基本信息 **面试常见问题：**

|  |  |  |  |
| --- | --- | --- | --- |
| 基本情况 | | | |
| **姓名** | 李瑞华 | 性别 | 男 |
| **出生日期** | 1992-07 | 属相 | 猴 |
| **现住地** | 高新区 |  |  |
| 大学情况 | | | |
| **学校** | 焦作大学 | 专业 | 应用电子技术 |
| **入学时间** | 2009-12 | 毕业时间 | 2012-07 |
| **专业课** | **计算机硬件实训、C语言、数据库原理、操作系统、Java程序设计** | | |
| **学历** | **专科** | 学位 | **学士学位** |
| **学校地址** | 河南省焦作市山阳区 | 校长 | **学校领导都没见过，所以也不知道校长的名字** |
| **去学校的方式：** |  | | |

|  |  |
| --- | --- |
| 第一家公司 ： | 上海点明信息技术有限公司 |
| **地址：** | 上海市闵行区光华路2118号 |
| **入职时间：** | **2015年12月** |
| **离职时间：** | **2017年5月** |
| **公司性质：** | **民营** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **王林** |
| **薪资方面：** | **刚刚过去工资是6.5K左右，过完年大概三月份涨到8K左右 辞职时9.5K** |
| **路线：** | 闵行35路 银康路富才路 昆阳路光华路 |
|  |  |

|  |  |
| --- | --- |
| 第二家公司 ： | 北京和悦科技有限公司 |
| **地址：** | 北京海淀区西三旗上奥世纪2A座 上奥世纪 |
| **入职时间：** | **2017年6月** |
| **离职时间：** | **2019年1月** |
| **公司性质：** | **民营** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **周伟** |
| **薪资方面：** | **刚刚过去工资是10K左右 辞职时13K** |
| **路线：** | 432路 天通苑站上 西三旗桥东站下车 |
|  |  |

|  |  |
| --- | --- |
| **第三家公司情况：** | 北京汇才卓远科技有限公司 |
| **公司地址**： | 北京市门头沟区石龙经济开发区永安路20号 石龙高科大厦 |
| **公司规模：** | **40人左右，我们项目组8人，一个项目经理，1名前端，1个测试，1个运维，其他都是Java工程师** |
| **公司性质：** | **民营** |
| **公司行业：** | 技术开发、技术咨询、技术转让、技术服务；软件开发，专业承包； |
| **去公司路线：** | 941路 石景山南宫站 石景山西公交场 （住绿城西府海棠） |
| **入职时间：** | **2019-03** |
| **离职时间：** | **2021-1** |
| **职位：** | **Java工程师** |
| **项目经理名字：** | **杨学** |
| **税前：** | **17K** |
| **税后：** | **15K** |
| **缴税：** | **1K左右(具体原因不清楚当时公司就是这样扣的税)** |
| **期望薪资：** | **16K** |
| **薪资的构成:** | **基本工资(6K)+绩效工资(9K)** |

|  |
| --- |
| 四年涨薪： **2016年工作的时候是6.5K，税后大概6K左右，**  **转正7.5K左右**  **2016上半年涨了0.5K左右 ，总8K左右。**  **2016下半年涨了0.5K 总共8.5K左右**  **2017过完年涨了1K 9.5K**  **2017年辞职9.5K**  **2017年6月工作10K左右**  **2017年底11K**  **2018年6月12K左右**  **2019年初离职时13K左右**  **2019年3月 13.5K**  **2019年年底为15.5K**  **2020年6月17K** |

|  |
| --- |
| **你为什么从上家公司离职？** |
| **上家公司的福利待遇也算不错，和同事相处的也蛮开心的，但是在技术提升方面不太符合自己发展的预期，另外离家也远不方便。** |
| **你交社保了吗？** |
| **交了** |
| |  | | --- | | **三个词形容自己**：有责任心强，交流表达能力强，团队协作能力强 | | **你的优缺点是啥？** |   **优点:**  **善于倾听，团队协作能力强，责任心强。**  **缺点: 这几年做程序，因为要不断的对代码进行验证确认，所有感觉自己现在有点强迫症。** |
| **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 16K 上家工资15K 税后14K左右 扣了1K左右的税 |
| 你的五年规划是啥？ |
| **近期的话我会先让自己的技术能力得到提高，当然，尽快的掌握公司的业务，投入到工作中；如果可以的话自己想网架构师方面发展** |
| 是统招吗 是 |
| 你们公司还有什么福利？多少薪： 14薪  **车补**  200左右  **项目奖金**  7k-10k，做完项目，收完尾款的时候。  **年终奖**  这个就是1个月的工资，  过年的时候和当月工资一块发  **饭/餐补**  30/天  **电脑补**  300 |

