# B简历：

# -------------------------------------

基本个人信息

|  |  |  |  |
| --- | --- | --- | --- |
| 基本情况 | | | |
| 姓名 | 韩旭 | 性别 | 男 |
| 出生日期 | 1995-03-28 | 属相 | 猪 |
| 现住地 |  |  |  |
| 大学情况 | | | |
| 学校 | 西安培华学院 | 专业 | 计算机科学与技术 |
| 入学时间 | 2013-03-28 | 毕业时间 | 2017-07-01 |
| 专业课 | **C语言、数据结构课程设计、数据库原理课程设 计、操作系统实训、Java程序设计** | | |
| 学历 | **本科** | 学位 | **学士学位** |
| 学校地址 | 西安市长安区常宁大街888号 | 校长 | **李映方** |
| 去学校的方式： | **我是在学校里的宿舍住的** | | |
| 上小学日期： |  | | |
| 上初中日期： |  | | |
| 上高中日期： |  | | |
| 上大学日期： | **2013年9月 18岁四年** | | |
| 毕业时间： | **2017年 7月 22岁** | | |
| 在校学生： | 6000**左右** | | |
| 有教师： | **不太清楚** | | |

|  |  |
| --- | --- |
| 第一家公司 ： | 郑州立信软件科技有限公司 (立信科技) |
| **地址：** | **郑州金水区商都世贸中心D座2202** |
| **入职时间：** | **2017年8月初** |
| **离职时间：** | **2018年4月** |
| **公司性质：** | **民营** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **李欣强** |
| **薪资方面：** | **刚刚毕业转正以后工资是6.5K左右，过完年涨到了 8K左右** |
| **路线：** | 从 郑州大学 坐1号线 到 二七广场换乘 到 通泰路下车走5分钟到商都世贸中心 1H |

|  |  |
| --- | --- |
| **第二家公司情况：** | 用友广信网络科技有限公司(只说公司简称) |
| **公司地址**： | 北京海淀区用友产业园中区8楼C座 |
| **公司规模：** | **100人左右,我们项目组8人，一个项目经理，4名开发人员，1名前端，1个测试，1个运维** |
| **公司性质：** | **民营** |
| **公司行业：** | 从事计算机科技、网络科技、电子科技、生物科技、信息技术领域内的技术开发。  北京住址: 西二旗智学苑(北门) |
| **去公司路线：** | 从 [地铁西二旗站南](https://www.amap.com/javascript:void(0)) 上车 [丰滢路口北](https://www.amap.com/javascript:void(0))  下车 步行 1.3公里 至 终点(约19分钟) 终点 用友产业园中区8号楼 约50左右 |
| **入职时间：** | **2018年4月** |
| **离职时间：** | **2021年3月** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **李瑞华** |
| **税前：** | **17K** |
| **税后：** | **16K** |
| **缴税：** | **1.5K左右(具体原因不清楚当时公司就是这样扣的税)** |
| **期望薪资：** | **18K** |
| **薪资的构成:** | **岗位工资(6K)+绩效工资(11K)** |

|  |
| --- |
| 四年涨薪： **2017年工作的时候是6.5K，税后大概6K左右，**  **转正8K左右**  **2018年2月跳槽涨了3.5K左右 总11.5K**  **2018年12月涨2K总 13.5K左右**  **2019年9月工作一年涨了2K左右 15.5k**  **2020年7月工作一年涨了1.5K左右 17k**  **2021年3月初离职时工资为17k** |

**面试常见问题：**

|  |
| --- |
| **你为什么从上家公司离职？** |
| **想换个工作环镜，提高提高自己的技术** |
| **你交社保了吗？为啥没交？** |
| **没交**  **在上家公司的时候，人事说咱们公司的五险一金如果要上的话都是从自己工资里面扣的，**  **当时感觉没啥必要也就没上。** |
| |  | | --- | | **三个词形容自己**： 交流表达能力强，团队协作能力强，抗压自学能力强 (**可以综合自己自学的知识表达**) | | **你的优缺点是啥？** |   **优点:**  **抗压能力强（有一次项目就要上线了，连着加班了一周，每天都到夜里很晚，但我最后也是成功完成任务。），**  **乐于分享，**  **团队协作能力强**  **缺点: 心急导致我说话有时候比较快同事有可能听不太清，**  **不过现在已经改进的差不多了** |
| **你知道五险一金都是啥吗？** |
| **养老保险、医疗保险、失业保险、工伤保险、生育保险，和住房公积金。** |
| **你能接受加班吗** |
| **能** |
| **你能接受出差吗** |
| **能** |
| 你期望薪资多少？你上家工资多少？税后拿到手的有多少？扣了多少钱的税？ |
| 期望薪资 17K 上家工资16K 税后15K左右 扣了1K左右的税 |
| 你的五年规划是啥？ |
| 前2,3年 继续加强自己的技术功底，  然后想朝着架构师或者项目经理方面走吧 |
| 是统招吗 是是本科吗 是能在学信网查到吗 我当时上的是民办大学，能在民教网查到！ |
| 你们公司还有什么福利？多少薪： 13薪  **车补**  220左右  **项目奖金**  8k-10k，做完项目，收完尾款的时候发。  **年终奖**  就是1个月的工资，  最后一个月和工资一块儿发/年后发  **电话补助**  260  **饭/餐补**  30/天  **电脑补**  200 |

**公司内其他部门：**

**技术部，人事部，销售部，财务部**

**---------------------------------------------------------------------**

**其他问题：**

**笔试问题：我工作这么多长的时间了，理论上的知识忘得差不多了，你和你们经理联系一下直接面试吧，不行的话那么我就去直接去下一家！**

## ----------------------------------------------------------主动说：

**你好！**今天来应聘咱们公司 java 开发工程师！

我最近做的一个项目是使用 **Spring Boot+springCloud 来做的微服务项目的开发**

首先先来说一下，这个springboot项目具体是怎么做的吧！

## springboot

1. 什么是spring boot？

<https://blog.csdn.net/qq_40147863/article/details/84194493>

从最根本上来讲，Spring Boot就是微服务(相当于接口开发)，它能够被任意项目的构建系统所使用，Spring Boot 的设计是为了让你尽可能更快的跑起来 Spring 应用程序并且尽可能减少你的配置文件。

它的优点：

1.快速搭建spring框架

2.零配置 遵守“约定大于配置”

3.集成了大量常用的第三方库的配置 做到了几乎开箱即用的能力

4.嵌入了web容器

（4 详细的第四步）提供一系列大型项目常用的非功能性特征，如嵌入式服务器、安全性、度量、运行状况检查、外部化配置等。

常用注解 https://www.cnblogs.com/nihaorz/p/10528121.html

1.项目配置注解

创建启动类 @springbootapplication

里面有三个注解@Configuraction @EnableAutoConfiguration @ComponentScan

三个注解解释

@Configuration

指出该类是 Bean 配置的信息源，相当于XML中的，一般加在主类上。

@EnableAutoConfiguration

让 Spring Boot 根据应用所声明的依赖来对 Spring 框架进行自动配置，一般加在主类上。

@ComponentScan

组件扫描。相当于，如果扫描到有@Component @Controller @Service等这些注解的类，则把这些类注册为bean。

## Springcloud

### 什么是Spring Cloud？

SpringCloud是一系列框架的有序集合，它是基于springboot的，通过springboot的开发便利性巧妙的简化了分布式系统基础设施的开发，如:注册中心与发现,配置中心,智能路由,消息总线,负载均衡,断路器,数据监控等,都可以用springboot的开发风格做到一键启动和部署，springcloud并没有重复的制造轮子,它只是将各家公司开发比较成熟、还经得起考验的框架组合起来，再次封装屏蔽掉了一些复杂的配置和实现原理，最后给开发者留出了简单易懂的、容易部署的、容易维护的分布式系统的开发工具包。

1. **为什么要使用springcloud？**

现在不论是商业应用还是用户应用，在开发的初期都是使用单体架构，但是随着逐渐业务的扩展，单体架构的应用也会变得越来越来复杂,这时会带来如下的几个问题:

1. 代码的结构混乱：

业务复杂，导致代码量增大，管理起来也会变得困难，同时，也会给业务的快速迭代来巨大的挑战;

1. 开发效率低：

因为单体架构是一套代码，大家同时开发一套，很难避免代码冲突，还伴随着不断解决冲突的过程，这样的话会严重影响开发效率;

1. 排查解决问题成本高:

线上业务发现bug，修复bug的话简单，但是需要重新编译、打包、上线，成本很高。

由于单体结构的应用随着系统复杂度的增高，会暴露出各种各样的问题。近些年来，微服务架构逐渐取代了单体架构，且这种趋势将会越来越流行。Spring Cloud是目前最常用的微服务开发框架，已经在企业级开发中大量的应用。

### 3.Spring Cloud的优缺点？

优点：

（1）前后端分离开发，便于快速开发

（2）采用去中心化思想，服务之间采用Restful等轻量级通讯，比ESB更轻量

（3）利于扩展模块，乐高积木一般

（4）Springcloud的节点不会影响另一个节点

（5）适于互联网时代，产品迭代周期更短，如果用来做电商网站的话，性能是非常好的

缺点： 配置复杂，节点多，涉及的知识点多，所以治理成本高，不利于维护系统，且对团队挑战大

总的来说优点大过于缺点，目前看来Spring Cloud是一套非常完善的分布式框架，目前很多企业开始用微服务、Spring Cloud的优势是显而易见的。因此对于想研究微服务架构的同学来说，学习Spring Cloud是一个不错的选择。

4.SpringBoot和SpringCloud的区别？

（1）Springboot是单体个体的开发，Springcloud是分布式（多个）的开发

（2）SpringBoot可以离开SpringCloud独立使用开发项目， 但是SpringCloud离不开SpringBoot ，属于依赖的关系.

