# 自我介绍

面试官你好，我叫许泽康，来自重庆邮电大学，是一名研二的在读研究生。研究生在校期间，我获得过两次学业奖学金。在22年8月到11月在北京京东集团实习过。在技术方面，我具有一定编程经验，能够熟练使用java编程语言，对于java语言的一些基础理论知识比较了解，了解基本数据结构和算法。有基本的深度学习背景，有过开发和测试相关的项目经历。性格方面，我比较乐观开朗，具有较强的抗压能力，以上是我的自我介绍，谢谢。

Hello interviewer, my name is Xu Zekang, from Chongqing University of Posts and Telecommunications, is a second-year postgraduate student. During my postgraduate study, I won two academic scholarships. Worked as an intern in Beijing Jingdong Group from August to November, 2022. In terms of technology, I have some programming experience and can skillfully use java programming language. I have a good understanding of some basic theoretical knowledge of java language as well as basic data structure and algorithm. Basic deep learning background with experience in development and test related projects. In terms of personality, I am optimistic and cheerful, with strong ability to work under pressure. The above is my self-introduction. Thank you.

# 一、Java基础篇

## 1.接口和抽象类的区别

相似点：

（1）接口和抽象类都不能被实例化

（2）实现接口或继承抽象类的普通子类都必须实现这些抽象方法

不同点：

（1）抽象类可以包含普通方法和代码块，接口里只能包含抽象方法，静态方法和默认方法，

（2）抽象类可以有构造方法，而接口没有

（3）抽象类中的成员变量可以是各种类型的，接口的成员变量只能是 public static final 类型的，并且必须赋值

## 2.重载和重写的区别

重载发生在同一个类中，方法名相同、参数列表、返回类型、权限修饰符可以不同

重写发生在子类中，方法名相、参数列表、返回类型都相同，权限修饰符要大于父类方法，声明异常范围要小于父类方法，但是final和private修饰的方法不可重写

## 3.==和equals的区别

==比较基本类型，比较的是值，==比较引用类型，比较的是内存地址

equlas是Object类的方法，本质上与==一样，但是有些类重写了equals方法，比如String的equals被重写后，比较的是字符值，另外重写了equlas后，也必须重写hashcode()方法

## 4.异常处理机制

（1）使用try、catch、finaly捕获异常，finaly中的代码一定会执行，捕获异常后程序会继续执行

（2）使用throws声明该方法可能会抛出的异常类型，出现异常后，程序终止

## 5.HashMap原理

1.HashMap在Jdk1.8以后是基于数组+链表+红黑树来实现的，特点是，key不能重复，可以为null，线程不安全

2.HashMap的扩容机制：

HashMap的默认容量为16，默认的负载因子为0.75，当HashMap中元素个数超过容量乘以负载因子的个数时，就创建一个大小为前一次两倍的新数组，再将原来数组中的数据复制到新数组中。当数组长度到达64且链表长度大于8时，链表转为红黑树

3.HashMap存取原理：

（1）计算key的hash值，然后进行二次hash，根据二次hash结果找到对应的索引位置

（2）如果这个位置有值，先进行equals比较，若结果为true则取代该元素，若结果为false，就使用高低位平移法将节点插入链表（JDK8以前使用头插法，但是头插法在并发扩容时可能会造成环形链表或数据丢失，而高低位平移发会发生数据覆盖的情况）

## 6.想要线程安全的HashMap怎么办？

（1）使用ConcurrentHashMap

（2）使用HashTable

（3）Collections.synchronizedHashMap()方法

## 7.ConcurrentHashMap原如何保证的线程安全？

JDK1.7:使用分段锁，将一个Map分为了16个段，每个段都是一个小的hashmap，每次操作只对其中一个段加锁

JDK1.8:采用CAS+Synchronized保证线程安全，每次插入数据时判断在当前数组下标是否是第一次插入，是就通过CAS方式插入，然后判断f.hash是否=-1，是的话就说明其他线程正在进行扩容，当前线程也会参与扩容；删除方法用了synchronized修饰，保证并发下移除元素安全

## 8.HashTable与HashMap的区别

（1）HashTable的每个方法都用synchronized修饰，因此是线程安全的，但同时读写效率很低

（2）HashTable的Key不允许为null

（3）HashTable只对key进行一次hash，HashMap进行了两次Hash

（4）HashTable底层使用的数组加链表

## 9.ArrayList和LinkedList的区别

ArratList的底层使用动态数组，默认容量为10，当元素数量到达容量时，生成一个新的数组，大小为前一次的1.5倍，然后将原来的数组copy过来；因为数组在内存中是连续的地址，所以ArrayList查找数据更快，由于扩容机制添加数据效率更低

LinkedList的底层使用链表，在内存中是离散的，没有扩容机制；LinkedList在查找数据时需要从头遍历，所以查找慢，但是添加数据效率更高

## 10.如何保证ArrayList的线程安全？

（1）使用collentions.synchronizedList（）方法为ArrayList加锁

（2）使用Vector，Vector底层与Arraylist相同，但是每个方法都由synchronized修饰，速度很慢

（3）使用juc下的CopyOnWriterArrayList，该类实现了读操作不加锁，写操作时为list创建一个副本，期间其它线程读取的都是原本list，写操作都在副本中进行，写入完成后，再将指针指向副本。

## 11.String、StringBuffer、StringBuilder的区别

String 由 char[] 数组构成，使用了 final 修饰，对 String 进行改变时每次都会新生成一个 String 对象，然后把指针指向新的引用对象。

StringBuffer可变并且线程安全

StringBuiler可变但线程不安全。

操作少量字符数据用 String；单线程操作大量数据用 StringBuilder；多线程操作大量数据用 StringBuffer。

## 12.hashCode和equals

两个对象相等，他们的hashCode和equals一定相等，但是hashCode相等的两个对象未必相等

如果重写equals()必须重写hashCode()，比如在HashMap中，key如果是String类型，String如果只重写了equals（）而没有重写hashcode（）的话，则两个equals()比较为true的key，因为hashcode不同导致两个key没有出现在一个索引上，就会出现map中存在两个相同的key

## 13.面向对象和面向过程的区别

面向对象有封装、继承、多态性的特性，所以相比面向过程易维护、易复用、易扩展，但是因为类调用时要实例化，所以开销大性能比面向过程低

## 14.深拷贝和浅拷贝

浅拷贝:浅拷贝只复制某个对象的引用，而不复制对象本身，新旧对象还是共享同一块内存

深拷贝:深拷贝会创造一个一摸一样的对象，新对象和原对象不共享内存，修改新对象不会改变原对对象。

## 15.多态的作用

多态就是方法或对象具有多种形态，多态的前提是两个对象（类）存在继承关系，多态是建立在封装和继承基础之上的。它的好处是可以消除类型之间的耦合关系，增加类的可扩充性和灵活性。

**多态的转型：**

[**向上转型**](https://so.csdn.net/so/search?q=%E5%90%91%E4%B8%8A%E8%BD%AC%E5%9E%8B&spm=1001.2101.3001.7020)：父类的引用指向子类的对象，编译类型看左边，运行类型看右边，可以调用父类的所有成员（需遵守访问权限），**不能调用子类的特有成员**，运行效果看子类的具体实现



**向下转型：**一个已经向上转型的子类对象，将父类引用转为子类引用，只能强制转换父类的引用，不能强制转换父类的对象，**要求父类的引用必须指向的是当前目标类型的对象**，当向下转型后，**可以调用**子类类型中所有的成员

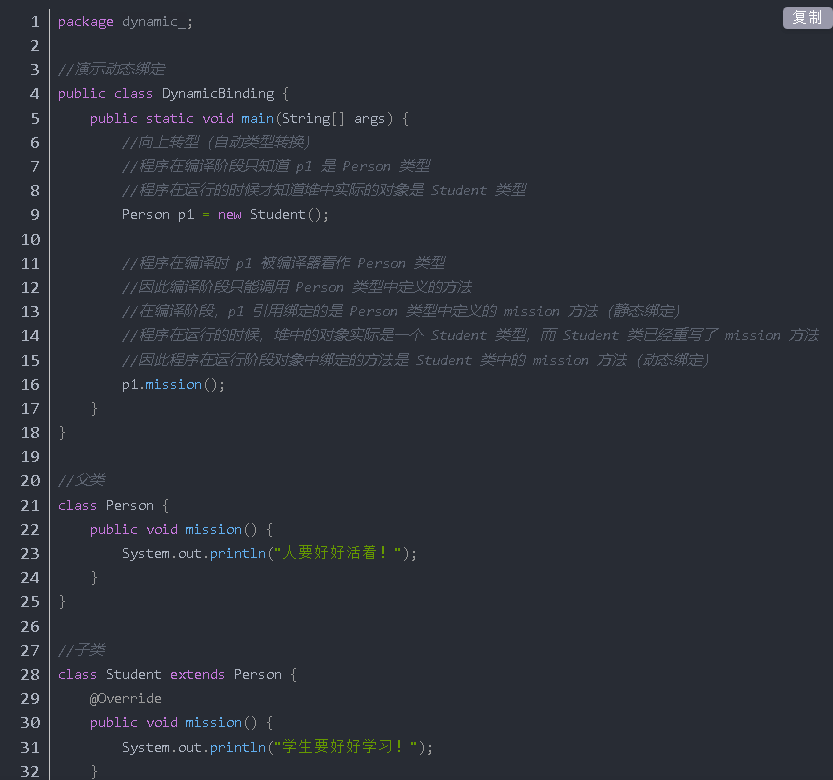


为了避免类型转换异常的问题，引出instanceof比较操作符，用于判断对象的类型是否为XX类型或XX类型的子类型。

**动态绑定：**

当调用对象方法的时候，该方法会和该对象的运行类型绑定

当调用对象属性时，没有动态绑定机制，即哪里声明，哪里使用。



## 16.什么是反射？

反射是通过获取类的class对象，然后动态的获取到这个类的内部结构，动态的去操作类的属性和方法。

应用场景有：要操作权限不够的类属性和方法时、实现自定义注解时、动态加载第三方jar包时、按需加载类，节省编译和初始化时间；

获取class对象的方法有：class.forName(类路径)，类.class()，对象的getClass（）

## 17.Java创建对象的五种方式?

(1)new关键字 (2)Class.newInstance (3)Constructor.newInstance

(4)Clone方法 (5)反序列化(1、**通过序列化，将对象的状态写在流里面**：先构造一个对象流ObjectOutputStream oss，使用oos的writeObject的方法将实例对象car1的状态写到流（字节流文件car1.ser）里面。

2、**根据需要把该流读取出来重新构造一个相同的对象**: 先构造一个对象流ObjectInputStream ois读取字节流文件car1.ser，使用ois的readObject的方法重新构造一个相同的对象car2。)

