1.三类语言：机器语言，汇编语言，高级语言（例：Java ）

机器语言性能最高

2.jdk1.8为主流

3.技术选型不会选最新版本，一般选稳定版

4.发展史，11个特点

5.\*\*\*\*java平台无关性：工作原理为：先编译统一的字节码class文件，后经各种不同jvm执行，屏蔽了硬件差异性。c/c++直接编译，与cpu绑死，所以与平台有关无跨平台。

6.\*\*\*\*jvm类加载机制流程：7步：加载、连接（验证、准备、解析）、初始化、使用、卸载（具体记忆）

加载指的是将类的class文件读入到内存，并为之创建一个java.lang.Class对象.

验证阶段用于检验被加载的类是否有正确的内部结构，并和其他类协调一致。

类准备阶段负责为类的静态变量分配内存，并设置默认初始值。

将类的二进制数据中的符号引用替换成直接引用。

初始化是为类的静态变量赋予正确的初始值

使用初始化后的类

进程执行完毕，部分垃圾回收

7.\*\*\*\*垃圾回收：垃圾回收是一种自动内存管理方法，垃圾收集器定期的扫描内存中不再被使用的对象，并将该对象所占用的内存资源释放出来以供其他程序使用。Jvm通过垃圾回收算法（finalize）自动回收。

8.\*\*\*\*final，finally，finalize的区分

final可以用来修饰类、方法、变量，分别有不同的意义， final修饰的class代表不可以继承扩展， final的变量是不可以修改的，而final的方法也是不可以重写的（override）。

finally则是Java保证重点代码一定要被执行的一种机制。我们可以使用try-finally或者try-catch-finally来进行类似关闭JDBC连接、保证unlock锁等动作。

finalize是基础类java.lang.Object的一个方法，它的设计目的是保证对象在被垃圾收集前完成特定资源的回收。 finalize机制现在已经不推荐使用，并且在JDK 9开始被标记为deprecated。

9．\*\*\*\*引用关系的四种类型：虚引用（与垃圾回收有关），强引用，软引用，弱引用

强引用：是指创建一个对象并把这个对象赋给一个引用变量。

弱引用：弱引用也是用来描述非必需对象的，当JVM进行垃圾回收时，无论内存是否充足，都会回收被弱引用关联的对象。

软引用：如果一个对象具有软引用，内存空间足够，垃圾回收器就不会回收它；如果内存空间不足了，就会回收这些对象的内存。只要垃圾回收器没有回收它，该对象就可以被程序使用。软引用可用来实现内存敏感的高速缓存,比如网页缓存、图片缓存等。使用软引用能防止内存泄露，增强程序的健壮性。

虚引用： 虚引用和前面的软引用、弱引用不同，它并不影响对象的生命周期。在java中用java.lang.ref.PhantomReference类表示。如果一个对象与虚引用关联，则跟没有引用与之关联一样，在任何时候都可能被垃圾回收器回收。

10．\*\*\*\*jvm的双亲加载模式

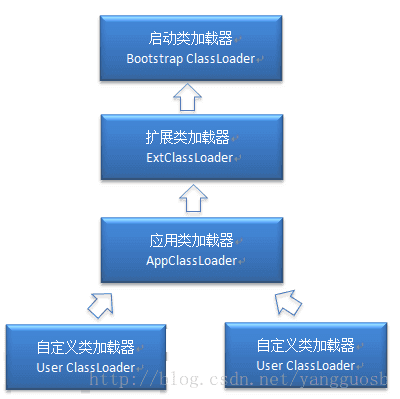
jvm三种类加载器：

1).Bootstrap ClassLoader:根类加载器，负责加载java的核心类，它不是java.lang.ClassLoader的子类，而是由JVM自身实现；

2).Extension ClassLoader:扩展类加载器，扩展类加载器的加载路径是JDK目录下jre/lib/ext,扩展类的getParent()方法返回null,实际上扩展类加载器的父类加载器是根加载器，只是根加载器并不是Java实现的；

3).System ClassLoader:系统(应用)类加载器，它负责在JVM启动时加载来自java命令的-classpath选项、java.class.path系统属性或CLASSPATH环境变量所指定的jar包和类路径。程序可以通过getSystemClassLoader()来获取系统类加载器；

双亲委派模型：



首先会先查找当前ClassLoader是否加载过此类，有就返回；

如果没有，查询父ClassLoader是否已经加载过此类，如果已经加载过,就直接返回Parent加载的类；

如果整个类加载器体系上的ClassLoader都没有加载过，才由当前ClassLoader加载(调用findClass)。

作用：保证JDK核心类的优先加载；

如何打破双亲模型：

自定义类加载器，重写loadClass方法；

使用线程上下文类加载器；

\*\*\*\*类的加载机制：从父类开始调用，一直到父类的object方法，在回到当前子类。

11.注释规范（参考阿里巴巴编程规范）

12.oop面向对象编程aop面向切面编程

13.\*\*\*\*资源获取---版本控制器（github，开源中国，svn，csv，vss，git，国内gitee）多试试

14.\*\*\*\*数据类型：八种基本数据类型和引用类型，及大小。

整型：byte 1

short 2

int 4

long 8

字符型：char 2

浮点型：float 4

double 8

boolean型:Boolean 由所在编译器决定大小

\*\*\*\*double比long大得多的原因

\*\*\*\*数据类型转换：从低到高byte，short，int，long,float,double

char,int.