**公司内其他部门：**

**技术部（开发部/研发部），行政部，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！**

坐稳再开口！！ 主动性！！ 开门见山！！大白话！！三秒钟原则！！学会反问！！

**Hr人事，leader领导，team团队，pm项目/产品经理，cto首席技术执行官，coo首席运营官，CEO董事长、**

# 事务

### 1、什么是事务？

事务是一组逻辑操作单元，即一组sql语句。要么都执行，要么都不执行。

### 2、事务的四大特性

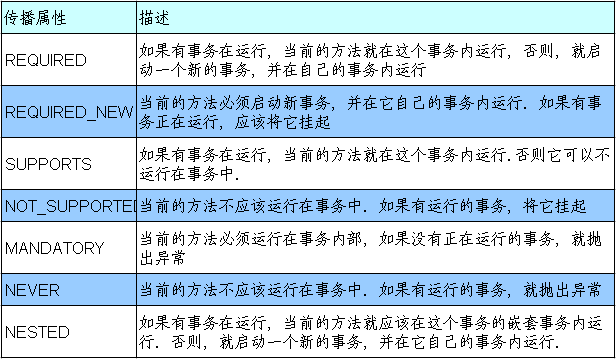
原子性：事务要么都执行，要么都不执行；

一致性：事务使数据库从一种一致性状态到另外一种一致性状态；

隔离性：事务之间是相互隔离的。一个事务内部使用的数据以及操作对其他 并发执行的事务是相互隔离的，不能相互干扰；

持久性：事务一旦提交对数据的改变就是永久的，接下来的任何故障都不会 对其有影响；

### 事务的传播特性：



### 事务的隔离级别

1）read\_uncommitted（读未提交）

可读取未提交事务的操作数据，最低的隔离级别，一般都没有用的。这种情况会出现脏读。

2）read\_committed（读已提交）

一个事务等另一个事务提交之后才可进行读取，解决了脏读问题，但会出现不可重复读

3）repeatable\_read（可重复读）

读取事务开启的时候不能对数据进行修改，解决了不可重复读问题，但是存在幻读问题；

4）serializable（序列化）：是最高的事务隔离级别，可以避免脏读、不可重复读与幻读。但是这种事务隔离级别效率低下，比较耗数据库性能，一般不使用；

# 基于token的登录认证

## 1、什么是JWT

JSON Web Token (JWT)是一个开放标准(RFC 7519)，它定义了一种紧凑的、自包含的方式，用于作为JSON对象在各方之间安全地传输信息。该信息可以被验证和信任，因为它是数字签名的。

## **2、基于token的鉴权机制**

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

## **3、JWT的构成**

第一部分我们称它为头部（header),

第二部分我们称其为载荷（payload, )，

第三部分是签证（signature).

## 基于token登录认证的步骤

1. 用户发送携带用户名及密码的请求到后台服务；
2. 后台服务进行验证成功后，调用JWT的sign方法生产token令牌；
3. 将token值返回给前台浏览器，并存入sessionStorage；
4. 后续客户端发起请求时都需要携带该token参数；
5. 后台拦截器进行token的uusign解密验证，若没有token值或解密失败则进行拦截；

## 5、相关问题

①**sessionStorage与localStorage的区别？**

localStorage的生命周期是永久性的。假若使用localStorage存储数据，即使关闭浏览器，也不会让数据消失，除非主动的去删除数据；

sessionStorage 的生命周期是在浏览器关闭前，在整个浏览器未关闭前，其数据一直都是存在的；

sessionStorage不在不同的浏览器窗口中共享；localstorage在所有同源窗口中都是共享的；

②**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. 建议将登录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
6. session保存在服务器，客户端不知道其中的信心；cookie保存在客户端，服务器能够知道其中的信息
7. session中保存的是对象，cookie中保存的是字符串
8. session不能区分路径，同一个用户在访问一个网站期间，所有的session在任何一个地方都可以访问到，而cookie中如果设置了路径参数，那么同一个网站中不同路径下的cookie互相是访问不到的