# 二.Java多线程篇

## 1.进程和线程的区别，进程间如何通信

进程：系统运行的基本单位，进程在运行过程中都是相互独立，但是线程之间运行可以相互影响。

线程：独立运行的最小单位，一个进程包含多个线程且它们共享同一进程内的系统资源

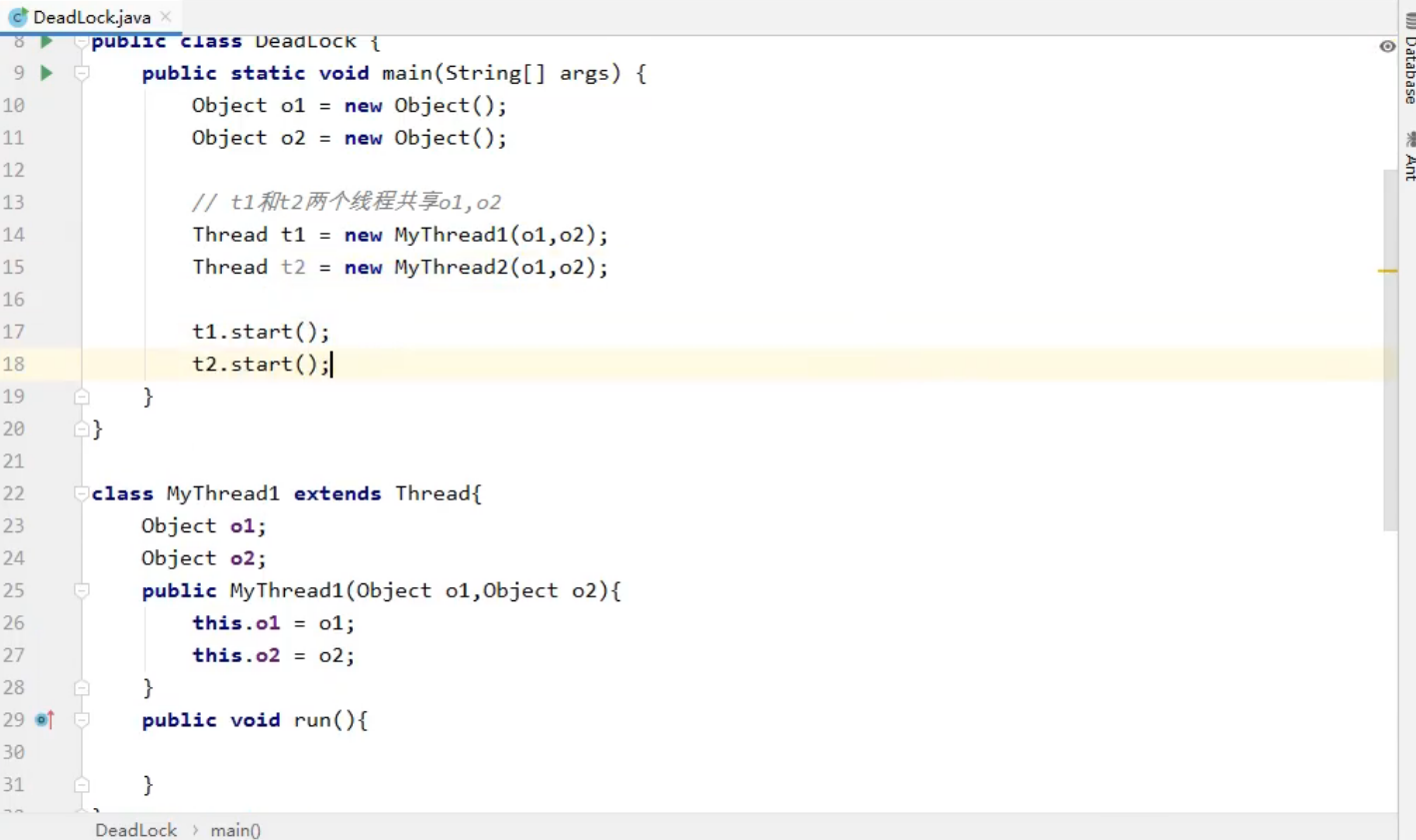
进程间通过管道、 共享内存、信号量机制、消息队列通信

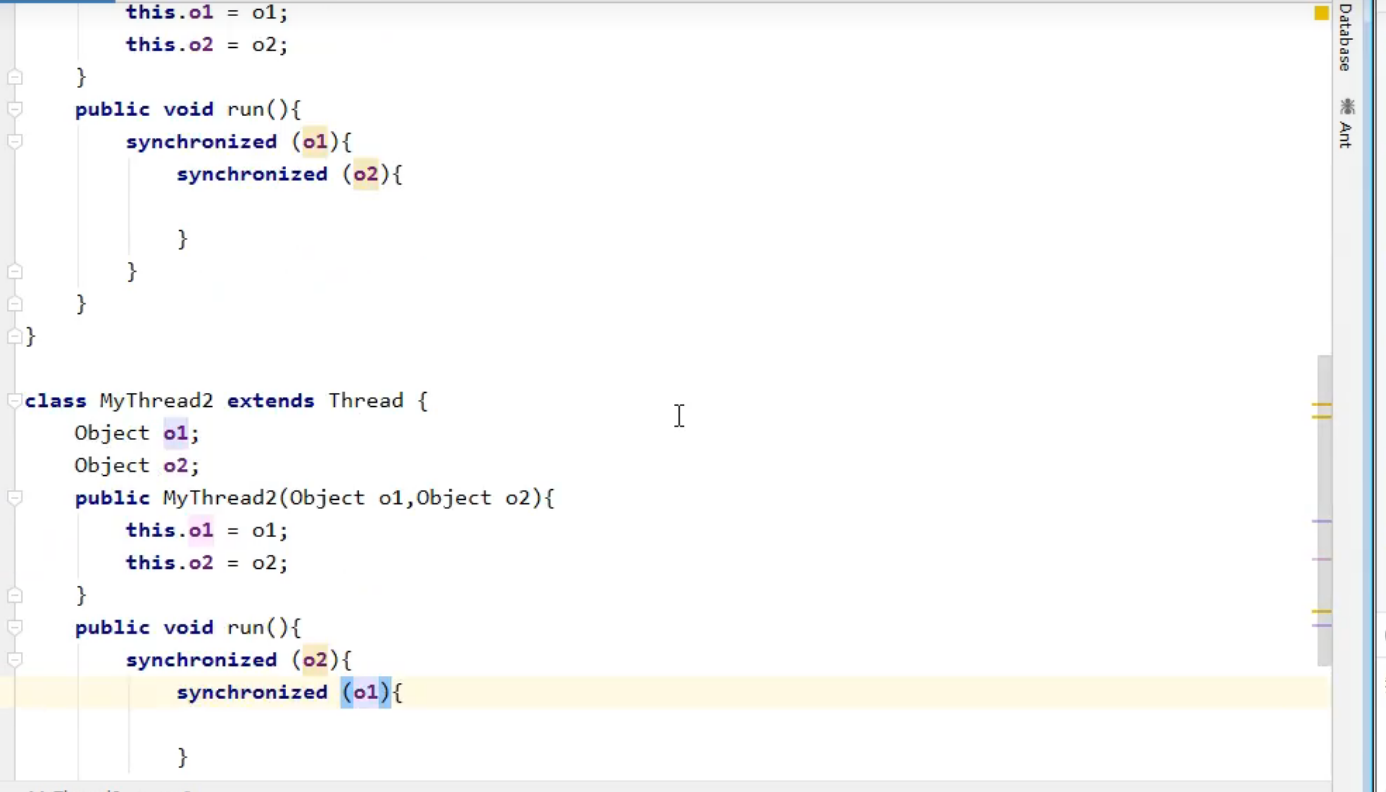
## 2.什么是线程上下文切换

当一个线程被剥夺cpu使用权时，切换到另外一个线程执行

## 3.什么是死锁

死锁指多个线程在执行过程中，因争夺资源造成的一种相互等待的僵局





## 4.死锁的必要条件

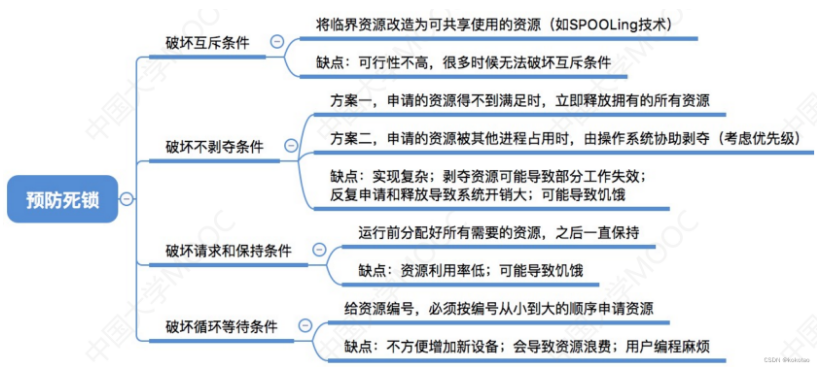
互斥条件：同一资源同时只能由一个线程读取

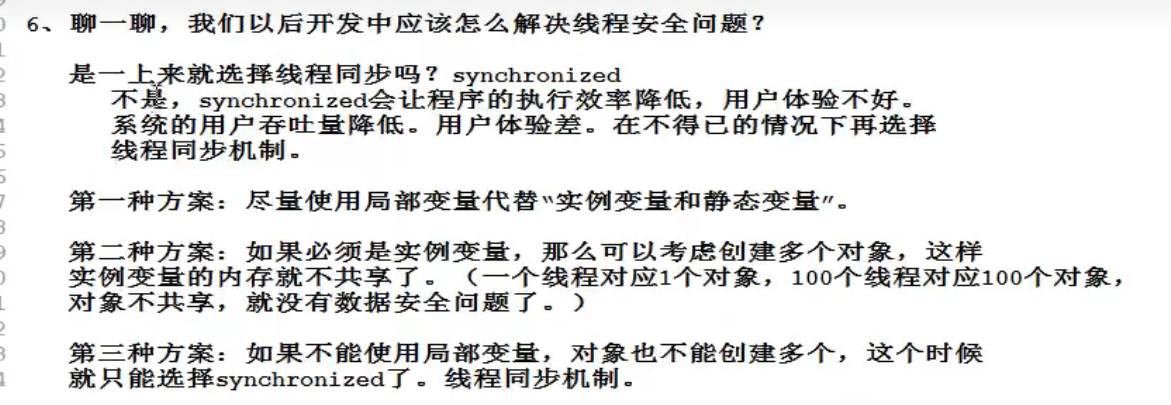
不可抢占条件：不能强行剥夺线程占有的资源

请求和保持条件：请求其他资源的同时对自己手中的资源保持不放

循环等待条件：在相互等待资源的过程中，形成一个闭环

想要预防死锁，只需要破坏其中一个条件即可，比如使用定时锁、尽量让线程用相同的加锁顺序，还可以用银行家算法（在进程提出资源申请时，先预判此次分配是否系统进入不安全状态。如果会进入不安全状态，就暂时不答应这次请求，让该进程先阻塞等待. 通俗来说，就是“我”是否有足够的钱来买一件衣服，并且还有判断剩余的钱是否让“我”的生活处于影响状态。如果有足够钱，并且不会对“我”的生活产生影响，“我”购买该衣服。）可以预防死锁





## 5.Synchrpnized

**Synchronized锁原理和优化**

**synchronized的三种应用方式：**

1.作用于实例方法，当前实例加锁，进入同步代码前要获得当前实例的锁；当一个线程正在访问一个对象synchronized实例方法时，别的线程是访问不了的。一个对象一把锁说的就是这个

2.作用于静态方法，当前类加锁，进去同步代码前要获得当前类对象的锁；静态方法是不属于当前实例的，而是属性类的，那么这个锁就是类锁

3.作用于代码块，这需要指定加锁的对象，对所给的指定对象加锁，进入同步代码前要获得指定对象的锁。

在JVM中，对象在内存中分为三块区域：对象头，实例数据和对齐填充

**对象头**：

Mark Word，用于存储对象自身运行时的数据，如哈希码(Hash Code)，GC分代年龄，锁状态标志，偏向线程ID、偏向时间戳等信息，它会根据对象的状态复用自己的存储空间。它是实现轻量级锁和偏向锁的关键。

类型指针，对象会指向它的类的元数据的指针，虚拟机通过这个指针确定这个对象是哪个类的实例。

Array length，如果对象是一个数组，还必须记录数组长度的数据 - 。

**实例数据**：

存放类的属性数据信息，包括父类的属性信息。

**对齐填充**：

由于虚拟机要求 对象起始地址必须是8字节的整数倍。填充数据不是必须存在的，仅仅是为了字节对齐。

Synchronized通过编译，会在同步块的前后分别形成monitorenter和monitorexit这个两个字节码指令。

**Synchronized底层原理**

1）在执行monitorenter指令时，首先要尝试获取对象锁。如果这个对象没被锁定，或者当前线程已经拥有了那个对象锁，把锁的计算器加1

2）相应的，在执行monitorexit指令时会将锁计算器减1，当计算器为0时，锁就被释放了

3）如果获取对象锁失败，那当前线程就要阻塞，直到对象锁被另一个线程释放为止。

**synchronized在JVM里的实现就是基于进入和退出monitor来实现的，底层则是通过成对的MonitorEnter和MonitorExit指令来实现，monitor内置在每一个对象中，任何一个对象都有一个monitor与之关联可以理解monitor是一个同步工具**

**Synchrpnized和lock的区别**

（1）synchronized是关键字，lock是一个类

（2）synchronized在发生异常时会自动释放锁，lock需要手动释放锁

（3）synchronized是可重入锁(可以多次获取相同的锁)、非公平锁(先到不一定先得)、不可中断锁(抢占过程不可以被中断的锁)，lock的ReentrantLock是可重入锁，可中断锁，可以是公平锁也可以是非公平锁

（4）synchronized是JVM层次通过监视器实现的，Lock是通过AQS实现的

**ReentrantLock** 和 synchronized 不一样，需要手动释放锁，所以使用 ReentrantLock的时候一定要**手动释放锁**，并且**加锁次数和释放次数要一样**

## 6.什么是AQS锁?

AQS就是基于CLH队列，用volatile修饰共享变量state，线程通过CAS去改变状态符，成功则获取锁成功，失败则进入等待队列，等待被唤醒。

核心思想是，如果被请求的共享资源空闲，则将当前请求资源的线程设置为有效的工作线程，并将共享资源设置为锁定状态，如果被请求的共享资源被占用，那么就需要一套线程阻塞等待以及被唤醒时锁分配的机制，这个机制AQS是用CLH队列锁实现的，即将暂时获取不到锁的线程加入到队列中。

**CLH队列是一个虚拟的双向队列，即不存在队列实例，仅存在节点之间的关联关系。**

AQS的原理是，AQS内部有三个核心组件，一个是state代表加锁状态初始值为0，一个是获取到锁的线程，还有一个阻塞队列。当有线程想获取锁时，会以CAS的形式将state变为1，CAS成功后便将加锁线程设为自己。当其他线程来竞争锁时会判断state是不是0，不是0再判断加锁线程是不是自己，不是的话就把自己放入阻塞队列。这个阻塞队列是用双向链表实现的

可重入锁的原理就是每次加锁时判断一下加锁线程是不是自己，是的话state+1，释放锁的时候就将state-1。当state减到0的时候就去唤醒阻塞队列的第一个线程。

## 7.为什么AQS使用的双向链表？

因为有一些线程可能发生中断 ，而发生中断时候就需要在同步阻塞队列中删除掉，这个时候用双向链表方便删除掉中间的节点

## 8.有哪些常见的AQS锁

AQS分为独占锁和共享锁

ReentrantLock（独占锁）：可重入，可中断，可以是公平锁也可以是非公平锁，非公平锁就是会通过两次CAS去抢占锁，公平锁会按队列顺序排队

Semaphore（信号量）:设定一个信号量，当调用acquire()时判断是否还有信号，有就获取一个信号量，没有就阻塞等待其他线程释放信号量，当调用release()时释放一个信号量，唤醒阻塞线程。

应用场景：允许多个线程访问某个临界资源时，如上下车，买卖票

CountDownLatch（倒计数器）:给计数器设置一个初始值，当调用CountDown()时计数器减一，当调用await() 时判断计数器是否归0，不为0就阻塞，直到计数器为0。

应用场景：启动一个服务时，主线程需要等待多个组件加载完毕，之后再继续执行

CyclicBarrier（循环栅栏）:给计数器设置一个目标值,当调用await() 时会计数+1并判断计数器是否达到目标值，未达到就阻塞，直到计数器达到目标值

应用场景：多线程计算数据，最后合并计算结果的应用场景

## 9.sleep()和wait()的区别

(1)wait()是Object的方法，sleep()是Thread类的方法

(2)wait()会释放锁，sleep()不会释放锁

(3)wait()要在同步方法或者同步代码块中执行，sleep()没有限制

(4)wait()要调用notify()或notifyall()唤醒,sleep()自动唤醒

## 10.yield()和join()区别

yield()调用后线程进入就绪状态，以允许具有相同优先级的其他线程获得运行机会

A线程中调用B线程的join() ,则A进入阻塞状态，直到线程B执行完毕后，才会继续执行线程A。

## 11.线程池七大参数

核心线程数：线程池中的基本线程数量

最大线程数：当阻塞队列满了之后，逐一启动

最大线程的存活时间：当阻塞队列的任务执行完后，最大线长的回收时间

最大线程的存活时间单位

阻塞队列：当核心线程满后，后面来的任务都进入阻塞队列

线程工厂：用于生产线程

任务拒绝策略：阻塞队列满后，拒绝任务，有四种策略（1）抛异常（2）丢弃任务不抛异常（3）打回任务（4）尝试与最老的线程竞争

## 12.Java内存模型

JMM（Java内存模型 ）屏蔽了各种硬件和操作系统的内存访问差异，实现让Java程序在各平台下都能达到一致的内存访问效果，它定义了JVM如何将程序中的变量在主存中读取

具体定义为：所有变量都存在主存中，主存是线程共享区域；每个线程都有自己独有的工作内存，线程想要操作变量必须从主从中copy变量到自己的工作区，每个线程的工作内存是相互隔离的

由于主存与工作内存之间有读写延迟，且读写不是原子性操作，所以会有线程安全问题

## 13.保证并发安全的三大特性？

原子性：一次或多次操作在执行期间不被其他线程影响

可见性：当一个线程在工作内存修改了变量，其他线程能立刻知道

有序性：JVM对指令的优化会让指令执行顺序改变，有序性是禁止指令重排

## 14.volatile

**保证共享变量的可见性**：保证了不同线程对这个变量进行操作时的可见性，即一个线程修改了某个变量的值，这新值对其他线程来说是立即可见的。

**防止局部指令重排序**：一个线程先去写一个volatile变量，然后一个线程去读这个变量，那么这个写操作的结果一定对读的这个线程可见。

volatile的底层通过插入内存屏障，强制更新缓存，将对缓存的修改操作立即写入主存

## 15.线程使用方式

(1)继承 Tread 类

(2)实现 Runnable 接口

(3)实现 Callable 接口：带有返回值

(4)线程池创建线程

## 16.ThreadLocal原理

ThreadLocal中填充的变量属于当前线程，原理是为每个线程创建变量副本，不同线程之间不可见，保证线程安全。每个线程内部都维护了一个Map，key为threadLocal实例，value为要保存的副本。

但是使用ThreadLocal会存在内存泄露问题，因为key为弱引用，而value为强引用，每次gc时key都会回收，而value不会被回收。所以为了解决内存泄漏问题，可以在每次使用完后删除value或者使用static修饰ThreadLocal，可以随时获取value

## 17.什么是CAS锁

CAS锁可以保证原子性，思想是更新内存时会判断内存值是否被别人修改过，如果没有就直接更新。如果被修改，就重新获取值，直到更新完成为止。这样的缺点是

（1）只能支持一个变量的原子操作，不能保证整个代码块的原子操作

（2）CAS频繁失败导致CPU开销大

（3）ABS问题:线程1和线程2同时去修改一个变量，将值从A改为B，但线程1突然阻塞，此时线程2将A改为B,然后线程3又将B改成A,此时线程1将A又改为B,这个过程线程2是不知道的，这就是ABA问题，可以通过版本号或时间戳解决

## 18. AtomicInteger的使用场景

AtomicInteger是一个提供原子操作的Integer类，使用CAS+volatile实来现线程安全的数值操作。

因为volatile禁止了jvm的排序优化,所以它不适合在并发量小的时候使用，只适合在一些高并发程序中使用

# 三.JVM篇

## 1.JVM运行时数据区（内存结构）

线程私有区：

（1）虚拟机栈：每次调用方法都会在虚拟机栈中产生一个栈帧，每个栈帧中都有方法的参数、局部变量、方法出口等信息，方法执行完毕后释放栈帧

（2）本地方法栈：为native修饰的本地方法提供的空间，在HotSpot中与虚拟机合二为一

（3）程序计数器：保存指令执行的地址，方便线程切回后能继续执行代码

线程共享区：

（4）堆内存：Jvm进行垃圾回收的主要区域，存放对象信息，分为新生代和老年代，内存比例为1：2，新生代的Eden区内存不够时时发生MinorGC,老年代内存不够时发生FullGC

（5）方法区：存放类信息、静态变量、常量、运行时常量池等信息。JDK1.8之前用持久代实现，JDK1.8后用元空间实现，元空间使用的是本地内存，而非在JVM内存结构中

## 2.内存泄漏、内存溢出

**内存泄漏**：指不再被使用的对象或者变量却一直占据在内存中，对象不会再被程序用到但是GC又不能回收他们的情况，就叫内存泄漏。

长生命周期的对象持有短生命周期对象的引用就很可能发生内存泄露，尽管短生命周期对象已经不再需要，但是因为长生命周期对象持有它的引用而导致不能被回收，这就是java中内存泄露的发生场景。

**内存溢出**：指程序申请内存时，没有足够的内存供申请者使用

堆内存溢出：（1）当对象一直创建而不被回收时（2）加载的类越来越多时（3)虚拟机栈的线程越来越多时

栈溢出：方法调用次数过多，一般是递归不当造成

## 3.JVM有哪些垃圾回收算法？

（1）标记清除算法： 标记不需要回收的对象，然后清除没有标记的对象，会造成许多内存碎片。**缺点：效率不高，无法清除垃圾碎片**。

（2）复制算法： 将内存分为两块，只使用一块，进行垃圾回收时，先将存活的对象复制到另一块区域，然后清空之前的区域。用在新生代。**缺点：效率不高，无法清除垃圾碎片。**

（3）标记整理算法： 与标记清除算法类似，但是在标记之后，将存活对象向一端移动，然后清除边界外的垃圾对象。用在老年代

（4）分代算法：根据对象存活周期的不同将内存划分为几块，一般是新生代和老年代，新生代基本采用复制算法，老年代采用标记整理算法。

## 4.GC如何判断对象可以被回收？

1. 引用计数法

给对象添加一个计数器，当对象增加一个引用时计数器加 1，引用失效时减 1。计数为 0 时，对象可被回收。

当两个对象出现循环引用时，计数器永远不为 0，导致无法对它们进行回收。

正因为循环引用的存在，因此 Java 虚拟机不使用引用计数算法。

.2. **可达性分析**

以GC Roots的根对象作为起点，根据对象的引用向下搜索，能搜索到的对象即为不可回收对象，搜索过程所走过的路径称为：引用链。

如果某个对象向上找不到任何GC Roots对象，则对象不可达，对象被回收。

## 5.典型垃圾回收器

**Serial GC**：最古老的垃圾收集器，其收集工作是单线程的，并且在进行垃圾收集过程中，会进入 “Stop-The-World”状态（指的是Gc事件发生过程中，会产生应用程序的停顿。停顿产生时整个应用程序线程都会被暂停，没有任何响应，有点像卡死的感觉，这个停顿称为STW）**采用标记 - 整理算法**

**CMS**:以最小的停顿时间为目标、只运行在老年代的垃圾回收器，**使用标记-清除算法**，可以并发收集。存在着内存碎片化问题，所以难以避免在长时间运行等情况下发生 full GC，导致恶劣的停顿。另外，既然强调了并发（Concurrent），CMS 会占用更多 CPU 资源，并和用户线程争抢。

Full GC通常意义上而言指的是一次特殊GC的行为描述，这次GC会回收整个堆的内存，包含老年代，新生代，metaspace等。

**G1** ：JDK1.9以后的默认垃圾回收器，注重响应速度，支持并发，兼顾吞吐量和停顿时间，采用标记整理+复制算法回收内存，使用可达性分析法来判断对象是否可以被回收。

**判断一个对象是否可以回收**

1. 引用计数法

给对象添加一个计数器，当对象增加一个引用时计数器加 1，引用失效时减 1。计数为 0 时，对象可被回收。

当两个对象出现循环引用时，计数器永远不为 0，导致无法对它们进行回收。

正因为循环引用的存在，因此 Java 虚拟机不使用引用计数算法。

.2. **可达性分析**

以GC Roots的根对象作为起点，根据对象的引用向下搜索，能搜索到的对象即为不可回收对象，搜索过程所走过的路径称为：引用链。

如果某个对象向上找不到任何GC Roots对象，则对象不可达，对象被回收。

## 6.类加载器和双亲委派机制

从父类加载器到子类加载器分别为：

BootStrapClassLoader 加载路径为：JAVA\_HOME/jre/lib

ExtensionClassLoader 加载路径为：JAVA\_HOME/jre/lib/ext

ApplicationClassLoader 加载路径为：classpath

还有一个自定义类加载器

当一个类加载器收到类加载请求时，会先把这个请求交给父类加载器处理，若父类加载器找不到该类，再由自己去寻找。该机制可以避免类被重复加载，保障了类的唯一性，还可以避免系统级别的类被篡改

## 7.JVM中有哪些引用？

强引用：new的对象。哪怕内存溢出也不会回收

软引用：通过WeakReference引用对象，只有内存不足时才会回收

弱引用：通过WeakReference引用对象，每次垃圾回收都会回收

虚引用：使用PlantomReference引用对象，必须配合引用队列使用，一般用于追踪垃圾回收动作

## 8.类加载过程

**（1）加载** ：

1.通过类的全名，获取类的二进制数据流

2.解析类的二进制数据流为方法区内的数据结构(Java 类模型)

3.创建 java.lang.Class 类的实例，表示该类型。作为方法区这个类的各种数据的访问入口

**（2）连接**：

验证：验证字节码文件的正确性。

准备：正式为类变量在方法区中分配内存，并设置初始值，final类型的变量在编译时已经赋值了

解析：将常量池中的符号引用（如类的全限定名）解析为直接引用（类在实际内存中的地址）

**（3）初始化** ：为类的静态变量赋予正确的初始值。执行类的静态代码块。

**类的初始化时机：只有类或接口被Java程序首次主动使用时才初始化他们。**

1.new getstatic putstatic invokestatic指令，访问final变量除外

2.java.lang.reflect对类进行反射调用时

3.初始化子类的时候，父类首先初始化

4.虚拟机启动时，被执行的主类必须初始化

5.动态语言支持java.lang.invoke.MethodHandle解析的结果为REF\_getstatic REF\_putstatic REF\_invokestatic的方法句柄时，该类必须初始化