byte，short,char混合运算结果是int型

强制数据类型转换会溢出。

C++可以自定义int大小，java在未来可能也会有这种功能，所以int未必就只能是四位

15.\*\*\*\*父子类中含有成员变量、静态成员变量，非静态语句块，静态语句块，构造方法，请问创建一个父类对象和一个子类对象时，成员变量、静态成员变量，非静态语句块，静态语句块，构造方法的执行顺序？

（1、父类的静态语句块、

父类静态成员变量  
（2、子类的静态语句块、

子类静态成员变量  
（3、父类的成员变量初始化、  
（4、父类的非静态语句块、父类的成员变量（两者同级看代码的先后顺序）  
（5、父类的构造方法  
（6、子类的成员变量初始化  
（7、子类的非静态语句块，子类的成员变量（两者同级看代码的先后顺序）  
（8、子类的构造方法

（1、如果一个类的成员变量在声明时就进行初始化处理， 则先执行成员变量的初始化，然后调用构造方法创建实例  
（2、如果一个类中有静态语句块，则在加载到jvm时（不一定创建实例），先执行静态语句块，（并且只执行一次），再执行成员变量的初始化，最后调用构造方法创建实例  
（3、如果一个类中有非静态语句块，则非静态语句块在类的成员变量初始化之后，构造方法执行之前进行执行。（并且每次使用new创建实例对象时都会执行一次）

16．i++，++i为表达式，有两种值：整体表达式的值，先增+1再用，后增用原值；表达式中的变量自己的值结束+1。

17.int i=0;i=i++;请问变量i的值是多少？结果为0。i被后面表达式的值覆盖。

18.$$与短路：一0就0;||或短路：一1就1.从左往右，短路后面的表达式不会参与运算

19.使用异或运算交换两个整型变量的值。（如何不使用第三者，完成两个整型变量值的交换）

完成两个整型变量值的交换三种方法：（不用）1.创建临时变量；2.求和做差；3.按位异或。

创建临时变量：引入一个变量tmp存放其中一个待交换变量a，再对待交换变量赋值即可。

例：tmp = a;

a = b;

b = tmp;

求和做差：将这两个数求和然后赋给其中一个变量a，变量a减去变量b就能得到原来的a的值。

例：a = a + b; //a的值被重新赋值

b = a - b; //a-b 的值是原来a的值

a = a - b; //a-b 的值是原来b的值

按位异或：按照（二进制）位异或 的方法交换数据。在计算机中，所有的数据都是以二进制0,1形式存储的；比如10 在计算机中的存储序列为1010， 20在计算中的存储序列为 10100。 按位异或的规则是：不同为1，相同为0，即对应二进制位上 是1和0时，结果为1，否则为0；



例： a = a^b;

b = a^b;

a = a^b;

20.\*\*\*\*计算机中所有的数都是以补码形式存在的，位运算是在补码的基础上进行的。

21.原码，补码，反码

原码：原码就是符号位加上真值的绝对值, 即用第一位表示符号, 其余位表示值.

例：[+1]原 = 0000 0001

[-1]原 = 1000 0001

反码：正数的反码是其本身

负数的反码是在其原码的基础上, 符号位不变，其余各个位取反.

例：[+1] = [00000001]原 = [00000001]反

[-1] = [10000001]原 = [11111110]反

补码：正数的补码就是其本身

负数的补码是在其原码的基础上, 符号位不变, 其余各位取反, 最后+1. (即在反码的基础上+1)

例：[+1] = [00000001]原 = [00000001]反 = [00000001]补

[-1] = [10000001]原 = [11111110]反 = [11111111]补

22.三种分类：顺序流程，分支流程，循环流程

\*\*\*\*面试考点：for(;;){}和while(true){}两个死循环哪个执行效率高？

证明for (;;)执行效率高的相对优势:1.指令少2.不占用寄存器3.没有判断跳转,并不是不能跳出

23.\*\*\*\*break和continue关机字的区别？什么是位置标签？

break中断的是其所在的循环体，中断后就不再循环，continue中断的是当前一次循环后，跳出当前循环再进行下一次循环。

（Java内的goto关键字为保留关键字，但没有使用。）

24.形参和实参？方法定义时给定的参数列表和方法被调用时主方法传递给被调用的子方法的参数。

形参和实参一一对应个数，类型和顺序。

25.void类型的返回方法是否可以返回return？

注意标注了void不代表没有返回类型，只是返回类型为空，表示完成当前void方法，方法内return后面如果有代码不再执行。

26.java中实参和形参参数值的传递方式？

（1.java的基本数据类型是传值调用，对象引用类型是传引用。

（2.当传值调用时，改变的是形参的值，并没有改变实参的值，实参的值可以传递给形参，但是，这个传递是单向的，形参不能传递回实参。

（3.当引用调用时（地址传递，给形参传递实参的地址，形参值的改变会影响实参），如果参数是对象，无论对对象做了何种操作，都不会改变实参对象的引用，但是如果改变了对象的内容，就会改变实参对象的内容。

27.\*\*\*\*何为方法重载？重载条件？重载的的调用方式？变长参数（jdk1.5）？变长参数与重载的关系？

方法重载:在同一个类中,如果方法名相同,但是参数（类型，个数，顺序）不同则为方法重载.

