.52），而不是私网ip

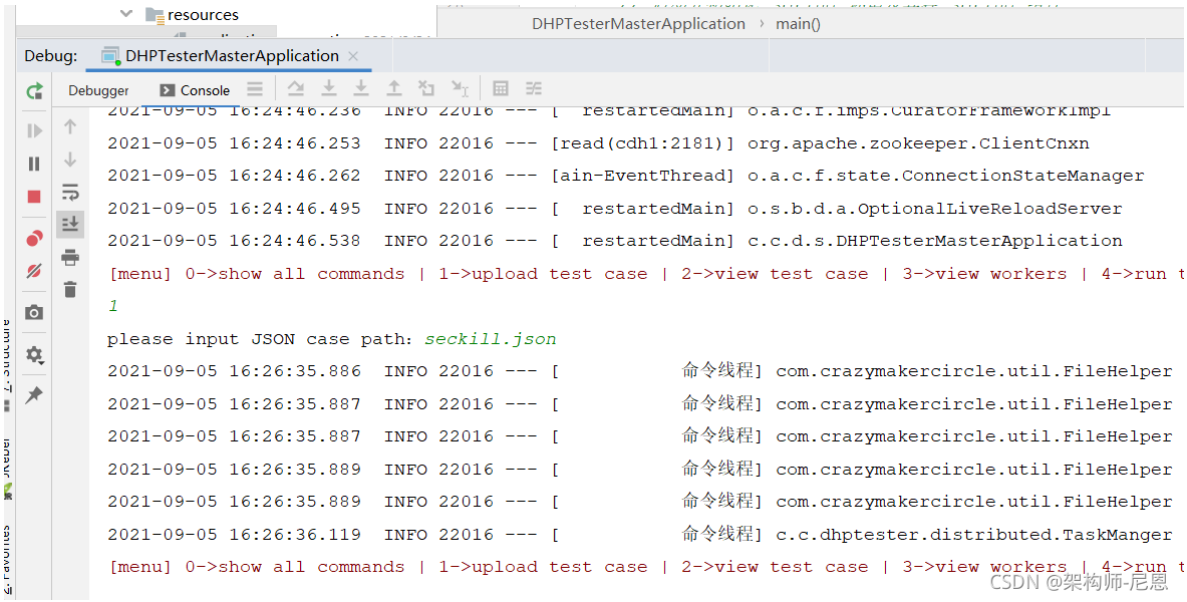
修改获取令牌的url，替换商品exposedkey



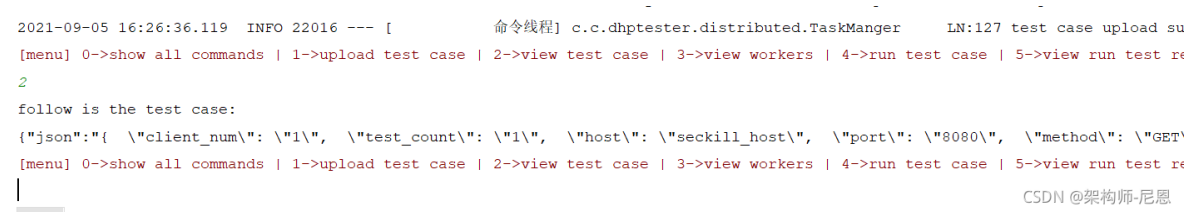


本地开始用例验证

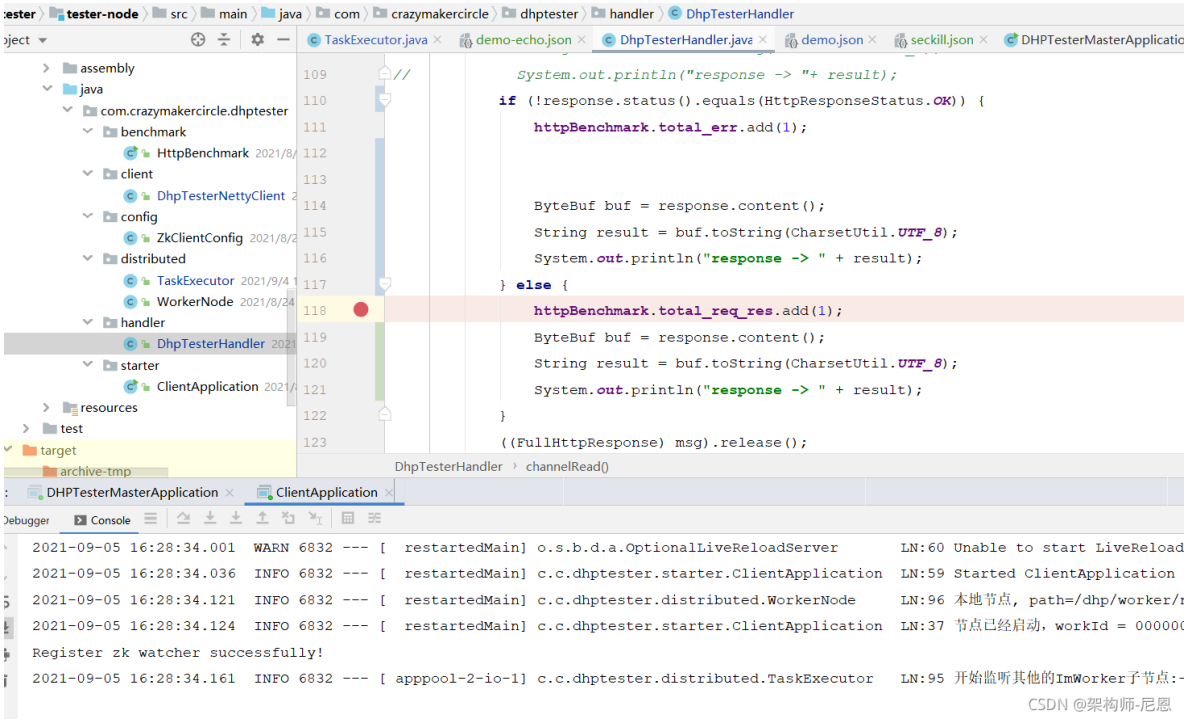