# 权限管理

### 权限管理的实现

权限管理对我们目前来说主要通过三块来进行实现。

1设型:RBAC 模型

RBAC 模型:也就是基于角色的权限控制，有别于传统模型中的直接把权限赋予账号. RBAC 模型增加了“角色"的概念，把权限赋予角色，再把角色赋予用户，提高了账号的管理效率，降低了出错的概率。

2数据库设计:权限五表<用户表，角色表，权限表，用户角色关系表，角色权限关系表>用户表、角色表、权限表都根据需求进行相应设计，同时由于用户和角色、角色和权限都是多对多的关系，所以引入了角色用户关系表和角色权限关系表来配置他们之间的多对多关系。

3、 后台系统设计:三个模块<用户模块、角色模块、权限模块>

1:用户新增或者修改的时候可以对角色进行勾选进行关联，角色是通过数据库查出来，然后展示在新增或者修改的弹窗中，以复选框或者下拉框进行展示，方使用户勾选。

2)角色模块:用户在新增或者修改角色的时候可以对权限勾选进行关联，权限是通过数据库查出来，然后以 zTree 插件展示在新增或者修改的弹窗中，另外对角色进行副除时还需要查看此角色是否被用户占用，如占用则不能删除。

3)权限模块:使用zTree插件进行展示，同时在删除时要把选中节点的所有子节点一并删除，另外还需要查看此权限是否被角色占用，如占用则不能删除，同时在新增或者修改时必须选中一个节点才可以新增或者修改，否则将进行弹窗提示。

# 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；

# AOP日志记录

### 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，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参数。

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的通知类型，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，使用它的来获取方法的签名，再根据方法签名来查找到对应的方法。（getSignature()获取方法签名对象，getMethod()获取切点方法）

通过反射的getAnnotation()获取方法上的注解，判断方法上注解的类对象是否与我们自定义注解的类对象一致，若一致通过反射获取到所需要记录的内容；

# 多线程导出

### POI导出步骤

做poi导出，我们同样需要使用反射和自定义注解，但不同的是，这时候自定义注解添加的位置成了属性，作用是为了得到需要导出的字段以及相应的数据。

最初，我只是做了一个简单的Excel表，使用XSSFWorkbook，生成sheet页、行、列，随之放入数据，最后写入Excel中，关流即可。这样写并没有什么问题，开发和测试都一切顺利。但是到了上线阶段数据量上去之后就不适用了。

后来我想到了生成多个excel文件多个sheet页并将其做成一个压缩包，这样再多的数据量也不会有事了。我先是尝试了使用流的方式进行操作。首先创建了ServletOutputStrea流和ZipOutputStream流，在此，我们需要确定每个sheet页数据的条数和sheet页的个数，以此来确定excel文件的个数，然后使用HSSFWorkbook通过循环，生成sheet页、行、列，随后放入数据，生成excel，依次写入zip压缩流下载完成。

再之后，需求发生了变化，需要做成多线程的poi导出，（此时先介绍多线程）这时，我之前做的那种思路的zip压缩流就不能再用了，因为它无法加到多线程中去。我换了一个思路，先在服务器上创建了一个临时文件夹，通过多线程去生成多个的excel文件，将其放到临时文件夹中。这些都完成之后，通过FileInputStream流依次读取临时文件夹中的数据，再通过ZipEntry来进行写入，最后下载zip压缩包，关流。至此，多线程poi导出便写完了。

# redis非关系型数据库

## **1、redis是什么**

redis是存储key-value格式数据，基于内存的非关系型数据库

因为redis是基于内存的效率非常快(读取11000次每秒 写入是8100次每秒)，它支持数据持久化，支持事务所有的操作都是原子性的，有多种数据类型（String、Hash、List、Set、ZSet）。

redis有两种持久化方案 rdb和aof

rdb基于快照形式，定时对redis进行拍照记录，效率高，但是有一个问题就是当你的缓存在两次快照的间隔时间内宕机的话，会造成该时间片内操作的数据丢失；

aof基于日志形式的，它是对redis的增删改操作一一进行记录，效率不如rdb，但不会丢失数据；

redis的所有操作都是原子性的 redsi是单线程 事务单个执行

## 2、redis的数据类型

**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rpop(弹出)redis数据库中存储的数据 就是去除数组中的最后一条数据

**zset(sorted set：有序集合) Set（集合）**

不好意思 没用过！！！

## 3、redis三大问题

### 1 缓存穿透

缓存穿透是指缓存服务器中没有缓存数据，数据库中也没有符合条件的数据，导致业务

系统每次都绕过缓存服务器查询下游的数据库，缓存服务器完全失去了其应有的作用。

**如何解决缓存穿透？**

一般有2种解决办法，分别是缓存空值和过滤器缓存空值在缓存中，之所以会发生穿透，就是因为缓存没有对那些不存在的值得Key缓存下来，从而导致每次查询都要请求到数据库。 那么我们就可以为这些key对应的值设置为null并放到缓存中，这样再出现查询这个key 的请求的时候，直接返回null即可。但是还需要注意的就是需要有一个失效时间，因为如果不设置失效的话，如果哪天有值了就会导致问题。