（1.变长参数（可变参数）：

1.使用 … 表示可变参数------ 三个下点

( int … i )

2.编译器会自动为可变参数隐式创建一个数组,通过操作数组的方式使用可变参数

所以可变参数本质上就是数组

3.可变参数必须放在参数列表的最后位置

( char i , short j , ---------代表省略号---------int … i )

4.可传递0个一个多个实参值。

5.普通方法和多加了变长参数的方法同时存在，优先运行普通方法。

（2.构成方法重载的必要条件：

方法重载是发生在同一个类中；

方法名必须相同；

参数列表必须不同；

访问修饰符、返回值可以相同也可以不同，和重载没有半毛钱关系。

（3.重载调用的方式

调用非静态方法：通过对象调用

调用静态的方法：通过类名调用

本类中

静态直接调用静态

静态调用非静态

非静态调用静态

非静态调用非静态

不同类中

静态调用静态

静态调用非静态

非静态调用静态

非静态调用非静态

28.什么是不规则数组？如何进行遍历？

对Java中不规则数组的理解：

Java实际上没有多维数组，只有一维数组，多维数组被解释为“数组的数组”。 “不规则”数组，即数组的每一行有不同的长度，例如第一行有一列，第二行有两列，依次类推。

例：

//创建不规则数组

final int MAX = 10;

int [][] arrs = new int[MAX + 1][]; //分配一个具有所含行数的数组

for (int n = 0; n <= MAX; n++) //分配每一行

arrs[n] = new int[n + 1];

//给数组赋值

for (int n = 0; n < arrs.length; n++) {

for (int k = 0; k < arrs[n].length; k++){

int lotteryArrs = 1;

arrs[n][k] = lotteryArrs;

}

}

//遍历二维数组

for (int[] row : arrs){

for (int arr : row){

System.out.print(arr + " ");

}

System.out.println();

}

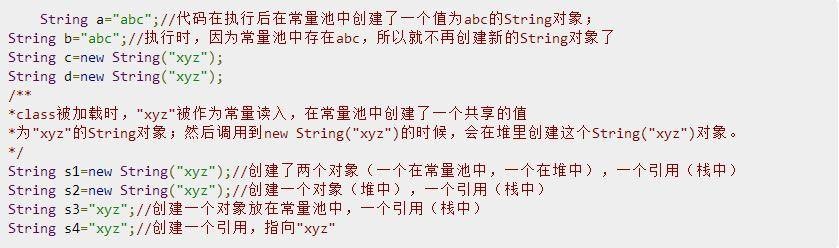
29.String类在内存中的存储方式？

String a = “abc”；

String b = new String("abc");

这两种声明方法得到的String是不一样的，第一种声明方法得到的String是存储在String常量池里面的，而第二种方式的声明很明显是正常的创建一个对象的使用方法，所以存储在堆中。

30.String str1=new String(“小明”)；创建几个对象？



31.==和equals的异同点

（1.equals只能比较引用类型，“==”既能比较引用类型又能比较基本数据类型；

（2.equals方法默认判断对象的哈希值是否相等（拓展：equals相等，hashcode一定相等，hashcode相等，equals不一定相等）；“==”运算符在判断引用数据类型对象时，是判断对象的地址是否相同。

（3.equals指的是值是否相同，“==”指引用是否相同。

32. \*\*\*\*什么是面向对象？（自己组织语言）

面向对象是一种思想，是基于面向过程而言的，就是说面向对象是将功能等通过对象来实现，将功能封装进对象之中，让对象去实现具体的细节；这种思想是将数据作为第一位，而方法或者说是算法作为其次，这是对数据一种优化，操作起来更加的方便，简化了过程。

\*\*\*\*抽象，继承，封装，多态

抽象:将相同的类的共同特征总结出来，构造成类的过程。包括数据抽象以及行为抽象。数据抽象变成类的成员变量，行为抽象变成类的成员函数。抽象只关心成员变量以及成员函数，并不关系具体的实现细节。

继承是从已有的类得到继承信息创建新的类的过程，继承可以表示为 is-a 关系 。比如动物就是父类或者说基类。那么动物有很多种，比如小狗，小猫，它们有很多共同的属性，它们都是动物。那么表现在代码里就是，它们可以直接把行为或者数据继承过来。

封装其实可以理解为：“装”与“封”，就是把数据以及操作方法封装在函数里/类，然后再装起来。也就是把数据以及操作数据的方法绑定起来，对外界提供已经定义好的接口进行访问。面向对象的本质就是将现实世界描绘成一系列的完全自治，封闭的对象。可以理解为我们无论内部实现有多复杂，使用者可以完全不用理，只需要知道怎么使用就可以了。

多态，就是指一个引用（类型）在不同情况下的多种状态，你也可以这样理解：多态是指通过指向父类的指针，来调用在不同子类中实现的方法。

\*\*\*\*成员变量（类变量和实例变量）和局部变量的区别

作用域不同，成员变量全局使用，局部只限定在方法内。

\*\*\*\*高内聚，低耦合

耦合：元素与元素之间的连接，感知和依赖量度。这里说的元素即是功能，对象，系统，子系统，模块。

低耦合方法：

元素A不能过度依赖元素B

合理的职责划分：让系统中的对象各司其职，不仅是提高内聚的要求，同时也可以有效地降低耦合

使用接口而不是继承：我们不难发现。继承就是一种耦合，假如子类A继承了父类B，不论是直接继承或者间接继承，一但父类B不存在或者发生任何变更，都将导致子类A不得不修改或者重写。假如父类B的子类数十上百的，这就是灾难性的变更。

内聚：更为专业的说法叫做功能内聚，是对系统中元素职责的相关性和集中度的量度。如果元素有高度的相关职责，除了这些职责在没有其他的工作，那么该元素就有高内聚，

也就是各司其职不越界。

33.\*\*\*\*构造方法的执行流程？父类子类构造方法的关系？

34.什么是继承？Java继承是单继承，如何实现多继承？

用内部类分别单继承不同的父类，就类似于当前类实现了多继承

35.什么是多态？多态的体现？（一是：父类指代子类（溯型）；二是重写）什么是重写？什么是溯型？上溯：用父类对象调用子类方法；下溯：用父类调用自己内部方法。

36.构成重写的条件？

方法重写：只存在于非静态非私有，方法在继承父类或实现接口时,对相同的方法名相同的参数类型的方法,进行重写

需要注意:静态方法没有重写.子类只是将父类的静态方法隐藏了.

