1. 三次握手，四次挥手，四次挥手的close_wait状态发生在什么时候

三次握手：

第三次握手是为了防止失效的连接请求到达服务器，让服务器错误打开连接。

客户端发送的连接请求如果在网络中滞留，那么就会隔很长一段时间才能收到服务器端发回的连接确认。客户端等待一个超时重传时间之后，就会重新请求连接。但是这个滞留的连接请求最后还是会到达服务器，如果不进行三次握手，那么服务器就会打开两个连接。如果有第三次握手，客户端会忽略服务器之后发送的对滞留连接请求的连接确认，不进行第三次握手，因此就不会再次打开连接。

四次挥手：

客户端接收到服务器端的 FIN 报文后进入此状态，此时并不是直接进入 CLOSED 状态，还需要等待一个时间计时器设置的时间 2MSL **（time-wait）**。这么做有两个理由：

• 确保最后一个确认报文能够到达。如果 B 没收到 A 发送来的确认报文，那么就会重新发送连接释放请求报文，A 等待一段时间就是为了处理这种情况的发生。

• 等待一段时间是为了让本连接持续时间内所产生的所有报文都从网络中消失，使得下一个新的连接不会出现旧的连接请求报文。

close-wait是第二次挥手报文和第三次挥手报文之间的一个时间段，服务器尽可能把未发送完的数据发送出去。

2.tcp/ip网络协议，对应的七层网络分别是，路由在第几层

各层作用及协议

分层	作用	协议
物理层	通过媒介传输比特，确定机械及电气规范（比特 Bit）	RJ45、CLOCK、IEEE802.3（中继器，集线器）
数据链路层	将比特组装成帧和点到点的传递（帧 Frame）	PPP、FR、HDLC、VLAN、MAC（网桥，交换机）
网络层	负责数据包从源到宿的传递和网际互连（包 Packet）	IP、ICMP、ARP、RARP、OSPF、IPX、RIP、IGRP（路由器）
运输层	提供端到端的可靠报文传递和错误恢复（段Segment）	TCP、UDP、SPX
会话层	建立、管理和终止会话（会话协议数据单元 SPDU）	NFS、SQL、NETBIOS、RPC
表示层	对数据进行翻译、加密和压缩（表示协议数据单元 PPDU）	JPEG、MPEG、ASII
应用层	允许访问OSI环境的手段（应用协议数据单元 APDU）	FTP、DNS、Telnet、SMTP、HTTP、WWW、NFS

路由在网路层

3.tcp/udp区别及上层协议

用户数据报协议 UDP（User Datagram Protocol）是无连接的，尽最大可能交付，没有拥塞控制，面向报文（对于应用程序传下来的报文不合并也不拆分，只是添加 UDP 首部），支持一对一、一对多、多对一和多对多的交互通信。

传输控制协议 TCP（Transmission Control Protocol）是面向连接的，提供可靠交付，有流量控制，拥塞控制，提供全双工通信，面向字节流（把应用层传下来的报文看成字节流，把字节流组织成大小不等的数据块），每一条 TCP 连接只能是点对点的（一对一）。

TCP应用场景：

效率要求相对低，但对准确性要求相对高的场景。因为传输中需要对数据确认、重发、排序等操作，相比之下效率没有UDP高。举几个例子：文件传输（准确高要求高、但是速度可以相对慢）、接受邮件、远程登录。

UDP应用场景：

效率要求相对高，对准确性要求相对低的场景。举几个例子：QQ聊天、在线视频、网络语音电话（即时通讯，速度要求高，但是出现偶尔断续不是太大问题，并且此处完全不可以使用重发机制）、广播通信（广播、多播）。

4.jvm内存模型，五个区主要做什么

堆：存放new的实体对象。

栈：存放线程的相关信息。里面存放栈帧，栈帧包括局部变量、操作数栈、动态链接、方法出口。

程序计数器：记录每个线程的执行情况，执行到第几行代码.

本地方法栈：存放本地方法的局部变量和操作数栈。native修饰的。

方法区：存放类信息常量池、字符串常量池和运行时常量池。

5.jvm调优实战：如果运行程序时电脑很卡，怎么找原因，有没有自己去调过，用哪些（top/jheap等都是什么意思）

1. **jstack 可查看java进程里的存活线程状态**
2. **jmap 可查看java进程堆内存使用情况**
3. **top linux 命令，可查看java某进程的线程**

6.hashmap，cocurrentHashMap区别，cocurrentHashMap结构组成，这个结构对应的jdk版本是多少

hashmap: 数组+链表/红黑树，会出现线程不安全的情况。

correntHashMap: (1.7) segment数组+hashmap，对于每个segment加reentrantlock，一个segment对应一个hashmap，线程安全。