**BloomFilter布隆过滤器**

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

## 4、redis主从复制

    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。

**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。

## 5、Sentinel哨兵模式

**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，导致系统故障。这时候我们就采用哨兵模式来解决该问题；

 哨兵是redis的一种特殊节点，它不存储数据，但可以做故障发现、故障转移；

**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， 并让其他的从服务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。

**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。

**自动故障切换（Automatic failover）**

**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。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。

## 6、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

# Docker容器引擎

## 1、docker是什么？

Docker是一款可以解决运行环境和配置问题的软件容器引擎；

Docker的组成部分（容器，镜像（image），仓库）

## 2、docker的相关命令

docker version 查看docker版本

docker info 查看docker的详细信息（cpu 内存 容器 镜像）

**镜像的操作命令：**

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 镜像名:tag 下载一个centos 的一个镜像 docker pull centos 下载镜像不指定版本

docker rmi 镜像名（id） 注意正在被容器使用的镜像是无法删除

**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）

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

退出容器 Ctrl+P+Q

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 如果想把运行中的容器也删掉 加参数 -f）

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

## 3、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:/容器目录 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 宿主名录: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

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

## 4、制作镜像

commit

build（Dockerfile）

编写Dockerfile文件

语法 ：

FROM 指定基础镜像

MAINTAINER 指定镜像维护者信息

RUN 用于执行指定脚本命令

CMD 指定启动容器时执行的命令

EXPOSE 指定容器暴露的端口

ENV 指定环境变量

ADD 将文件从宿主机复制到容器指定位置，同时对压缩文件有自动解压功能

COPY 将文件从宿主机复制到容器指定位置

ENTRPOINT 设置容器启动时需要运行的命令

WORKDIR 为后续的如RUN、CMD、ENTRYPOINT、COPY、ADD指定工作目录

# **定时任务 发邮件 发短信 会员续费）**

## 1、ssm下设置定时器TimerTask

在springMVC.xml配置文件中配置：

xmlns下添加：

xmlns:task="http://www.springframework.org/schema/task"

在xsi中添加

http://www.springframework.org/schema/task

http://www.springframework.org/schema/task/spring-task-3.2.xsd

这里要注意添加位置，加在xsi的最后面，位置不当有可能报错。

在中间添加

<!-- 设置定时任务 -->

<task:annotation-driven/>

然后新建我们的定时器类TimerTask

在需要定时执行的方法上添加我们的注解 来设置我们的定时时间

@Scheduled(cron = "0 30 22 ? \* \*")

设置上之后这个方法的方法体就变成了我们定时任务的执行体，等定时时间达到之后 就会去执行方法体里面的内容，注意类上需要添加我们的component注解、component注解是把我们这个类交给aop进行管理

## 2、应用场景

**发送邮件**

我们需要在定时任务的执行体里面添加一个分布式锁，目的是为了防止多台服务器集群同时触发我们的定时任务造成的邮件重复问题，这个分布式锁我们用到了redis中的getset方法，redis的getset方法redis中string类型下的它会存进去新的值并且返回我们原来的值，原来没有值的话返回null； 通过getset方法把我们的当前时间作为key存到我们的redis中，并且给设置一个过期时间，之后判断getset的返回值是否为空，不为空那我们就给他直接返回，并且提示一下我们的任务已经执行过了为空的话我们就先去查询我们数据库中今天过生日的用户，之后创建一个线程池来发送邮箱线程池核心线程数根据自己的用户数量和服务器配置来均衡一个中间点，我们这用到了三个核心线程数。