条件：

（1必须是在具有继承或者实现关系的两个类之间；

（2方法名、参数列表必须相同；

`(3子类重写方法的返回值类型如果是基本数据类型必须和父类相同，如果是引用类型必须是父类方法返回值的子类或和父类的返回值相同；

(4重写方法若要抛出异常，必须是被重写方法抛异常类的异常子类或者相同或者和比被重写方法抛得异常要少，父类方法不抛异常子类重写也不抛异常；

`(5重写方法的访问修饰符范围要大于等于被重写方法的访问修饰符。

37.方法重载与重写的异同？

重载和重写都是实现多态的方式，区别在与重载是编译时期的多态，重写是运行时期的多态，重载发生在一个类中，同名的方法不同的参数列表视为重载（同名不同命），重写则是实现或者继承的时候重写父类的方法，要求参数列表和返回值相同。

（1方法重载的特点：

1.参数列表不同（参数列表包括：参数的个数，顺序，类型）

2.重载与返回值无关，可以存在父类，子类，同类中

3.可以抛出不同的异常

4.可以有不同的修饰符

（2方法重写的特点：

1.参数列表完全相同（个数，顺序，类型）

2.返回值类型必须和父类一致

3.访问修饰符的权限必须宽于父类（大）

4.抛出的异常需得是父类的同异常或自异常（小）

38.\*\*\*\*this和super关键字的区别？

This访问本方法内的变量，super访问的时父类的成员变量

39.整理static,final,abstract,interface

40.接口多继承：类与接口：类implements接口1，接口2，。。。

接口与接口：接口extends接口1，接口2.。。。

接口的多继承解决类与类单继承的不足

41.接口和抽象类的异同？了解Default关键字？

1-成员区别：

接口：

成员变量：可以变量也可以常量

构造方法：有构造方法，用于子类访问父类数据的初始化。

成员方法：可以抽象也可以非抽象

抽象类：

成员变量：只能常量public static final

构造方法：没有构造方法

成员方法：只能抽象 public abstract

2-关系区别：

1- 类与类：

继承关系，只能单继承，可以多层继承

2-类与接口：

实现关系，可以单实现，可以多实现，也可以在继承类的同时实现单个或多个接口。

例如：class son extends Object implements Father，Mather{}

3-接口与接口：

继承关系，可以单继承和多继承

例如：interface Sister extends Father, Mather{};3-设计理念不同：

抽象类被继承的体现是is a的关系，该继承体系中共性的功能

接口被实现体现的是like a 的关系，该继承体系中拓展功能。

Default关键字，java8新特性，虚拟扩展方法，可以默认实现接口中的方法体，子类调用时不用重写，用new一个类直接调用，出现多接口相同default方法，不直接调用的要重写覆盖。

42. （自己整理的）实现Runnable接口比继承Thread类所具有的优势：

（1适合多个相同的程序代码的线程去共享同一资源

（2可以避免java中的单继承的局限性

（3增加程序的健壮性，实现解耦操作，任务可被多个线程共享，任务和线程独立

（4线程池只能放入实现Runnable或Callable类线程，不能直接放入继承Thread的类

43.匿名内部类？事件处理（例如按钮监听器）？委托？

匿名内部类：可以使用new的方法调用接口或抽象类的行为方法

可以new，但是，将实现和new创建合二为一

使用原则：当只关于行为动作的结果，而不关于行为动作的发出者时，

使用匿名内部类进行抽象方法调用；

事件处理：为要触发事件的对象（source）定义事件对象；

其次，为事件对象定义事件监听器；

最后，定义事件源（source）的类，指定添加监听器的方法。

委托：委托指的是在A类中以各种方式利用B类，完成类的功能。

委托的类型：1.A use BB类对象在A类中出现，但是是以局部变量或是方法参数的形式出现的。A类中并没有B类的对象作为域。一般称这种delegation为临时性的delegation。

2.A has BB类对象在A类中出现，B类的对象是A类的域之一。B类对象通过A类对象的constructor方法或其它方法从外部输入A类对象。A has B有两种情况。

1）Association。A类对象和B类对象之间并没有从属关系。

2）Aggregation。A类对象由B类聚合而成，但是B类可以脱离A类单独存在。一般称这种delegation为永久性的delegation。

3.A ispartof BB类对象在A类中出现，B类的对象是A类的域之一。B类对象在A类对象内创建。B类对象不能脱离A类对象独立存在。一般称这种delegation为永久性的delegation。

44.异常的分类？异常与错误的区别？

（1）Exception：编译时异常（检查性，需要异常处理）和运行时异常（非检查性，修改代码bug）

（2）区别：

Exception和Error都是继承了Throwable类，在Java中只有Throwable类型的实例才可以被抛出（throw）或者捕获（catch），它是异常处理机制的基本组成类型。

Exception和Error体现了Java平台设计者对不同异常情况的分类。 Exception是程序正常运行中，可以预料的意外情况，可能并且应该被捕获，进行相应处理。

+

Error是指在正常情况下，不大可能出现的情况，绝大部分的Error都会导致程序（比如JVM自身）处于非正常的、不可恢复状态。既然是非正常情况，所以不便于也不需要捕获，常见的比如OutOfMemoryError之类，都是Error的子类。