启动master，上传用例



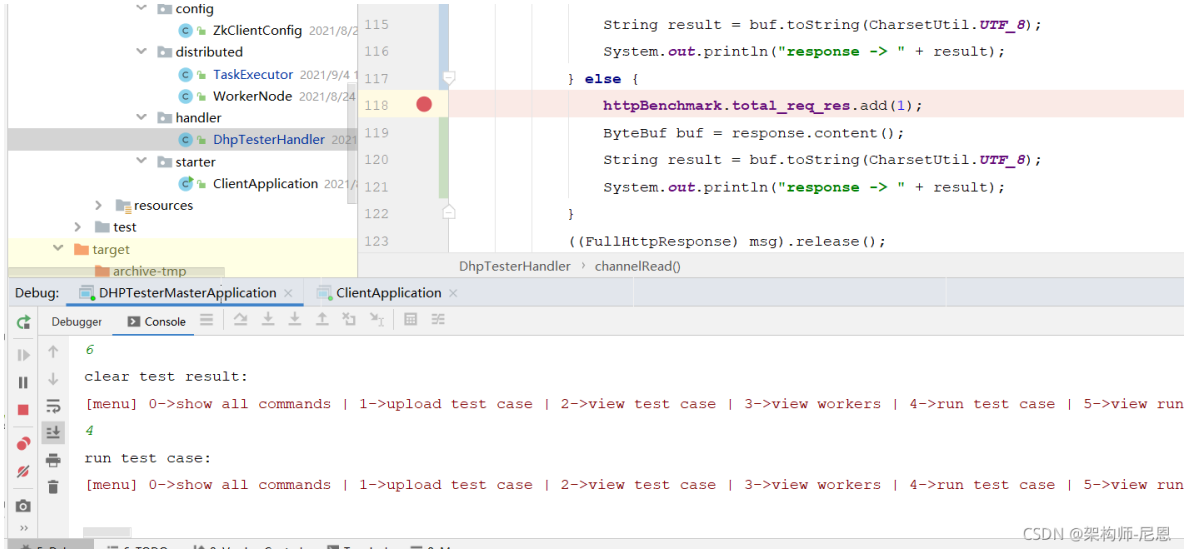
查看用例



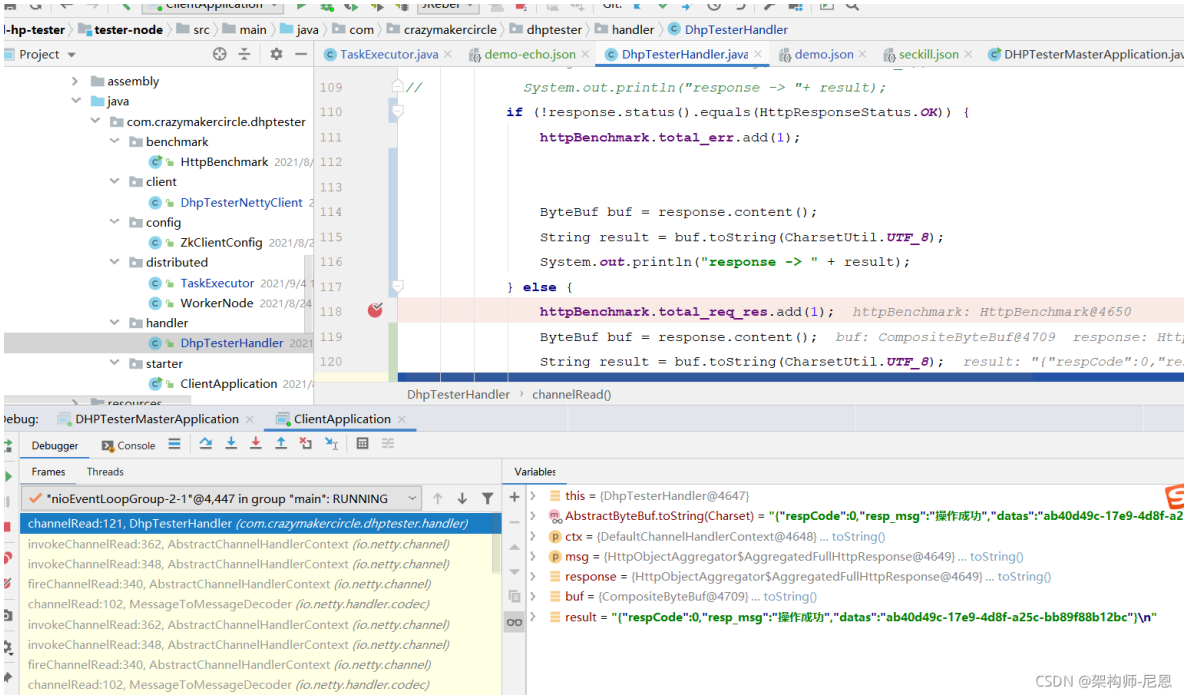
启动工作节点



master执行命令



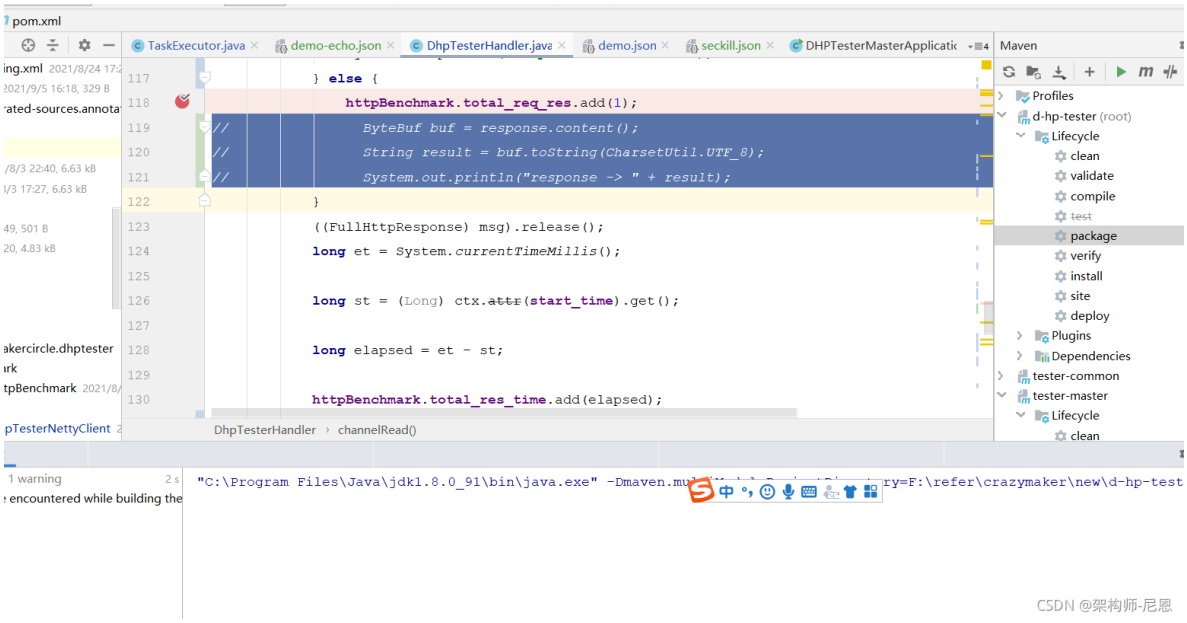
worker请求发送成功