循环我们的用户数据，创建我们的线程穿过去几个参数1.我们邮件的标题2.邮件的内容3.邮件接收者，之后再我们的线程执行体中（run方法中）去调用一下我们邮件的工具类，工具类中配置号我们发送方的邮箱号，设置我们的发送方的邮箱类型（smtp.qq.com）配置发件人的邮箱密码 （密码是我们在账号设置中设置的smtp的密匙） 设置发送人，接收人，标题，内容，这就实现了我们的邮箱发送

## 3、多线程是什么？

多线程就是说我们任务执行的时间有些长，为了不让用户长时间等待，我们将这个时间繁琐点的任务放到多线程中去执行 主线程直接返回给用户

## 4、创建线程的有哪些方式？

1）继承Thread类创建线程类

继承thread重写run方法

直接new 线程对象.start执行

2）通过Runnable接口创建线程类

实现runnable重写run方法

new thread

new 线程类 吧线程类放到thread中

然后通过start启动

3）通过线程池创建

线程池就可以理解为我们的线程容器

线程池的目的是为了减少我们线程的运行时间

不需要每次都去创建线程 执行后再销毁线程了 而是每次过来都去线程池中取一

个线程，执行完之后吧线程归还给线程池中

//创建一个线程池

ExecutorService executorService = Executors.newFixedThreadPool(3);//设置主线程3Java线程具有五中基本状态

1）新建状态（New）：当线程对象对创建后，即进入了新建状态，如：

Thread t = new MyThread()；

2）就绪状态（Runnable）：当调用线程对象的start()方法（t.start();），线程

即进入就绪状态。处于就绪状态的线程，只是说明此线程已经做好了准备，随时

等待CPU调度执行，并不是说执行了t.start()此线程立即就会执行；

3）运行状态（Running）：当CPU开始调度处于就绪状态的线程时，此时线程

才得以真正执行，即进入到运行状态。注：就

绪状态是进入到运行状态的唯一入口，也就是说，线程要想进入运行状态执行，

首先必须处于就绪状态中；

4）阻塞状态（Blocked）：处于运行状态中的线程由于某种原因，暂时放弃对

CPU的使用权，停止执行，此时进入阻塞状态，直到其进入到就绪状态，才 有

机会再次被CPU调用以进入到运行状态。

根据阻塞产生的原因不同，阻塞状态又可以分为三种：

a.等待阻塞：运行状态中的线程执行wait()方法，使本线程进入到等待阻塞状

态；

b.同步阻塞 – 线程在获取synchronized同步锁失败(因为锁被其它线程所占用)，

它会进入同步阻塞状态；

c.其他阻塞 – 通过调用线程的sleep()或join()或发出了I/O请求时，线程会进入到

阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完

毕时，线程重新转入就绪状态。5）死亡状态（Dead）：线程执行完了或者因异常退出了run()方法，该线程结

束生命周期。

## 5、线程安全问题

（1）运行结果错误

我们多线程成一起对变量进行操作 进行i++ 一千个线程同时执行 得到的结

果不一定是1000 会有得到重复数据的 是因为同时读到变量的值 （200）同时加

一 最终的结果都是201

我们可以加一个synchronized锁 目的是为了让线程走到有锁的地方单个顺

序执行

synchronized是加在容易产生线程安全问题的地方

但是枷锁也会碰到逻辑安全问题

死锁

死锁是一种状态，当两个（或多个）线程（或进程）相互持有对方所需要的

资源，却又都不主动释放自己手中所持有的资源，导致大家都获取不到自己想要

的资源，所有相关的线程（或进程）都无法继续往下执行，在未改变这种状态之

前都不能向前推进，我们就把这种状态称为死锁状态，认为它们发生了死锁。

还有优先级不够 一直拿不到cpu资源 从而导致进程不运行

# Java基础

**1、String和StringBuilder和StringBuffer的区别**

1）String是一个字符串常量，长度固定

2）StringBuffer是一个长度可变的量，线程安全，运行效率低高于String

3）StringBuilder是一个长度可变的量，线程不安全，运行效率高于StringBuffer高于String

**2、集合**

1）collection （顶层接口） 开来可深

作用：进行数据储存

collection是所有集合的所有集合接口

·创建对象

Collection集合名 = new 实现类();

Collection<泛型> 集合名 = new 实现类<泛型>();

·注意：泛型只能是类，如果要使用基本数据类型，请使用基本数据类型的包装类

·常用方法:

1.add(); 添加

2. size(); 集合的元素的个数

3. remove() 移除指定元素

4. iterator(); 迭代器

a. hasNext() 判断是否有下一个元素（true为有下一个元素。false为没有 下一个元素）

b. next() 取得迭代器中的下一个元素

2）List集合是Java.util包中的类

·创建对象的方法：

List 集合名 = new 实现类();