（3）SpringBoot专注于快速、方便的开发单个微服务个体，SpringCloud关注全局的服务治理框架

### 5.springcloud五大组件

现在介绍一下springcloud的五大组件：

Eureka注册中心与发现、ribbon负载均衡、feign接口调用、hystrix熔断器、zuul服务网关

1.Eureka

Eureka是服务注册中心，分别是eurekaserver服务端和eurekaclient客户端

eurekaserver服务端：里面有个注册中心，注册中心保存的是各个服务所在的机器和端口号

eurekaclient客户端：将服务注册到注册中心中并且发现服务

工作原理：在启动应用后，eureka客户端会向服务器注册自己的服务，同时服务端的信息会缓存到本地,客户端会向服务器周期性的心跳交互更新服务信息。

2.ribbon组建介绍

ribbon在springcloud中主要做负载均衡的作用.它是基于http协议和tcp协议的客户端负载均衡,使得面向REST请求时变成客户端的负载均衡调用.

3.feign组建介绍

feign 组建其实是整合和ribbon的.feign主要是进行远程调用,它通过@FeginClient的注解,进行动态代理(要是某个接口上面标注有这个注解,那么feign会对这个接口进行动态代理)

feign的动态代理,会根据你请求的@RequestMapping 和参数,来动态构造你要请求的服务地址,然后根据地址发送请求.

如果服务部署了多台机器,那么feign会通过负载均衡ribbon的轮询机制确实服务机器(ribbon默认采用轮询机制).

4.Hystrix组建

在微服务架构中,一个系统会有多个服务,服务之间的相互调用如果不做任何保护就会导致服务器雪崩问题.那么Hystrix就是用来解决这些问题的.Hystrix是隔离,熔断以及降级的一个框架,

在某个服务挂掉之后,Hystrix会熔断这个服务,直接返回.(降级:在这个服务熔断之后,记录这期间服务请求的信息,以便后期维护).

5.zuul网关

zuul:微服务网关,主要是负责网络路由的.

在微服务的架构中,可能后台会部署几百个服务,所有的服务名称前端不可能全部记住.那么现在就需要网关,前端通过浏览器发送请求,进入到网关zuul,网关会根据请求的一些特征,将请求转发到后端的服务.

网关还可以做统一的服务降级,限流,认证权限安全等等.

6.springcloud 常见面试问题

1.springcloud 与dubbo的区别

服务调用方式不同,dubbo是采用rpc远程调用,srpingcloud是采用rest api远程调用.

注册中心:dubbo是采用zookeeper,springcloud是采用eureka

服务网关不同:dubbo本身是没有实现的,只是整合了第三方的技术.而springcloud是有一个组建zuul作为路由网关,springcloud还支持断路器,与git完美集成分布式配置文件支持版本控制.

2.微服务之间是如何独立通信的?

feign远程调用.优点:没有中间代理件,简单常见.缺点:可用性低,在请求过程中服务端和客户端都必须是可用的

消息中间件.优点:解耦,提高可用性,可以支持多种相应模式(如:发布/订阅,请求/异步响应等等).缺点:增加系统的复杂度,万一消息中间件挂了,整个服务就挂了.

3.eureka和zookeeper的区别

eureka取用cap的ap,而zookeeper选用cp注重一致性(C:一致性.A:可用性.P:分区容错性.)

在服务崩溃时,eureka有自我保护机制,各个节点都是平等关系,在一个节点崩溃时,微服务并不会盲目删除这个节点,而是进入保护状态,此时可以正常注册,也可以提供查询服务,但是不与其他节点同步信息,其他节点可以正常使用.zookeeper分为leader和follower两种角色,采用半数存活的原则,超过一般的节点挂机就会使整个服务崩溃 而且在leader崩溃之后需要重新选举,在此期间服务是不可用的.

## Docker

Docker是一个开源的应用容器引擎，它可以解决运行环境和配置问题，方便做持续集成并且有助于整体发布的容器虚拟化技术。

可以让我们打包他们的应用以及依赖包到一个轻量级、可移植的容器中，然后发布到任何流行的Linux机器上，也可以实现虚拟化，容器是完全使用沙箱机制，相互之间不会有任何接口，重要的是容器性能开销极低，docker的主要目标就是：一次封装，到处运行。

Docker主要有三部分镜像，容器，仓库

**镜像**是一种轻量级、可执行的独立软件包，用来打包软件运行环境和基于运行环境开发的软件，它包含运行某个软件所需的所有内容，包括代码、库、环境变量，配置文件等等，

docker的镜像实际上由一层一层的文件系统组成，这种层级的文件系统UnionFS。这种分层文件最大的一个好处就是：共享资源。比如：有多个镜像都从相同的 base 镜像构建而来，那么宿主机只需在磁盘上保存一份base镜像，同时内存中也只需加载一份 base 镜像，就可以为所有容器服务了。而且镜像的每一层都可以被共享。

在Docker中镜像是一种完全相同的副本，镜像是只读的。我们能写入的只有容器**。**在使用镜像的时候我们可以理解成就是一个模板，它里面帮我们安装好了一些应用程序并且配置好了程序的运行环境和配置，比如我们获取了一个centos的镜像，我们在获取到镜像后通过docker run -it --name centostest centos 命令就可以生成一个叫centostest的容器并运行这个容器，这时这个叫centostest的容器就相当于一个新的centos系统，在docker中容器，就是一个运行时环境，运行容器，然后在容器中运行程序，至于仓库就是我们存放镜像的地方

在linux上面安装docker 之后可以通过docker version来查看docker是否安装成功以及版本号

### docker 的一些操作

docker images 列出本机的所有镜像 镜像名 版本号 id 创建时间 大小

docker search 镜像名 搜索镜像 收藏数

docker pull mysql:5.6(版本号) 下载一个centos 的一个镜像

docker pull centos 下载镜像不指定版本 （下载最新的）

docker rmi 镜像名(id) 注意正在被容器使用的镜像是无法删除

docker rmi -f 镜像名(id) 可以删除正在被容器使用的镜像

**容器的操作命令（有镜像才能创建容器）**

docker run -it --name 容器名称（唯一的） 镜像（创建容器并进入终端一旦退出容器结束）

docker run -itd --name 容器名称（唯一的） 镜像（创建容器进入后台运行不会进行终端 （exec 进入终端）终端退出 容器继续运行）

容器端口映射：

docker run -itd --name (名称) -p (服务端口):(映射端口) /bin/bash

退出容器 ctrl+p+q 不会停止容器 exit 会停止容器

进入后台运行容器的终端 docker exec -it 容器名称(容器id) /bin/bash

查看运行中的容器 docker ps

查看所有的容器 docker ps -a

删除容器 docker rm 容器id（名称）（运行中的容器不能删）

删除容器 docker rm -f 容器id（名称）（可以删除运行中的容器）

停止容器 docker stop 容器id（名称）

启动容器 docker start 容器id（名称）

**拷贝 宿主机向容器中拷贝文件**

docker cp 宿主机的文件 容器id:目录

**拷贝 容器中的文件 拷贝到宿主机上**

docker cp 容器id:目录 宿主机的目录

### 数据卷怎么创建 (创建一个容器 并和宿主机共享目录)

docker run -itd --name 容器名 -v 宿主目录(/opt/docker/gx):/容器目录(/gx) 镜像 /bin/bash

数据卷容器（多个容器共享一个目录 （修改一个目录 多个容器跟着改变））

先创建一个共享的容器

docker run -v 宿主目录:容器目录 --name 容器名称 镜像 /bin/bash

创建容器指定共享的容器 实现共享目录的

docker run -itd --volumes-from 共享容器名 --name容器名称 镜像 /bin/bash

### Docker 容器配置mysql

docker run -id --name memysqla -p 4000:3306 -e MYSQL\_ROOT\_PASSWORD=root mysql:5.6

memysqla 容器名

4000:3306 将容器的 3306 端口映射到宿主机的 4000端口

root mysql密码

mysql:5.6 mysql版本号

### 制作镜像

创建DockerFile文件

内容如下：

FROM centos

MAINTAINER liuwenhao

RUN mkdir -p /docker/doc

ADD jdk1.8.0\_45 /docker/doc/jdk

ADD springcloud\_product\_eureka\_server-1.0-SNAPSHOT.jar /docker/doc/

ENV JAVA\_HOME /docker/doc/jdk

ENV PATH $PATH:$JAVA\_HOME/bin

EXPOSE 3400

ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/docker/doc/springcloud\_product\_eureka\_server-1.0-SNAPSHOT.jar"]

生成镜像：docker build -f DockerFile文件 -t 生成的镜像名称:版本号 .