(1.8) 结构和hashmap相似，只不过每个table元素会加上synchronized来满足线程同步，通过cas来修改元素内容。

7.hashMap在几的时候会转成红黑树，几的时候又转回来，为什么这么设置？为什么是线程不安全的，初始大小，扩容可能会出现什么问题？ （扩容可能出现死循环）

当链表长度>=8转红黑树，当红黑树元素<=6转为链表。

根据泊松分布，在负载因子默认为0.75的时候，单个hash槽内元素个数为8的概率小于百万分之一。

初始大小为16，扩容容易死锁，线程不安全因为可能会丢失修改（两个线程同时修改value，后面会把前面的覆盖）。

死锁原因（Java8之前）：多线程扩容导致的死锁

HashMap是非线程安全，死锁一般都是产生于并发情况下。我们假设有二个进程T1、T2，HashMap容量为2,T1线程放入key A、B、C、D、E。在T1线程中A、B、C Hash值相同，于是形成一个链接，假设为A->C->B，而D、E Hash值不同，于是容量不足，需要新建一个更大尺寸的hash表，然后把数据从老的Hash表中迁移到新的Hash表中(refresh)。这时T2进程闯进来了，T1暂时挂起，T2进程也准备放入新的key，这时也发现容量不足，也refresh一把。refresh之后原来的链表结构假设为**C->A（头插法）**，之后T1进程继续执行，链接结构为A->C,这时就形成A.next=C,C.next=A的环形链表。一旦取值进入这个环形链表就会陷入死循环。

JAVA8取消了头插法解决了死锁，并且也不用rehash来扩容。方法如下：首先假设扩容前容量为8，其中的一个链表假设为A->B->C->D，用四个指针来直接完成迁移扩容，先计算ABCD中的部分hashcode && 8是否为1，如果不为1则代表为低位元素，扩容后还是在指定的key处，可以通过低位Low指针和低位high指针直接迁移过去；如果为1则代表为高位元素，扩容后放到key + 8的位置处，可以通过高位Low指针和高位high指针迁移至key+8处。

8.算法题：判断一个字符串是否是回文字符串

```
public static boolean isHuiWen(String text) {
        int length = text.length();
        for (int i = 0; i < length / 2; i++) {
            if (text.toCharArray()[i] != text.toCharArray()[length - i - 1]) {
                return false;
            }
        }
        return true;
    }
```