//

类的卸载条件：

堆中不含有该类的任何实例  
该类的classloader被回收  
该类对应的class对象没有被任何地方所引用，即没有通过反射访问该类方法。

## 9.JVM类初始化顺序

父类静态代码块和静态成员变量->子类静态代码块和静态成员变量->父类代码块和普通成员变量->父类构造方法->子类代码块和普成员变量->子类构造方法

## 10..对象的创建过程

（1）检查类是否已被加载，没有加载就先加载类

（2）为对象在堆中分配内存，使用CAS方式分配，防止在为A分配内存时，执行当前地址的指针还没有来得及修改，对象B就拿来分配内存。

（3）初始化，将对象中的属性都分配0值或null

（4）设置对象头

（5）为属性赋值和执行构造方法

## 11.对象头中有哪些信息

二-18.Synchronized锁原理和优化 也有

对象头中有两部分，一部分是MarkWork,存储对象运行时的数据，如对象的hashcode、GC分代年龄、GC标记、锁的状态、获取到锁的线程ID等；另外一部分是表明对象所属类，如果是数组，还有一个部分存放数组长度

## 12.JVM内存参数

-Xmx[]:堆空间最大内存

-Xms[]:堆空间最小内存，一般设置成跟堆空间最大内存一样的

-Xmn[]:新生代的最大内存

-xx:[survivorRatio=3]:eden区与from+to区的比例为3：1，默认为4：1

-xx[use 垃圾回收器名称]：指定垃圾回收器

-xss:设置单个线程栈大小

一般设堆空间为最大可用物理地址的百分之80

## 13.GC的回收机制和原理

GC的目的实现内存的自动释放，使用可达性分析法判断对象是否可回收，采用了分代回收思想，

将堆分为新生代、老年代，新生代中采用复制算法，老年代采用整理算法，当新生代内存不足时会发生minorGC,老年代不足时会发送fullGC

# 四.数据结构和算法

## 1.栈与队列的相同点：

1.都是线性结构。

2.插入操作都是限定在表尾进行。

3.都可以通过顺序结构和链式结构实现。、

4.插入与删除的时间复杂度都是O（1），在空间复杂度上两者也一样。

5.多链栈和多链队列的管理模式可以相同。

## 2.栈与队列的不同点：

1.删除数据元素的位置不同，栈的删除操作在表尾进行，队列的删除操作在表头进行。

2.应用场景不同；常见栈的应用场景包括括号问题的求解，表达式的转换和求值，函数调用和递归实现，深度优先搜索遍历等；常见的队列的应用场景包括计算机系统中各种资源的管理，消息缓冲器的管理和广度优先搜索遍历等。

3.顺序栈能够实现多栈空间共享，而顺序队列不能。

## 栈实现队列



## 队列实现栈



## 3.简述二叉排序树

答：二叉排序树(Binary Sort Tree)，又称二叉查找树（Binary Search Tree)，也称二叉搜索树。它或者是一棵空树，或者是具有以下性质的二叉树：

（1）若它的左子树不空，则左子树上所有节点的值均小于它的根节点的值；

（2）若它的右子树不空，则右子树上所有节点的值均大于它的根节点的值；

（3）它的左右子树也分别为二叉排序树。

左<中<右

## 4.简述线索二叉树

答：对于n个结点的二叉树，在二叉链存储结构中有n+1个空链域，利用这些空链域存放在某种遍历次序下该结点的前驱结点和后继结点的指针，这些指针称为线索，加上线索的二叉树称为线索二叉树。

## 5.二叉树与度为2的树的区别有哪些？

（1）度为2的树中至少有一个结点的度为2（即至少有3个结点），而二叉树没有这种要求，它可以为空。

（2）度为2的有序树的孩子的左右次序是相对于另一孩子而言的，若某个结点只有一个孩子，则这个孩子就无须区分其左右次序，而二叉树无论其孩子数是否为2，均需确定其左右次序，即二叉树的结点次序不是相对于另一结点而言，而是确定的。

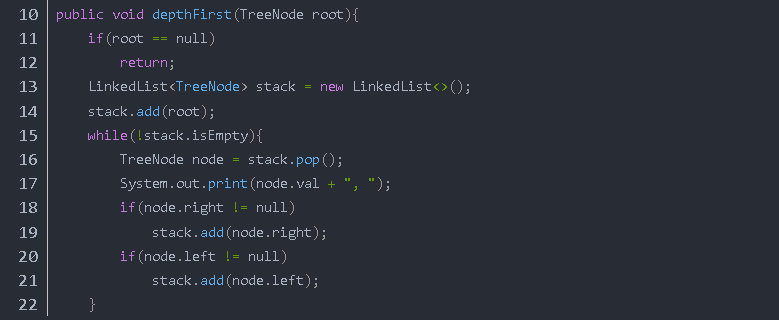
## 6.二叉树的存储结构有哪些？

答：顺序存储结构和链式存储结构。

## 深度优先搜索（DFS）

深度优先搜索（DFS，Depth First Search），就是“一条路走到黑”。对每一个可能的分支路径深入到不能再深入为止，当访问某个节点到尽头时，返回上一个还没访问的节点继续进行深度优先搜索。

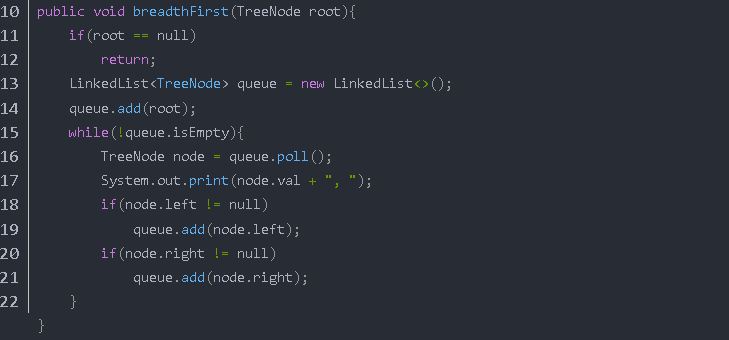
深度优先搜索常用栈（Stack）这种数据结构配合实现，利用 Stack 先进后出的特点。



## [广度优先搜索](https://so.csdn.net/so/search?q=%E5%B9%BF%E5%BA%A6%E4%BC%98%E5%85%88%E6%90%9C%E7%B4%A2&spm=1001.2101.3001.7020)（BFS）

广度优先搜索（BFS，Breadth First Search），广度优先是连通图的一种遍历策略。它并不考虑结果的可能位置，而是彻底地搜索整张图，检查图中所有的节点，直到找到结果为止。

广度优先搜索常用队列（queue）这种数据结构配合实现，利用 queue 先进先出的特点。

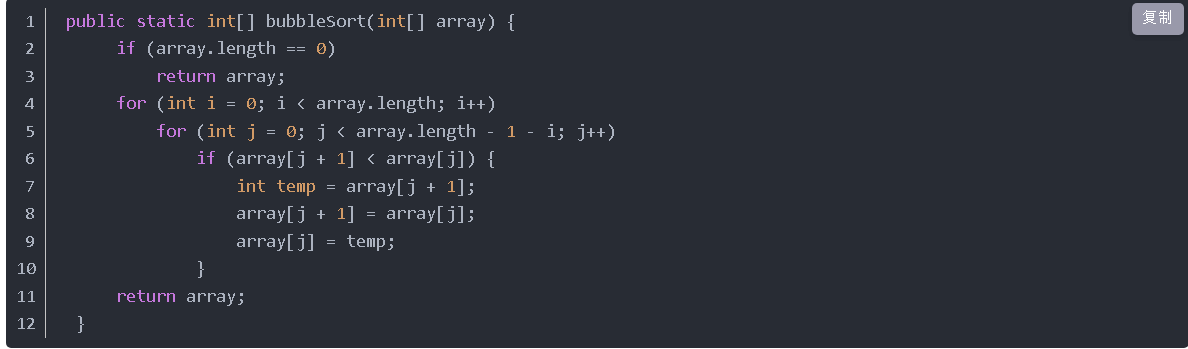


## 7.常见排序算法复杂度



## 冒泡排序

比较相邻的元素。如果第一个比第二个大，就交换它们两个；对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对，这样在最后的元素应该会是最大的数；针对所有的元素重复以上的步骤，除了最后一个；重复步骤1~3，直到排序完成。



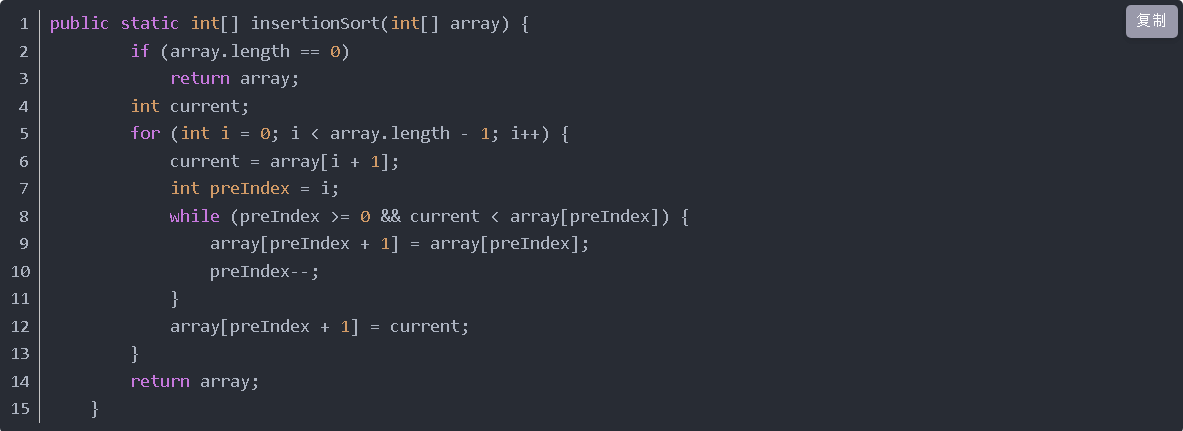
## 选择排序

首先在未排序序列中找到最小（大）元素，存放到排序序列的起始位置，然后，再从剩余未排序元素中继续寻找最小（大）元素，然后放到已排序序列的末尾。以此类推，直到所有元素均排序完毕。



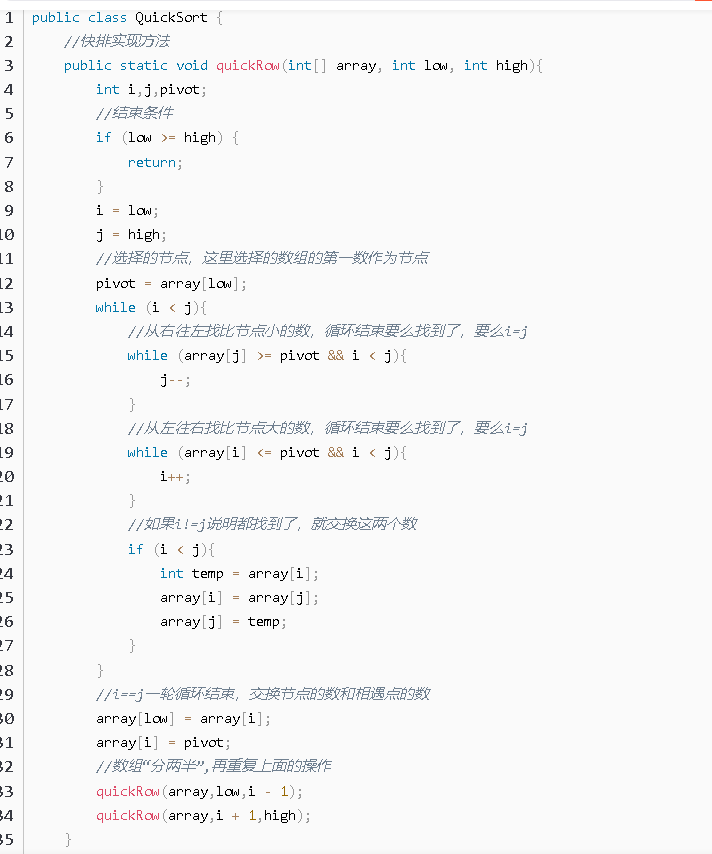
## 插入排序

比较相邻的元素。如果第一个比第二个大，就交换它们两个；对每一对相邻元素作同样的工作，从开始第一对到结尾的最后一对，这样在最后的元素应该会是最大的数；针对所有的元素重复以上的步骤，除了最后一个；重复步骤1~3，直到排序完成。



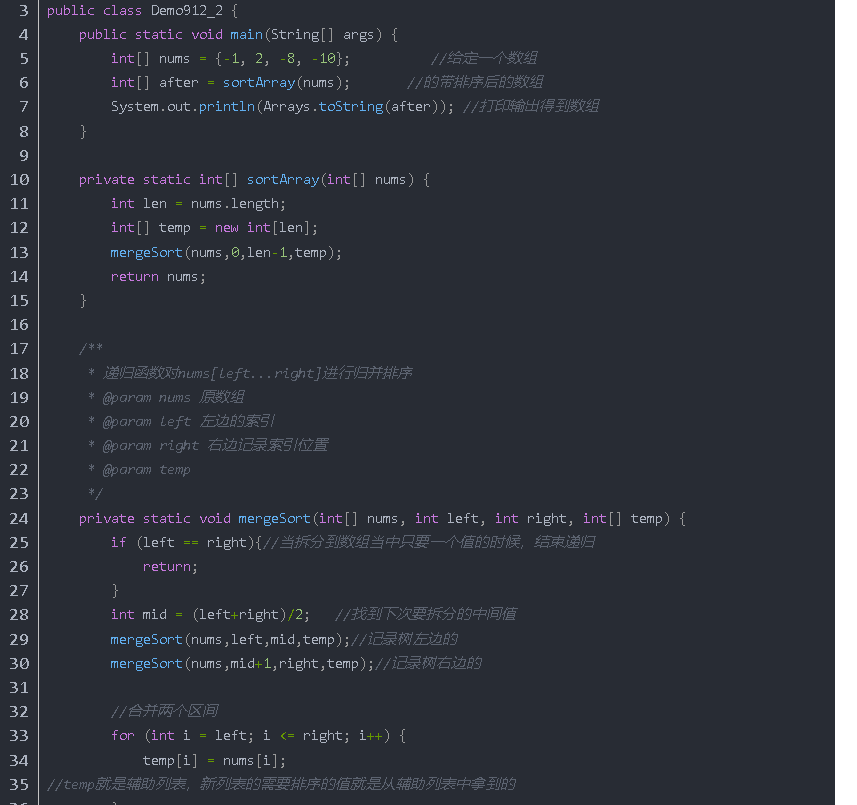
## 快速排序

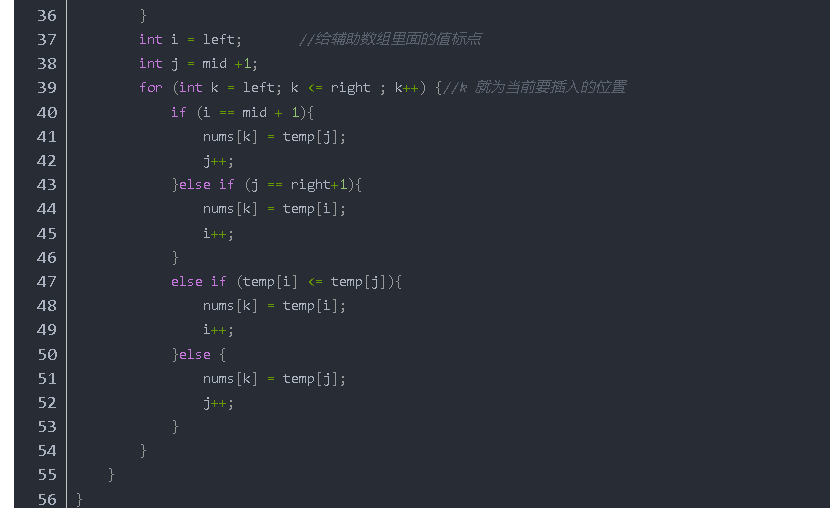
从数列中挑出一个元素，称为 “基准”（pivot）；重新排序数列，所有元素比基准值小的摆放在基准前面，所有元素比基准值大的摆在基准的后面（相同的数可以到任一边）。在这个分区退出之后，该基准就处于数列的中间位置。这个称为分区（partition）操作；递归地（recursive）把小于基准值元素的子数列和大于基准值元素的子数列排序。



## 归并排序

把长度为n的输入序列分成两个长度为n/2的子序列；对这两个子序列分别采用归并排序；将两个排序好的子序列合并成一个最终的排序序列。





## 希尔排序

希尔排序通过比较相距一定间隔的元素来进行，各趟比较所用的距离随着算法的进行而减小，直到只比较相邻元素的最后一趟排序为止。希尔排序目的为了加快速度改进了[插入排序](https://so.csdn.net/so/search?q=%E6%8F%92%E5%85%A5%E6%8E%92%E5%BA%8F&spm=1001.2101.3001.7020)，交换不相邻的元素对数组的局部进行排序，并最终用插入排序将局部有序的数组排序。



# 五.Mysql篇

## 1.MyIsAm和InnoDB的区别

InnoDB有三大特性，分别是事务、外键、行级锁，这些都是MyIsAm不支持的，

另外InnoDB是聚簇索引，MyIAm是非聚簇索引，

InnoDB不支持全文索引，MyIAm支持

InnoDB支持自增和MVCC模式的读写，MyIAm不支持

MyIsAM的访问速度一般InnoDB快，差异在于innodb的mvcc、行锁会比较消耗性能，还可能有回表的过程（先去辅助索引中查询数据，找到数据对应的key之后，再通过key回表到聚簇索引树查找数据）

## 2.mysql事务特性

**事务是用户定义的一个数据库操作序列，这些操作要么全做，要么全不做，是一个不可分割的工作单位**

原子性：一个事务内的操作统一成功或失败

一致性：事务前后的数据总量不变

隔离性：事务与事务之间相互不影响

持久性：事务一旦提交发生的改变不可逆

为什么要有事务回滚机制：在 MySQL 中，**恢复机制是通过回滚日志（undo log）实现的**，所有事务进行的修改都会先记录到这个回滚日志中，然后在对数据库中的对应行进行写入。**当事务已经被提交之后，就无法再次回滚了。**