启动容器：docker run -itd --name (名称) -p (服务端口):(映射端口) 镜像id /bin/bash

### 私有仓库

1、拉取私有仓库镜像

docker pull registry

2、启动私有仓库容器

docker run -id --name=registry -p 5000:5000 registry

3、打开浏览器 输入地址

http://私有仓库服务器ip:5000/v2/\_catalog，看到{"repositories":[]} 表示私有仓库搭建成功

4、修改daemon.json vim /etc/docker/daemon.json

在上述文件中添加一个key，保存退出。此步用于让 docker 信任私有仓库地址；注意将私有仓库服务器ip修改为自己私有仓库服务器真实ip

{"insecure-registries":["私有仓库服务器ip:5000"]}

5、重启docker 服务

systemctl restart docker

docker start registry

把镜像推到私有仓库

1. 标记镜像为私有仓库的镜像

docker tag 本地镜像 私有仓库服务器IP:5000/放到仓库的镜像名称

1. 上传标记的镜像

docker push 私有仓库服务器IP:5000/放到仓库的镜像名称

从私有仓库拉镜像

docker pull 私有仓库服务器IP:5000/放到仓库的镜像名称

Docker rabbitmq

拉取 管理版本 进行

docker pull rabbitmq:management

运行容器

docker run -dit --name Myrabbitmq -e RABBITMQ\_DEFAULT\_USER=admin -e RABBITMQ\_DEFAULT\_PASS=admin -p 15672:15672 -p 5672:5672 rabbitmq:management

## redis的总结

什么是redis?

redis是非关系型性数据库,完全开源的,高性能的key value 数据库

redis的优势?

redis支持数据的持久化(rdb和aof)

Rdb : 基于快照的方式将某个时间段的数据进行存储 适用于大数据的场合使用 数据量过小的话可能会丢失某一时间段的数据

 AOF : 将对数据库数据造成改变的操作已文本的方式记录到aof文件中

Rdb和Aof的区别

 RDB持久化是指在指定的时间间隔内将内存中的数据集快照写入磁盘，实际操作过程是fork一个子进程，先将数据集写入临时文件，写入成功后，再替换之前的文件，用二进制压缩存储。

 RDB持久化保存键空间的所有键值对(包括过期字典中的数据),并以二进制形式保存，符合rdb文件规范，根据不同数据类型会有不同处理。

 AOF持久化以日志的形式记录服务器所处理的每一个写、删除操作，查询操作不会记录，以文本的方式记录，可以打开文件看到详细的操作记录。

 AOF持久化保存redis服务器所执行的所有写命令来记录数据库状态,在写入之前命令存储在aof\_buf缓冲区

 RDB 在恢复大数据集时的速度比 AOF 的恢复速度要快，AOF在运行效率上往往会慢于RDB。

redis不仅提供了KeyValue数据存储结构,还给我们提供了String,list,set,zset,hash等数据存储结构

String字符串

我们用string类型做过解决接口幂等性 还可以用来计数 incr key自增 decr key自减

还存放了我们的地区数据 部门数据 这些数据的特性是什么？（数据访问量大切数据不经常改变）

Hash（哈希）购物车

我们的购物车数据是用hash存在redis中的

List（列表）优惠券

就是做消息队列 就是来限流,因为list是先进先出的原则，所以可以做消息队列

zset(sorted set：有序集合) Set（无序集合）

不好意思 没用过！！！

redis支持备份

redis的优点?

读写速度快 性能极高 Redis能读的速度是110000次/s,写的速度是81000次/s

丰富的数据类型

redis支持原子性,多个操作也支持事务即原子性

redis的三大问题?

分别是 穿透 击穿 雪崩

1、穿透

穿透就是多个请求去查询缓存中和数据库中都不存在的数据,这样的话就会导致我们的数据库压力比较大甚至导致数据库崩溃

针对于穿透这种的话我们有两种解决方案

第一种解决方案

就是我们可以将请求查询的key在缓存中设置一个空值并且在设置一个失效时间,这样的话查询直接返回一个空值

第二种解决方案

很多时候，缓存穿透是因为有很多恶意流量的请求，这些请求可能随机生成很多Key来请求查询，这些肯定在缓存和数据库中都没有，那就很容易导致缓存穿透。

在缓存穿透防治上常用的技术是布隆过滤器(Bloom Filter)。布隆过滤器是一种比较巧妙的概率性数据结构，它可以告诉你数据一定不存在或可能存在，相比Map、Set、List等传统数据结构它占用内存少、结构更高效。

对于缓存穿透，我们可以将查询的数据条件都哈希到一个足够大的布隆过滤器中，用户发送的请求会先被布隆过滤器拦截，一定不存在的数据就直接拦截返回了，从而避免下一步对数据库的压力。

2、缓存击穿

缓存击穿是指当某一key的缓存过期时大并发量的请求同时访问此key，

瞬间击穿缓存服务器直接访问数据库，让数据库处于负载的情况。

如何解决缓存击穿一般有2种解决办法，分别是异步定时更新和互斥锁

异步定时更新

在缓存处理上，同理，比如某一个热点数据的过期时间是1小时，那么每59分钟，通过定时任务去更新这个热点key，并重新设置其过期时间。

互斥锁

就是当Redis中根据key获得的value值为空时，先锁上，然后从数据库

加载，加载完毕，释放锁。若其他线程也在请求该key时，发现获取锁失败，则先阻塞。

3、雪崩

缓存雪崩是指当大量缓存同时过期或缓存服务宕机，所有请求的都直接

访问数据库，造成数据库高负载，影响性能，甚至数据库宕机。

如何避免缓存雪崩？

避免的方法：设置不同的过期时间或者使用缓存集群。

设置不同的过期时间

为了避免大量的缓存在同一时间过期，可以把不同的key过期时间设置

成不同的， 并且通过定时刷新的方式更新过期时间。

使用缓存集群

在缓存雪崩问题防治上面，一个比较典型的技术就是采用集群方式部

署，使用集群可以避免服务单点故障。

我之前做项目还碰到过因为redis崩溃 导致的雪崩问题

所以有了解过redis集群

redis集群分为三种方式

如何redis的高可用?

Redis集群部署有三种模式：主从复制模式、哨兵模式、集群模式。

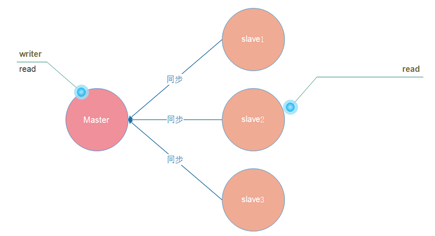
redis主从复制

1.1        概念

    主从复制模型中，有多个redis节点。

    其中，有且仅有一个为主节点Master。从节点Slave可以有多个。

只要网络连接正常，Master会一直将自己的数据更新同步给Slaves，保持主从同步。



1.1        特点

（1）主节点Master可读、可写.