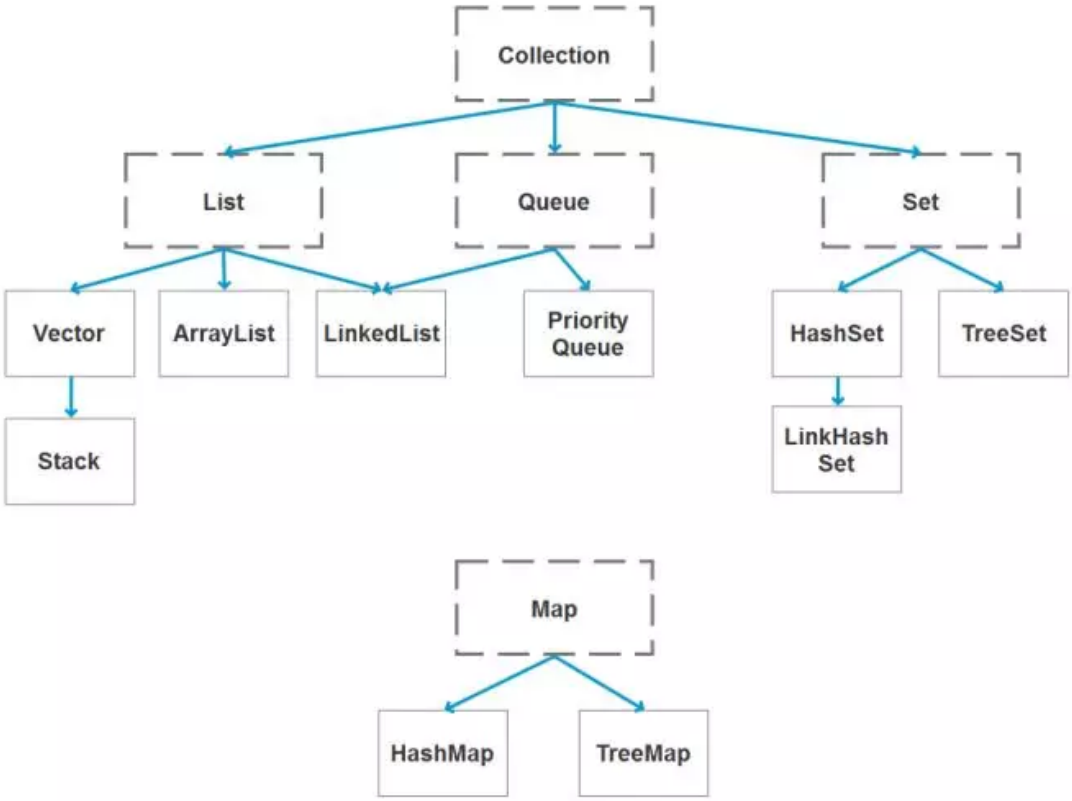
9.说一下常见的算法思想？算法思想在实践中常用

递归与分治思想:折半查找算法

贪心算法:求解最小生成树的普利姆（Prim）算法，求解图的单源最短路径的迪克斯特拉（Dijkstra）算法都是基于贪心算法的思想设计的

穷举法

10.Java 集合有哪些，区别



比较	List	Set	Map
继承接口	Collection	Collection	
常见实现类	AbstractList(其常用子类有ArrayList、LinkedList、Vector)	AbstractSet(其常用子类有HashSet、LinkedHashSet、TreeSet)	HashMap、HashTable
常见方法	add( )、remove( )、clear( )、get( )、contains( )、size( )	add( )、remove( )、clear( )、contains( )、size( )	put( )、get( )、remove( )、clear( )、containsKey( )、containsValue( )、keySet( )、values( )、size( )
元素	可重复	不可重复(用 `equals()` 判断)	不可重复
顺序	有序	无序(实际上由HashCode决定)	
线程安全	Vector线程安全		Hashtable线程安全

11.Java常用package，什么时候会引入这个包

java.lang 该包提供了Java编程的基础类，例如 Object、Math、String、StringBuffer、System、Thread等，不使用该包就很难编写Java代码了。

java.util 该包提供了包含集合框架、遗留的集合类、事件模型、日期和时间实施、国际化和各种实用工具类（字符串标记生成器、随机数生成器和位数组）。

java.io ： 该包通过文件系统、数据流和序列化提供系统的输入与输出。

java.net： 该包提供实现网络应用与开发的类。

java.sql： 该包提供了使用Java语言访问并处理存储在数据源（通常是一个关系型数据库）中的数据API。

12.Java基本类型有哪些

String 不属于基础类型，基础类型有 8 种：byte、boolean、char、short、int、float、long、double，而 String 属于对象。

13.算法题：给定一个数组和x，找这个数组中连续子区间之和小于x的个数（双指针思想）

14.TCP中如果接收缓冲区满了，发送端怎么操作，会报错吗

TCP的发送缓冲区中的数据，如果收不到接收方的ACK就不会删除，导致发送缓冲区溢出。

如果接收方的缓冲区满了：

1. 只要收到了包，就会ACK。
2. TCP在ACK的同时会带有window大小值，表示这边能接受的数据量。发送方会根据这个调整数据量。
3. 接收方缓冲区满时，回给发送方的window值就是0。
4. 发送方看到window为0的包，会启动一个定时器，隔一段时间发一个包试探。
5. 一旦接收方缓冲区有足够空间了，就会给window赋上非0值。发送方就又开始发送了。

15.发送端发送socket阻塞对发送端接收数据有影响吗

客户端的输出流和服务器端的输入流是一对，客户端的输入流和服务器端的输出流又是一对，他们操作的对象是网络文件。在任何一端读取数据时，另一端必须先写数据到网络文件中，否则就会阻塞。

只要接收端没有发送socket阻塞就不会影响其接数据。

16.快排的时间复杂度，什么时候最坏，什么时候最好（为什么这样选择哨兵）

相对有序或者倒序最坏，每次哨兵哨兵调整后正好在中间最好。

哨兵的位置通过三数取中法来确定，而不是只设置为头或者尾，因为最佳的划分是将待排序的序列分成等长的子序列。

举例：待排序序列为：8 1 4 9 6 3 5 2 7 0

左边为：8，右边为0，中间为6.

我们这里取三个数排序后，中间那个数作为枢轴，则枢轴为6

17.算法题：循环升序链表（首位相接），给你链表中的任意一个位置的指针（不一定是最小的）和一个准备插入的值，找到要插入的位置使插入后的链表仍然是循环升序的。

两个指针一起往前走，找到后插在两个指针之间。

18.算法题：最长连续子序列

```
public int lengthOfLIS(int[] nums) {
    int n = nums.length;
    int[] dp = new int[n];
    for (int i = 0; i < n; i++) {
        int max = 1;
        for (int j = 0; j < i; j++) {
            if (nums[i] > nums[j]) {
                max = Math.max(max, dp[j] + 1);
            }
        }
        dp[i] = max;
    }
    int ret = 0;
    for (int i = 0; i < n; i++) {
        ret = Math.max(ret, dp[i]);
    }
    return ret;
}
```

19.死锁产生的条件

四个必要条件：

- 互斥条件：进程对所分配到的资源不允许其他进程进行访问，若其他进程访问该资源，只能等待，直至占有该资源的进程使用完成后释放该资源
- 请求和保持条件：进程获得一定的资源之后，又对其他资源发出请求，但是该资源可能被其他进程占有，此事请求阻塞，但又对自己获得的资源保持不放
- 不可剥夺条件：是指进程已获得的资源，在未完成使用之前，不可被剥夺，只能在使用完后自己释放
- 环路等待条件：是指进程发生死锁后，若干进程之间形成一种头尾相接的循环等待资源关系

20.你在计算机方面的理解（操作系统、计算机组成）

21.写一个查找abc.exe文件的命令，统计当前目录下所有文件名中包含某个字符串的文件一个多少个?