List<泛型> 集合名 = new 实现类<泛型>();

·List集合的特点：

List集合有下标，数据是有序的，可以存放重复数据

·List集合的常用方法：

add() 添加元素

remove() 移除元素 瑞姆

size() 获得集合的长度

get(index) 获得指定下标的元素

iterators() 获得迭代器 艾特瑞特

3）Set集合

①HashSet ②TreeSet

特点：无序 无下标 不能放重复数据

·ArrayList 和 LinkedList，Vector的区别：

·ArrayList基于数组的存储结构，查询效率快，删除和添加数据比较慢，线程不安全

· LinkedList基于链式存储结构，查询效率慢，删除和添加效率快， 线程不安全

·Vector线程安全的，已经不建议使用

**3、Map映射(不能被实例化，通过实现类创建Map对象）**

Map接口的实现类 HashMap，HashTable

1.对象的创建

Map<泛型，泛型> 变量名 = new 实现类<泛型，泛型>();

存值得方式：以键值对的形式存在，并且，键对象不允许重复，值对象 允许重复

map.put("键对象"，“值对象”);

常用方法

取值方式: get（） 通过键对象获取值对象 map.get(键对象);

得到所有的值对象：map.values();

得到所有的键对象：map.keySet();

2.HashMap和HashTable的区别

HashMap 线程不安全的，效率高，允许键对象和值对象为null

HashTable线程安全的，效率低，不允许键对象和值对象为空，否则会报异常

Map是无序的，它的存储结构是哈希表<key,value>键值对，map中插入元素是根据key计算出的哈希值来存储元素的，因此他不是按照元素的添加顺序来存储对象的，所以Map是无序的

1. hashMap的底层原理

HashMap存储方式在jdk8之前用的是数组加链表，在之后用的是数组+链表+红黑树。当我们往hashMap里放值时，首先计算key的hashcode，再%（取模）数组的长度，从而获得该值在数组上的位置下标，若该位置上没有放置元素，则直接存放在该位置上；若已有元素，则插入该位置下的链表头部。Jdk1.8之后在此基础上增加了红黑树的存储方式，当链表上的key达到8个的时候将链表变为红黑树的存储方式，小于6个时候会再转换成链表的存储方式。（补充：我后来在网上查阅资料的时候看到个问题，就是说这个头插法有可能会在扩容时候出现问题，扩容的时候会重新放置数据，并且它会将链表中的数据倒序插入到一个新的链表，这时候当有一个新的元素存储过来的时候，会造成新元素和从链表尾部过来的值相互指向，造成死锁，jdk8之后采用了尾插法就避免了这个问题）

**4、get和post区别**

1）get是从服务器上获取数据，post是向服务器传送数据。

2）get传送的数据量较小，不能大于2KB。

post传送的数据量较大，一般被默认为不受限制。

3）get安全性非常低，post安全性较高。但是执行效率却比post方法好。

4）在进行文件上传时只能使用post而不是get。

5、**反射**

1. 通过类对象获取类的所有属性和方法；
2. 通过对象可以获取到属性的值和调用它的方法；

反射中常用的几个对象：

Class 类（可以获取所有的属性（getDeclaredFields()、getFields()、获取所有方法 getDeclaredMethods()））

Field 属性（获取属性名getName()、获取属性的数据类型getType() ））

Method 方法（get、获取返回值类型 getReturnType() 、执 行方法invoke() ）

Constructors 构造方法 （获取构造方法 getConstructors()）

Annotation 注解 （获取注解 getAnnotations()）

6、自定义注解

自定义注解的关键字@Interface

自定义注解的四个元注解

* + 1. Target 声明注解的使用范围
    2. Retention 声明注解的生命周期

7、Lambda表达式

Lambda表达式是jdk1.8的新特性，是一种简单快捷实现接口的新写法，但是不是所有的接口都适合lambda表达式，需要以函数接口为前提（函数接口也就是有且只有一个抽象方法，都可以有多个普通方法），使用如下：

//接口

public interface EnumDemo {  
 public void query(String a);  
}

//实现方法及调用

String abb = "888";  
EnumDemo enumDemo = (a)->{  
 System.*out*.println(a);  
};  
enumDemo.query(abb);

8、Stream流

Stream就是用来操作集合的；

创建Stream流的方法

集合名.stream();