## 3.事务靠什么保证

原子性：由undolog日志保证，他记录了需要回滚的日志信息，回滚时撤销已执行的sql

一致性：由其他三大特性共同保证，是事务的目的

隔离性：由MVCC保证

持久性：由redolog日志和内存保证，mysql修改数据时内存和redolog会记录操作，宕机时可恢复

## 4.事务的隔离级别

在高并发情况下，并发事务会产生脏读、不可重复读、幻读问题，这时需要用隔离级别来控制, 事务的隔离性是由锁和mvcc实现的。

**脏读**：A 事务读取到了 B 事务未提交的内容，但是之后B事务满足一致性等特性而做了回滚操作，那么读取事务得到的结果就是脏数据了。

**幻读**：A 事务读取了一个范围的内容，而同时 B 事务在此期间插入（删除）了一条数据。造成"幻觉"。

**丢弃修改**：两个写事务T1 T2同时对A=0进行递增操作，结果T2覆盖T1，导致最终结果是1 而不是2，事务被覆盖

**不可重复读**：当设置T2事务只能读取 T1 事务已经提交的部分，T2 读取一个数据，然后T1 对该数据做了修改。如果 T2 再次读取这个数据，此时读取的结果和第一次读取的结果不同。

读未提交： 允许一个事务读取另一个事务已提交的数据，可能出现不可重复读，幻读。

读提交： 只允许事务读取另一个事务没有提交的数据可能出现不可重复读，幻读。

可重复读： 确保同一字段多次读取结果一致，可能出现欢幻读。

可串行化： 所有事务逐次执行，没有并发问日

Inno DB 默认隔离级别为可重复读级别，分为快照度和当前读，并且通过间隙锁解决了幻读问题。

## 5.什么是快照读和当前读

**\*快照读**读取的是当前数据的可见版本，可能是会过期数据，像不加锁的 select 操作就是快照读，即不加锁的非阻塞读；快照读的前提是隔离级别不是串行级别，串行级别下的快照读会退化成当前读；之所以出现快照读的情况，是基于提高并发性能的考虑，快照读的实现是基于MVCC ,可以认为 MVCC 是行锁的一个变种，但它在很多情况下，避免了加锁操作，降低了开销

**\*当前读**读取的是数据的最新版本，并且当前读返回的记录都会上锁，保证其他事务不会并发修改这条记录。如update、insert、delete、select for undate（排他锁）、select lockin share mode（共享锁） 都是当前读

**「快照读就是 MySQL 实现 MVCC 理想模型的其中一个非阻塞读功能」**。而相对而言，当前读就是悲观锁的具体功能实现

## 6.MVCC是什么

**MVCC 就是为了实现读-写冲突不加锁，而这个读指的就是快照读, 而非当前读，当前读实际上是一种加锁的操作，是悲观锁的实现**

MVCC是多版本并发控制，维持一个数据的多个版本，使得读写操作没有冲突。只在读已提交和可重复读中生效。

它的底层实现主要是分为了三个部分：

**隐藏字段**：在mysql中给每个表都设置了隐藏字段，有一个是trx\_id(事务id)，记录每一次操作的事务id，是自增的；另一个字段是roll\_pointer(回滚指针)，指向上一个版本的事务版本记录地址

**undo log：**主要的作用是记录回滚日志，存储老版本数据，在内部会形成一个版本链，在多个事务并行操作某一行记录，记录不同事务修改数据的版本，通过roll\_pointer指针形成一个链表

**readView**:解决的是一个事务查询选择版本的问题，在内部定义了一些匹配规则和当前的一些事务id判断该访问那个版本的数据，不同的隔离级别快照读是不一样的，最终的访问的结果不一样。如果是rc隔离级别，每一次执行快照读时生成ReadView，如果是rr隔离级别仅在事务中第一次执行快照读时生成ReadView，后续复用

## 7.MySQL有哪些索引

索引是一种数据结构，可以帮助我们快速的进行数据的查找。数据库索引好比是一本书前面的目录，能加快数据库的查询速度。一般来说索引本身也很大，不可能全部存储在内存中，因此索引往往是存储在磁盘上的文件中的（可能存储在单独的索引文件中，也可能和数据一起存储在数据文件中）。

**我们通常所说的索引，包括聚集索引、覆盖索引、组合索引、前缀索引、唯一索引等，没有特别说明，默认都是使用B+树结构组织（多路搜索树，并不一定是二叉的）的索引**

优点：加快数据的检索速度、加速表和表之间的连接 缺点：索引需要占物理空间。

主键索引：一张表只能有一个主键索引，主键索引列不能有空值和重复值

唯一索引：唯一索引不能有相同值，但允许为空

普通索引：允许出现重复值

组合索引：对多个字段建立一个联合索引，减少索引开销，遵循最左匹配原则

全文索引：myisam引擎支持，通过建立倒排索引提升检索效率，广泛用于搜索引擎

## 8.聚簇索引和非聚簇索引的区别

聚簇索引：聚簇索引就是指数据与索引放到一块，同时叶子节点中存放行数据**，**也将聚簇索引的叶子节点称为数据页。

在 InnoDB 中，**只有主键索引是聚簇索引**，如果没有主键，则挑选一个唯一键建立聚簇索引。如果没有唯一键，则MySQL自动为InnoDB表生成一个隐含字段来建立聚簇索引，这个字段长度为6个字节，类型为长整形。

**优点**：根据索引可以直接获取值，所以他获取数据更快；对于主键的排序查找和范围查找效率更高；当查询使用聚簇索引时，在对应的叶子节点，可以获取到整行数据，因此不用再次进行回表查询。

缺点：如果主键值很大的话，辅助索引也会变得很大；如果用uuid作为主键，数据存储会很稀疏；修改主键或乱序插入会让数据行移动导致页分裂；所以一般我们定义主键时尽量让主键值小，并且定义为自增和不可修改。

非聚簇索引（辅助索引）：数据与索引分开存储，叶子节点存放的是数据行地址，先根据索引找到数据地址，再根据地址去找数据。一般我们自己定义的索引都是非聚簇索引。非聚簇索引不一定会回表查询，这涉及到查询语句所要求的字段是否全部命中了索引，如果全部命中了索引，那么就不必再进行回表查询。

举个简单的例子，假设我们在员工表的年龄上建立了索引，那么当进行select age from employee where age < 20的查询时，在索引的叶子节点上，已经包含了age信息，不会再次进行回表查询。

他们都是b+数结构

**什么是回表查询：**，回表的意思就是通过二级索引找到对应的主键值，然后再通过主键值找到聚集索引 中所对应的整行数据，这个过程就是回表

**什么叫覆盖索引：**覆盖索引是指select查询语句使用了索引，在返回的列，必须在索引中全部能够找到，如果我们使用id查询，它会直接走聚集索引查询，一次索引扫描，直接返回数据，性能高。 如果按照二级索引查询数据的时候，返回的列中没有创建索引，有可能会触 发回表查询，尽量避免使用select \*，尽量在返回的列中都包含添加索引的字 段

## 9.怎么**防止SQL注入**

发生SQL注入的根本原因是因为fasd or ‘1’=‘1’这条语句被当作了sql语句的一部分被编译进去了

1. 使用参数化查询：参数化查询（使用预编译语句或者存储过程来处理 SQL 查询语句）可以防止SQL注入，因为参数化查询会对用户输入的数据进行过滤和转义，从而保护查询语句免受攻击。

2.对用户输入的数据进行验证和过滤：在接收到用户输入数据时，进行数据验证和过滤可以有效地防止SQL注入。例如，检查用户输入是否包含特殊字符或SQL关键字，并且对输入进行转义。

3.不要使用管理员权限运行应用程序：为了防止SQL注入，不要使用管理员权限运行应用程序。应该为应用程序分配最小的权限，以避免黑客利用注入漏洞获得管理员权限。

4.定期更新应用程序和数据库：及时更新应用程序和数据库可以修补已知的安全漏洞，并增强安全性

## 10.为什么要用内连接而不用外连接？

用外连接的话连接顺序是固定死的，比如left join，他必须先对左表进行全表扫描，然后一条条到右表去匹配；而内连接的话mysql会自己根据查询优化器去判断用哪个表做驱动。

子查询的话同样也会对驱动表进行全表扫描，所以尽量用小表做驱动表。

内连接取量表交集部分，只会显示主表和从表相关联的字段，不会出现null

左连接取左表全部 右表匹部分，如果左侧的主表中没有关联字段，会用null 填满

右连接取右表全部 左表匹部分和letf join相反

## 11.MySQL整个查询的过程

（1）客户端向 MySQL 服务器发送一条查询请求

（2）服务器首先检查查询缓存，如果命中缓存，则返回存储在缓存中的结果。否则进入下一阶段

（3）服务器进行 SQL 解析、预处理、再由优化器生成对应的执行计划

（4）MySQL 根据执行计划，调用存储引擎的 API 来执行查询

（5）将结果返回给客户端，同时缓存查询结果

注意：只有在8.0之前才有查询缓存，8.0之后查询缓存被去掉了

## 12.执行计划中有哪些字段？

我们想看一个sql的执行计划使用的语句是explain+SQL，表中的字段包括：

id:select查询的优先级，id越大优先级越高，子查询的id一般会更大

select\_type:查询的类型,是普通查询还是联合查询还是子查询，常见类型有simple（不包含子查询），primary（标记复杂查询中最外层的查询），union(标记primart只后子查询)

table：者一行的数据是数哪张表的

type:扫描类型，效率从底到高为

ALL（全表扫描)>index(全索引扫描，我们的需要的数据在索引中可以获取)>range(范围索引扫描)>ref(使用非唯一索引列进行了关联查询)> eq\_ref (使用唯一索引进行关联查询)>const(查询结果只有一行，体现在使用唯一索引进行单表查询)>system(表中只有一行数据)

possible\_keys（可能的）:当前查询语句可能用到的索引，可能为null(如果用了索引但是为null有可能是表数据太少innodb认为全表扫描更快)

key:实际使用到的索引

ref（编号）:显示索引的哪一行被使用了

rows:估算大概多少行数据被查找了

excess（额外的）:mysql如何查询额外信息，常见的有：

Using filesort:排序时使用外部排序而不是索引排序，需要回表查询数据，然后在排序缓冲区中进行排序

backward index scan:排序时使用了索引排序，但如果是按照降序排序的话就会使用反向扫描索引

Using index:表示使用覆盖索引（覆盖索引：如果key中显示有索引，select后面查询的字段都可以从通过这个索引获取）

Using temporary:查询时要建立一个临时表存放数据

## 13.哪些情况索引会失效

（1）where条件中有or，除非所有查询条件都有索引，否则失效

（2）在字符串 like 时左边是通配符。类似于’%aaa’。，索引失效

（3）索引列参与数学运算或者函数，索引失效

（4）违背最左匹配原则，索引失效

（5）索引字段发生类型转换，索引失效

（6）mysql觉得全表扫描更快时（数据少），索引失效

## 14.B和B+树的区别，为什么使用B+树

**第一：在B树中，非叶子节点和叶子节点都会存放数据，而B+树的 所有的数据都会出现在叶子节点，在查询的时候，B+树查找效率更加稳定**

**第二：在进行范围查询的时候，B+树效率更高，因为B+树都在叶子节点存 储，并且叶子节点是一个双向链表**

## 15.MySQL有哪些锁

**当数据库有并发事务的时候,可能会产生数据的不一致,这时候需要一些机制来保证访问的次序,锁机制就是这样的一个机制.**

**基于粒度（锁的级别）：**

表级锁：对整张表加锁，粒度大并发小，开销小，加锁快，不会出现死锁。发生锁冲突的概率最高。

行级锁：只针对当前操作的行进行加锁，粒度小并发大开销大，加锁慢，会出现死锁。发生锁冲突的概率最低。

页级锁：锁定粒度介于行级锁和表级锁中间，开销和加锁时间界于表锁和行锁之间；会出现死锁；锁定粒度界于表锁和行锁之间，并发度一般。

死锁出现的案例？

批量入库，存在则更新，不存在则插入，insert into tab(xx,xx) on duplicate key update xx=‘xx’

如何处理死锁？

1.通过innodblockwait\_timeout来设置超时时间，一直等待直到超时

2.发起死锁检测，发现死锁之后，主动回滚死锁中的事务，不需要其他事务继续

**innodb默认是使用设置死锁时间来让死锁超时的策略，默认innodblockwait\_timeout设置的时长是50s**

如何避免死锁？

1.为了在单个innodb表上执行多个并发写入操作时避免死锁，可以在事务开始时，通过为预期要修改行，使用select …for update语句来获取必要的锁，即使这些行的更改语句是在之后才执行的

2.在事务中，如果要更新记录，应该直接申请足够级别的锁，即排他锁，而不应先申请共享锁，更新时在申请排他锁。因为这时候当用户在申请排他锁时，其他事务可能又已经获得了相同记录的共享锁

3.如果事务需要修改或锁定多个表，则应在每个事务中以相同的顺序使用加锁语句。在应用中，如果不同的程序会并发获取多个表，应尽量约定以相同的顺序来访问表，这样可以大大降低产生死锁的机会

4.通过 select …lock in share mode获取行的读锁后，如果当前事务在需要对该记录进行更新操作，则很有可能造成死锁

5.改变事务隔离级别

MySQL常用存储引擎的锁机制？

MyISAM和MEMORY采用表级锁(table-level locking)

BDB采用页面锁或表级锁，默认为页面锁

InnoDB支持行级锁和表级锁,默认为行级锁

**基于属性（使用方式）：**

共享锁：又称读锁，一个事务为表加了读锁，其它事务只能加读锁，不能加写锁，直到释放

SELECT … LOCK IN SHARE MODE; 在查询语句后面增加LOCK IN SHARE MODE，MySQL 就会对查询结果中的每行都加共享锁

排他锁：又称写锁，一个事务加写锁之后，其他事务不能再加任何锁，避免脏读问题

SELECT … FOR UPDATE; 在查询语句后面增加FOR UPDATE，MySQL 就会对查询结果中的每行都加排他锁

**基于思想：**

乐观锁：假设不会发生并发冲突，只在提交操作时检查是否违反数据完整性。 乐观锁不能解决脏读的问题。

悲观锁：假定会发生并发冲突，屏蔽一切可能违反数据完整性的操作

数据库悲观锁和乐观锁的原理和应用场景分别有什么？

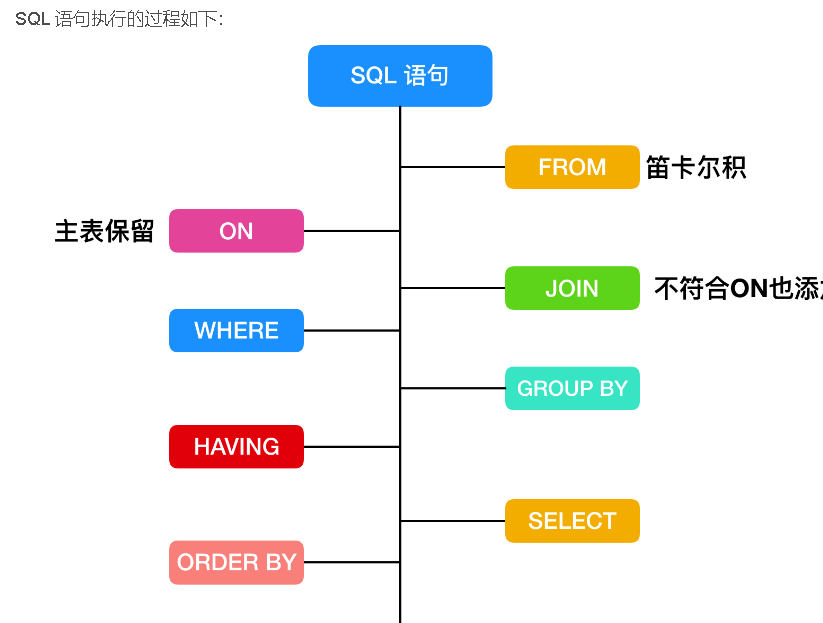
乐观锁，先进行业务操作，只在最后实际更新数据时进行检查数据是否被更新过。Java 并发包中的 AtomicFieldUpdater 类似，也是利用 CAS 机制，并不会对数据加锁，而是通过对比数据的时间戳或者版本号，来实现乐观锁需要的版本判断。

悲观锁，先获取锁，再进行业务操作，一般就是利用类似 SELECT … FOR UPDATE 这样的语句，对数据加锁，避免其他事务意外修改数据。当数据库执行SELECT … FOR UPDATE时会获取被select中的数据行的行锁，select for update获取的行锁会在当前事务结束时自动释放，因此必须在事务中使用。

## 16. undo log和redo log的区别

redo log日志记录的是数据页的物理变化，服务宕机可用来同步数据，而undo log 不同，它主要记录的是逻辑日志，当事务回滚时，通过逆操作恢复原来的数据，比如我们删除一条数据的时候，就会在undo log日志文件中新增一条delete语句，如果发生回滚就执行逆操作；redo log保证了事务的持久性，undo log保证了事务的原子性和一致性

## 17.sql执行顺序



## 18.如何设计数据库？

（1）抽取实体，如用户信息，商品信息，评论

（2）分析其中属性，如用户信息：姓名、性别...