通过文件名查找

find 查找范围 -name 要查找的文件的文件名

```
find /home/test -name emp.data
```

无错误查找

普通用户查找大范围时，会报Permission denied通过2>/dev/null来解决

```
find / -name emp.data 2>/dev/``null
find / -name ``"sqlite*"` `2>/dev/``null
```

查找完后用ctrl+c来结束

根据部分文件名查找

1、查找以emp开头的文件；

```
find / -name ``"emp*"` `2>/dev/``null
```

2、查找以p.data结束的文件；

```
find / -name ``"*p.data"` `2>/dev/``null
```

3、查找包含mp.d的文件。

```
find / -name ``"*mp.d*"` `2>/dev/``null
```

```
# 统计当前文件夹下文件的个数
ls -l | grep "^-" | wc -l

# 统计当前文件夹下目录的个数
ls -l | grep "^d" | wc -l

# 统计当前文件夹下文件的个数，包括子文件夹里的
ls -lR | grep "^-" | wc -l

# 统计文件夹下目录的个数，包括子文件夹里的
ls -lR | grep "^d" | wc -l

# 递归的查看某目录下所有文件及目录数目
find dir/* | wc -l
```

22.算法题：先定义一个双向循环链表，然后写一个函数，输入链表头和要插入的节点，将节点插入到链表头。

23.有一个二叉树，有k层，他的节点数是怎么计算的?

思路：求第k层节点数量，等于求以根节点的左孩子为根的k-1层节点数，加上以根节点的右孩子为根的k-1层节点数。

```
    /**
     * 求二叉树第k层的节点个数
     *
     * @param root
     * @param k
```

```
     * @return
     */
    public int getNodeNumKthLevel(TreeNode root, int k) {
        if (root == null || k < 1) {
            return 0;
        }
        if (k == 1) {
            return 1;
        }
        return getNodeNumKthLevel(root.left, k - 1) +
getNodeNumKthLevel(root.right, k - 1);
    }
```

24.算法题：给定一个字符串，统计字符出现的次数。按字符出现的顺序，输出字符和频率。

```
public static Map countChar(String str){

        char[] buf=str.toCharArray();

        Map<Character, Integer> map=new TreeMap<Character, Integer>();

        for(Character c:buf){
            if(map.containsKey(c)){
                map.put(c, map.get(c)+1);
            }else{
                map.put(c, 1);
            }
        }
        return map;
    }

    public  static  void show(Map map){
        //遍历
        Set entrySet=map.entrySet();
        Iterator<Entry> ite=entrySet.iterator();
        while(ite.hasNext()){
            Entry entry=ite.next();
            System.out.println("Key:"+entry.getKey()+"
Value:"+entry.getValue());
        }
    }
```

25.如何具体说了下基数排序的过程

26.重载和覆盖的区别

重载和重写**都是多态**的一种体现方式。

1、重载是编译期间的活动，重写是运行期间的活动。

2、重载是在一个类中定义相同的名字的方法，方法的参数列表或者类型要互相不同，但是返回值类型不作为是否重载的标准，可以修改可见性；

重写是不同的，要求子类重写基类的方法时要与父类方法具有相同的参数类型和返回值，可见性需要大于等于基类的方法

关于可见性：

子类重写父类方法，返回值为基本类型时，必须相同；返回值为引用类型时，要小于等于父类。

同理，子类抛出的异常也要小于等于父类； 子类重写方法的修饰词要大于等于父类； 两小、一大、一同原则

27.按值和按引用的区别

1.按值调用：方法接收的是调用者提供的值。

2.按引用调用：方法接收的是调用者提供的变量地址。

Java中非对象数据类型按值传递，对象数据类型按"引用传递"，但是你只可以修改它的值，不可以指向新的地址！对象类型是按引用调用。

28.接口和抽象的区别

实现：抽象类的子类使用 extends 来继承；接口必须使用 implements 来实现接口。

构造函数：抽象类可以有构造函数；接口不能有。

main 方法：抽象类可以有 main 方法，并且我们能运行它；接口不能有 main 方法。

实现数量：类可以实现很多个接口；但是只能继承一个抽象类。

访问修饰符：接口中的方法默认使用 public 修饰；抽象类中的方法可以是任意访问修饰符。

29.数组跟链表的优缺点

最明显的区别是 ArrrayList底层的数据结构是数组，支持随机访问，而 LinkedList 的底层数据结构是双向循环链表，不支持随机访问。使用下标访问一个元素，ArrayList 的时间复杂度是 O(1)，而 LinkedList 是 O(n)。

30.深度优先和广度优先算法。BFS的实现

BFS（Breadth First Search），其主要思想是从起始点开始，将其邻近的所有顶点都加到一个队列（FIFO）中去，然后标记下这些顶点离起始顶点的距离为1.最后将起始顶点标记为已访问，今后就不会再访问。然后再从队列中取出最先进队的顶点A，也取出其周边邻近节点，加入队列末尾，将这些顶点的距离相对A再加1，最后离开这个顶点A。依次下去，直到队列为空为止。