Stream的常用方法

forEach()用来遍历集合

filter()条件过滤

Limit() 取定量数据（取几条）

Skip()跳过几条

Sorted()对集合进行排序

9、多线程

* 1. 什么是进程？什么是线程？

进程：进程指正在运行的程序。确切的来说，当一个程序进入内存运行，即变成一个进程，进程是处于运行过程中的程序，并且具有一定独立功能，进程主要用于获取CPU资源。

线程：线程是程序中最小的执行单元，一个进程至少有一个线程；

多线程：一个进程中有多个线程同时执行任务；

* 1. 线程的状态
     1. 创建（new之后）
     2. 就绪（执行start之后）
     3. 运行（执行run之后）
     4. 阻塞（执行sleep或wait之后，sleep不释放锁，wait释放锁）
     5. 死亡（run方法执行结束后）
  2. 创建线程的方法
     1. 实现Runable类
     2. 继承Thread类
     3. 实现callable类以及Future；(此线程可以有返回值)
  3. 线程安全问题
     1. 当多个线程同时操作同一个资源时会造成线程不安全问题；此时我们通过synchronized给代码块加锁
     2. 还有就是死锁，A线程拿着B线程需要资源不释放，B线程拿着A线程需要的资源也不释放，造成死锁，导致程序无法往下进行；尽量避免死锁的情况，或者线程礼让（线程礼让也不完全能解决死锁的问题）

10、线程池

* 1. 什么是线程池？
     1. 线程池就是事先创建若干个可执行的线程放入一个容器中，当有任务需要执行时，直接从线程池拿空闲线程使用，执行完任务后将线程归还给线程池，这样就实现了线程的复用，从而减少了创建和销毁线程对象的开销。
  2. 线程池的优点？
     1. 降低系统消耗；
     2. 提升响应速度；
     3. 提高线程的管理性；
  3. 线程池运行原理
     1. 提交多线程任务，判断线程池核心线程数是否已经满，若没有满足则创建线程执任务，
     2. 若核心线程都在占用，判断等待队列是否已满，未满则进入队列；
     3. 队列已满则判断线程数是否达到最大线程，若为达到则直接创建线程执行任务；
     4. 未达到最大线程数则进入拒绝策略；
  4. 线程池的核心类ThreadPoolExecutor的主要参数
     1. c**orePoolSize：线程池中核心线程数**
     2. **maximumPoolSize：线程池最大线程数**
     3. **keepAliveTime：非核心线程闲置超时时间**
     4. Unit**：keepAliveTime的时间单位**
     5. **workQueue：线程池中的任务队列**
     6. **threadFactory：线程池创建线程时使用的线程工厂**
     7. **handler：线程池的饱和策略**
  5. 常用的线程池：

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 |

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 |

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 |

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 |

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 |

# springcloud分布式框架

### spring-cloud-eureka：

spring-cloud-eureka是spring-cloud-nettfix微服务套件中的一部分， 它基于nettfix-

eureka做了二次封装，主要负责微服务架构中的服务治理功能。

服务治理一般都会有两个功能：服务注册、服务发现。通常会有一个注册中心，每个服务单

元向注册中心登记自己信息，比如提供的服务，ip, 端口以及一些附近加信息等。注册中心

会将新的服务实例发送给其它依赖此服务的实例。

核心注解@EnableEurekaClient

### springcloud-Feign:

Feign是一个声明式WebService客户端。使用Feign能让编写Web Service客户端更加简单, 它的使用方法是定义一个接口，然后在上面添加注解，同时也支持JAX-RS标准的注解。Feign也支持可拔插式的编码器和解码器。Spring Cloud对Feign进行了封装，使其支持了Spring MVC标准注解和HttpMessageConverters。Feign可以与Eureka和Ribbon组合使用以支持负载均衡。

Feign也是运行在消费者端的，使用 Ribbon 进行负载均衡，所以 Feign 直接内置了 Ribbon。而与Ribbon不同的是，通过feign只需要定义服务绑定接口且以声明式的方法，优雅而简单的实现了服务调用。