（2）从节点Slave只读。（read-only）

因此，主从模型可以提高读的能力，在一定程度上缓解了写的能力。因为能写仍然只有Master节点一个，可以将读的操作全部移交到从节点上，变相提高了写能力。

数据冗余：主从复制实现了数据的热备份，是持久化之外的一种数据冗余方式。

故障恢复：当主节点出现问题时，可以由从节点提供服务，实现快速的故障恢复；实际上是一种服务的冗余。

负载均衡：在主从复制的基础上，配合读写分离，可以由主节点提供写服务，由从节点提供读服务（即写Redis数据时应用连接主节点，读Redis数据时应用连接从节点），分担服务器负载；尤其是在写少读多的场景下，通过多个从节点分担读负载，可以大大提高Redis服务器的并发量。

读写分离：可以用于实现读写分离，主库写、从库读，读写分离不仅可以提高服务器的负载能力，同时可根据需求的变化，改变从库的数量。

高可用基石：除了上述作用以外，主从复制还是哨兵和集群能够实施的基础，因此说主从复制是Redis高可用的基础。

Sentinel哨兵模式

1.1        主从模式的缺陷

当主节点宕机了，整个集群就没有可写的节点了。

由于从节点上备份了主节点的所有数据，那在主节点宕机的情况下，如果能够将从节点变成一个主节点，是不是就可以解决这个问题了呢？

答：是的，这个就是Sentinel哨兵的作用。

1.2        哨兵的任务

Redis 的 Sentinel 系统用于管理多个 Redis 服务器（instance）， 该系统执行以下三个任务：

监控（Monitoring）： Sentinel 会不断地检查你的主服务器和从服务器是否运作正常。

提醒（Notification）： 当被监控的某个 Redis 服务器出现问题时， Sentinel 可以通过 API 向管理员或者其他应用程序发送通知。

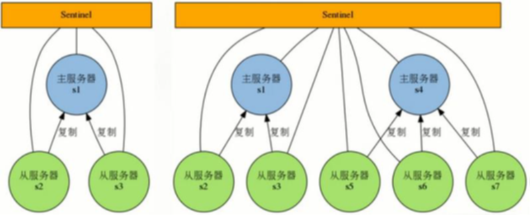
自动故障迁移（Automatic failover）： 当一个主服务器不能正常工作时， Sentinel 会开始一次自动故障迁移操作， 它会进行选举，将其中一个从服务器升级为新的主服务器， 并让失效主服务器的其他从服务器改为复制新的主服务器； 当客户端试图连接失效的主服务器时， 集群也会向客户端返回新主服务器的地址， 使得集群可以使用新主服务器代替失效服务器。

1.2.1       监控（Monitoring）

（1）Sentinel可以监控任意多个Master和该Master下的Slaves。（即多个主从模式）

（2）同一个哨兵下的、不同主从模型，彼此之间相互独立。

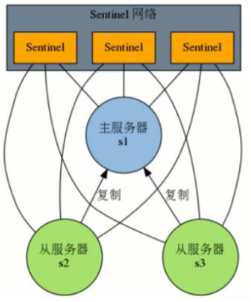
（3）Sentinel会不断检查Master和Slaves是否正常。



1.2.2       自动故障切换（Automatic failover）

1.2.2.1    Sentinel网络

监控同一个Master的Sentinel会自动连接，组成一个分布式的Sentinel网络，互相通信并交换彼此关于被监视服务器的信息。下图中，三个监控s1的Sentinel，自动组成Sentinel网络结构。



疑问：为什么要使用sentinel网络呢？

答：当只有一个sentinel的时候，如果这个sentinel挂掉了，那么就无法实现自动故障切换了。

在sentinel网络中，只要还有一个sentinel活着，就可以实现故障切换。

（2）选举

当Sentinel确定Master下线后，会在所有的Slaves中，选举一个新的节点，升级成Master节点。

其它Slaves节点，转为该节点的从节点。

（3）原Master重新上线

当原Master节点重新上线后，自动转为当前Master节点的从节点。

Cluster模式介绍（多服务器集群）

sentinel模式基本可以满足一般生产的需求，具备高可用性。但是当数据量过大到一台服务器存放不下的情况时，主从模式或sentinel模式就不能满足需求了，这个时候需要对存储的数据进行分片，将数据存储到多个Redis实例中。cluster模式的出现就是为了解决单机Redis容量有限的问题，将Redis的数据根据一定的规则分配到多台机器。

cluster可以说是sentinel和主从模式的结合体，通过cluster可以实现主从和master重选功能，所以如果配置两个副本三个分片的话，就需要六个Redis实例。因为Redis的数据是根据一定规则分配到cluster的不同机器的，当数据量过大时，可以新增机器进行扩容。

使用集群，只需要将redis配置文件中的cluster-enable配置打开即可。每个集群中至少需要三个主数据库才能正常运行，新增节点非常方便。

cluster集群特点：

\* 多个redis节点网络互联，数据共享

\* 所有的节点都是一主一从（也可以是一主多从），其中从不提供服务，仅作为备用

\* 不支持同时处理多个key（如MSET/MGET），因为redis需要把key均匀分布在各个节点上，

并发量很高的情况下同时创建key-value会降低性能并导致不可预测的行为

\* 支持在线增加、删除节点

\* 客户端可以连接任何一个主节点进行读写

## 事务

1 什么是事务？ （事务相关的概念）

事务就是一组有逻辑操作单元的SQL集合,要不都成功,要么全部失败回滚

1、原子性（Atomicity）

　　原子性是指事务包含的所有操作要么全部成功，要么全部失败回滚，因此事务的操作如果成功就必须要完全应用到数据库，如果操作失败则不能对数据库有任何影响。

2、一致性（Consistency）

　　一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

　　拿转账来说，假设用户A和用户B两者的钱加起来一共是5000，那么不管A和B之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是5000，这就是事务的一致性。

3、隔离性（Isolation）

　　隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

　　即要达到这么一种效果：对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

4、持久性（Durability）

　　持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

　　例如我们在使用JDBC操作数据库时，在提交事务方法后，提示用户事务操作完成，当我们程序执行完成直到看到提示后，就可以认定事务以及正确提交，

即使这时候数据库出现了问题，也必须要将我们的事务完全执行完成，否则就会造成我们看到提示事务处理完毕，但是数据库因为故障而没有执行事务的重大错误。

事务的传播特性:

传播特性

1. PROPAGATION\_REQUIRED(瑞快润特)

如果事务不存在则创建，如果存在使用存在的事务

1. PROPAGATION\_SUPPORTS(死普特死)

如果当前存在事务，则使用存在的事务，如果不存在则使用非事务方式处理。

1. PROPAGATION\_MANDATORY(慢的脆)

如果当前存在事务，则使用存在的事务，如果不存在则抛出异常。

4. PROPAGATION\_REQUIRES\_NEW(瑞快润)

永远创建新的事务，如果当前存在事务则挂起

5. PROPAGATION\_NOT\_SUPPORTED

永远使用非事务方式执行，如果当前存在事务则挂起

6. PROPAGATION\_NEVER

不支持事务执行方式，如果存在事务则抛出异常

7. PROPAGATION\_NESTED

如果存在一个活动的事务，则运行在一个嵌套的事务中. 如果没有活动事务, 安按照PROPAGATION\_REQUIRED 事务属性执行

事务的隔离级别:

参考地址:https://blog.csdn.net/zhouym\_/article/details/90381606

第一种隔离级别：Read uncommitted(读未提交)

如果一个事务已经开始写数据，则另外一个事务不允许同时进行写操作，但允许其他事务读此行数据，该隔离级别可以通过“排他写锁”，但是不排斥读线程实现。这样就避免了更新丢失，却可能出现脏读，也就是说事务B读取到了事务A未提交的数据

解决了更新丢失，但还是可能会出现脏读

第二种隔离级别：Read committed(读提交)

如果是一个读事务(线程)，则允许其他事务读写，如果是写事务将会禁止其他事务访问该行数据，该隔离级别避免了脏读，但是可能出现不可重复读。事务A事先读取了数据，事务B紧接着更新了数据，并提交了事务，而事务A再次读取该数据时，数据已经发生了改变。

解决了更新丢失和脏读问题

第三种隔离级别：Repeatable read(可重复读取)

可重复读取是指在一个事务内，多次读同一个数据，在这个事务还没结束时，其他事务不能访问该数据(包括了读写)，这样就可以在同一个事务内两次读到的数据是一样的，因此称为是可重复读隔离级别，读取数据的事务将会禁止写事务(但允许读事务)，写事务则禁止任何其他事务(包括了读写)，这样避免了不可重复读和脏读，但是有时可能会出现幻读。(读取数据的事务)可以通过“共享读镜”和“排他写锁”实现。

解决了更新丢失、脏读、不可重复读、但是还会出现幻读

第四种隔离级别：Serializable(可序化)

提供严格的事务隔离，它要求事务序列化执行，事务只能一个接着一个地执行，但不能并发执行，如果仅仅通过“行级锁”是无法实现序列化的，必须通过其他机制保证新插入的数据不会被执行查询操作的事务访问到。序列化是最高的事务隔离级别，同时代价也是最高的，性能很低，一般很少使用，在该级别下，事务顺序执行，不仅可以避免脏读、不可重复读，还避免了幻读

解决了更新丢失、脏读、不可重复读、幻读(虚读)

以上四种隔离级别最高的是Serializable级别，最低的是Read uncommitted级别，当然级别越高，执行效率就越低，像Serializeble这样的级别，就是以锁表的方式(类似于Java多线程中的锁)使得其他线程只能在锁外等待，所以平时选用何种隔离级别应该根据实际情况来，在MYSQL数据库中默认的隔离级别是Repeatable read（可重复读）。