DFS（Depth First Search）深度优先搜索是从起始顶点开始，递归访问其所有邻近节点，比如A节点是其第一个邻近节点，而B节点又是A的一个邻近节点，则DFS访问A节点后再访问B节点，如果B节点有未访问的邻近节点的话将继续访问其邻近节点，否则继续访问A的未访问邻近节点，当所有从A节点出去的路径都访问完之后，继续递归访问除A以外未被访问的邻近节点

```
public static void bfs(HashMap<Character, LinkedList<Character>> graph,
HashMap<Character, Integer> dist, Character s) {
        //建立队列
        Queue<Character> q = new LinkedList<>();
        //给定起始节点
        Character start = s;
        //起始节点放到距离表中
        dist.put(start, 0);
        ((LinkedList<Character>) q).add(start);
        //遍历,一定要取出栈顶节点再加入
        while (q != null) {
            //取出栈顶节点和栈顶节点到起始节点的距离
            Character poll = q.poll();
            if(poll == null){
                break;
            }
            Integer distance = dist.get(poll);
            System.out.println("节点" + poll + "到起始节点" + start + "的距离为" +
distance);
            distance++;
            //将邻接节点加入
            for (Character c : graph.get(poll)) {
                //如果没遍历过这个节点,加入到队列和距离表中
                if (!dist.containsKey(c)) {
                    dist.put(c, distance);
                    q.offer(c);
                }
            }
        }
```

31.对华为狼性文化的看法。对美国打压华为的看法。

敏锐的嗅觉，敏捷的反应，以及敢于往前扑的勇气。还有就是像狼群一样团结上进，上下一心。我觉得狼性文化的加班对于刚毕业的应届生来说很契合，年轻有活力的我们应该在年轻的时间节点多努力多沉淀沉淀自己，但是现在很多公司虽然在学华为的狼性文化，但是很多跑偏了，接着狼性文化肆意剥削员工，不给调休不给加班费。我的看法是，只要加班费到位，身体扛得住，我是接受加班的，但是如果不给调休不给加班费就属实剥削劳动力了。

老美大力打压华为，恰恰证明老美慌了，害怕世界霸主的地位被撼动，也证明了华为的实力，是国内顶级科技的代表。表面上打压的是华为，实际上打压的是中国。我们国家一直都顶着世界第一加工厂的头衔，虽然手机电脑行业做的红热，但是核心科技，比如芯片和操作系统还是用的别人的，这次打压是逆境也是重生的好机会，比如台积电断供其实也是在逼着华为往更强大的方向走，只有最核心的技术掌握在自己手上才不怕被别人牵着走。

32.知道DML、DDL吗？知道两表关联、多表关联吗？

DML（data manipulation language）数据操纵语言：就是我们最经常用到的 SELECT、UPDATE、INSERT、DELETE。

DDL（data definition language）数据库定义语言：其实就是我们在创建表的时候用到的一些sql，比如说：CREATE、ALTER、DROP等。DDL主要是用在定义或改变表的结构，数据类型，表之间的链接和约束等初始化工作上

DCL（Data Control Language）数据库控制语言：是用来设置或更改数据库用户或角色权限的语句，包括（grant,deny,revoke等）语句

left join：在两张表进行连接查询时，会返回左表所有的行，即使在右表中没有匹配的记录。

right join：在两张表进行连接查询时，会返回右表所有的行，即使在左表中没有匹配的记录。

inner join：在两张表进行连接查询时，只保留两张表中完全匹配的结果集。

full join：在两张表进行连接查询时，返回左表和右表中所有没有匹配的行。

33.MySQL存储引擎，数据结构，事务

索引是一种数据结构，不是单单的一本书的目录而已，在查询数据时如果不用索引的话就是一条一条比对的查询。如果查询的数据就是千万级别的话，那么效率低下，就需要引入索引来帮助快速查询数据，索引是一条数据的一列或者多列，索引对应地是数据的地址。一般关系型数据库用B/B+树来构建索引（**一层可以存放更多的元**），不用红黑树或者平衡二叉树是因为树的高度还是太高了，磁盘IO读取次数过多。**B+树中叶子结点的指针是为了支持范围查找、模糊查找，所以不用每次查找一个元素都从根节点开始查找了**