（3）分析表与表之间的关联关系

然后可以参考三大范式进行设计，设计主键时，主键要尽量小并且定义为自增和不可修改。

## 19.where和having的区别？

where是约束声明，having是过滤声明，where早于having执行，并且where不可以使用聚合函数，having可以

## 20.三大范式

第一范式：每个列都不可以再拆分。

第二范式：在第一范式的基础上，非主键列完全依赖于主键，而不能是依赖于主键的一部分。

第三范式：在第二范式的基础上，非主键列只依赖于主键，不依赖于其他非主键。

## 21.char和varchar的区别

char是不可变的，最大长度为255，varchar是可变的字符串，最大长度为2^16

varchar(10) 和 int(10) 代表什么含义?

varchar(10)中的10代表字符串最大长度为10，存进库里的字符串长度超过10将报错无法进行存储

int(10)表示的是存储数字长度，如果长度不够10就会按照0进行填充

## 22. MySQL存储过程

存储过程：（PROCEDURE）是事先经过编译并存储在数据库中的一段SQL语句的集合。调用存储过程可以简化应用开发人员的很多工作，减少数据在数据库和应用服务器之间的传输，对于提高数据处理的效率是很有好处的。

就是数据库 SQL 语言层面的代码封装与重用。

存储过程就类似于Java中的方法，需要先定义，使用时需要调用。存储过程可以定义参数，参数分为IN、OUT、INOUT类型三种类型。

1.IN类型的参数表示接受调用者传入的数据；

2.OUT类型的参数表示向调用者返回数据；

3.INOUT类型的参数即可以接受调用者传入的参数，也可以向调用者返回数据。

## 23.MySQL 删除自增 id，随后重启 MySQL 服务，再插入数据，自增 id 会从几开始？

innodb 引擎：

MySQL8.0前，下次自增会取表中最大 id + 1。原理是最大id会记录在内存中，重启之后会重新读取表中最大的id

MySQL8.0后，仍从删除数据 id 后算起。原理是它将最大id记录在redolog里了

myisam：

自增的 id 都从删除数据 id 后算起。原理是它将最大id记录到数据文件里了

## 24.优化SQL

MySQL插入百万级的数据如何优化？

（1）一次sql插入多条数据，可以减少写redolog日志和binlog日志的io次数（sql是有长度限制的，但可以调整）

（2）保证数据按照索引进行有序插入

怎么优化SQL

**表结构优化**

1.尽量使用数字型字段：若只含数值信息的字段尽量不要设计为字符型，这会降低查询和连接的性能，并会增加存储开销。这是因为引擎在处理查询和连接时会逐个比较字符串中每一个字符，而对于数字型而言只需要比较一次就够了。

2.尽可能的使用 varchar 代替 char：变长字段存储空间小，可以节省存储空间。

3.当索引列大量重复数据时，可以把索引删除掉：比如有一列是性别，几乎只有男、女、未知，这样的索引是无效的。

**查询优化**

1.应尽量避免在 where 子句中使用!=或<>操作符

2.应尽量避免在 where 子句中使用 or 来连接条件

3.任何查询也不要出现select \*

4.避免在 where 子句中对字段进行 null 值判断

**索引优化**

1.对作为查询条件和 order by的字段建立索引

2.避免建立过多的索引，多使用组合索引

**慢查询优化**

1.分析语句，是否加载了不必要的字段/数据

2.分析 SQL 语句，有没有走适当的索引

3.如果 SQL 很复杂，优化 SQL 结构

4.如果表数据量太大，考虑分表

## 25.视图

**视图**是一种**虚拟的表**，通常是有一个表或者多个表的行或列的子集，具有和物理表相同的功能，使用视图可以简化复杂的 sql 操作，隐藏具体的细节，保护数据，视图创建后，可以使用与表相同的方式利用它们。 视图不能被索引，也不能有关联的触发器或默认值，如果视图本身内有order by 则对视图再次order by将被覆盖。

**create view xxx as xxxx**

**游标**是对查询出来的结果集作为一个单元来有效的处理。

一般不使用游标，但是需要逐条处理数据的时候，游标显得十分重要。

## 26. 数据库中的主键、超键、候选键、外键是什 么？

超键：在关系中能唯一标识元组的属性集称为关系模式的超键

候选键：不含有多余属性的超键称为候选键。也就是在候选键中，若再删除属性，就不是键了！

**主键**：主键是这行记录在这张表中的唯一标识。（就像身份证号码）。即使业务上本张表没有主键,也建议添加一个自增长的 ID 列作为主键.设定了主键之后,在后续的删改查的时候可能更加快速以及确保操作数据范围安全。

一张表的主键约束只能有一个

外键：如果关系模式R中属性K是其它模式的主键，那么k在模式R中称为外键。

## 27.常见约束

## 28.如果让你做分库与分表的设计，简单说说你会怎么做？

分库分表方案:

水平分库：以字段为依据，按照一定策略（hash、range等），将一个库中的数据拆分到多个库中。

水平分表：以字段为依据，按照一定策略（hash、range等），将一个表中的数据拆分到多个表中。

垂直分库：以表为依据，按照业务归属不同，将不同的表拆分到不同的库中。

垂直分表：以字段为依据，按照字段的活跃性，将表中字段拆到不同的表（主表和扩展表）中。

常用的分库分表中间件：

sharding-jdbc

Mycat

分库分表可能遇到的问题：

事务问题：需要用分布式事务

跨节点Join的问题：解决这一问题可以分两次查询实现

跨节点的count,order by,group by以及聚合函数问题：分别在各个节点上得到结果后在应用程序端进行合并。

数据迁移，容量规划，扩容等问题

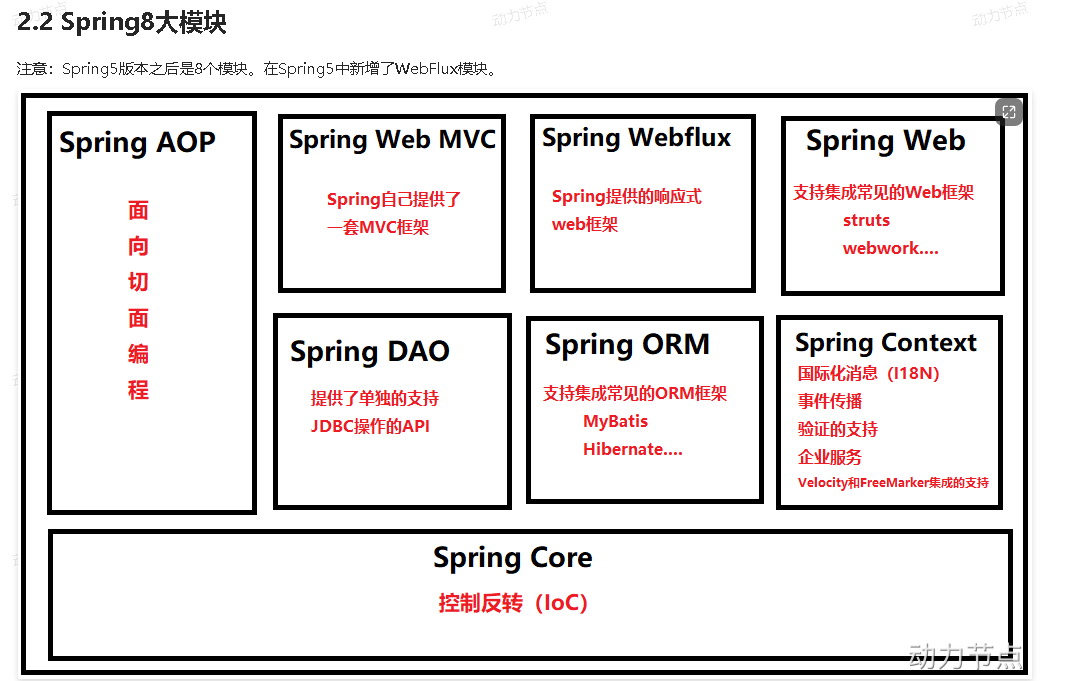
ID问题：数据库被切分后，不能再依赖数据库自身的主键生成机制，最简单可以考虑UUID

跨分片的排序分页问题

# 六.常用开发框架系列

## 1.什么是Spring？

Spring是个轻量级的框架，通过IOC达到松耦合的目的，通过AOP可以分离应用业务逻辑和系统服务进行内聚性的开发，不过配置各种组件时比较繁琐，所以后面才出选了SpringBoot的框架。



## 2.IOC是什么？

控制反转是为了降低程序耦合度，提高程序扩展力，达到OCP原则(对扩展开放，对修改关闭)，达到DIP原则(依赖倒置原则,倡导面向抽象编程，面向接口编程，不要面向具体编程，让上层不再依赖下层，下面改动了，上面的代码不会受到牵连)

把对象的创建和调用从程序员手中交由IOC容器管理，降低对象之间的依赖关系。

在没有spring之前是应用程序直接自己控制资源而有了spring之后应用程序就不需要自己去了解资源底层的一切，由容器去控制资源了，这里对资源的控制权发生了改变，这就叫控制反转

**Spring IoC容器的实现原理：工厂模式 + 解析XML + 反射机制。**

控制反转这种思想如何实现呢？

DI（依赖注入）：现在应用程序如果需要资源就直接向容器要，它不用管这个资源怎么来的只需要用它就可以了

这里容器向应用程序提供资源就叫依赖注入

## 3. Bean

**Bean是一个由Spring IoC容器实例化、组装和管理的对象**

**创建一个bean的方式**

1.通过反射调用构造方法创建bean对象

调用类的构造方法获取对应的bean实例，是使用最多的方式，这种方式只需要在xml bean元素中指定class属性，spring容器内部会自动调用该类型的构造方法来创建bean对象，将其放在容器中以供使用

2.通过静态工厂方法创建bean对象

创建一个静态工厂类(不通过new，而是用一个静态方法来对外提供自身实例的方法，即为我们所说的静态工厂方法)，用于生成Blog对象。

3. 通过实例工厂方法创建bean对象

让spring容器去调用某些对象的某些实例方法来生成bean对象放在容器中以供使用。

4.通过FactoryBean来创建bean对象

BeanFactory是spring容器的顶层接口，而这里要说的是FactoryBean，也是一个接口， FactoryBean可以让spring容器通过这个接口的实现来创建我们需要的bean对象。

我们在对一个bean进行实例化后，要对他的属性进行填充，大多数我们都是使用 @Autowire直接的填充依赖注入的，他是有限按照类型进行匹配。

**Bean 的作用域**

（1）Singleton: 默认情况下，Spring的IoC容器创建的Bean对象是单例的,一个IOC容器只有一个

（2）Prototype: 在bean标签中指定scope属性的值为：prototype,每次调用getBean()都会生成一个新的对象, 调用几次则创建几次

仅限于在WEB应用中使用:

（3）request:每个http请求都会创建一个自己的bean, 一个请求对应一个Bean

（4）session:一个session对应一个Bean

（5）application: 一个应用对应一个Bean

（6）webSocket: 一个websocket生命周期对应一个bean。WebSocket是

（HTML5下一种新的协议，基于tcp，实现了浏览器与服务器全双工通信，节省资源和带宽并达到实时通讯，Websocket是一个持久化的协议）

**Bean的实例化方式**

1.通过构造方法实例化。默认情况下，会调用Bean的无参数构造方法。

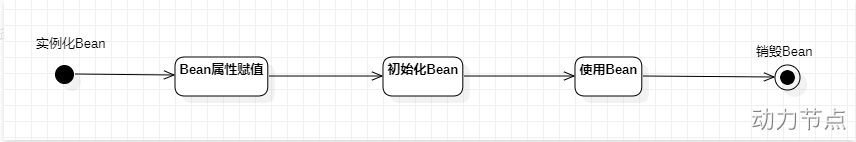
2.通过简单工厂模式实例化。编写简单工厂模式当中的工厂类，在Spring配置文件中指定创建该Bean的方法（使用factory-method属性指定）

3.通过factory-bean实例化。本质上是通过工厂方法模式进行实例化。定义具体工厂类，工厂类中定义实例方法，在Spring配置文件中指定factory-bean以及factory-method

以上的第三种方式中，factory-bean是我们自定义的，factory-method也是我们自己定义的。在Spring中，当你编写的类直接实现FactoryBean接口之后，factory-bean不需要指定了，factory-method也不需要指定了。factory-bean会自动指向实现FactoryBean接口的类，factory-method会自动指向getObject()方法

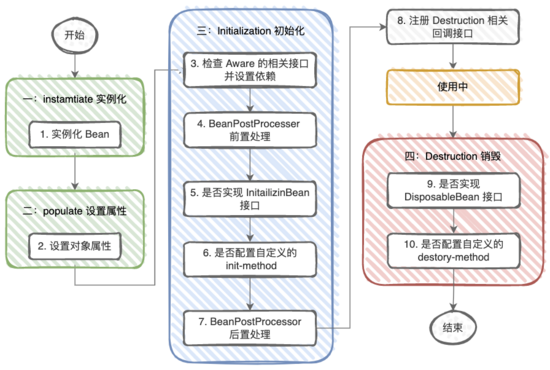
4.通过FactoryBean接口实例化。编写一个类实现FactoryBean接口，在Spring配置文件中配置FactoryBean。

**Bean的生命周期**



我们描述 Bean 的生命周期，都是指的 Singleton Bean。

对 Prototype Bean 来说，当用户 getBean 获得 Prototype Bean 的实例后，IOC 容器就不再对当前实例进行管理，而是把管理权交由用户，此后再 getBean 生成的是新的实例。



**实例化**：第 1 步，实例化一个 Bean 对象；一般使用ApplicationContext，它是建立在BeanFactory之上的，拥有更多的方法

BeanFactory：延迟加载（使用时加载）

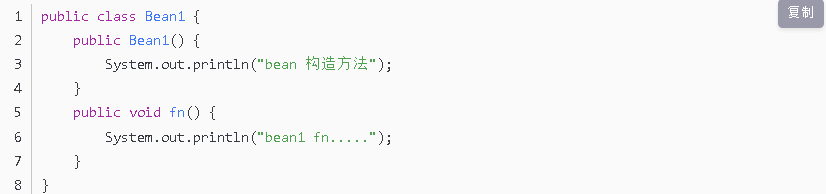
ApplicationContext：立即加载（初始化时加载）

**属性赋值**：第 2 步，为 Bean 设置相关属性和依赖；

**初始化**：初始化的阶段的步骤比较多，5、6 步是真正的初始化，第 3、4 步为在初始化前执行，第 7 步在初始化后执行，初始化完成之后，Bean 就可以被使用了；

初始化的三种方法：

1.类构造器初始化：最常用的方法，也是最简单的方法，构造器可以不用写可以用默认的2.静态工厂初始化 3. 静态工厂初始化



**销毁**：第 8~10 步，第 8 步其实也可以算到销毁阶段，但不是真正意义上的销毁，而是先在使用前注册了销毁的相关调用接口，为了后面第 9、10 步真正销毁 Bean 时再执行相应的方法。

实例化 Instantiation->属性赋值 Populate->初始化 Initialization->销毁 Destruction

在这四步的基础上面，Spring 提供了一些拓展点：

## 4.Bean的循环依赖是什么，怎么解决的？

循环依赖就是A对象中有B属性，B对象中有A属性。在创建 A 实例的时候里面包含着 B 属性实例，所以这个时候就需要去创建 B 实例，而创建B 实例过程中也包含着 A 实例。 这样 A 实例还在创建的过程当中，所以就导致 A 和 B 实例都创建不出来。

Spring为什么可以解决set + singleton模式下循环依赖？

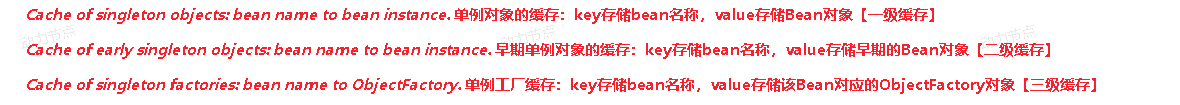
根本的原因在于：这种方式可以做到将“实例化Bean”和“给Bean属性赋值”这两个动作分开去完成。也就是说，Bean都是单例的，我们可以先把所有的单例Bean实例化出来，放到一个集合当中（我们可以称之为缓存），所有的单例Bean全部实例化完成之后，以后我们再慢慢的调用setter方法给属性赋值。这样就解决了循环依赖的问题。

spring通过三级缓存来解决循环依赖：**这三个缓存其实本质上是三个Map集合。**

一级缓存：缓存经过完整的生命周期的Bean

二级缓存 ：缓存未经过完整的生命周期的Bean

三级缓存：缓存的是ObjectFactory，其中存储了一个生成代理类的拉姆达表达式



spring会先从一级缓存中获取Bean，如果获取不到，则从二级缓存中获取Bean，如果二级缓存还是获取不到，则从三级缓存中获取，三级缓存存放了提前曝光**(使用addSingletonFactory()方法)**的ObjectFactory对象，通过ObjectFactory对象获取Bean实例，这样就解决了循环依赖的问题。

## 5.spring的bean是线程安全的吗？

spring的默认bean作用域是单例的，单例的bean不是线程安全的，但是开发中大部分的bean都是无状态的，不具备存储功能，比如controller、service、dao，他们不需要保证线程安全。

如果要保证线程安全，可以将bean的作用域改为prototype，比如像Model View。

另外还可以采用ThreadLocal来解决线程安全问题。ThreadLocal为每个线程保存一个副本变量，每个线程只操作自己的副本变量。

**ThreadLocal原理：**

ThreadLocal中填充的变量属于当前线程，原理是为每个线程创建变量副本，不同线程之间不可见，保证线程安全。每个线程内部都维护了一个Map，key为threadLocal实例，value为要保存的副本。

但是使用ThreadLocal会存在内存泄露问题，因为key为弱引用，而value为强引用，每次gc时key都会回收，而value不会被回收。所以为了解决内存泄漏问题，可以在每次使用完后删除value或者使用static修饰ThreadLocal，可以随时获取value

## 6.注解

**注解的存在主要是为了简化XML的配置**

负责声明Bean的注解，常见的包括四个：

● @Component

● @Controller

● @Service

● @Repository

这四个注解的功能都一样，另外三个都是@Component注解的别名

声明后这些Bean将被实例化，给Bean属性赋值需要用到这些注解：

● @Value 用来注入简单类型的属性

● @Autowired 用来注入非简单类型，可以出现在：属性上、构造方法上、构造方法的参数上、setter方法上。当带参数的构造方法只有一个，@Autowired注解可以省略。单独使用@Autowired注解，默认根据类型装配

● @Qualifier @Autowired注解默认根据类型注入。如果要根据名称注入的话，需要配合@Qualifier注解一起使用。

● @Resource 用来注入非简单类型，用在属性上、setter方法上，默认根据名称装配

**springboot常用注解**

@RestController ：修饰类，该控制器会返回Json数据

@RequestMapping("/path") ：修饰类，该控制器的请求路径

@Autowired : 修饰属性，按照类型进行依赖注入

@PathVariable : 修饰参数，将路径值映射到参数上

@ResponseBody :修饰方法，该方法会返回Json数据

@RequestBody（需要使用Post提交方式） :修饰参数，将Json数据封装到对应参数中

@Controller@Service@Compont: 将类注册到ioc容器

@Transaction：开启事务

## 7.AOP是什么？

AOP是面向切面编程，是一种编程技术，可以将那些与业务不相关但是很多业务都要调用的代码(例如：日志、事务管理、安全等交叉业务)抽取出来，思想就是不侵入原有代码的情况下对功能进行增强。

AOP的优点：

● 第一：代码复用性增强。

● 第二：代码易维护。

● 第三：使开发者更关注业务逻辑。

SpringAOP是基于动态代理实现的，一种是jdk动态代理，一种是cglib动态代理:

**jdk动态代理**是原理是利用反射来实现的，需要调用反射包下的Proxy类的newProxyInstance方法来返回代理对象，这个方法中有三个参数，分别是用于加载代理类的类加载器，被代理类实现的接口的class数组和一个用于增强方法的InvocaHandler实现类。

**cglib动态代理原理**是利用asm开源包来实现的，是把被代理类的class文件加载进来，通过修改它的字节码生成子类来处理

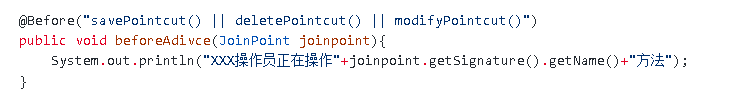
jdk动态代理要求被代理类必须有实现的接口，生成的动态代理类会和代理类实现同样的接口，cglib则生成的动态代理类会继承被代理类。Spring默认使用jdk动态代理，当被代理的类没有接口时就使用cglib动态代理