## 多线程导出

当时在公司做一个报表,需要使用excel导出,做了好几个版本,进行优化,第一版 就选择了poi导出,通过反射+注解+zip来做,

反应慢,我们加上线程池;

反射是什么

自定义注解

线程池

Zip

1. 配置:我们需要配置上poi的POM依赖,直接可以使用
2. 思路:
3. 首先需要用户传来两个参数,即我们每个sheet的数据量(datasize)以及每个excel中有多少个sheet页即(sheetsize)
4. 查出我们需要导出的(数据量);

sheetAllCount(sheet总数)=总数据量/datasize;

excelCount(excel数量)= (sheet总数)/ (sheetsize);

1. 循环excel总数 每次循环 相当于创造一个excel的数据 , 创建workbox

再循环我们excel中sheet页的数量;得到当前excel中的每个sheet;创建之前 我们需要这个sheet的数据;我们需要通过分页查询按批次查询数据; 那么我们就需要 数据的起始下标与条数 条数我们可以传 每个sheet的数据量(datasize)

起始下标 :当前excel-1 \* sheetAllCount(sheet总数) x2 我们得到每个sheet数据,然后通过集合中的对象得到类对象;

1. 通过反射得到我们javabean的属性,给要添加的属性设置上自定义注解,value为我们的表头信息, getDeclaredFields获取所有字段的集合;循环判断加了注解;将注解的value设为表头;

Createcell.setcellvalue();

1. 循换每个sheet的数据; 创建row 在循环我们反射的bean属性;如果加了注解就创建一个cell;并判断类型;放入对应cell中;
2. 写入我们服务器文件中
3. 为了导出速度;我们加入线程池; 并使用contdownlach;
4. 等所有线程执行完成;我们通过java提供的zip进行压锁并响应; 将响应流加入我们的zip输出流;读写我们excel文件夹; 完成多线程导出;

线程池:

线程是系统非常珍贵的资源;线程池就是一个容器;用来存放线程;并帮我们线程执行任务;

可以减少我们线程创建与销毁的资源消耗;

创建方式 自定义型

Threadpoolexctor t=new Thredddpoolexctor

参数 :

1.核心线程数 即空闲也不会销毁的线程 核心数不足 提交任务会创建

2.最大线程数;核心数满了 就创建非核心 ;想家不能超过最大线程数

3.非核心超时时间: 非核心线程限制多久被回收;

4.线程队列:队列怎么处理线程 是直接创建新线程还是等待还是直接交给核心执行

5,饱和策略:丢弃任务 丢弃其他任务 新建线程 报异常

### 常用四种线程程池:

* 1. **FixedThreadPool**

创建一个固定工作线程数量的线程池。核心线程数和最大线程数是一样的，任务如果过多会在队列中阻塞。如果某个线程因为执行异常而结束，那么线程池会补充一个新线程。

|  |
| --- |
| * 线程池中的线程处于一定的量，可以很好的控制线程的并发量 * 线程可以重复被使用，在显式关闭之前，都将一直存在 * 超出一定量的任务被提交时候需在队列中等待 |

* 1. **CachedThreadPool**

创建一个可缓存的线程池。核心线程数是0，最大线程数是 Integer.MAX\_VALUE。当线程池中的线程都处于活动状态的时候，线程池会创建新的线程来处理新任务，否则会用空闲的线程来处理新任务。这类线程池的空闲线程都是有超时机制的，keepAliveTime在这里是有效的，时长为60秒，超过60秒的空闲线程就会被回收，当线程池都处于闲置状态时，线程池中的线程都会因为超时而被回收，所以几乎不会占用什么系统资源。

|  |
| --- |
| * 线程池中线程的数量没有固定大小，可达到最大值（Interger. MAX\_VALUE） * 线程池中的线程可进行缓存重复利用和回收（回收默认时间为1分钟） * 当线程池中没有可用线程，会重新创建一个线程 |

* 1. **SingleThreadExecutor**

创建一个单线程的线程池。这个线程池的核心线程数和最大线程数都是1，也就是相当于单线程串行执行所有任务。如果这个唯一的线程因为异常结束，那么会有一个新的线程来替代它。此线程池保证所有任务的执行顺序按照任务的提交顺序执行。

|  |
| --- |
| * 线程池中最多有1个线程，之后提交的任务将会排在队列中依次执行 |

* 1. **ScheduledThreadPool**

创建一个支持定时及周期性的任务执行的线程池。

|  |
| --- |
| * 可定时或者延迟执行线程活动 |

* 1. **SingleThreadScheduledExecutor**

创建一个单线程的支持定时及周期性的任务执行的线程池。

|  |
| --- |
| * 可定时或者延迟执行线程活动 * 线程池中最多有1个线程 |

### 反射

反射是什么 反射就是可以让我们识别一个未知的类,一个未知的对象;

对于类来说就是 我们可以知道他里面的所有属性和所有方法

对于对象来说 我们可以得到所有的属性值和调用它里面的方法

Java反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法；这种动态获取的以及动态调用对象的方法功能称为java的反射机制

**反射机制的优缺点**

优点：

运行期类型的判断，动态类的加载，动态代理使用反射

缺点：

性能是一个

**获取类对象的三种方式**

类名点 . class 接收返回值 获取到类对象

Class areaclass= Area.class;

第二种方式

Class areaclass2=Class.forName(“com.jgxi.entity.po.Area”); 填写类的全路径

第三种方式

New对象  通过对象名 get获取class对象

Area a = new Area;

Class areaClass3=a.getClass();

自定义注解 :

java的一种标识 帮我识别需要的类 属性 方法;

注解

元注解：是java提供的修饰自定义注解的四种注解，叫元注解，

自定义注解 @in

包括以下四种：

@Target:用于指定自定义注解的使用范围。

ElementType.FIELD：应用于全局属性

ElementType.METHOD：应用于方法

ElementType.PARAMETER：应用于方法的参数

ElementType.TYPE：应用于类、接口或者枚举声明

@Retention：用于修饰自定义注解的生存周期，或者可以保留多久

RetentionPolicy.SOURCE：源代码时保留，其他直接丢弃

RetentionPolicy.CLASS：默认值，编译器将把注解记录在class文件中，程序运行时，不会留下

RetentionPolicy.RUNTIME：编译器将把注解记录在class文件中，当运行java程序时，虚拟机保留注解，程序可以通过反射获取该注解；

@Documented：执行javadoc命令时，被该元注解修饰的自定义注解也会生成在文档中

@Inherited：如果父类所使用的注解有@Inherited修饰，则子类可以继承该注解，否则不能继承。

Zip 就是java提供的一个压缩包工具

Zipoutputstream zip输出流

Zipentry 相当于zip 中的一个位置

## Mysql优化

### Mysql优化哪些方面？

### 表设计上 存储引擎和字段类型

* 1. 存储引擎常用的有两种MylSam和InnoDB
     1. Mylsam不支持事务，不支持外键，采用表锁，侧重于性能；
     2. InnoDB支持事务，支持外键，采用行锁，侧重于事务；根据需求选择合适的存储引擎；
  2. 字段类型 给字段设置合适的数据类型，在合适的情况下使用整数型来代替字符串

### 功能上 创建索引

1. 索引简单说在列上添加标识，帮助Mysql高效获取数据。索引就是一种数据结构。
2. 索引的优点：

提高数据检索的效率，降低数据库的IO成本。

通过索引列对数据进行排序，降低数据排序的成本，降低了CPU的消耗

1. 索引的缺点：

虽然索引大大提高了查询速度，同时却会降低更新表的速度，如对表进行INSERT、UPDATE和DELETE。因为在更新表时，MySQL不仅要保存数据，还要保存一下索引文件每次更新添加了索引列的字段，都会调整因为更新所带来的键值变化后的索引信息。

1. 什么情况下应该创建索引
   1. **主键自动建立唯一索引**
   2. **频繁作为查询条件的字段应该创建索引**
   3. **查询中与其它表关联的字段，外键关系建立索引**
   4. **单键/组合索引的选择问题， 组合索引性价比更高**
   5. **查询中排序的字段，排序字段若通过索引去访问将大大提高排序速度**
   6. **查询中统计或者分组字段**
2. **什么情况不应该建立索引**
   1. **表记录太少**
   2. 经**常增删改的表或者字段**
   3. **Where 条件里用不到的字段不创建索引**
   4. **过滤性不好的不适合建立索引(字段重复的值比较多，超过50%)**

### SQL语句上 合理的sql

1. 对查询进行优化，尽量避免全表扫描，首先在where和order by涉及的字段上添加索引；
2. 避免在索引列上进行计算，导致索引失效；
3. 避免在where条件上对字段的空值进行判断，否则引擎放弃索引而进行全表扫描；

## 基于token的登陆

token的意思是“令牌”，是服务端生成的一串字符串，作为客户端进行请求的一个标识

Token Auth的优点

Token机制相对于Cookie机制又有什么好处呢？

支持跨域访问: Cookie是不允许垮域访问的，这一点对Token机制是不存在的，前提是传输的用户认证信息通过HTTP头传输.