本地测试通过

工具打包

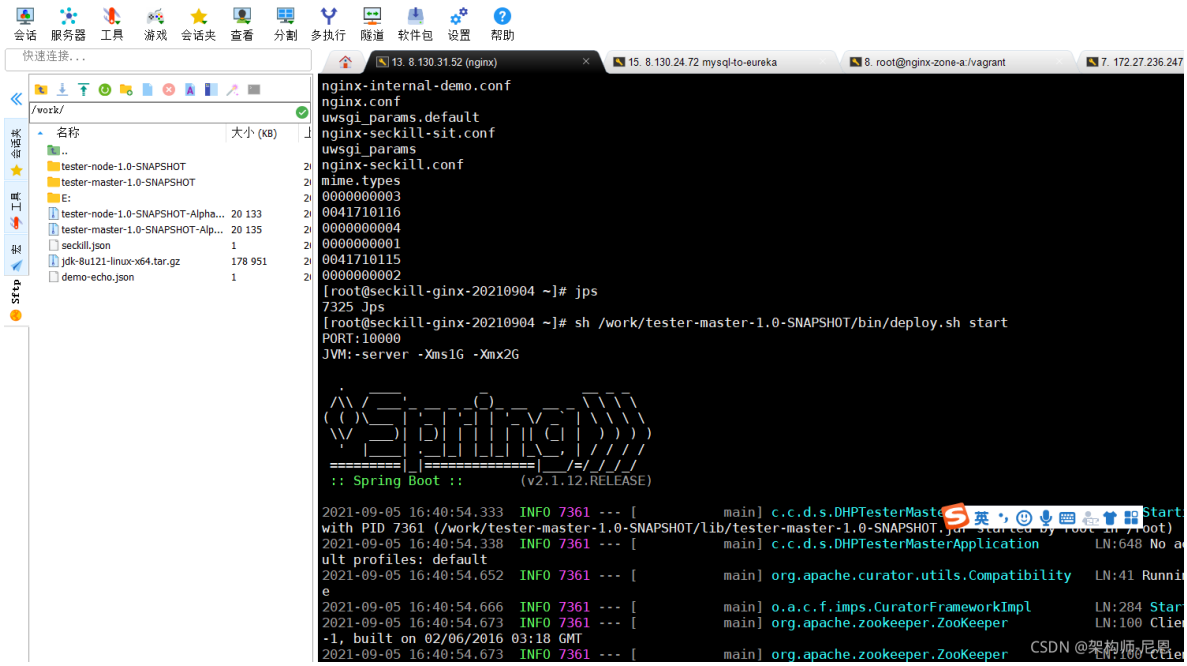
打包之前，去掉worker中刚才的结果打印代码



部署master

上传到/work 解压缩，然后启动

```
sh /work/tester-master-1.0-SNAPSHOT/bin/deploy.sh start
```