存储引擎是修饰在数据库表的，不是数据库的。（InnoDB和MyISAM）

MyISAM：有三个文件，一个存放索引，一个存放数据，一个存放表结构。

查找数据时会先根据索引文件判断索引对应的行记录地址，然后再查找行记录文件找到所对应的行记录。叶子结点存放的数据是**一个行记录的地址指针。**

InnoDB：有两个文件，一个存放表结构，一个是数据和索引的合并文件。（主索引+辅助索引来查询数据）

表数据文件本身就是用B+树来组织的索引结构文件，所以必须要有主键，而且设置为自增的整型。叶子结点存放的数据是**一个完整的行记录数据。**

聚集索引：叶子结点存放的完整的记录数据，所以InnoDB是聚集而MyISAM不是。

1. 原子性（Atomicity）

事务被视为不可分割的最小单元，事务的所有操作要么全部提交成功，要么全部失败回滚。

回滚可以用回滚日志（Undo Log）来实现，回滚日志记录着事务所执行的修改操作，在回滚时反向执行这些修改操作即可。

2. 一致性（Consistency）

数据库在事务执行前后都保持一致性状态。在一致性状态下，所有事务对同一个数据的读取结果都是相同的。

3. 隔离性（Isolation）

一个事务所做的修改在最终提交以前，对其它事务是不可见的。

4. 持久性（Durability）

一旦事务提交，则其所做的修改将会永远保存到数据库中。即使系统发生崩溃，事务执行的结果也不能丢失。

未提交读（READ UNCOMMITTED）

事务中的修改，即使没有提交，对其它事务也是可见的。（更新数据时加入了共享锁即读锁，只能读不能写，解决丢失修改）

提交读（READ COMMITTED）

一个事务只能读取已经提交的事务所做的修改。换句话说，一个事务所做的修改在提交之前对其它事务是不可见的。（读数据时加入了共享锁但是读操作完了以后就释放共享锁，使得不能重复读，更新数据时加入了排他锁即写锁，解决脏读的问题）

可重复读（REPEATABLE READ）

保证在同一个事务中多次读取同一数据的结果是一样的。（读数据时加入了共享锁，一直到事务执行完毕才释放共享锁，这样就可以解决不可重复读问题，更新数据时加入了排他锁即写锁。）

可串行化（SERIALIZABLE）

强制事务串行执行，这样多个事务互不干扰，不会出现并发一致性问题。

该隔离级别需要加锁实现，因为要使用加锁机制保证同一时间只有一个事务执行，也就是保证事务串行执行。（读数据时对**整个数据表**加入了共享锁，更新数据时对**整个数据表**加入了排他锁即写锁，解决幻影读问题）

34.linkedHashMap的工作原理，底层实现机理。

LinkedHashMap是HashMap和双向链表合二为一，hashmap中的数组是双向链表结构，同时链表也是双向的。LinkedHashMap保证有序和查询效率。

35.class文件中常量池存储什么样的数据，它的结构构造。

直接引用和符号引用简述：

举个例子：现在我要在A类中引用到B类，符号引用就是我只要知道B类的全类名是什么就可以了，而不用知道B类在内存中的那个具体位置（有可能B类还没有被加载进内存呢）。直接引用就相当于是一个指针，能够直接或者间接的定位到内存中的B类的具体位置。将符号引用转换为直接引用简单来说就是：在A类中可以通过使用B类的全类名转换得到B类在内存中的具体位置。

主要存放两类数据，一是字面量、二是符号引用。

字面量：比如String类型的字符串值或者定义为final类型的常量的值。

符号引用：

1. 类或接口的全限定名（包括他的父类和所实现的接口）
2. 变量或方法的名称
3. 变量或方法的描述信息
4. this

- 方法的描述：参数个数、参数类型、方法返回类型等等
- 变量的描述信息：变量的返回值

Class文件结构采用类似C语言的结构体来存储数据的，主要有两类数据项，无符号数和表，无符号数用来表述数字，索引引用以及字符串等，比如 u1,u2,u4,u8分别代表1个字节，2个字节，4个字节，8个字节的无符号数，而表是有多个无符号数以及其它的表组成的复合结构。

36.介绍一下java中的虚拟机栈和本地方法栈的区别。

虚拟机栈是用于描述java方法执行的内存模型。

每个java方法在执行时，会创建一个“栈帧（stack frame）”，栈帧的结构分为“局部变量表、操作数栈、动态链接、方法出口”几个部分（具体的作用会在字节码执行引擎章节中讲到，这里只需要了解栈帧是一个方法执行时所需要数据的结构）。我们常说的“堆内存、栈内存”中的“栈内存”指的便是虚拟机栈，确切地说，指的是虚拟机栈的栈帧中的局部变量表，因为这里存放了一个方法的所有局部变量。

本地方法栈的功能和特点类似于虚拟机栈，均具有线程隔离的特点以及都能抛出StackOverflowError和OutOfMemoryError异常。

不同的是，本地方法栈服务的对象是JVM执行的native方法，而虚拟机栈服务的是JVM执行的java方法。

37.jvm中常见的垃圾回收算法，都用在什么样子的场合。

- 标记-清除算法**（多次标记效率低，会引起内存碎片）**。
- 标记-整理法：首先标记出所有需要回收的对象，让所有存活的对象都向一端移动，然后直接清理掉端边界以外的内存。**(适用于老年代，多次标记效率低)**
- 引用计数法**（无法解决循环引用问题）**
- 复制算法**（适用于新生代，不适用于垃圾少的情况如老年代，且内存使用率低只有一半）**