Exception又分为可检查（checked）异常和不检查（unchecked）异常，可检查异常在源代码里必须显式地进行捕获处理，这是编译期检查的一部分。前面我介绍的不可查的Error，是Throwable不是Exception。

不检查异常就是所谓的运行时异常，类似 NullPointerException、 ArrayIndexOutOfBoundsException之类，通常是可以编码避免的逻辑错误，具体根据需要来判断是否需要捕获，并不会在编译期强制要求。

45.异常处理机制？finally语句块的特点？

Try-catch-finally和throws；

Finally无论发生什么异常都会被执行。

46.什么是包装类？什么是装箱和拆箱？

八种基本类型的包装类：Byte，Short，Integer，Long，Character，Float，Double，Boolean。

自动装箱即自动将基本数据类型转换成包装类型。

自动拆箱即自动将包装类型转换成基本数据类型。

47.为什么要有包装类？包装类存在的价值？

48.Java中那些地方必须使用包装类？泛型，synchronize（包装类）{}，集合

49.Object中equals和==方法一样。

50.死锁的必要条件（4个）？

（1.互斥条件：进程要求对所分配的资源进行排它性控制，即在一段时间内某资源仅为一进程所占用。

（2.请求和保持条件：当进程因请求资源而阻塞时，对已获得的资源保持不放。

（3.不剥夺条件：进程已获得的资源在未使用完之前，不能剥夺，只能在使用完时由自己释放。

（4.环路等待条件：在发生死锁时，必然存在一个进程--资源的环形链。

51.finally和return的用法？

Finally语句在return之前执行

52.常用集合类有哪些？具有那些特性？是否有序?

数组：读写快增删慢

链表：增删快读写慢

53.comparator与comparable接口的作用和异同点？

1.Comparator位于包java.util下，而Comparable位于包java.lang下。

2.Comparable接口将比较代码嵌入需要进行比较的类的自身代码中，而Comparator接口在一个独立的类中实现比较。

3.Comparable接口强制进行自然排序，而Comparator接口不强制进行自然排序，可以指定排序顺序。

54.什么是泛型？泛型的好处？泛型和方法重载有什么联系？泛型可以使用基本类型吗？

声明定义时：（jdk定义声明时，或者自己声明定义一个泛型类或者泛型方法）

是程序在编程时为了达到代码复用，在类、方法声明定义时，而使用泛型符号用于指代某种类型的一种编程方式；

编程使用时：（你自己写代码时）

在具体编程时，可以把泛型符号替换为具体的类型，这样在编程可以避免类型之间的转换过程；

使用泛型时，类型转换由编译器完成，如果不使用泛型，类型转换由编程人员自己显式处理；

使用泛型，可以避免大量的方法的重载；

在编程时，如果指明了具体的泛型类型，则所用到的数据类型必须与泛型类型一致，即使可以进行类型自动转换也不允许；

在使用泛型时，指明的泛型的具体类型只能是引用类型，不能是基本类型，基本类型应该使用其对应的包装类类型；

建议，对于集合等使用泛型定义的类或方法，在使用时指明具体的类型；（要求：在编程时使用具体的泛型类型）

55.HashMap的底层实现的数据结构？1.7-数组+链表；1.8+ 数组+链表+红黑树

56.什么是红黑树？在HashMap中链表是如何转换红黑树？为什么使用红黑树？

(1:什么是红黑树？

红黑树是一颗二叉搜索树，也是多种平衡搜索树的一种，可以保证最坏情况下时间复杂度为O(lgn)。

红黑树的特性：

1.节点是红色或黑色。

2.根节点是黑色。

3.每个叶子节点都是黑色的空节点（NIL节点）。

4 每个红色节点的两个子节点都是黑色。(从每个叶子到根的所有路径上不能有两个连续的红色节点)

5.从任一节点到其每个叶子的所有路径都包含相同数目的黑色节点。

(2:长度超过8

（3： 如果元素小于8个，查询成本高，新增成本低

如果元素大于8个，查询成本低，新增成本高

保证平衡性的最大的目的就是降低树的高度，因为树的查找性能取决于树的高度。所以树的高度越低搜索的效率越高！

57.HashMap的负载因子是0.75；HashMap的初始长度是16；HashMap是如何扩容的？按照16的倍数扩容；Map<String,Object> map=new HashMap<>(30);内存中初始化map长度为32.

58.解决Hash冲突：

（HashMap）拉链法：长度超过8后，转化为红黑树。

再散列法

依次后推

59.ConcurrentHashMap如何实现多线程并发的？CAS无锁化解决多线程并发：

【乐观锁】；Synchronize关键字解决多线程并发：【悲观锁】。

60.什么是乐观锁？什么是悲观锁？

悲观锁

总是假设最坏的情况，每次去拿数据的时候都认为别人会修改，所以每次在拿数据的时候都会上锁，这样别人想拿这个数据就会阻塞直到它拿到锁（共享资源每次只给一个线程使用，其它线程阻塞，用完后再把资源转让给其它线程）。传统的关系型数据库里边就用到了很多这种锁机制，比如行锁，表锁等，读锁，写锁等，都是在做操作之前先上锁。Java中synchronized和ReentrantLock等独占锁就是悲观锁思想的实现。

乐观锁

总是假设最好的情况，每次去拿数据的时候都认为别人不会修改，所以不会上锁，但是在更新的时候会判断一下在此期间别人有没有去更新这个数据，可以使用版本号机制和CAS算法实现。乐观锁适用于多读的应用类型，这样可以提高吞吐量，像数据库提供的类似于write\_condition机制，其实都是提供的乐观锁。在Java中java.util.concurrent.atomic包下面的原子变量类就是使用了乐观锁的一种实现方式CAS实现的。

61.什么是对象流？将内存中的某个时间节点的状体信息以字节流的形式持久化到硬盘文件中。

什么样的数据会进行序列化到硬盘进行持久化？

①在很多框架中就会有这种.object结尾的文件,因为很多对象都不创建,创建起来太麻烦,直接读取,而且有些对象的值你不知道,框架封存在.object文件中,直接读取这个文件中的这个值就行了,不需要传这个值.