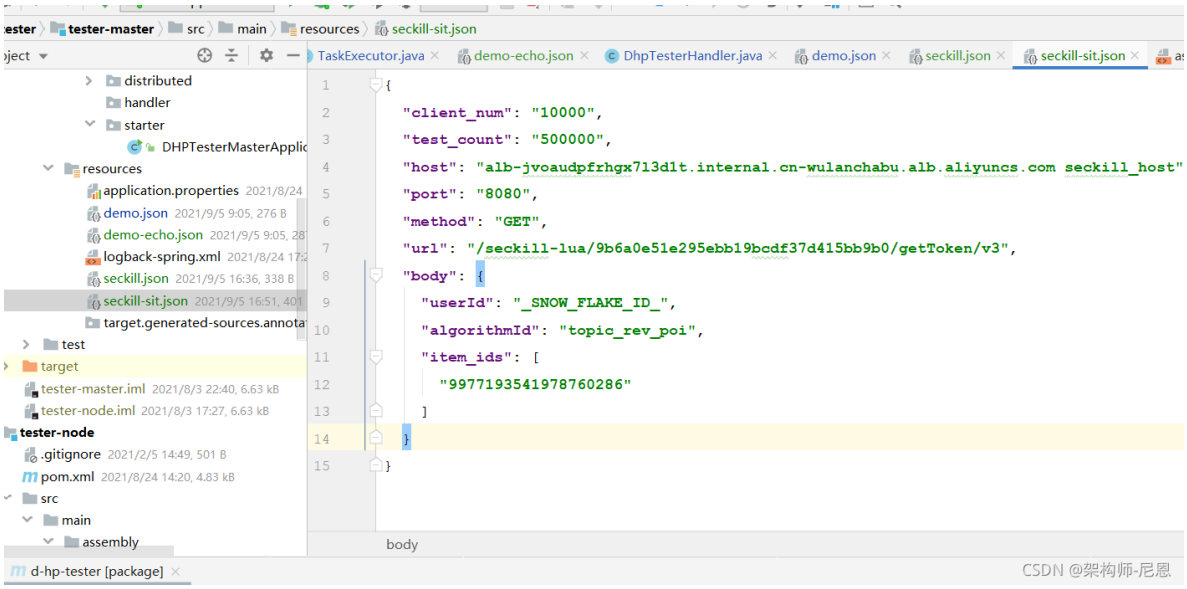



```
[root@seckill-ginx-20210904 work]# tar -zxvf tester-master-1.0-SNAPSHOT-
Alpha.tar.gz
tester-master-1.0-SNAPSHOT/bin/
tester-master-1.0-SNAPSHOT/bin/deploy.sh
tester-master-1.0-SNAPSHOT/lib/tester-master-1.0-SNAPSHOT.jar
tester-master-1.0-SNAPSHOT/logs/
[root@seckill-ginx-20210904 work]# sh /work/tester-master-1.0-
SNAPSHOT/bin/deploy.sh start
```