无状态(也称：服务端可扩展行):Token机制在服务端不需要存储session信息，因为Token 自身包含了所有登录用户的信息，只需要在客户端的cookie或本地介质存储状态信息.

更适用CDN: 可以通过内容分发网络请求你服务端的所有资料（如：javascript，HTML,图片等），而你的服务端只要提供API即可.

去耦: 不需要绑定到一个特定的身份验证方案。Token可以在任何地方生成，只要在你的API被调用的时候，你可以进行Token生成调用即可.

更适用于移动应用: 当你的客户端是一个原生平台（iOS, Android，Windows 8等）时，Cookie是不被支持的（你需要通过Cookie容器进行处理），这时采用Token认证机制就会简单得多。

CSRF:因为不再依赖于Cookie，所以你就不需要考虑对CSRF（跨站请求伪造）的防范。

性能: 一次网络往返时间（通过数据库查询session信息）总比做一次HMACSHA256计算 的Token验证和解析要费时得多.

不需要为登录页面做特殊处理: 如果你使用Protractor 做功能测试的时候，不再需要为登录页面做特殊处理.

基于标准化:你的API可以采用标准化的 JSON Web Token (JWT). 这个标准已经存在多个后端库（.NET, Ruby, Java,Python, PHP）和多家公司的支持（如：Firebase,Google, Microsoft）.

### token的运行原理

#### token基本原理[#](https://www.cnblogs.com/loveer/p/11429034.html#3382584390)

Request指在一次请求的全过程中有效，即从http请求到服务器处理结束，返回响应的整个过程，存放在HttpServletRequest对象中。

Session是用户全局变量，在整个会话期间都有效。只要页面不关闭就一直有效（或者直到用户一直未活动导致会话过期，默认session过期时间为30分钟）

token 认证过程 \*

客户端使用用户名跟密码请求登录

服务端收到请求，去验证用户名与密码

验证成功后，服务端会签发一个 Token，再把这个 Token 发送给客户端

客户端收到 Token 以后可以把它存储起来，比如放在 session Storage 里或者 Local Storage 里

客户端每次向服务端请求资源的时候需要带着服务端签发的 Token

服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，就向客户端返回请求的数据

因为token是被签名的，所以我们可以认为一个可以解码认证通过的token是由我们系统发放的，其中带的信息是合法有效的

### cookie 和session的区别

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗  
   考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能  
   考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、所以个人建议：  
   将登陆信息等重要信息存放为SESSION  
   其他信息 如果需要保留，可以放在COOKIE中

### 基于JWT的Token认证[#](https://www.cnblogs.com/loveer/p/11429034.html#1352540416)

JSON Web Token（JWT）是一个非常轻巧的规范。这个规范允许我们使用JWT在用户和服务器之间传递安全可靠的信息。

JWT的组成：  
一个JWT实际上就是一个字符串，它由三部分组成，头部、载荷与签名

头部（Header）

用于描述关于该JWT的最基本的信息，例如其类型以及签名所用的算法等。这也可以被表示成一个JSON对象。

然后将其进行base64编码，得到第一部分

{"typ": "JWT",

"alg": "HS256"

}

载荷（Payload）

一般添加用户的相关信息或其他业务需要的必要信息。但不建议添加敏感信息，因为该部分在客户端可解密

（base64是对称解密的，意味着该部分信息可以归类为明文信息）

然后将其进行base64编码，得到第二部分

{ "iss": "JWT Builder",

"iat": 1416797419,

"exp": 1448333419,

"aud": "www.example.com",

"sub": "aaa@example.com",

"Email": "aaa@example.com",

"Role": [ "admin", "user" ]

}

iss:

该JWT的签发者，是否使用是可选的；

sub:

该JWT所面向的用户，是否使用是可选的；

aud:

接收该JWT的一方，是否使用是可选的；

exp(expires):

什么时候过期，这里是一个Unix时间戳，是否使用是可选的；

iat(issued at):

在什么时候签发的(UNIX时间)，是否使用是可选的；

nbf (Not Before):

如果当前时间在nbf里的时间之前，则Token不被接受；是否使用是可选的；

jti:

JWT的唯一身份标识，主要用来作为一次性token,从而回避重放攻击。

签名（Signature）

需要base64加密后的header和base64加密后的payload使用"."连接组成的字符串，

然后通过header中声明的加密方式进行加盐secret组合加密（在加密的时候，我们还需要提供一个密钥（secret），加盐secret组合加密）

然后就构成了jwt的第三部分。

最后，将这一部分签名也拼接在被签名的字符串后面，我们就得到了完整的JWT

注意：secret就是你服务端的私钥，在任何场景都不应该流露出去。

一旦客户端得知这个secret, 那就意味着客户端是可以自我签发jwt了

起源

说起JWT，我们应该来谈一谈基于token的认证和传统的session认证的区别。

### 传统的session认证

我们知道，http协议本身是一种无状态的协议，而这就意味着如果用户向我们的应用提供了用户名和密码来进行用户认证，那么下一次请求时，用户还要再一次进行用户认证才行，因为根据http协议，我们并不能知道是哪个用户发出的请求，所以为了让我们的应用能识别是哪个用户发出的请求，我们只能在服务器存储一份用户登录的信息，这份登录信息会在响应时传递给浏览器，告诉其保存为cookie,以便下次请求时发送给我们的应用，这样我们的应用就能识别请求来自哪个用户了,这就是传统的基于session认证。

但是这种基于session的认证使应用本身很难得到扩展，随着不同客户端用户的增加，独立的服务器已无法承载更多的用户，而这时候基于session认证应用的问题就会暴露出来.

### 基于session认证所显露的问题

**Session**: 每个用户经过我们的应用认证之后，我们的应用都要在服务端做一次记录，以方便用户下次请求的鉴别，通常而言session都是保存在内存中，而随着认证用户的增多，服务端的开销会明显增大。

**扩展性**: 用户认证之后，服务端做认证记录，如果认证的记录被保存在内存中的话，这意味着用户下次请求还必须要请求在这台服务器上,这样才能拿到授权的资源，这样在分布式的应用上，相应的限制了负载均衡器的能力。这也意味着限制了应用的扩展能力。

**CSRF**: 因为是基于cookie来进行用户识别的, cookie如果被截获，用户就会很容易受到跨站请求伪造的攻击。

### 基于token的鉴权机制

基于token的鉴权机制类似于http协议也是无状态的，它不需要在服务端去保留用户的认证信息或者会话信息。这就意味着基于token认证机制的应用不需要去考虑用户在哪一台服务器登录了，这就为应用的扩展提供了便利。

流程上是这样的：

* 用户使用用户名密码来请求服务器
* 服务器进行验证用户的信息
* 服务器通过验证发送给用户一个token
* 客户端存储token，并在每次请求时附送上这个token值
* 服务端验证token值，并返回数据

这个token必须要在每次请求时传递给服务端，它应该保存在请求头里， 另外，服务端要支持CORS(跨来源资源共享)策略，一般我们在服务端这么做就可以了Access-Control-Allow-Origin: \*。

## 日志记录步骤

要做aop日志记录，首先我们需要的是自定义注解，自定义注解可以说它是一个标记点，可以添加在类、方法、属性上，注解不会对所修饰的代码产生直接的影响。便于在其他地方找到这个标记点，得到这个标记点所在位置的各项数据。创建自定义注解，先创建@Interface类，创建aop日志记录的自定义注解要使用的相关的注解有@Target，@Rentention，这两个注解的作用分别是：

@Target：是专门用来限定某个自定义注解能够被应用在哪些Java元素上面的（类、方法、属性）。

@Rentention：声明这个自定义注解的生命周期，我在使用的时候一般使用RUNTIME。

创建好自定义注解后，将其添加到controller层中需要进行日志记录方法上，之后就可以写日志记录的核心代码了。

创建一个类，在类上添加两个注解，分别是@Component @Aspect，这两个注解的作用分别是：

@Component：将本类交给ioc管理

@Aspect：声明这个是切面类。

在该类中新建方法，方法上添加注解，声明aop的通知，

aop的通知共有五个，@Before前置通知，@After后置通知，@AfterThrowing异常通知， @Around环绕通知，@AfterReturning最终通知，设置参数，表明横切那个层，在这里我使用@Before前置通知，横切controller层。