**如何使用aop自定义日志？**

第一步：创建一个切面类，把它添加到ioc容器中并添加@Aspect注解

第二步： 在切面类中写一个通知方法，在方法上添加通知注解并通过切入点表达式来表示要对哪些方法进行日志打印，然后方法参数为JoinPoint

第三步：通过JoinPoint这个参数可以获取当前执行的方法名、方法参数等信息，这样就可以根据需求在方法进入或结束时打印日志



## 8. Spring中的八大设计模式

**1. 简单工厂模式**: BeanFactory的getBean()方法，通过唯一标识来获取Bean对象。是典型的简单工厂模式（静态工厂模式）；

**2. 工厂方法模式**: FactoryBean是典型的工厂方法模式。在配置文件中通过factory-method属性来指定工厂方法，该方法是一个实例方法。

**3. 单例模式**: Spring用的是双重判断加锁的单例模式

**4. 代理模式:** Spring的AOP就是使用了动态代理实现的。

**5. 装饰器模式: 在尽可能少修改原有类代码下的情况下，做到动态切换不同的数据源.**JavaSE中的IO流是非常典型的装饰器模式, Spring中类名中带有：Decorator和Wrapper单词的类，都是装饰器模式

**6. 观察者模式:** 定义对象间的一对多的关系，当一个对象的状态发生改变时，所有依赖于它的对象都得到通知并自动更新。Spring中观察者模式一般用在listener的实现。

**7. 策略模式:** 策略模式是行为性模式，调用不同的方法，适应行为的变化 ，强调父类的调用子类的特性 。

**8. 模板方法模式:** Spring中的JdbcTemplate类就是一个模板类。它就是一个模板方法设计模式的体现。在模板类的模板方法execute中编写核心算法，具体的实现步骤在子类中完成。

## 9.Spring 事务原理？

Spring实现事务的两种方式**: 基于AOP实现的,一般使用声明式**

1.编程式事务:通过编写代码的方式来实现事务的管理。

2**.声明式事务**:**基于注解方式**,基于XML配置方式

在service类上或方法上增加@Transactional注解，这个方法中的sql会统一成功或失败。

原理是：

当一个方法加上@Transactional注解，spring会基于这个类生成一个代理对象并将这个代理对象作为bean，当使用这个bean中的方法时，如果存在@Transactional注解，就会将事务自动提交设为false，然后执行方法，执行过程没有异常则提交，有异常则回滚、

## 10.spring事务失效场景

（1）事务方法所在的类没有加载到容器中

（2）事务方法不是public类型

（3）同一类中，一个没有添加事务的方法调用另外以一个添加事务的方法，事务不生效

（4）spring事务默认只回滚运行时异常，可以用rollbackfor属性设置



（5）业务自己捕获了异常，事务会认为程序正常秩序

## 11.spring事务的传播行为

1.REQUIRED：支持当前事务，如果不存在就新建一个(默认)【没有就新建，有就加入】

2.SUPPORTS：支持当前事务，如果当前没有事务，就以非事务方式执行【有就加入，没有就不管了】

3.MANDATORY：必须运行在一个事务中，如果当前没有事务正在发生，将抛出一个异常【有就加入，没有就抛异常】

4.REQUIRES\_NEW：开启一个新的事务，如果一个事务已经存在，则将这个存在的事务挂起【不管有没有，直接开启一个新事务，开启的新事务和之前的事务不存在嵌套关系，之前事务被挂起】

5.NOT\_SUPPORTED：以非事务方式运行，如果有事务存在，挂起当前事务【不支持事务，存在就挂起】

6.NEVER：以非事务方式运行，如果有事务存在，抛出异常【不支持事务，存在就抛异常】

7.NESTED：如果当前正有一个事务在进行中，则该方法应当运行在一个嵌套式事务中。被嵌套的事务可以独立于外层事务进行提交或回滚。如果外层事务不存在，行为就像REQUIRED一样。【有事务的话，就在这个事务里再嵌套一个完全独立的事务，嵌套的事务可以独立的提交和回滚。没有事务就和REQUIRED一样】

## 12.SpringMVC工作原理

Spring MVC是Spring框架的一部分，是一个基于MVC模式的Web框架，用于构建Web应用程序。它提供了处理请求、路由、视图渲染等功能。

SpringMVC工作过程围绕着前端控制器DispatchServerlet，几个重要组件有HandleMapping（处理器映射器）、HandleAdapter（处理器适配器）、ViewReslover（试图解析器）

工作流程：

（1）DispatchServerlet接收用户请求将请求发送给HandleMapping

（2）HandleMapping根据请求url找到具体的handle和拦截器，返回给DispatchServerlet

（3）DispatchServerlet调用HandleAdapter,HandleAdapter执行具体的controller，并将controller返回的ModelAndView返回给DispatchServler

（4）DispatchServerlet将ModelAndView传给ViewReslover,ViewReslover解析后返回具体view

（5）DispatchServerlet根据view进行视图渲染，返回给用户

## 13.springboot自动配置原理

启动类@SpringbootApplication注解下，有三个关键注解

1、@SpringBootConfiguration：SpringBoot的配置类，将此注解标注在某个类上表示这个是一个SpringBoot的配置类。

2、@EnableAutoConfiguration：开启自动配置功能，工程中需要配置的东西,SpringBoot会帮我们自动配置；@EnableAutoConfiguration告诉SpringBoot开启自动配置功能。这样自动配置才会生效。

3、@ComponentScan：开启包的扫描功能，相当于spring中.xml配置中的<context:component-scan>,并没有指定basepackage，如果没有指定，spring底层会自动

## 14. MyBatis

MyBatis本质上就是对JDBC的封装，通过MyBatis完成CRUD，在三层架构中负责持久层的，属于持久层框架。

**（1）优点：**

① 与JDBC相比，减少了50%以上的代码量，消除了JDBC大量冗余的代码，不需要手动开关连接；

② 基于SQL语句编程，相当灵活，不会对应用程序或者数据库的现有设计造成任何影响，SQL写在XML里，解除sql与程序代码的耦合，便于统一管理；提供XML标签，支持编写动态SQL语句，并可重用。

③ 很好的与各种数据库兼容（因为MyBatis使用JDBC来连接数据库，所以只要JDBC支持的数据库MyBatis都支持）。

④ 能够与Spring很好的集成；

⑤ 提供映射标签，支持对象与数据库的ORM字段关系映射；提供对象关系映射标签，支持对象关系组件维护。

**（2）缺点**：

① SQL语句的编写工作量较大，尤其当字段多、关联表多时，对开发人员编写SQL语句的功底有一定要求。

② SQL语句依赖于数据库，导致数据库移植性差，不能随意更换数据库。

# 七.Redis系列

## 1.redis为什么快？

（1）完全基于内存操作，内存的速度远大于硬盘的速度，Redis的瓶颈是内存和带宽

（2）数据结构简单高效，简单动态字符串，双向链表，压缩列表，哈希表，跳表和整数数组

（3）redis执行命令是单线程的，避免了上下文切换带来的性能问题，不存在加锁释放锁操作，没有死锁问题导致的性能消耗

**Redis单线程指网络请求模块是一个线程处理所有网络请求，其他模块仍会使用多个线程。**

(4) 采用了非阻塞的io多路复用机制，使用了单线程来处理并发的连接;

**io多路复用**就是一个进程就能够处理多个套接字，解决了上下文切换的问题

select、poll、epoll都是IO多路复用的机制。I/O多路复用就是通过一种机制，可以监视多个描述符，一旦某个描述符就绪，能够通知程序进行响应的操作。

redis 的iO模型默认采用epoll实现的。

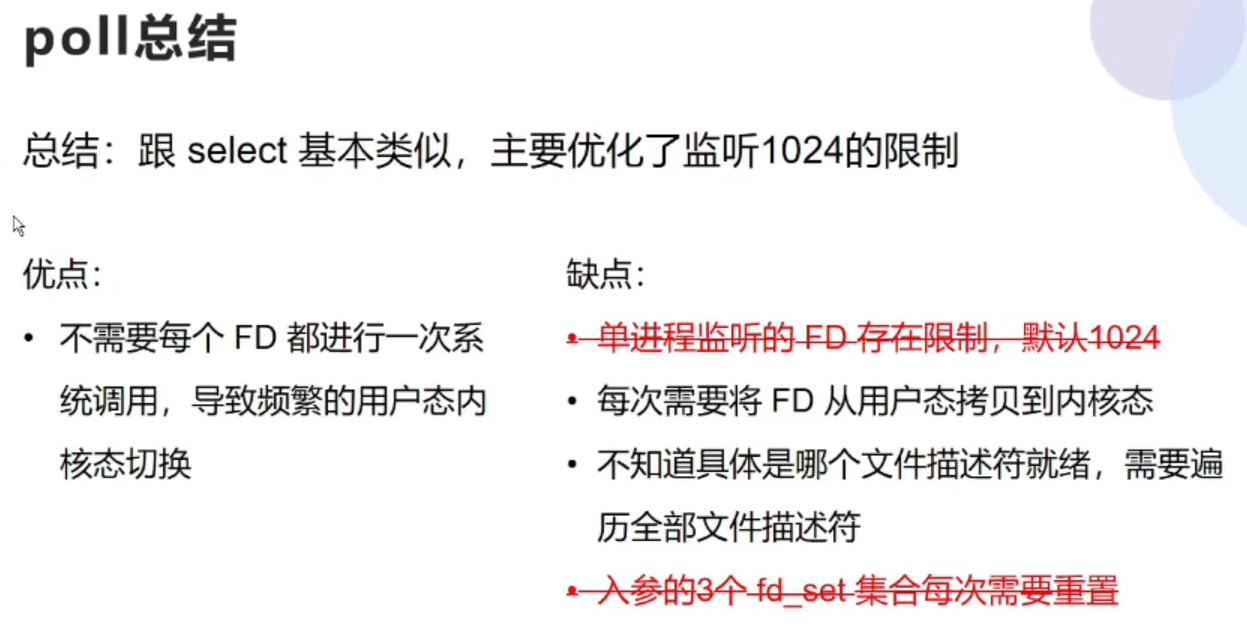
其发展可以分select->poll→epoll三个阶段来描述：

**select就是轮询，在Linux上限制个数一般为1024个**

select 函数监视的文件描述符分3类，分别是writefds、readfds、和exceptfds。调用后select函数会阻塞，直到有描述符就绪（有数据可读、可写、或者有except），或者超时（timeout指定等待时间，如果立即返回设为null即可），函数返回。当select函数返回后，可以通过遍历fd\_set，来找到就绪的描述符。

**poll解决了select的个数限制，但是依然是轮询**

poll改变了文件描述符集合的描述方式，使用了pollfd结构而不是select的fd\_set结构，使得poll支持的文件描述符集合限制远大于select的1024。



**epoll解决了个数的限制，同时解决了轮询的方式**

第一：epoll将文件描述符维护在内核态，不需要每次将想要监听的FD拷贝到内核态，直接通过epoll control将FD添加到内核态，后续一直维护epoll对象即可

第二：epoll通过就绪链表可以知道哪些文件描述符是就绪的，不需要轮询

**epoll\_create：**创建一个epoll文件描述符，创建eventpoll，其中包含红黑树cache和双向链表

**epollctl：**对指定描述符fd执行op的绑定操作，把fd写入红黑树，同时在内核注册回调函数

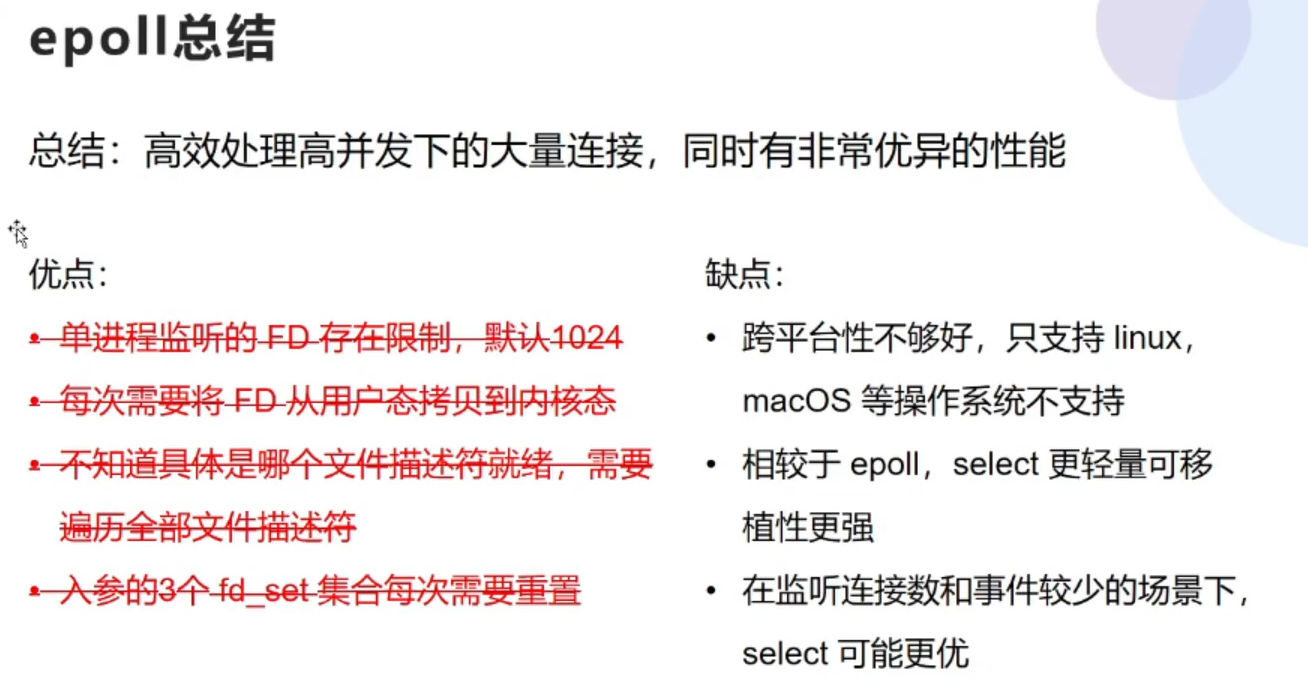
**epollwait：**获取epfd上的io事件

**为什么epoll快？**

**我们在调用epoll\_create时，内核除了帮我们在epoll文件系统里建了个file结点，在共享空间里建了个红黑树用于存储以后epoll\_ctl传来的socket外，还会再建立一个链表，用于存储准备就绪的事件，当epoll\_wait调用时，仅仅观察这个链表里有没有数据即可。有数据就返回，没有数据就sleep，等到timeout时间到后即使链表没数据也返回**

**就绪链表怎么维护？**

**当我们执行epoll\_ctl时，除了把socket放到epoll文件系统里file对象对应的红黑树上之外，还会给内核中断处理程序注册一个回调函数，告诉内核，如果这个句柄的中断到了，就把它放到准备就绪list链表里。所以，当一个socket上有数据到了，内核在把网卡上的数据copy到内核中后就来把socket插入到准备就绪链表里了。**



Redis坚持单线程是因为Redis的性能瓶颈是网络延迟而不是CPU，多线程对数据读取不会带来性能提升。

redis6.0引入多线程IO，只是用来处理网络数据的读写和协议的解析，而执行命令依旧是单线程。

## 2.redis持久化机制

**Redis持久化就是把内存的数据写到磁盘中去，防止服务宕机了内存数据丢失；**

（1）快照持久化RDB

redis的默认持久化机制，通过父进程fork一个子进程，子进程将redis的数据快照写入一个临时文件，等待持久化完毕后替换上一次的rdb文件。整个过程主进程不进行任何的io操作。持久化策略可以通过save配置单位时间内执行多少次操作触发持久化。所以RDB的优点是保证redis性能最大化，恢复速度数据较快，缺点是可能会丢失两次持久化之间的数据

（2）追加持久化AOF

以日志形式记录每一次的写入和删除操作，策略有每秒同步、每次操作同步、不同步，优点是数据完整性高，缺点是运行效率低，恢复时间长

## 3.Redis如何实现key的过期删除？

采用的定期过期+惰性过期

定期删除 ：Redis 每隔一段时间从设置过期时间的 key 集合中，随机抽取一些 key ，检查是否过期，如果已经过期做删除处理。

惰性删除 ：Redis 在 key 被访问的时候检查 key 是否过期，如果过期则删除。

## 4.Redis数据类型应用场景

String：可以用来缓存json信息，可以用incr命令实现自增或自减的计数器

Hash：与String一样可以保存json信息

List：可以用来做消息队列，list的pop是原子性操作能一定程度保证线程安全

Set：可以做去重，比如一个用户只能参加一次活动 ;可以做交集求共友

SortSet ：有序的。可以实现排行榜

## 5.Redis缓存穿透如何解决？

缓存穿透是指频繁请求客户端和缓存中都不存在的数据，缓存永远不生效，请求都到达了数据库。

解决方案：

（1）在接口上做基础校验，比如id<=0就拦截

（2）缓存空对象：找不到的数据也缓存起来，并设置过期时间，可能会造成短期不一致

（3）布隆过滤器：在客户端和缓存之间添加一个过滤器，拦截掉一定不存在的数据请求

## 6.Redis如何解决缓存击穿？

缓存击穿是值一个key非常热点，key在某一瞬间失效，导致大量请求到达数据库

解决方案：

（1）设置热点数据永不过期

（2）给缓存重建的业务加上互斥锁，缺点是性能低

## 7.Redis如何解决缓存雪崩？

缓存雪崩是值某一时间Key同时失效或redis宕机，导致大量请求到达数据库

解决方案：

（1）搭建集群保证高可用

（2）进行数据预热，给不同的key设置随机的过期时间

（3）给缓存业务添加限流降级，通过加锁或队列控制操作redis的线程数量

（4）给业务添加多级缓存

## 8.Redis分布式锁的实现原理

原理是使用setnx+setex命令来实现，但是会有一系列问题：

（1）任务时常超过缓存时间，锁自动释放。可以使用Redision看门狗解决

（2）加锁和释放锁的不是同一线程。可以在Value中存入uuid，删除时进行验证。但是要注意验证锁和删除锁也不是一个原子性操作，可以用lua脚本使之成为原子性操作

（3）不可重入。可以使用Redision解决（实现机制类似AQS,计数）

（4）redis集群下主节点宕机导致锁丢失。使用红锁解决

## 9.Redis集群方案

(1)主从模式：个master节点，多个slave节点，master节点宕机slave自动变成主节点

(2)哨兵模式：在主从集群基础上添加哨兵节点或哨兵集群，用于监控master节点健康状态，通过投票机制选择slave成为主节点

(3)分片集群：主从模式和哨兵模式解决了并发读的问题，但没有解决并发写的问题，因此有了分片集群。分片集群有多个master节点并且不同master保存不同的数据，master之间通过ping相互监测健康状态。客户端请求任意一个节点都会转发到正确节点，因为每个master都被映射到0-16384个插槽上，集群的key是根据key的hash值与插槽绑定