上传测试用例json

修改一下连接数 10000,测试次数50000

修改成slb的域名

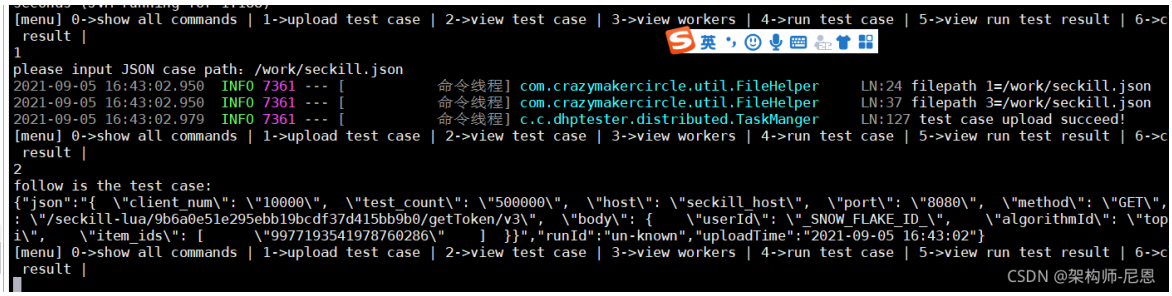


上传

/root/demo-echo.json

/root/seckill-sit.json

加载测试用例json

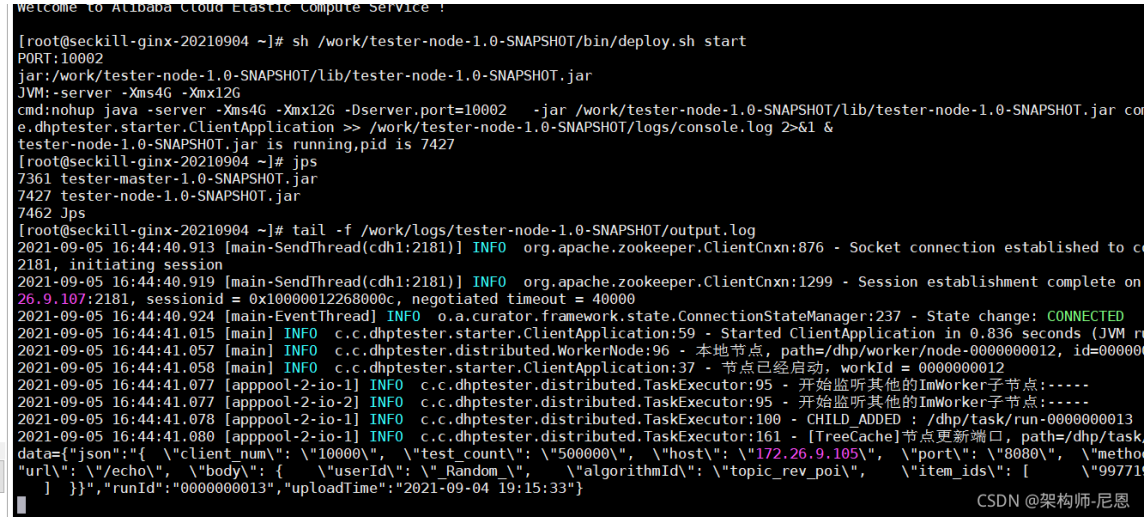


上传、解压、启动worker

> sh /work/tester-node-1.0-SNAPSHOT/bin/deploy.sh start

打印日志

> tail -f /work/logs/tester-node-1.0-SNAPSHOT/output.log



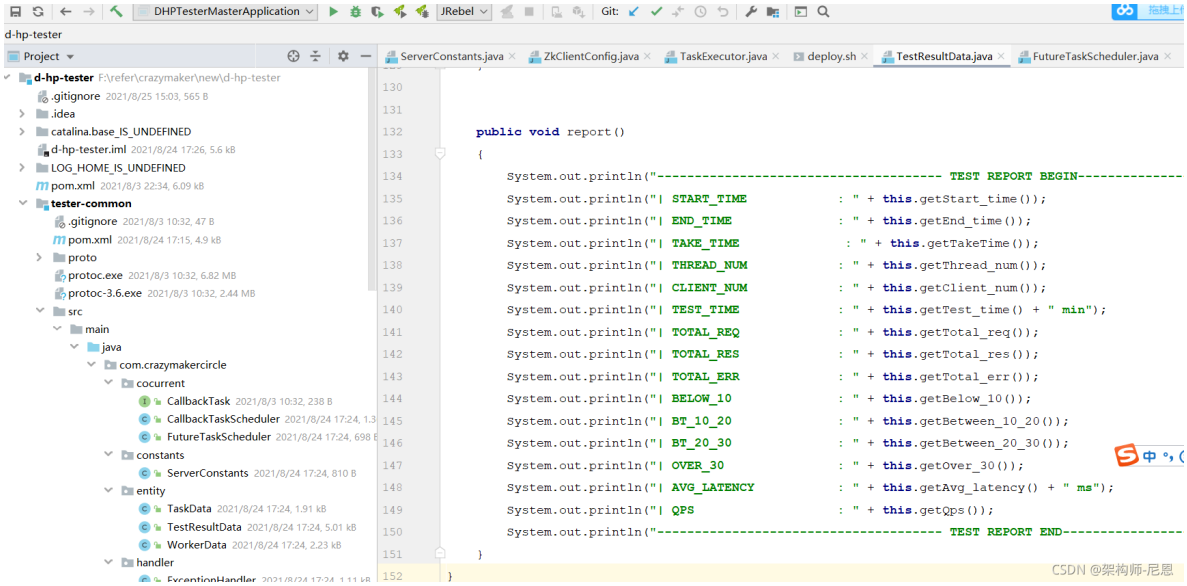
启动测试

master 使用4就可以了

手写基于ZOOKeeper+Netty的分布式测试工具：工作节点原理与实战

工具的源码路径

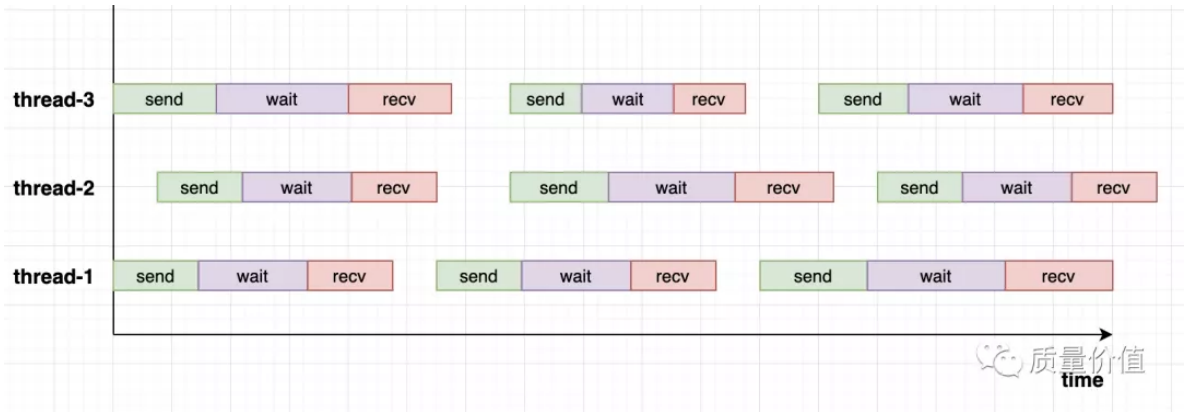
https://gitee.com/crazymaker/d-hp-tester.git



基于ZOOKeeper+Netty手写分布式测试工具：系统架构设计

基于多线程并发的ab、JMeter 性能差的根因

ab、JMeter分别是用C、Java开发的、基于多线程并发模型的压测工具，也是目前最流行的开源压测工具，两者的工作原理类似，如下图：



多线程并发

- 不管ab还是JMeter，其所谓的虚拟用户(vuser)就是对应一个线程
- 在单个线程中，每个请求（query）都是同步调用的，下一个请求要等待前一个请求完成才能进行
- 一个请求（query）分成三部分：
    - send - 施压端发送开始，直到承压端接收完成
    - wait - 承压端接收完成开始，直至业务处理结束
    - recv - 承压端返回数据，直至施压端接收完成
- 同一线程中连续的两个请求之间存在**等待时间**这种概念，即图中的空白处

在多线程并发模型下，是不是可以通过不断增加线程数量生产出更大的压力?

