# 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

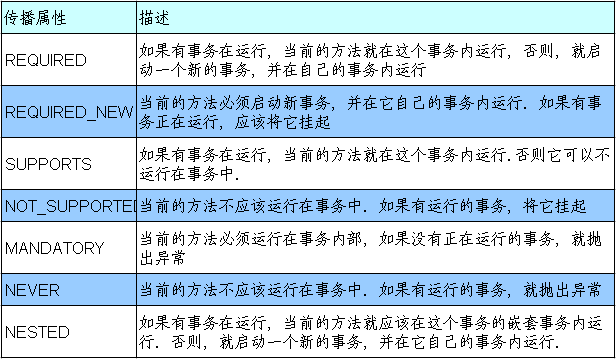
原子性：事务要么都执行，要么都不执行；

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，用户权限关系表>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，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。

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。

# 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是非关系型数据库 也是一个key value的数据库

redis效率非常快 读取11000次每秒 写入是8100次每秒

为什么效率这么快 因为redis是基于内存的 所以效率快

redis有两种持久化方案 rdb和aof

reb相当于快照形式 效率快 但是丢失数据会丢失一整片

aof是日志形式的 效率慢 但是最多丢失最后一条数据

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

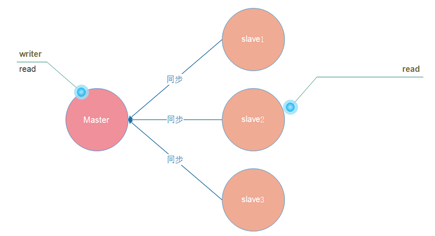
## 4、redis主从复制

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。

**1           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。

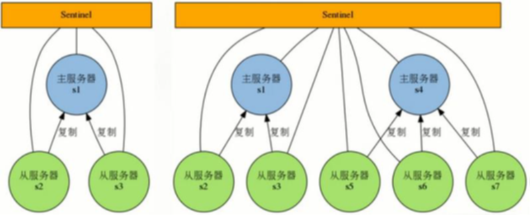
**自动故障迁移**（Automatic failover）： 当一个主服务器不能正常工作时， Sentinel 会开始一次自动故障迁移操作， 它会进行选举，将其中一个从服务器升级为新的主服务器， 并让失效主服务器的其他从服务器改为复制新的主服务器； 当客户端试图连接失效的主服务器时， 集群也会向客户端返回新主服务器的地址， 使得集群可以使用新主服务器代替失效服务器。

**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。**

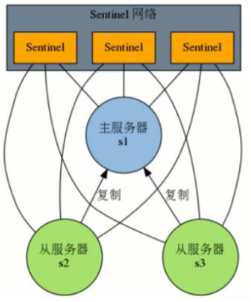
（3）Sentinel会不断检查Master和Slaves是否正常。



**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。

## 5、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