在方法上添加参数JoinPoint，

使用它，来获取方法的签名，再根据方法签名来查找到所有的方法。这时候在之前设置的自定义注解便要发挥作用了。

代码如下:

//构建日志对象  
LogContent logContent = new LogContent();  
//获取对象签名  
MethodSignature methodSignature = (MethodSignature) joinPoint.getSignature();  
//获取方法对象  
Method method = methodSignature.getMethod();

一般来说自定义注解会和反射一起使用，以此发挥出最大的作用。所谓反射，便是：对类来说，能知道它的所有属性和方法；对于对象来说，能得到所有的属性值和调用的方法，对象，即万物皆对象，注解也不例外。在此，使用getAnnotation，便可以得到添加自定义注解方法的各个数据，如操作数据，请求路径等。

得到相应数据后，将其保存即可，自此，日志记录完成。

## HashMap的底层原理：

我们常用的数据结构有三种：数组结构、链表结构、哈希表结构

数组结构：查询和修改比较快，原因是数组是连续的，但是插入和删除比较慢

链表的结构：插入和删除比较快，查询和修改比较慢

哈希表结构：就是把数组和链表的优点组合起来

HashMap就是使用哈希结构，当我们往HashMap中put元素时，会根据key调用hashCode()方法得到一个HashCode值，在通过哈希算法取模数组的长度得到一个下标，得到下标后查看当前下标位置是否有元素，如果没有元素的话将值放到当前位置,否则就放入改位置对应的链表中，采用的是头插法

Map中get()获取元素时，会通过key调用HashCode()方法得到HashCode值，通过哈希算法取模数组的长度得到下标，根据下标快速的定位到某个位置，如果这个位置什么都没有就返回一个null，。

Jdk1.8之前采用的是头插法，jdk1.8之后是尾插法，当我们的链表长度达到8的时候就会变成红黑树

## 基于token的登陆

token的意思是“令牌”，是服务端生成的一串字符串，作为客户端进行请求的一个标识

Token Auth的优点

Token机制相对于Cookie机制又有什么好处呢？

支持跨域访问: Cookie是不允许垮域访问的，这一点对Token机制是不存在的，前提是传输的用户认证信息通过HTTP头传输.

无状态(也称：服务端可扩展行):Token机制在服务端不需要存储session信息，因为Token 自身包含了所有登录用户的信息，只需要在客户端的cookie或本地介质存储状态信息.

更适用CDN: 可以通过内容分发网络请求你服务端的所有资料（如：javascript，HTML,图片等），而你的服务端只要提供API即可.

去耦: 不需要绑定到一个特定的身份验证方案。Token可以在任何地方生成，只要在你的API被调用的时候，你可以进行Token生成调用即可.

更适用于移动应用: 当你的客户端是一个原生平台（iOS, Android，Windows 8等）时，Cookie是不被支持的（你需要通过Cookie容器进行处理），这时采用Token认证机制就会简单得多。

CSRF:因为不再依赖于Cookie，所以你就不需要考虑对CSRF（跨站请求伪造）的防范。

性能: 一次网络往返时间（通过数据库查询session信息）总比做一次HMACSHA256计算 的Token验证和解析要费时得多.

不需要为登录页面做特殊处理: 如果你使用Protractor 做功能测试的时候，不再需要为登录页面做特殊处理.

基于标准化:你的API可以采用标准化的 JSON Web Token (JWT). 这个标准已经存在多个后端库（.NET, Ruby, Java,Python, PHP）和多家公司的支持（如：Firebase,Google, Microsoft）.

### token的运行原理

#### token基本原理[#](https://www.cnblogs.com/loveer/p/11429034.html#3382584390)

Request指在一次请求的全过程中有效，即从http请求到服务器处理结束，返回响应的整个过程，存放在HttpServletRequest对象中。

Session是用户全局变量，在整个会话期间都有效。只要页面不关闭就一直有效（或者直到用户一直未活动导致会话过期，默认session过期时间为30分钟）

token 认证过程 \*

客户端使用用户名跟密码请求登录

服务端收到请求，去验证用户名与密码

验证成功后，服务端会签发一个 Token，再把这个 Token 发送给客户端

客户端收到 Token 以后可以把它存储起来，比如放在 session Storage 里或者 Local Storage 里

客户端每次向服务端请求资源的时候需要带着服务端签发的 Token

服务端收到请求，然后去验证客户端请求里面带着的 Token，如果验证成功，就向客户端返回请求的数据

因为token是被签名的，所以我们可以认为一个可以解码认证通过的token是由我们系统发放的，其中带的信息是合法有效的

### cookie 和session的区别

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗  
   考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能  
   考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、所以个人建议：  
   将登陆信息等重要信息存放为SESSION  
   其他信息 如果需要保留，可以放在COOKIE中

### 基于JWT的Token认证[#](https://www.cnblogs.com/loveer/p/11429034.html#1352540416)

JSON Web Token（JWT）是一个非常轻巧的规范。这个规范允许我们使用JWT在用户和服务器之间传递安全可靠的信息。

JWT的组成：  
一个JWT实际上就是一个字符串，它由三部分组成，头部、载荷与签名

头部（Header）

用于描述关于该JWT的最基本的信息，例如其类型以及签名所用的算法等。这也可以被表示成一个JSON对象。

然后将其进行base64编码，得到第一部分

{"typ": "JWT",

"alg": "HS256"

}

载荷（Payload）

一般添加用户的相关信息或其他业务需要的必要信息。但不建议添加敏感信息，因为该部分在客户端可解密

（base64是对称解密的，意味着该部分信息可以归类为明文信息）

然后将其进行base64编码，得到第二部分

{ "iss": "JWT Builder",

"iat": 1416797419,

"exp": 1448333419,

"aud": "www.example.com",

"sub": "aaa@example.com",

"Email": "aaa@example.com",

"Role": [ "admin", "user" ]

}

iss:

该JWT的签发者，是否使用是可选的；

sub:

该JWT所面向的用户，是否使用是可选的；

aud:

接收该JWT的一方，是否使用是可选的；

exp(expires):

什么时候过期，这里是一个Unix时间戳，是否使用是可选的；

iat(issued at):

在什么时候签发的(UNIX时间)，是否使用是可选的；

nbf (Not Before):

如果当前时间在nbf里的时间之前，则Token不被接受；是否使用是可选的；

jti:

JWT的唯一身份标识，主要用来作为一次性token,从而回避重放攻击。

签名（Signature）

需要base64加密后的header和base64加密后的payload使用"."连接组成的字符串，

然后通过header中声明的加密方式进行加盐secret组合加密（在加密的时候，我们还需要提供一个密钥（secret），加盐secret组合加密）

然后就构成了jwt的第三部分。

最后，将这一部分签名也拼接在被签名的字符串后面，我们就得到了完整的JWT

注意：secret就是你服务端的私钥，在任何场景都不应该流露出去。

一旦客户端得知这个secret, 那就意味着客户端是可以自我签发jwt了

### 起源

说起JWT，我们应该来谈一谈基于token的认证和传统的session认证的区别。

### 传统的session认证

我们知道，http协议本身是一种无状态的协议，而这就意味着如果用户向我们的应用提供了用户名和密码来进行用户认证，那么下一次请求时，用户还要再一次进行用户认证才行，因为根据http协议，我们并不能知道是哪个用户发出的请求，所以为了让我们的应用能识别是哪个用户发出的请求，我们只能在服务器存储一份用户登录的信息，这份登录信息会在响应时传递给浏览器，告诉其保存为cookie,以便下次请求时发送给我们的应用，这样我们的应用就能识别请求来自哪个用户了,这就是传统的基于session认证。

但是这种基于session的认证使应用本身很难得到扩展，随着不同客户端用户的增加，独立的服务器已无法承载更多的用户，而这时候基于session认证应用的问题就会暴露出来.