答案是否定的。

事实上一个进程在一个时间点只能执行一个线程，而所谓的**并发**是指在进程里不断切换线程实现了看上去的多个任务的**并发**，但是线程上下文切换有很高的成本，过多的线程数反而会造成性能的严重下滑。

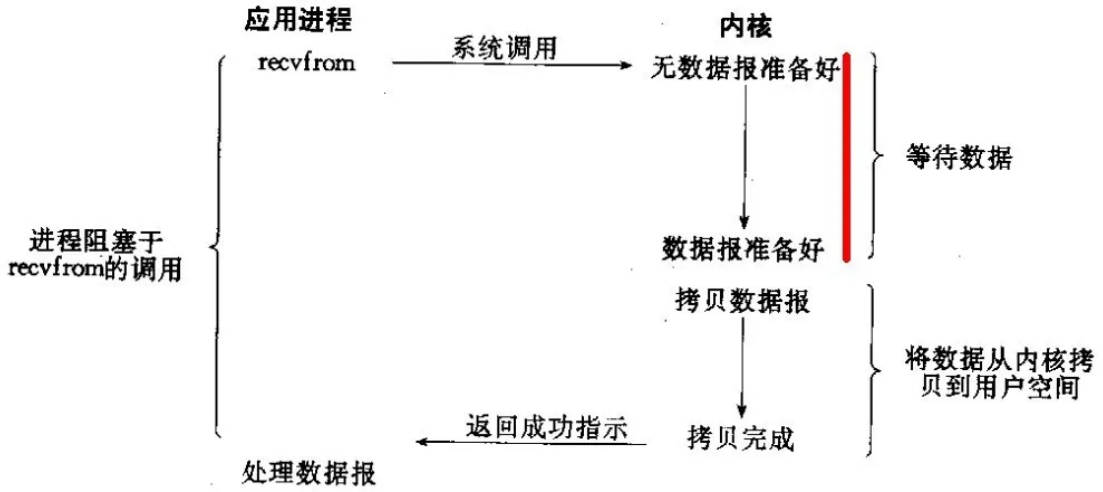


图 6.1 阻塞 I/O 模型

BIO

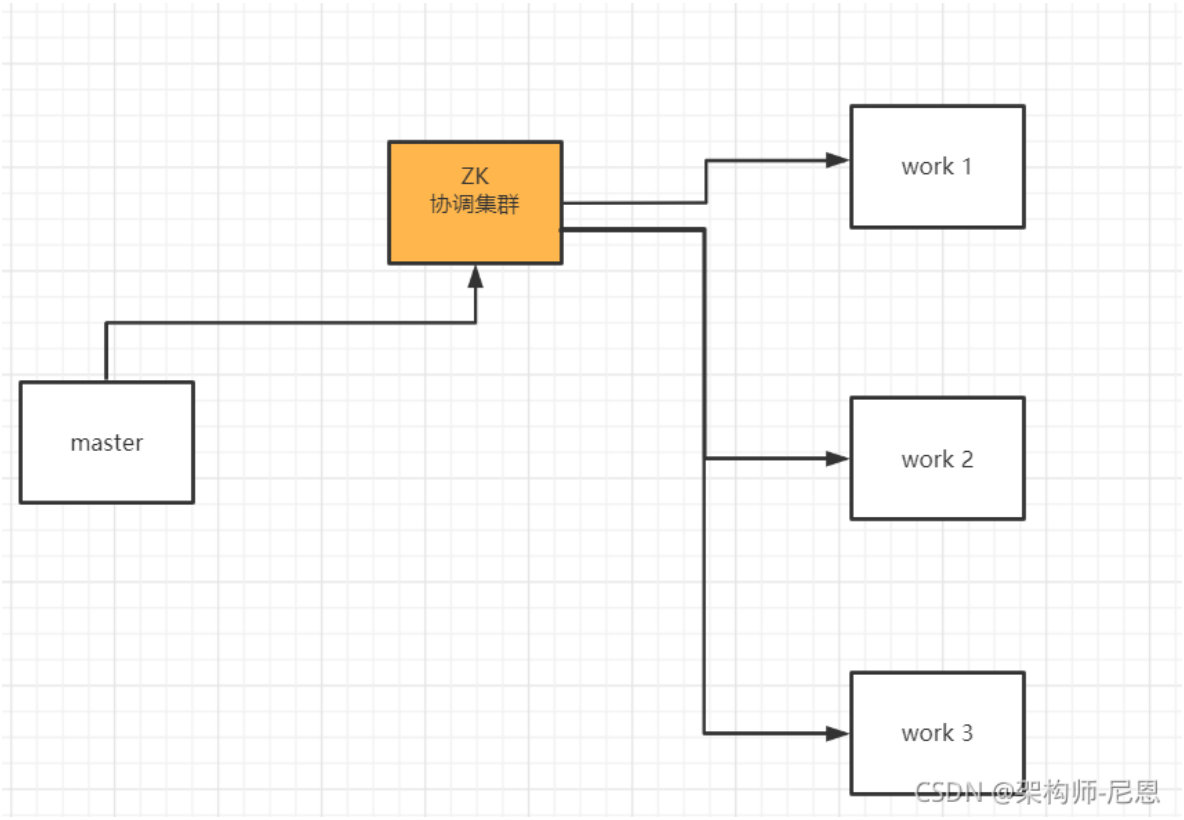
从应用角度来看，基于多线程的并发模型，往往需要设置**最大并发数**参数，而如果压测场景需要不断往上加压，那这类工具其实挺难应付的。