核心注解：@EnableFeignClients//声明fegin

### Spring Cloud Ribbon:

Spring Cloud Ribbon是基于Netflix Ribbon实现的一套客户端负载均衡的工具。

简单的说，Ribbon是Netflix发布的开源项目，主要功能是提供客户端的软件负载均衡算法，将Netflix的中间层服务连接在一起。Ribbon客户端组件提供一系列完善的配置项如连接超时，重试等。简单的说，就是在配置文件中列出Load Balancer（简称LB）后面所有的机器，Ribbon会自动的帮助你基于某种规则（如简单轮询，随机连接等）去连接这些机器。我们也很容易使用Ribbon实现自定义的负载均衡算法。核心：

启动类中注入bean

@Bean

@LoadBalanced //声明 ribbon （并对resttemmplate进行负载均衡 调用的时候直接写服务名就行 不用写ip和端口）

RestTemplate restTemplate() {

return new RestTemplate();// RestTemplate对httpclient发送请求 进行封装了

}

### Spring Cloud Hystrix：

Hystrix中文是豪猪的意思，在Spring Cloud中防雪崩的利器就是Spring Cloud Hystrix，Spring Cloud Hystrix是基于Netflix Hystrix实现的。在分布式系统里，许多依赖不可避免的会调用失败，比如超时、异常等，Hystrix能够保证在一个依赖出问题的情况下，不会导致整体服务失败，避免级联故

Hystrix能干啥

l 服务熔断

当Hystrix Command请求后端服务失败数量超过一定比例(默认50%), 断路器会切换到开路状态(Open). 这时所有请求会直接失败而不会发送到后端服务. 断路器保持在开路状态一段时间后(默认5秒), 自动切换到半开路状态(HALF-OPEN).

l 服务降级

Fallback相当于是降级操作, 我们可以实现一个fallback方法, 当请求后端服务出现异常或者服务熔断后的时候, 直接调用本地fallback方法，告知后面的请求服务不可用了，不要再来了。

l 服务容错

l 线程和信号隔离

在Hystrix中, 主要通过线程池来实现资源隔离. 通常在使用的时候我们会根据调用的远程服务划分出多个线程池.比如说，一个服务调用两外两个服务，你如果调用两个服务都用一个线程池，那么如果一个服务卡在哪里，资源没被释放。后面的请求又来了，导致后面的请求都卡在哪里等待，导致你依赖的A服务把你卡在哪里，耗尽了资源，也导致了你另外一个B服务也不可用了。这时如果依赖隔离，某一个服务调用A B两个服务，如果这时我有100个线程可用，我给A服务分配50个，给B服务分配50个，这样就算A服务挂了，我的B服务依然可以用。

l 请求缓存

比如一个请求过来请求我userId=1的数据，你后面的请求也过来请求同样的数据，这时我不会继续走原来的那条请求链路了，而是把第一次请求缓存过了，把第一次的请求结果返回给后面的请求。

l 请求合并

我依赖于某一个服务，我要调用N次，我发了N条请求发然后拿到一堆结果，这时候我们可以把多个请求合并成一个请求，这样我们只需调用一次远程服务，提升了效率。

l 近乎实时的服务监控

### Hystrix 客户端核心：

@EnableHystrix// 熔断 启动类中

对你要熔断的方法

// 此注解标识是要熔断 fallbackMethod 熔断后进行执行的方法

@HystrixCommand(fallbackMethod = "errorMethod",commandProperties = {

@HystrixProperty(name = "execution.isolation.thread.timeoutInMilliseconds",value = "3000")

}) //对此方法进行熔断

Hystrix 服务端：

启动类

@EnableHystrixDashboard

启动类中

/\* 提供监控平台的路径\*/

@Bean

public ServletRegistrationBean getServlet() {

HystrixMetricsStreamServlet streamServlet = new HystrixMetricsStreamServlet();

ServletRegistrationBean registrationBean = new ServletRegistrationBean(streamServlet);

registrationBean.setLoadOnStartup(1);

registrationBean.addUrlMappings("/hystrix.stream");

registrationBean.setName("HystrixMetricsStreamServlet");

return registrationBean;

### Spring Cloud zull：

Zuul是Netflix开源的微服务网关，他可以和Eureka,Ribbon,Hystrix等组件配合使用。Zuul组件的核心是一系列的过滤器，这些过滤器可以完成以下功能：

身份认证和安全:

识别每一个资源的验证要求，并拒绝那些不符的请求审查与监控：

***动态路由：***

动态将请求路由到不同后端集群

***压力测试：***

逐渐增加指向集群的流量，以了解性能

***负载分配：***

为每一种负载类型分配对应容量，并弃用超出限定值的请求

***静态响应处理：***

边缘位置进行响应，避免转发到内部集群