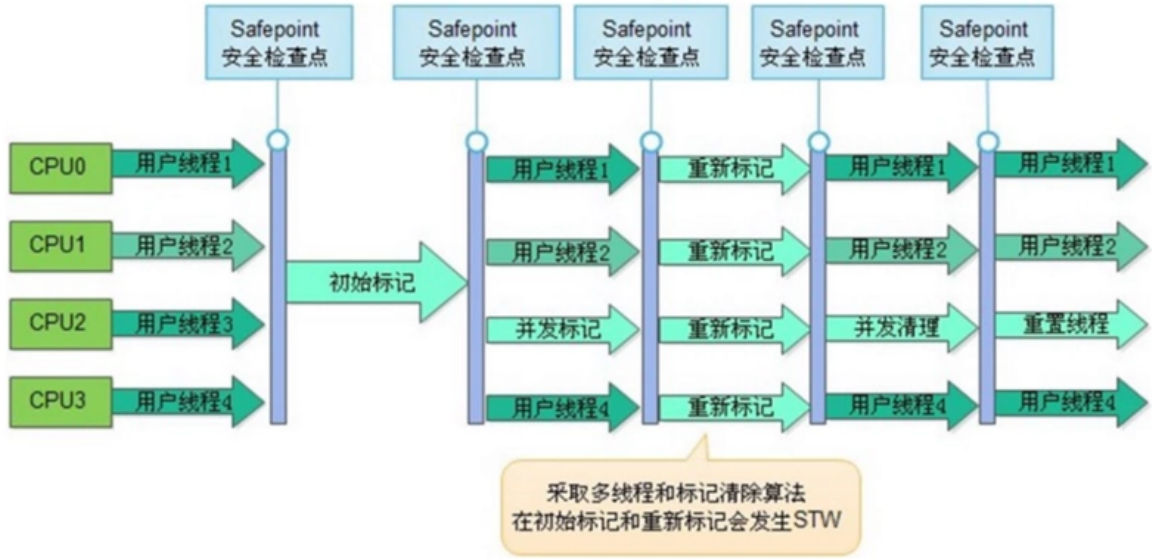
38.垃圾回器g1，cms

- Serial：最早的单线程串行垃圾回收器。
- Serial Old：Serial 垃圾回收器的老年版本，同样也是单线程的，可以作为 CMS 垃圾回收器的备选预案。
- ParNew：是 Serial 的多线程版本。
- Parallel 和 ParNew 收集器类似是多线程的，但 Parallel 是吞吐量优先的收集器，可以牺牲等待时间换取系统的吞吐量。
- Parallel Old 是 Parallel 老生代版本，Parallel 使用的是复制的内存回收算法，Parallel Old 使用的是标记-整理的内存回收算法。 **(JDK8默认使用的是Parallel回收，新生代用Parallel new ，老生代用Parallel old)**
- CMS：标记清除法一种以获得最短停顿时间为目标的收集器，非常适用 B/S 系统。
- G1：一种兼顾吞吐量和停顿时间的 GC 实现，是 JDK 9 以后的默认 GC 选项。

1、CMS

CMS(Concurrent Mark Sweep)，我们可以轻易地从命名上看出，它是一个并发的，然后是基于标记——清理的垃圾回收器，它清理垃圾的步骤大致分为四步：

1. 初始标记
2. 并发标记
3. 重新标记
4. 并发清理



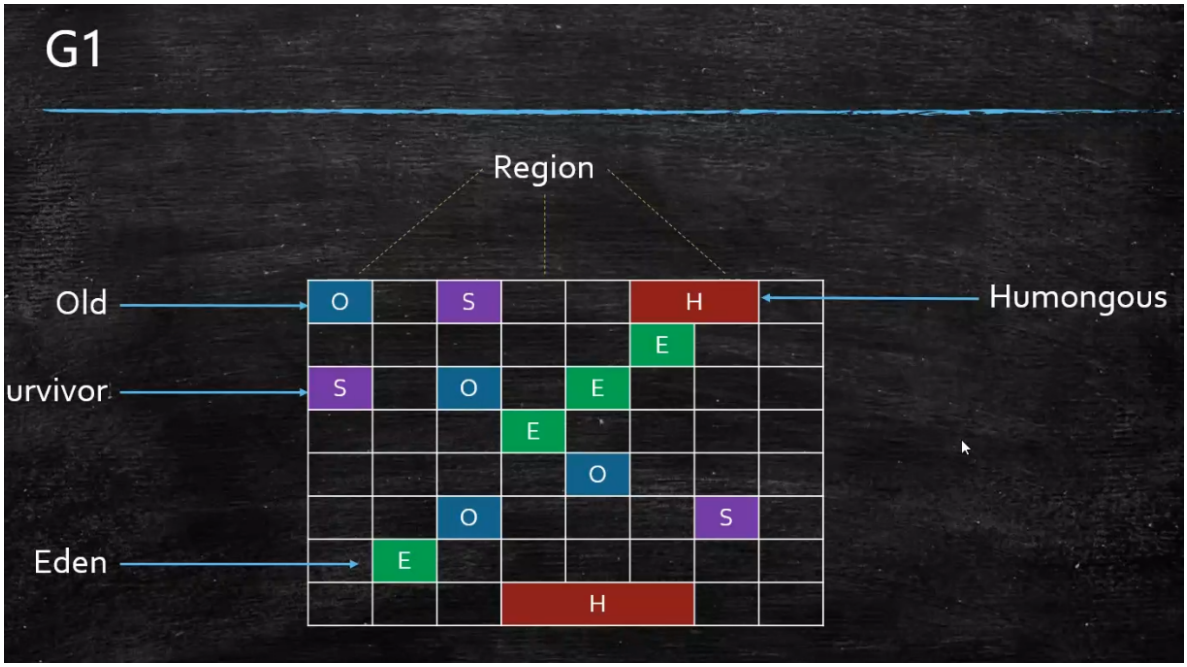
初始标记只要是找到GC Roots，所以是一个很快的过程，并发标记和用户线程一起，通过GC Roots找到存活的对象，重新标记主要是修复在并发标记阶段的发生了改变的对象，这个阶段会Stop the World；