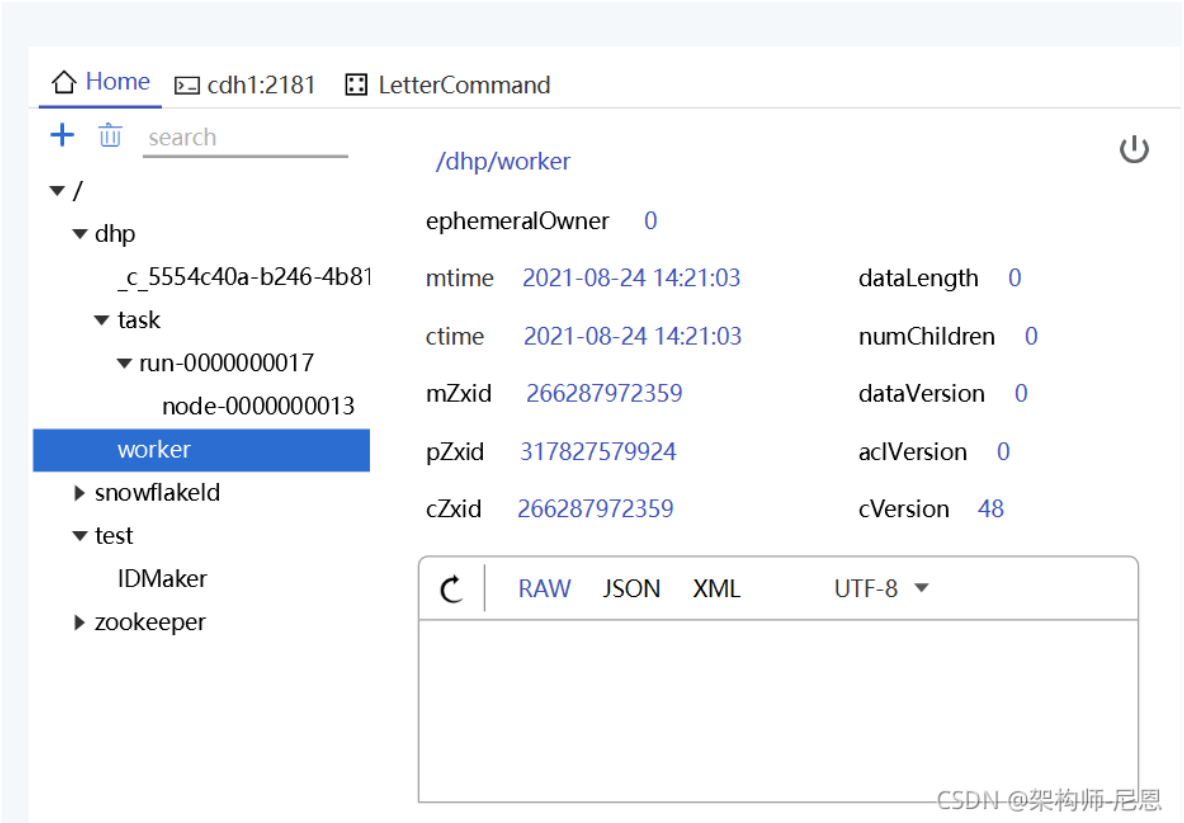
根因：

1 一个线程对应一个用户，线程数量大的时候，性能低

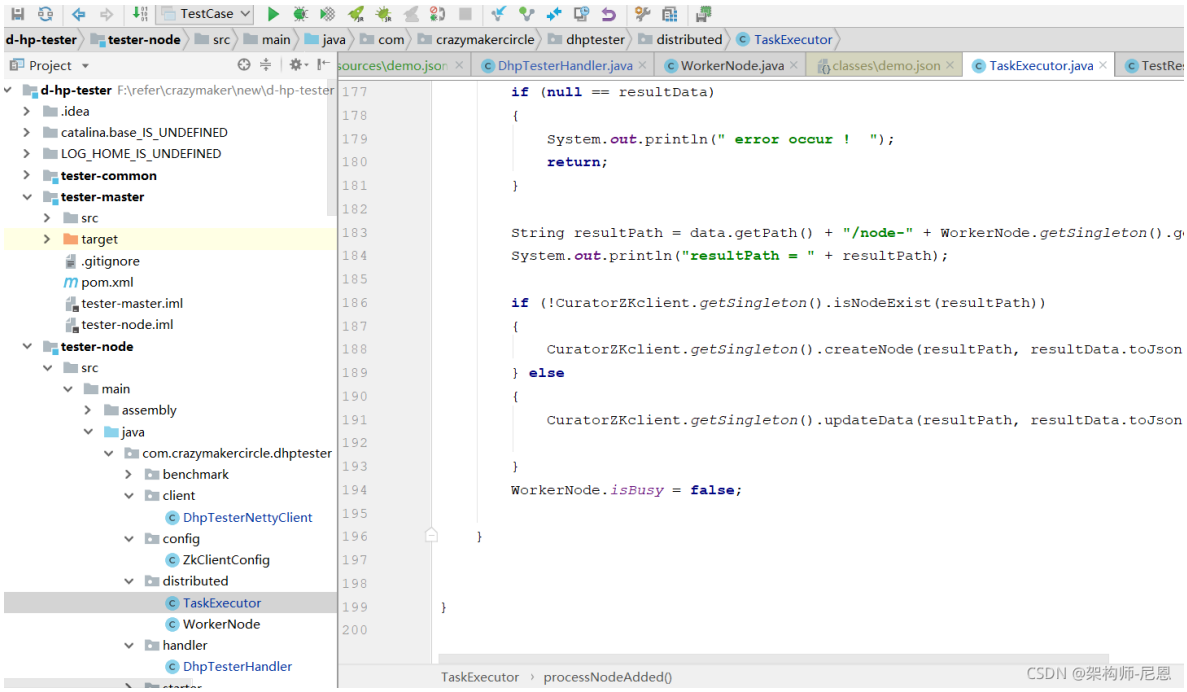
2 bio模型，性能低

基于ZOOKeeper+Netty的分布式测试工具的系统架构





基于ZOOKeeper+Netty手写分布式测试工具：工作节点的设计与实现



体验：执行一下工作节点的 请求测试

准备工作

准备好测试接口

```
[root@cdh1 ~]# /vagrant/LuaDemoProject/sh/linux/openresty-restart.sh
shell dir is: /vagrant/LuaDemoProject/sh/linux
openrestry/nginx is not running!
OPENRESTRY_PATH:/usr/local/openresty
PROJECT_PATH:/vagrant/LuaDemoProject/src
nginx: [alert] lua_code_cache is off; this will hurt performance in
/vagrant/LuaDemoProject/src/conf/nginx-seckill.conf:90