### 基于session认证所显露的问题

**Session**: 每个用户经过我们的应用认证之后，我们的应用都要在服务端做一次记录，以方便用户下次请求的鉴别，通常而言session都是保存在内存中，而随着认证用户的增多，服务端的开销会明显增大。

**扩展性**: 用户认证之后，服务端做认证记录，如果认证的记录被保存在内存中的话，这意味着用户下次请求还必须要请求在这台服务器上,这样才能拿到授权的资源，这样在分布式的应用上，相应的限制了负载均衡器的能力。这也意味着限制了应用的扩展能力。

**CSRF**: 因为是基于cookie来进行用户识别的, cookie如果被截获，用户就会很容易受到跨站请求伪造的攻击。

### 基于token的鉴权机制

基于token的鉴权机制类似于http协议也是无状态的，它不需要在服务端去保留用户的认证信息或者会话信息。这就意味着基于token认证机制的应用不需要去考虑用户在哪一台服务器登录了，这就为应用的扩展提供了便利。

流程上是这样的：

* 用户使用用户名密码来请求服务器
* 服务器进行验证用户的信息
* 服务器通过验证发送给用户一个token
* 客户端存储token，并在每次请求时附送上这个token值
* 服务端验证token值，并返回数据

这个token必须要在每次请求时传递给服务端，它应该保存在请求头里， 另外，服务端要支持CORS(跨来源资源共享)策略，一般我们在服务端这么做就可以了Access-Control-Allow-Origin: \*。

## 多线程导出

当时在公司做一个报表,需要使用excel导出,做了好几个版本,进行优化,第一版 就选择了poi导出,通过反射+注解+zip来做,

反应慢,我们加上线程池;

反射是什么

自定义注解

线程池

Zip

1. 配置:我们需要配置上poi的POM依赖,直接可以使用
2. 思路:
3. 首先需要用户传来两个参数,即我们每个sheet的数据量(datasize)以及每个excel中有多少个sheet页即(sheetsize)
4. 查出我们需要导出的(数据量);

sheetAllCount(sheet总数)=总数据量/datasize;

excelCount(excel数量)= (sheet总数)/ (sheetsize);

1. 循环excel总数 每次循环 相当于创造一个excel的数据 , 创建workbox

再循环我们excel中sheet页的数量;得到当前excel中的每个sheet;创建之前 我们需要这个sheet的数据;我们需要通过分页查询按批次查询数据; 那么我们就需要 数据的起始下标与条数 条数我们可以传 每个sheet的数据量(datasize)

起始下标 :当前excel-1 \* sheetAllCount(sheet总数) x2 我们得到每个sheet数据,然后通过集合中的对象得到类对象;

1. 通过反射得到我们javabean的属性,给要添加的属性设置上自定义注解,value为我们的表头信息, getDeclaredFields获取所有字段的集合;循环判断加了注解;将注解的value设为表头;

Createcell.setcellvalue();

1. 循换每个sheet的数据; 创建row 在循环我们反射的bean属性;如果加了注解就创建一个cell;并判断类型;放入对应cell中;
2. 写入我们服务器文件中
3. 为了导出速度;我们加入线程池; 并使用contdownlach;
4. 等所有线程执行完成;我们通过java提供的zip进行压锁并响应; 将响应流加入我们的zip输出流;读写我们excel文件夹; 完成多线程导出;

线程池:

线程是系统非常珍贵的资源;线程池就是一个容器;用来存放线程;并帮我们线程执行任务;

可以减少我们线程创建与销毁的资源消耗;

创建方式 自定义型

Threadpoolexctor t=new Thredddpoolexctor

参数 :

1.核心线程数 即空闲也不会销毁的线程 核心数不足 提交任务会创建

2.最大线程数;核心数满了 就创建非核心 ;想家不能超过最大线程数

3.非核心超时时间: 非核心线程限制多久被回收;

4.线程队列:队列怎么处理线程 是直接创建新线程还是等待还是直接交给核心执行

5,饱和策略:丢弃任务 丢弃其他任务 新建线程 报异常

### 常用线程:

Fixedthredpool=Excutors.newfixedthredpool

固定大小的线程;核心与最大一致 控制并发量 异常 建新的

Cachedthredpool可缓存的线程池; 核心0 最大 max 有任务创建线程; 闲置回收 节约资源

Singlethredexecutor

一个单线程的线程池; 保证顺序

**ScheduledThreadPool**

创建一个可以定时 或 延时的线程池

还有一个单线程的定时延时线程

### 反射

反射是什么 反射就是可以让我们识别一个未知的类,一个未知的对象;

对于类来说就是 我们可以知道他里面的所有属性和所有方法

对于对象来说 我们可以得到所有的属性值和调用它里面的方法

Java反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意一个方法；这种动态获取的以及动态调用对象的方法功能称为java的反射机制

**反射机制的优缺点**

优点：

运行期类型的判断，动态类的加载，动态代理使用反射

缺点：

性能是一个

**获取类对象的三种方式**

类名点 . class 接收返回值 获取到类对象

Class areaclass= Area.class;

第二种方式

Class areaclass2=Class.forName(“com.jgxi.entity.po.Area”); 填写类的全路径

第三种方式

New对象  通过对象名 get获取class对象

Area a = new Area;

Class areaClass3=a.getClass();

自定义注解 :

一种标识 帮我识别需要的类 属性 方法;

@interf

元注解

@target 声明作用未知

@retention 声明生命周期

String value( ) default “”;

Zip 就是java提供的一个压缩包工具

Zipoutputstream zip输出流

Zipentry 相当于zip 中的一个位置

# RabbitMQ

削峰 异步 解耦

面对我们一些高并发场景 ;我们的项目可能会扛不住;

响应慢;

请求量大;

消息队列是什么 就是可以暂时存储生产者的数据;暂存 等待消费者来消费;

削峰 就是我们有3000个请求 生产者将数据交给消息队列 就可以返回了; 我们消费者可以先拿1000个数据消费;

解耦 我们有 ABCD 四个服务 a 生产出一个id ;

放入 消息队列中 其他服务调用与否 我们a是不用关心的

异步 就是我们数据生产出来了 ;我们可以一会消费它;

生产者将数据放入 消息队列中就可以做其他的了;

消费者可以根据策略再去选择消费数据;

我们的消息队列 有两大核心 交换机 和队列组成的

有三种推送模式

**Direct Exchange**

直连型交换机，根据消息携带的路由键将消息投递给对应队列。

大致流程，有一个队列绑定到一个直连交换机上，同时赋予一个路由键 routing key 。

然后当一个消息携带着路由值为X，这个消息通过生产者发送给交换机时，交换机就会根据这个路由值X去寻找绑定值也是X的队列。

**Fanout Exchange**

扇型交换机，这个交换机没有路由键概念，就算你绑了路由键也是无视的。 这个交换机在接收到消息后，会直接转发到绑定到它上面的所有队列。

**Topic Exchange**

主题交换机，这个交换机其实跟直连交换机流程差不多，但是它的特点就是在它的路由键和绑定键之间是有规则的。

**消息确认:Message acknowledgment**

在实际应用中，可能会发生消费者收到Queue中的消息，但没有处理完成就宕机（或出现其他意外）的情况，这种情况下就可能会导致消息丢失。为了避免这种情况发生，我们可以要求消费者在消费完消息后发送一个回执给RabbitMQ，RabbitMQ收到消息回执（Message acknowledgment）后才将该消息从Queue中移除；如果RabbitMQ没有收到回执并检测到消费者的RabbitMQ连接断开，则RabbitMQ会将该消息发送给其他消费者（如果存在多个消费者）进行处理。这里不存在Timeout概念，一个消费者处理消息时间再长也不会导致该消息被发送给其他消费者，除非它的RabbitMQ连接断开。

这里会产生另外一个问题，如果我们的开发人员在处理完业务逻辑后，忘记发送回执给RabbitMQ，这将会导致严重的问题，Queue中堆积的消息会越来越多，消费者重启后会重复消费这些消息并重复执行业务逻辑。

如果我们采用no-ack的方式进行确认，也就是说，每次Consumer接到数据后，而不管是否处理完成，RabbitMQ会立即把这个Message标记为完成，然后从queue中删除了。

**Dead Letter Exchange（死信交换器）**

在队列上指定一个Exchange，则在该队列上发生如下情况，

1.消息被拒绝（basic.reject or basic.nack)，且requeue=false

2.消息过期而被删除（TTL）

3.消息数量超过队列最大限制而被删除

4.消息总大小超过队列最大限制而被删除

就会把该消息转发到指定的这个exchange

需要定义了x-dead-letter-exchange属性，同时也可以指定一个可选的x-dead-letter-routing-key，表示默认的routing-key，如果没有指定，则使用消息原来的routeing-key进行转发

当定义队列时指定了x-dead-letter-exchange（x-dead-letter-routing-key视情况而定），并且消费端执行拒绝策略的时候将消息路由到指定的Exchange中去。

我们知道还有二种情况会造成消息转发到死信队列。

一种是消息过期而被删除，可以使用这个方式使的rabbitmq实现延迟队列的作用。还有一种就是消息数量超过队列最大限制而被删除或者消息总大小超过队列最大限制而被删除

# linux命令

ip addr 查看ip地址

pwd 查看当前所在的路径

systemctl status firewalld 查看防火墙状态

systemctl satrt firewalld 启动防火墙

systemctl stop firewalld 关闭防火墙

mkdir 创建文件夹 加 -p的话是创建多级目录

ps –ef|grep 进程名 查看某个进程的运行状态

rz 上传文件

tar –zxvf 文件 解压压缩包

cp file1 file2 复制一个文件

rm –rf file1 删除文件