在搞web开发的时候一些类就需要实现序列化接口,因为服务器就会对你的对象进行临时本地存储.它怕服务器崩了的以后,你的会话都被消失了.所以存储在了硬盘上,你重新启动服务器会恢复之前的会话,恢复对象,你之前运行的东西都在.

②对某些特点的对象,比如数据库连接对象,存储特定数据的对象 ,这样对象你不想创建他们,想存储起来,让他们的生命周期延长,可以把他们放在硬盘当中.每次系统启动的时候都到.object中读取对象和里面的数据,这个时候就可以把他们序列化来完成.

62.什么是序列化和反序列化？Serarizable接口的作用？

（1：序列化：指将一个对象给定一个唯一的编号，让其他人也可以通过序列号使用它。

反序列化：将序列化的对象再重新转换为对象。

（2：一个对象有对应的一些属性,把这个对象保存在硬盘上的过程叫做”持久化”.

对象的默认序列化机制写入的内容是：对象的类，类签名，以及非瞬态和非静态字段的值。(因为静态static的东西在方法区.)

序列化能把堆内存中的对象的生命周期延长,做持久化操作.当下次再需要这个对象的时候,我们不用new了,直接从硬盘中读取就可以了.(存储到硬盘是一个文件,不需要我们去解析,如果用记事本打开解析会出现乱码,解析要用特定的方式,不用我们管. 我们只需要读取). 把对象存储到硬盘上的一个文件中,这个文件的标准扩展名是(.object).

63.Transient关键字的作用？是否可以和final，static搭配使用？以及它们之间的区别？

1）一旦变量被transient修饰，变量将不再是对象持久化的一部分，该变量内容在序列化后无法获得访问。

2）transient关键字只能修饰变量，而不能修饰方法和类。注意，本地变量是不能被transient关键字修饰的。变量如果是用户自定义类变量，则该类需要实现Serializable接口。

3）被transient关键字修饰的变量不再能被序列化，一个静态变量不管是否被transient修饰，均不能被序列化。

\*\*\*\*64.编程题：输入一个文件目录，将该文件目录中的文件内容汇总含有“java”关键字的文件进行输出。（递归）（FileFilter接口）

65.扩展：NIO,NIO2的编程方式？

NIO

一般来说, 流IO表现得很好, 对于大部分的IO场景, 它都能适应. 但是, 由于它的阻塞性, 每一个流的读写都需要占用一个线程. 这意味着, 流IO的可伸缩性很差. 因此, 引入非阻塞IO就再正常不过了. 实际上, NIO就是IO Multiplexing在Java中的实现. IO Multiplexing在系统级语言如C/C++中应用了很长时间. 使用IO Multiplexing, IO的伸缩性大大提高, 使用单个线程, 就可以处理大量的IO对象.

在介绍NIO的非阻塞IO之前, 先大致了解一下NIO提供的IO模型.

NIO的概念概念有三个, Buffers/Channels/Selectors. 其中, Channels是输入/输出的管道, 所有的读写操作都需要通过它来完成. Channel读写的粒度是Block, 而不是像流IO一样, 提供一个字节流或者字符流的抽象. 这个Block的抽象即Buffer. 所有的读操作会由Channel将数据读入Buffer, 然后用户来处理Buffer, 所有的写操作需要先将数据填到Buffer中, 再由Channel来消费Buffer中的数据. NIO的第三个核心概念是Selector, 它是一个事件监控器, 我们将它注册我们所感兴趣的IO事件, 并且对其进行Polling, 来确定事件是否发生, 发生则做相应的IO操作. 其中, Selector所监控的对象是Channel, 我们在Selector上声明我们关心哪一个Channel的什么事件, Selector会监控这些Channels, 并在事件发生时通知我们.

现在, 考虑三个问题:

为什么要引入Channel, 直接扩展已有的Stream类不行吗?: 流的抽象已经很完备了, 添加更多的特性与概念只会将流的概念进一步复杂化, API更加难以使用, 这是一种很不好的API设计方式. 因此, NIO引入了一套新的抽象. Do one thing, and do it well.

为什么引入Buffer? 直接用byte数组可以吗?: 实际上肯定是可以的, 但Buffer类提供了更加方便的操作. 同时, Buffer提供了很多性能上的优化.

为什么引入Buffer? 直接读写byte不行吗?: 如果直接操作byte, 性能会很低, 实际上还是需要buffering来提供性能, 与其加一层buffering抽象, 不如直接给用户提供Buffer. 最重要的是, 基于Buffer的IO操作, 某些情况下可以直接映射成系统调用, 性能极高!

NIO支持阻塞与非阻塞两种模式. 阻塞模式下, 实际上与流IO差不多, 非阻塞模式下, Channels与Selector配合, 才是它最大的威力所在.

我们可以大体将Channel分成两类, 一种是支持SelectableChannel(除了FileChannel以外都是, 一般是网络相关的操作.), 另一种与non-SelectableChannel(即FileChannel). 前者可以与Selector一起使用, 提供强伸缩性的IO.