## 10.Redis集群主从同步原理

主从同步第一次是全量同步：slave第一次请求master节点会根据replid判断是否是第一次同步，是的话master会生成RDB发送给slave。

后续为增量同步：在发送RDB期间，会产生一个缓存区间记录发送RDB期间产生的新的命令,slave节点在加载完后，会持续读取缓存区间中的数据

## 11.Redis缓存一致性解决方案

Redis缓存一致性解决方案主要思考的是删除缓存和更新数据库的先后顺序

先删除缓存后更新数据库存在的问题是可能会数据不一致，一般使用延时双删来解决，即先删除缓存，再更新数据库，休眠X秒后再次淘汰缓存。第二次删除可能导致吞吐率降低，可以考虑进行异步删除。

先更新数据库后删除缓存存在的问题是会可能会更新失败，可以采用延时删除。但由于读比写快，发生这一情况概率较小。

但是无论哪种策略，都可能存在删除失败的问题，解决方案是用中间件canal订阅binlog日志提取需要删除的key，然后另写一段非业务代码去获取key并尝试删除，若删除失败就把删除失败的key发送到消息队列，然后进行删除重试。

## 12.Redis内存淘汰策略

noeviction（默认策略）：不淘汰数据，返回错误OOM，只能读取不能写入

volatile-ttl：针对设置了过期时间的key，淘汰最早过期的

volatile-random: 针对设置了过期时间的key，随机淘汰

volatile-lru：针对设置了过期时间的key，淘汰最近最少访问的

volatile-lfu：针对设置了过期时间的key，淘汰访问频率最低的

allkeys-random: 从所有key中随机删除

allkeys-lru：从所有key中淘汰最近最少访问的

allkeys-lfu：从所有key中淘汰访问频率最低的

# 八.计算机网络系列

## 1. OSI七层模型及TCP/IP网络四层模型

1.物理层：在物理媒体上为数据端设备透明地传递原始比特流。传输单位是比特。

2.数据链路层：功能可以概括为成帧、差错控制、流量控制和传输管理等。传输单位是帧。

3.网络层：将网络层的协议数据单元（分组）从源端传到目的端，为分组交换网上的不同主机提供通信服务。对分组进行路由选择，并实现流量控制、阻塞控制、差错控制和国际互联等功能。传输单位是数据报。

常用协议：

- ARP:根据IP地址获取对应的MAC地址，通过一问一答实现交互，问是通过广播形式实现，答是通过单播形式。

- ICMP互联网控制报文协议:在ip网络设备之间发送控制报文，传递差错、控制、查询等信息。可根据ICMP查看要发送的数据是否成功送达。若未成功送达的话，可查看发送失败的原因

- DHCP动态主机配置协议:给主机动态的分配ip地址，无需人为参与

- IP协议:寻址和路由，在IP 数据包中会携带源 IP 地址和目的 IP 地址来标识该数据包的源主机和目的主机。IP 数据报在传输过程中，每个中间节点（IP 网关、路由器）只根据网络地址进行转发，如果中间节点是路由器，则路由器会根据路由表选择合适的路径。IP 协议根据路由选择协议提供的路由信息对 IP 数据报进行转发，直至抵达目的主机。

4.传输层：为端到端提供可靠的传输服务，为端到端连接提供流量控制、差错控制、服务质量、数据传输管理等服务。传输单位是报文段（TCP)或者用户数据报（UDP)。

- TCP:可靠的、面向连接、传输效率低

- UDP:不可靠的、无连接、传输效率高

5.会话层：负责管理主机间的通话进程，包括建立、管理以及终止进程间的通话。

6.表示层：主要处理在两个通信系统中交换信息的表示方法。

7.应用层：为特定类型的网络应用提供访问OSI参考模型环境的手段。传输单位是报文。

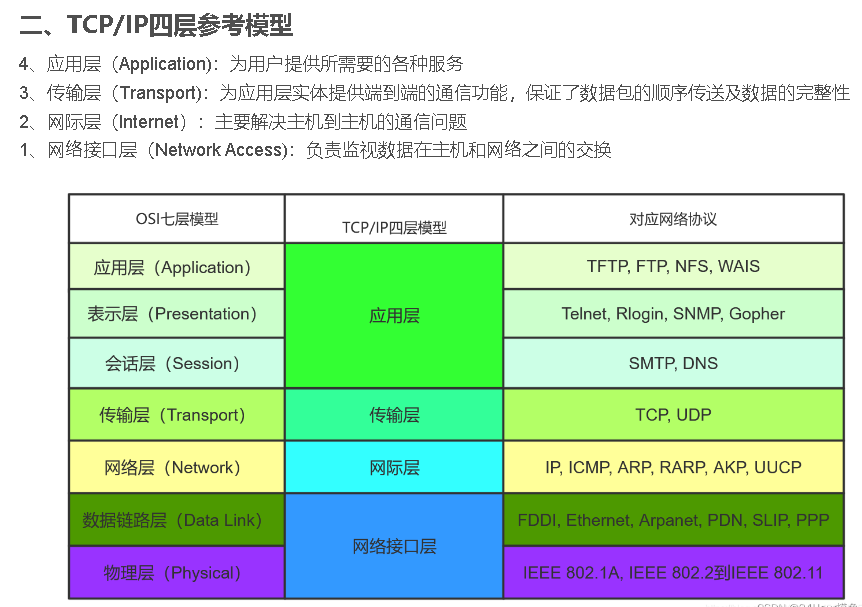
- DNS:将域名转换为ip地址

- FTP:用于ftp客户端和ftp服务器之间进行文本、文件传输

- SMTP: 简单邮件传输协议,提供可靠且有效的电子邮件传输的协议

- HTTP:规定了浏览器和服务器之间请求和响应的格式与规则

- HTTPS:在Http和Tcp之间加入SSL/TLS协议，对传输信息进行加密

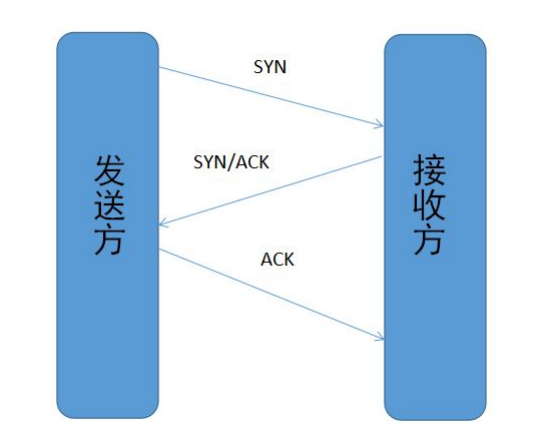


## 2.浏览器输入地址后做了什么？



## 3.TCP三次握手

三次握手：服务端新建套接字(是对网络中不同主机上的应用进程之间进行双向通信的端点的抽象，是应用程序通过网络协议进行通信的接口，是应用程序与网络协议根进行交互的接口 。)，绑定地址信息后开始监听，进入LISTEN状态。客户端新建套接字绑定地址信息后调用connect，发送连接请求SYN（建立联机），并进入SYN\_SENT状态，等待服务器的确认。服务端一旦监听到连接请求，就会将连接放入内核等待队列中，并向客户端发送SYN和确认报文段ACK，进入SYN\_RECD状态。客户端收到SYN+ACK报文后向服务端发送确认报文段ACK，并进入ESTABLISHED状态，开始读写数据。服务端一旦收到客户端的确认报文，就进入ESTABLISHED状态，就可以进行读写数据了

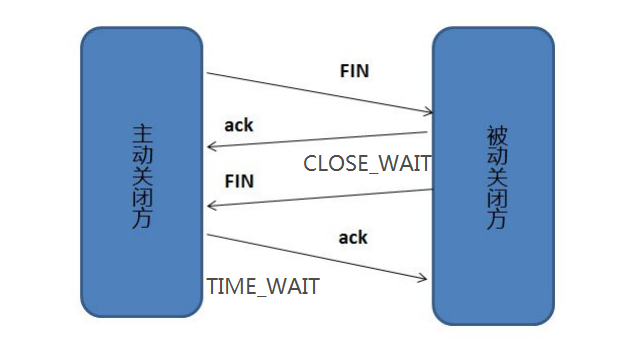


## 4.为什么TCP不能两次握手

两次不安全，四次没必要。tcp通信需要确保双方都具有数据收发的能力，得到ACK响应则认为对方具有数据收发的能力，因此双方都要发送SYN确保对方具有通信的能力。第一次握手是客户端发送SYN，服务端接收，服务端得出客户端的发送能力和服务端的接收能力都正常；第二次握手是服务端发送SYN+ACK，客户端接收，客户端得出客户端发送接收能力正常，服务端发送接收能力也都正常，但是此时服务器并不能确认客户端的接收能力是否正常；第三次握手客户端发送ACK，服务器接收，服务端才能得出客户端发送接收能力正常，服务端自己发送接收能力也都正常。

## 5.TCP四次挥手

客户端主动调用close时，向服务端发送结束报文段FIN报，同时进入FIN\_WAIT1状态；服务器会收到结束报文段FIN报，服务器返回确认报文段ACK并进入CLOSE\_WAIT状态，此时如果服务端有数据要发送的话，客户端依然需要接收。客户端收到服务器对结束报文段的确认，就会进入到FIN\_WAIT2状态，开始等待服务器的结束报文段；服务器端数据发送完毕后，当服务器真正调用close关闭连接时，会向客户端发送结束报文段FIN包，此时服务器进入LAST\_ACK状态，等待最后一个ACK的到来；客户端收到服务器发来的结束报文段, 进入TIME\_WAIT, 并发出送确认报文段ACK；服务器收到了对结束报文段确认的ACK，进入CLOSED状态，断开连接。而客户端要等待2MSL（MSL指的是报文在网络中最大生存时间。在客户端发送对服务端的FIN确认包ACK后，这个ACK包有可能到达不了，服务器端如果接收不到ACK包就会重新发送FIN包。所以客户端发送ACK后需要留出2MSL时间（ACK到达服务器器+服务器发送FIN重传包，一来一回）等待确认服务器端缺失收到了ACK包。也就是说客户端如果等待2MSL时间也没收到服务器端重传的FIN包，则就可以确认服务器已经收到客户端发送的ACK包）的时间，才会进入到CLOSED状态



## 6.为什么要进入时间等待状态？

若客户端发送确认释放包后直接关闭，而服务端因为某种原因没有收到客户端的确认释放包，就会一直发送确认请求，而客户端永远不会再响应该请求。

## 7.TCP 滑动窗口

**在没有接收到上次发送数据的ACK,还可以发送下一个数据报,可以通过后面的ACK来确认前面发送的数据报也称为累计确认.窗口是将等待ACK的时间重叠在一起,可以一次发送多条数据,就大大的提高性能.**

窗口大小是由接收方告诉发送方自己还有多少缓冲空间可以接收数据,以至于发送方根据接收方的处理能力控制窗口大小来发送数据,防止接收方处理不过来.所以,窗口大小是由接收方窗口大小确定的. 如果可用窗口大小为0,那证明发送方需要等待收到确认应答之后才能发送数据,否则不能发送数据.

**中途ACK丢包可以通过下一确认应答进行确认.**

**如果最后一个ACK丢失,那就是滑动窗口机制最坏的情况下,也就触发超时重传.**

**当某个报文段丢失,窗口大小又比较大的情况下,接收方在没有收到自己预期的序号,会对上一次收到的数据进行确认应答,如果接收方连续三次收到重复的确认应答,那么就需要发送方重传数据包,这样的机制也叫做快速重传机制**

## 8.TCP拥塞控制

**对网络中某一资源的需求超过了该资源所能提供的可用部分，就叫做网络拥塞。**

发送方会维护一个拥塞窗口大小的状态变量，大小取决于网络的拥塞程度。发送方的发送窗口大小是取接收方接收窗口和拥塞窗口中较小的一个

拥塞控制有四种算法：

慢开始：从小到大主键发送窗口，每收到一个确认报文窗口大小指数增长

拥塞避免：当窗口大小到达一定阈值时，转为拥塞避免，每收到一个确认报文窗口大小+1。若此时网络超时，就把阈值调小一半，重新慢开始

快重传：要求接收方收到请求后要立即回复

快恢复：发送方连续收到多个确认时，就把拥塞避免阈值减小，然后直接开始拥塞避免

## 9.TCP超时重传

发送方在发送按数据后一定时间(超时重传时间)内没有收到接收方响应报文，就会重新发送刚刚的报文，接收方收到报文后会对该报文的序列号进行检验，已存在就抛弃

## 10. TCP和UDP有什么区别



### TCP协议保证数据传输可靠性的方式主要有：

**1.大小控制**

应用数据被分割成 TCP 认为最适合发送的数据块。

**2.序列号**

TCP 给发送的每一个包进行编号，接收方对数据包进行排序，把有序数据传送给应用层。

TCP 的接收端会丢弃重复的数据，数据是否重复就是根据序列号进行判断的。

**3.校验和**

TCP 将保持它首部和数据的检验和。

这是一个端到端的检验和，目的是检测数据在传输过程中的任何变化。如果收到段的检验和有差错，TCP 将丢弃这个报文段和不确认收到此报文段。

**4.流量控制**

TCP 连接的每一方都有固定大小的缓冲空间，TCP的接收端只允许发送端发送接收端缓冲区能接纳的数据。当接收方来不及处理发送方的数据，能提示发送方降低发送的速率，防止包丢失。TCP 使用的流量控制协议是可变大小的滑动窗口协议。 （TCP 利用滑动窗口实现流量控制）

**5.拥塞控制**

当网络拥塞时，减少数据的发送。

**6.确认应答**

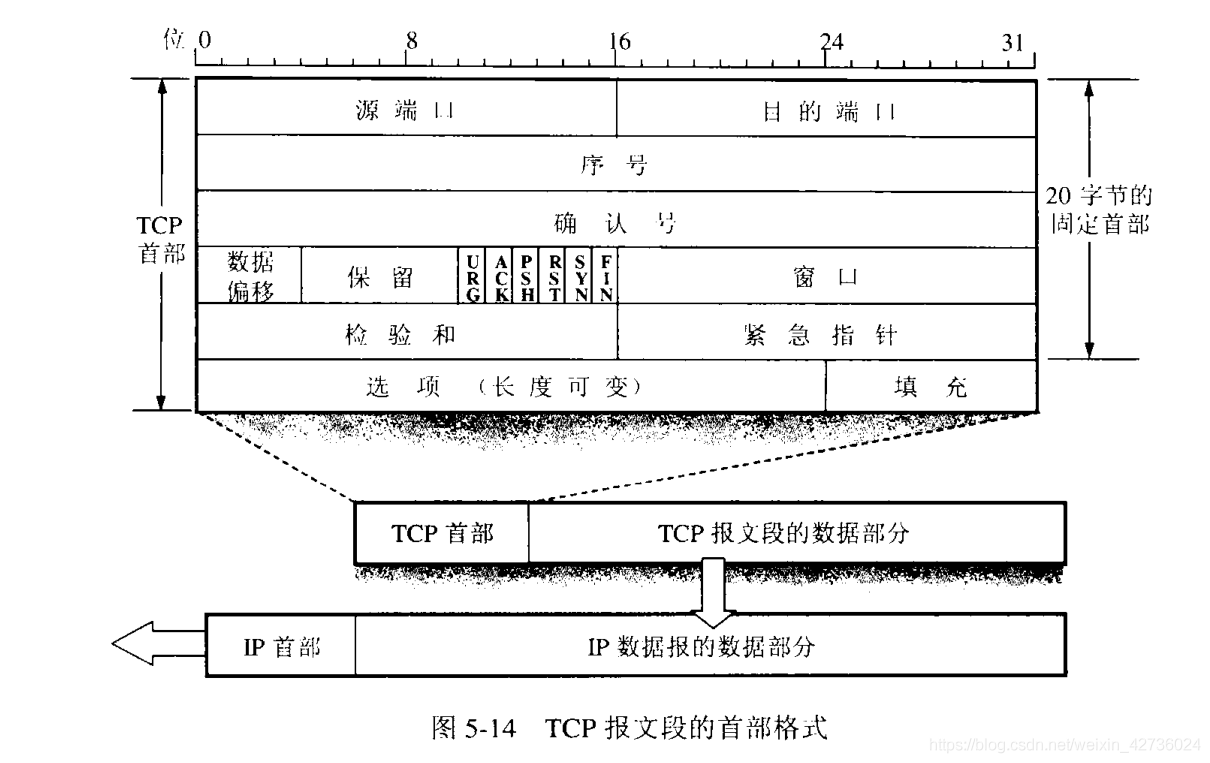
是通过ARQ协议实现的

基本原理是每发完一个分组就停止发送，等待对方确认。在收到确认后再发下一个分组。

**7.超时重传**

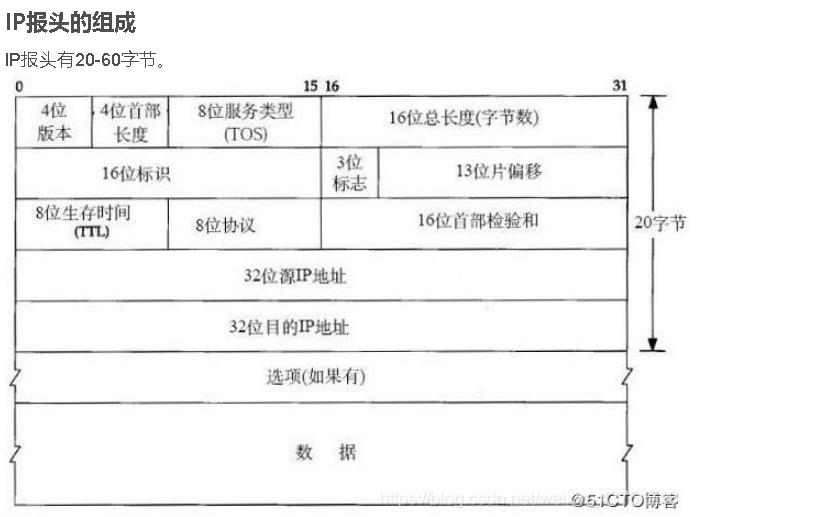
当 TCP 发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段。