openrestry/nginx starting succeeded!
pid is 11777


[root@cdh1 ~]# curl http://cdh1:8080/echo
echo
```

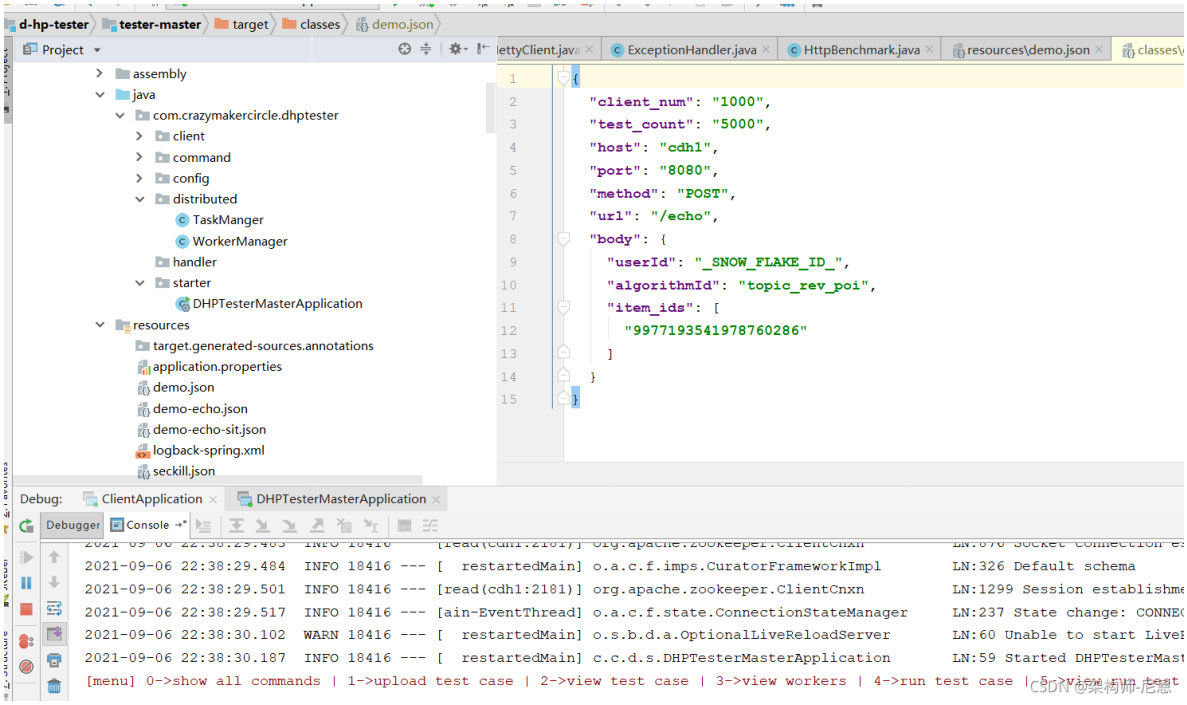
准备好测试脚本

```
{
```

```
  "client_num": "1000",
  "test_count": "10000",
  "host": "cdh1",
  "port": "80",
  "method": "POST",
  "url": "/echo",
  "body": {
    "userId": "_SNOW_FLAKE_ID_",
    "algorithmId": "topic_rev_poi",
    "item_ids": [
      "9977193541978760286"
    ]
  }
}
```

执行用例，查看结果

基于ZOOKeeper+Netty手写分布式测试工具：Master节点的设计与实现



功能设计

1 管理用例

上传、 查看

2 查看工作节点

3 启动任务

4 汇总结果

功能实现