IO VS NIO

考虑IO与NIO的区别. 除了在概念模型的差别, IO与NIO在性能上也会有很大差异. 我们从三个方面来考虑性能问题:

可伸缩性: 流IO的在IO对象数较少及大规模IO的情况下, 表现得很好, 但是当需要处理成百上千的IO对象时, 它的性能会Drop得很快. 相反, NIO在非阻塞模式下(阻塞模式下应该与流IO具有相同的特点, 这是阻塞IO的共性), 即使用Selector, 它可以处理大量的非活跃连接, 是实现C10K的关键技术.

GC: 许多号称高性能的服务器实现, 都以Zero Allocation作为一个重要的功能点. 理想情况上, 如果没有GC的开销, 服务器可以将所有时间花在有效地工作上, 并且保持一个可靠的延迟. 然后GC是不可避免的, Zero Allocation也只能是尽力而为. 而相比较而言, NIO只需要申请一个Buffer, 可以反复使用, 而字符流在这方便表现的就比较差了, 如readLne()这类接口, 需要分配大量临时的String对象.

API抽象层次: 相对而言, 基于Buffer的NIO抽象层次比流IO在低一些. 特别的, 系统调用级别的IO, 都是基于Buffer的. 当使用DirectBuffer时, 某些平台下, OS可以直接将数据复杂到DirectBuffer中, 避免了流IO中, OS将数据复制到OS Buffer后, 又需要向JVM Heap复制地过程. Zero Copy与Zero Allocation都是高性能服务器的重点技术. 特别的, 在使用Channel时, 需要使用DirectBuffer, 因为Channel内部使用的是DirectBuffer. 如果使用HeapBuffer, 则读写时, Channel会申请一个临时的DirectBuffer, 造成性能开销.

Memory Mapping

前面提到, FileChannel不支持非阻塞模式. 那么, 它是不是用处不大呢? 毕竟, NIO与IO相比最大的优势是非阻塞.

NIO中, FileChannel都一些属于自己的特性. 即, Memory Mapping. Memory Mapping是一个比较觉见, 在此不加多说. 无论是在顺序读写, 还是随机读写中, Memory Mapping都能够提供不弱于BufferedInputStream或者RandomAccessFile的性能.

特别强调的是, Memory Mapping可以Map的容量仅与虚拟内存大小有关, 与物理内存大小及JVM堆大小都没有关系. 因此, 在64位平台下, Memory Mapping可以工作得非常好

NIO2

聊过非阻塞IO后, 再来看看异步IO. IO方面的概念很多, 阻塞性与异步性是其关键概念. 简单而言, 凡是需要由应用程序将数据读写到应用程序内存中的IO, 都是同步IO, 比如上面的流IO与NIO. 相对的, 凡是由OS来完成读写的, 就是异步IO. 这个说法有些迷惑. 举例而言, 在NIO中, 当应用程序检测到某个Channel有可读数据时, 必须显示发起一个read请求. 而在异步IO中, 应用程序仅仅需要告诉OS, 我需要什么数据, 并提供给OS一个Buffer和一个回调. OS会自己检测Channel的可读性, 但其发起其可读, 会自动将数据复制到Buffer中, 并通知应用程序任务完成. 异步IO的典型实现是NodeJS及Boost.ASIO. 显然, 由于将任务进一步下发到了OS, 应用程序的可伸缩性及性能会大大增强. 并且, 比起非阻塞的NIO, 异步IO编程更加容易一些, 性能也基本上总是优于它的.

NIO2最大的改进是引入了四个异步Channel, 用于支持异步读写. 同时, 它还增加了对文件系统和文件属性的支持, 提供了WatchService/FileVisitor这些高级功能.