## 11.TCP、UDP、IP报头组成



1. **源端口**：源端口和IP地址的作用是标识报文的返回地址。
2. **目的端口**：端口指明接收方计算机上的应用程序接口。
3. **序号**：是TCP可靠传输的关键部分。序号是本报文段发送的数据组的第一个字节的序号。在TCP传送的流中，每一个字节一个序号。e.g.一个报文段的序号为300，此报文段数据部分共有100字节，则下一个报文段的序号为400。所以序号确保了TCP传输的有序性。
4. **确认号**，即ack：是TCP可靠传输的关键部分。确认号指明下一个期待收到的字节序号，表明该序号之前的所有数据已经正确无误的收到。确认号只有当ACK标志为1时才有效。比如建立连接时，SYN报文的ACK标志位为0。
5. **数据偏移／首部长度**：4bits。由于首部可能含有可选项内容，因此TCP报头的长度是不确定的，报头不包含任何任选字段则长度为20字节，4位首部长度字段所能表示的最大值为1111，转化为10进制为15，15\*32/8 = 60，故报头最大长度为60字节。首部长度也叫数据偏移，是因为首部长度实际上指示了数据区在报文段中的起始偏移值。
6. **保留**：为将来定义新的用途保留，现在一般置0。
7. **控制位**：URG ACK PSH RST SYN FIN，共6个，每一个标志位表示一个控制功能。
8. **URG**：紧急指针标志，为1时表示紧急指针有效，为0则忽略紧急指针。
9. **ACK**：确认序号标志，为1时表示确认号有效，为0表示报文中不含确认信息，忽略确认号字段。
10. **PSH**：push标志，为1表示是带有push标志的数据，指示接收方在接收到该报文段以后，应尽快将这个报文段交给应用程序，而不是在缓冲区排队。
11. **RST**：重置连接标志，用于重置由于主机崩溃或其他原因而出现错误的连接。或者用于拒绝非法的报文段和拒绝连接请求。
12. **SYN**：同步序号，用于建立连接过程，在连接请求中，SYN=1和ACK=0表示该数据段没有使用捎带的确认域，而连接应答捎带一个确认，即SYN=1和ACK=1。
13. **FIN**：finish标志，用于释放连接，为1时表示发送方已经没有数据发送了，即关闭本方数据流。
14. **窗口**：滑动窗口大小，用来告知发送端接受端的缓存大小，以此控制发送端发送数据的速率，从而达到流量控制。窗口大小时一个16bit字段，因而窗口大小最大为65535。
15. **校验和**：奇偶校验，此校验和是对整个的 TCP 报文段，包括 TCP 头部和 TCP 数据，以 16 位字进行计算所得。由发送端计算和存储，并由接收端进行验证。
16. **紧急指针**：只有当 URG 标志置 1 时紧急指针才有效。紧急指针是一个正的偏移量，和顺序号字段中的值相加表示紧急数据最后一个字节的序号。 TCP 的紧急方式是发送端向另一端发送紧急数据的一种方式。
17. **选项和填充**：最常见的可选字段是最长报文大小，又称为MSS（Maximum Segment Size），每个连接方通常都在通信的第一个报文段（为建立连接而设置SYN标志为1的那个段）中指明这个选项，它表示本端所能接受的最大报文段的长度。选项长度不一定是32位的整数倍，所以要加填充位，即在这个字段中加入额外的零，以保证TCP头是32的整数倍。
18. **数据部分**： TCP 报文段中的数据部分是可选的。在一个连接建立和一个连接终止时，双方交换的报文段仅有 TCP 首部。如果一方没有数据要发送，也使用没有任何数据的首部来确认收到的数据。在处理超时的许多情况中，也会发送不带任何数据的报文段。





1. **版本号（Version）**：长度为4位（bit），IP v4的值为0100，IP v6的值为0110。
2. **首部长度**：指的是IP包头长度，用4位（bit）表示，十进制值就是[0,15]，一个IP包前20个字节是必有的，后40个字节根据情况可能有可能没有。如果IP包头是20个字节，则该位应是20/4=5。
3. **服务类型（Type of Service TOS）**：长度为8位（bit），其组成：前3位为优先级（Precedence），后4位标志位，最后1位保留未用。优先级主要用于QoS，表示从0（普通级别）到7（网络控制分组）的优先级。标志位可分别表示D（Delay更低的时延）、T（Throughput 更高的吞吐量）、R（Reliability更高的可靠性）、C（Cost 更低费用的路由）。TOS只表示用户的请求，不具有强制性，实际应用中很少用，路由器通常忽略TOS字段。
4. **总长度（Total Length）**：指IP包总长度，用16位（bit）表示，即IP包最大长度可以达216=65535字节。在以太网中允许的最大包长为1500B，当超过网络允许的最大长度时需将过长的数据包分片。
5. **标识符（Identifier）**：长度为16位，用于数据包在分段重组时标识其序列号。将数据分段后，打包成IP包，IP包因走的路由上不同，会产生不同的到达目地的时间，到达目地的后再根据标识符进行重新组装还原。该字段要与标志、段偏移一起使用的才能达到分段组装的目标。
6. **标志（Flags）**：长度为3位，三位从左到右分别是MF、DF、未用。MF=1表示后面还有分段的数据包，MF=0表示没有更多分片（即最后一个分片）。DF=1表示路由器不能对该数据包分段，DF=0表示数据包可以被分段。
7. **偏移量（Fragment Offset）**：也称段偏移，用于标识该数据段在上层初始数据报文中的偏移量。如果某个包含分段的上层报文的IP数据包在传送时丢失，则整个一系列包含分段的上层数据包的IP包都会要求重传。
8. **生存时间（TTL）**：长度为8位，初始值由操作系统设置，每经过一个路由器转发后其值就减1，减至0后丢弃该包。这种机制可以避免数据包找不到目地时不断被转发，堵塞网络。
9. **协议（Protocol）**：长度为8位，标识上层所使用的协议。
10. **首部校验和（Header Checksum）**：长度为16位，首部检验和只对IP数据包首部进行校验，不包含数据部分。数据包每经过一个中间节点都要重新计算首部校验和，对首都进行检验。
11. **源IP地址（Source IP）**：长度为32位，表示数据发送的主机IP。
12. **目的IP地址（Destination IP）**：长度为32位，表示数据要接收的主机IP。
13. **选项字段（Options）**：长度为0-40字节（Byte），主要有：安全和处理限制（Security）、记录路径（Record Route）、时间戳（Timestamps）、宽松源站选路（Loose Source Routing）、严格的源站选路（Strict Source Routing）等。

## 12.状态码

1xx:请求正在处理

2xx:请求成功处理

3xx:请求重定向 301：永久重定向 302：临时重定向 304：使用本地缓存

4xx:客户端错误 400：请求格式错误 403：没有访问权限 415：请求体过大

5xx:服务端错误

## 13.http与https的区别

1、HTTPS 协议需要到 CA （Certificate Authority，证书颁发机构）申请证书，一般免费证书较少，因而需要一定费用。(以前的网易官网是http，现在是 https )

2、HTTP 是超文本传输协议，信息是明文传输，HTTPS 则是具有安全性的 SSL 加密传输协议。

3、HTTP 和 HTTPS 使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、HTTP 的连接很简单，是无状态的。HTTPS 协议是由 SSL+HTTP 协议构建的可进行加密传输、身份认证的网络协议，比 HTTP 协议安全。(无状态的意思是其数据包的发送、传输和接收都是相互独立的。无连接的意思是指通信双方都不长久的维持对方的任何信息。)

## 14.socket通信流程

（1）服务端创建socket并调用bind()方法绑定ip和端口号

（2）服务端调用listen()方法建立监听，此时服务的scoket还没有打开

（3）客户端创建socket并调用connect()方法像服务端请求连接

（4）服务端socket监听到客户端请求后，被动打开，调用accept()方法接收客户端连接请求，当accept()方法接收到客户端connect()方法返回的响应成功的信息后，连接成功

（5）客户端向socket写入请求信息，服务端读取信息

（6）客户端调用close()结束链接，服务端监听到释放连接请求后，也结束链接

## 15. 域名解析过程

1. 主机先查浏览器缓存；

2. 没有的话去查本地hosts文件；

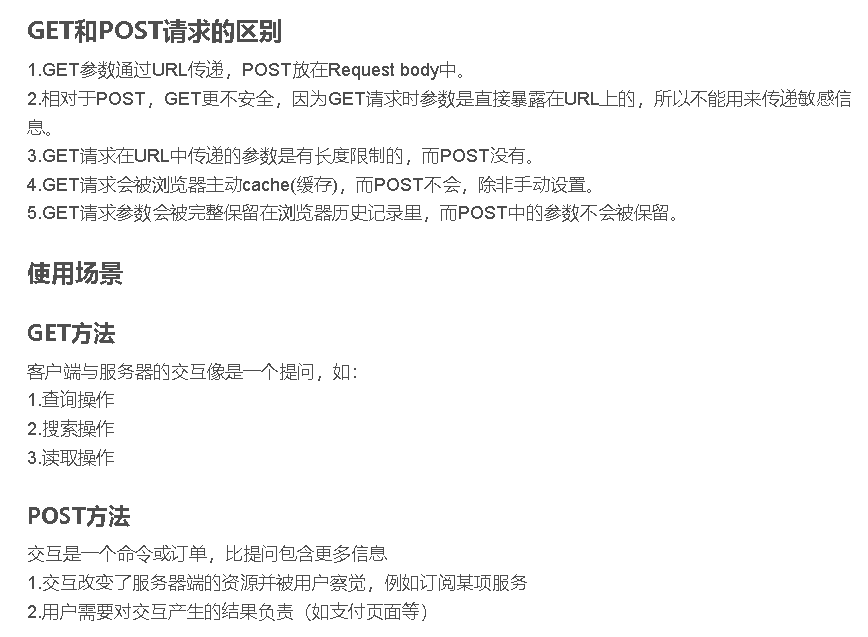
3. 然后去查本地DNS服务器，DNS服务器的缓存。（递归解析：主机先向本地DNS服务器查询，接下里的查询工作都交给本地DNS服务器，本地DNS服务器帮忙找到结果返回给主机）；

4. 本地DNS服务器采用迭代解析（本地DNS服务器向一个根域名服务器查询，根域名服务器没有，然后告诉本地DNS服务器接下来应该去查询的顶级域名服务器的IP，本地域名服务器根据这个IP继续向上查询，仍采用迭代的方式）。

递归解析：找别人帮忙干活，自己只要结果；

迭代解析：全程都要自己干，中途会有人告诉咋干。

## 16. GET 和 POST的区别



# 九.linux系列

## 1.linux常用命令

ifconfig:查看网络接口详情

ping：查看与某主机是否能联通

ps -ef|grep 进程名称：查看进程号

ps -ef ：查看进程使用标准语法

ps -aux 查询内存中进程信息

lost -i 端口 ：查看端口占用情况

top:查看系统负载情况，包括系统时间、系统所有进程状态（动态显示内存中的进程信息）、cpu情况

free:查看内存占用情况

kill:正常杀死进程，发出的信号可能会被阻塞

kill -9:强制杀死进程，发送的是exit命令，不会被阻塞

wc -l app.log 获取文件中行数

wc -w app.log 获取文件中单词数

wc -c app.log 获取文件中字节

chmod 改变文件或目录权限

执行命令 chmod 755，设置用户的权限为：

⑴ 设置文件所有者的权限为 7，就是“可读可写可执行”

⑵ 设置文件所属用户组的权限为 5，即设置与文件所有者同属一个用户组的其他用户的权限为 5，就是“可读可执行”

⑶ 设置其它用户组的权限为 5，也是“可读可执行”

## 2.l的io模型

IO是对磁盘或网络数据的读写，用户进程读取一次IO请求分为两个阶段：等待数据到达内核缓冲区和将内核缓冲区数据拷贝到用户空间，当用户去内核中拷贝数据时，要从用户态转为核心态

5种io模型:

(1)同步阻塞IO模型

用户进程发起io调用后会被阻塞，等待内核缓冲区数据准备完毕时就被唤醒，将内核数据复制到用户进程。这两个阶段都是阻塞的

(2)同步非阻塞IO模型

用户进程发起IO调用后，若内核缓冲区数据还未准备好，进程会继续干别的事，每隔一段时间就去看看内核数据是否准备好。不过将内核数据复制到用户进程这个阶段依旧是阻塞的

(3)IO多路复用模型

linux中把一切都看成文件，每个文件都有一个文件描述符（FD）来关联， IO多路复用模型就是复用单个进程同时监测多个文件描述符，当某个文件描述符可读或可写，就去通知用户进程。

(4)信号IO模型

用户进程发起IO调用后，会向内核注册一个信号处理函数然后继续干别的事，当内核数据准备就绪时就通知用户进程来进行拷贝。

（5）异步非阻塞模型

前面四种全是同步的。进程在发起IO调用后，会直接返回结果。待内核数据准备好时，由内核将数据复制给用户进程。两个阶段都是非阻塞的

## 4.IO多路复用详解

linux中把一切都看成文件，每个文件都有一个文件描述符（FD）来关联， IO多路复用模型就是复用单个进程同时监测多个文件描述符，当某个文件描述符可读或可写，就去通知用户进程。IO多路复用有三种方式

（1）select：采用数组结构，监测的fd有限，默认为1024；当有文件描述符就绪时，需要遍历整个FD数组来查看是哪个文件描述符就绪了，效率较低；每次调用select时都需要把整个文件描述符数组从用户态拷贝到内核态中来回拷贝，当fd很多时开销会很大；

（2）poll：采用链表结构，监测的文件描述符没有上限，其它的根select差不多

（3）epoll：采用红黑树结构，监测的fd没有上限，它有三个方法，epoll\_create() 用于创建一个epoll实例，epoll实例中有一颗红黑树记录监测的fd，一个链表记录就绪的fd;epoll\_ctl() 用于往epoll实例中增删要监测的文件描述符，并设置回调函数，当文件描述符就绪时触发回调函数将文件描述符添加到就绪链表当中；epoll\_wait() 用于见擦汗就绪列表并返回就绪列表的长度，然后将就绪列表的拷贝到用户空间缓冲区中。

所以epoll的优点是当有文件描述符就绪时，只把已就绪的文件描述符写给用户空间，不需要每次都遍历FD集合；每个FD只有在调新增的时候和就绪的时候才会在用户空间和内核空间之间拷贝一次。

## 5.epoll的LT和ET模式

LT（默认）：水平触发，当FD有数据可读的时候，那么每次 epoll\_wait都会去通知用户来操作直到读完

ET：边缘触发，当FD有数据可读的时候，它只会通知用户一次，直到下次再有数据流入才会再通知，所以在ET模式下一定要把缓冲区的数据一次读完

# 十.场景题

## 1.Java如何实现统计在线人数的功能？

java监听器

## 2.电商网站可以分成哪些模块（或订单模块要完成哪些功能）？

用户模块（用户账户、会员等级、收货信息）、订单模块（订单编号、类型信息、状态信息、时间信息等）、商品模块（店铺信息、数量、价格等）、支付模块（支付方式、支付时间、支付单号等）、物流模块（物流公司、物流单号、物流状态等）

# 其他问题：

## 印象最深的问题？

**外卖项目**

**问题描述**：在开发“启用/禁用员工账号”功能模块时，在Controller中创建一个通用的修改员工信息的update方法，启用、禁用员工账号，本质上就是一个更新操作，也就是对status状态字段进行操作。开发完毕后，测试过程中没有报错，但是功能并没有实现，查看数据库中的数据也没有变化。观察控制台输出的SQL发现SQL执行的结果是更新的数据行数为0，仔细观察update语句中id的值，和数据库中对应记录的id值并不相同，多次尝试后发现id末位总是为0，于是想到是否为精度问题，通过F12工具发现分页查询时服务端响应给页面的数据中id的值为19位数字，类型为long，而页面中js处理long型数字只能精确到前16位，所以最终通过ajax请求提交给服务端的时候id的最后几位被自动调整为0.

**解决方法**：可以在服务端给页面响应json数据时进行处理，将long型数据统一转为String字符串。

1. 使用对象转换器JacksonObjectMapper，基于Jackson进行Java对象到json数据的转换

2. 在WebMvcConfig配置类中扩展Spring mvc的消息转换器，将Jackson转换器追加到mvc框架的转换器集合中，并设置优先使用

## 2.什么是前后端分离？有哪些优点？

后端只负责提供数据接口，不再渲染模板，前端获取数据并呈现

1.前后端解耦，接口复用（前端和客户端公用接口），减少开发量

2.各司其职，前后端同步开发，提升工作效率。定义好接口规范。

3.更有利于调度（mock）、测试和运维部署

4.缺点是对于单页面不好做seo（搜索引擎优化）

## 3.什么是RESTful

Restful是一种资源定位及资源操作的风格，其实就是一套编写接口的“协议”，协议规定如何编写以及如何设置返回值、状态码等信息。

最显著的特点：restful给用户一个url，根据method不同在后端做不同的处理，比如：post 创建数据、get获取数据、put和patch修改数据、delete删除数据。基于这个风格设计的软件可以更简洁，更有层次，更易于实现缓存等机制。用URL定位资源，用HTTP动词（GET，POST，DELETE，PUT）描述操作

## 4.提升qps的操作

（1）页面动静分离，静态页面缓存到redis

（2）分布式锁拦截不同用户的重复

（3）限流算法

（4）验证码限流

（5）rabbitMq流量削峰

（6）接口隐藏

5. 你在京东集团的工作经历中，最具挑战性的任务是什么？你是如何解决的？

6. 你在外卖点餐项目中的职责是什么？你是如何使用SpringCache框架和Redis数据库完成缓存优化的？

7. 你提到了了解机器学习算法，能否举例说明你在实际项目中如何应用这些算法？

8. 为何转行来计算机专业

人才缺口大，发展空间广阔一些，待遇还不错，最重要的是你对技术本身很感兴趣。

进入这个行业我如何让自己变得专业呢？一是因为我的兴趣，第二呢在这个行业当中正好也有我的朋友，我在学习技术的过程当中朋友也给予了很多技术以及经验上的支持，让我的技术得以快速的提升，让我有了从事本行业的一个资本，同时未来呢，我也希望我能在这个行业走的更长更远，我也希望我的能力能给咱们的公司或者团队带来更多的价值。

# 正则表达式

## 1、什么是正则表达式，有什么用？

正则表达式：RegularExpression，正则表达式主要用在字符串格式匹配方面。

## 2、常见的正则表达式符号？

.匹配除换行符以外的任意字符

\w匹配字母或数字或下划线或汉字

\s匹配任意的空白符

\d匹配数字

\b匹配单词的开始或结束

^匹配字符串的开始

$匹配字符串的结束

\*重复零次或更多次

+重复一次或更多次

?重复零次或一次

{n}重复n次

{n,}重复n次或更多次

{n,m}重复n到m次

\W匹配任意不是字母，数字，下划线，汉字的字符

\S匹配任意不是空白符的字符

\D匹配任意非数字的字符

\B匹配不是单词开头或结束的位置

[^x]匹配除了x以外的任意字符

[^aeiou]匹配除了aeiou这几个字母以外的任意字符

正则表达式当中的小括号()优先级较高。

[1-9]表示1到9的任意1个数字（次数是1次。）

[A-Za-z0-9]表示A-Za-z0-9中的任意1个字符

[A-Za-z0-9-]表示A-Z、a-z、0-9、-，以上所有字符中的任意1个字符。

|表示或者

QQ号的正则表达式：^[1-9][0-9]{4,}$