并发清理则是保留上一步骤标记出的存活对象，清理掉其他对象，正因为采用并发清理，所以在清理的过程中用户线程又会产生垃圾，而导致浮动垃圾，只能通过下次垃圾回收进行处理；

因为cms采用的是标记清理，所以会导致内存空间不连续，从而产生内存碎片

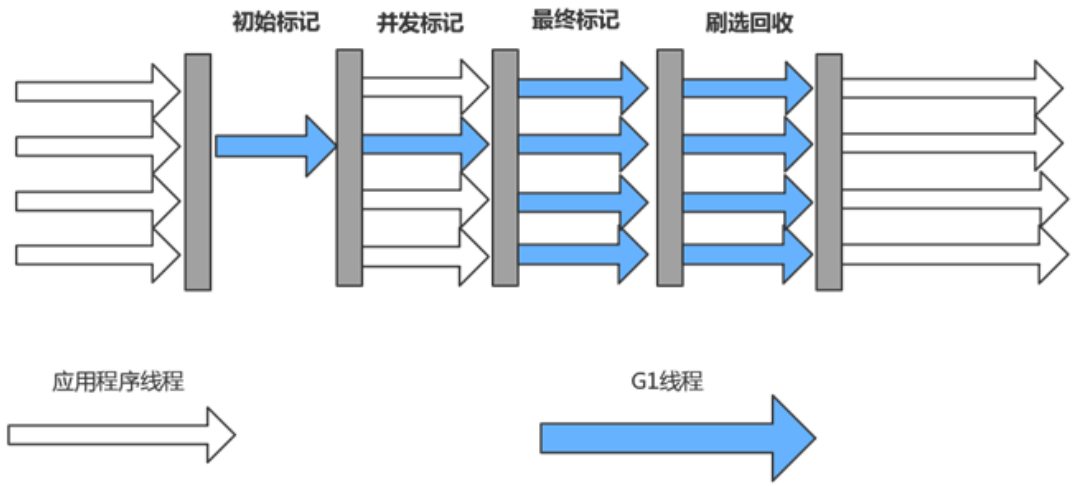
2.G1

G1(Garbage-First)，以分而治之的思想将堆内存分为若干个等大的Region块，虽然还是保留了新生代，老年代的概念，但是G1主要是以Region为单位进行垃圾回收，G1的分块大体结果如下图所示：



G1垃圾回收器的它清理垃圾的步骤大致分为四步：

1. 初始标记
2. 并发标记
3. 最终标记
4. 复制回收



初始标记和并发标记和CMS的过程是差不多的，最后的筛选回收会首先对各个Region的回收价值和成本进行排序，根据用户所期望的GC停顿时间来制定回收计划

因为采用的标记——整理的算法，所以不会产生内存碎片，最终的回收是STW的，所以也不会有浮动垃圾，Region的区域大小是固定的，所以回收Region的时间也是可控的

同时G1 使用了Remembered Set来避免全堆扫描，G1中每个Region都有一个与之对应的RememberedSet，在各个 Region 上记录自家的对象被外面对象引用的情况。当进行内存回收时，在GC根节点的枚举范围中加入RememberedSet 即可保证不对全堆扫描也不会有遗漏。

39.八大排序算法稳定性。

排序方法	时间复杂度（平均）	时间复杂度（最坏）	时间复杂度（最好）	空间复杂度	稳定性
插入排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
希尔排序	$O(n^{1.3})$	$O(n^2)$	$O(n)$	$O(1)$	不稳定
选择排序	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	不稳定
堆排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(1)$	不稳定
冒泡排序	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	稳定
快速排序	$O(nlog_2n)$	$O(n^2)$	$O(nlog_2n)$	$O(nlog_2n)$	不稳定
归并排序	$O(nlog_2n)$	$O(nlog_2n)$	$O(nlog_2n)$	$O(n)$	稳定

40.完全二叉树有n层，第n层有m个叶子，问一共多少叶子

1、一棵深度为k且有

$$2^k-1$$

个结点的二叉树称为满二叉树。

2、 满二叉树每一层的结点个数都达到了最大值, 即满二叉树的第i层上有

$$2^{i-1}$$

个结点 (i≥1) 。

3、具有n个结点的完全二叉树的深度

$$\lfloor \log 2k \rfloor +1$$