66.什么是进程？什么是线程？进程和线程之间的异同？

进程是cpu进行资源分配的基本单位。

线程又叫作微进程，进程中有多个线程，一个进程可以独占cpu的suoyou资源

67.线程的状态以及转换方式？

三态：就绪，运行，等待

五态：就绪，挂起就绪，运行，等待，挂起等待

68.java多线程编程的实现方式？

继承Thread类；

实现Runnable接口；

通过继承Thread类重写run方法或者是通过实现Runnable接口实现run方法都可以进行多线程编程；

多线程的业务逻辑都是在run方法中实现的；

多线程的启动都是通过start方法启动一个多线程，而不能直接调用run方法（直接调用run方法其实就是一个普通方法调用，不会走多线程的运行机制）

69.多线程中可以直接调用run方法，执行线程吗？

run方法可以创建一个线程，但是相当于同步的方式，没有多线程的存在。

只有调用start方法才是交给jvm管理，才是多线程。

70.线程类实现之后，调用run方法和start方法有什么区别？

run()方法:在本线程内调用该Runnable对象的run()方法，可以重复多次调用；

start()方法:启动一个线程，调用该Runnable对象的run()方法，不能多次启动一个线程；

71.线程几个常用方法？

run方法：线程的业务逻辑处理方法，必须通过start方法启动，才是多线程，直接调用run方法不是多线程；

start方法：线程启动方法；只有通过start方法启动，才是多线程，直接调用run方法不是多线程；

sleep静态方法：线程休眠终端方法；

join方法：线程之间的调用方法，一个线程可以通过join方法调用两外一个线程，进行调用的主线程会被阻塞，等待被join子线程执行完成之后，才会继续执行；

yield方法：调用yield方法，当前线程主动放弃cpu资源，让与其他的线程执行；（主要作用就是让当前线程主动放弃cpu资源，让与其他线程执行）。

72.线程死锁？线程之间都持有部分关键资源，但没法全部持有，就进入全部等待，产生死锁。线程死锁产生的条件？四个，同50题

73.如何避免线程产生死锁？Synchronized关键字

74.线程同步？

同步又叫做互斥访问；

多个进程在申请资源时，是互斥访问的；

同步一般使用同步锁来进行实现，是解决死锁的一中方式；

同步实现是使用同步锁将可能引发死锁的这些个资源作为一个整体进行通过管理，主要是使用互斥访问的方式进行同步管理；

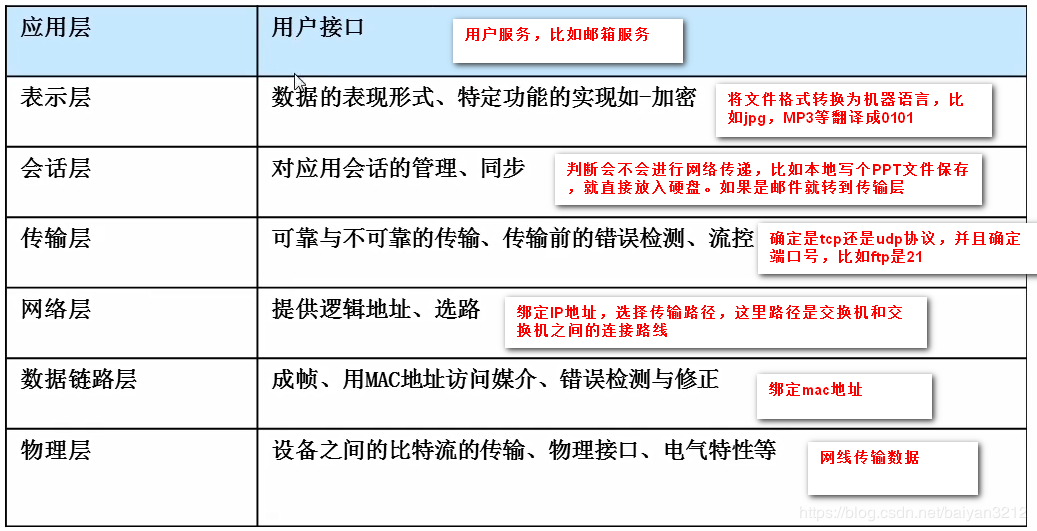
如果一个进程首先申请到了同步锁中的一种资源，那么其他进程就不可以再申请和占有同步锁所管理的其他资源，只能等待当前进程全部释放这些资源之后才能重新申请；

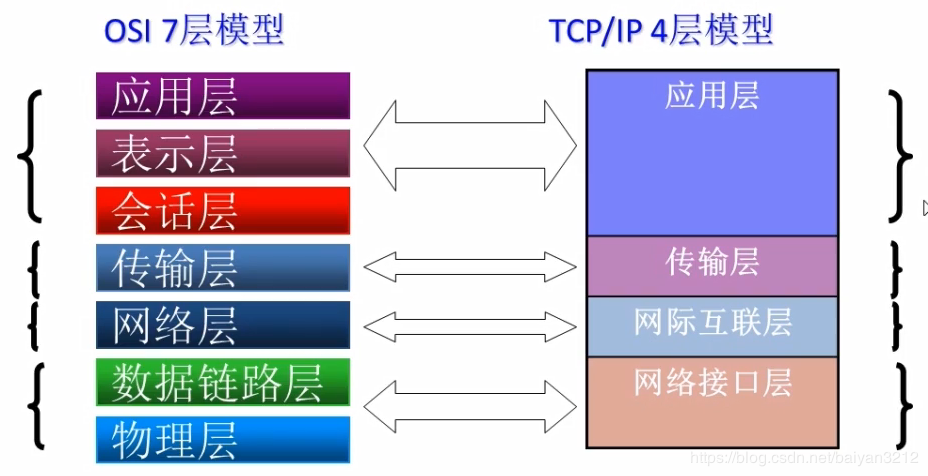
java中使用synchronized关键字实现同步锁；注意，synchronized关键字只能给对象类型加同步锁，不能给基本类型加同步锁；

使用同步锁是有系统开销的；

75.线程的通信机制？（wait-notify机制）

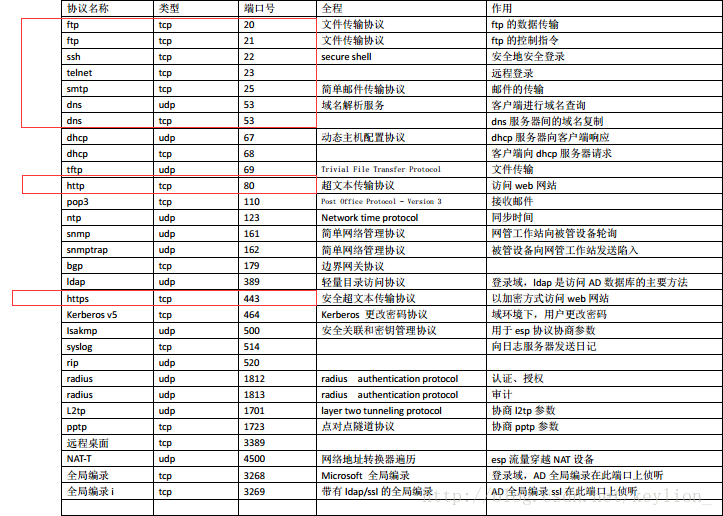
76.OIS7层模型和Tcp/Ip4层模型？

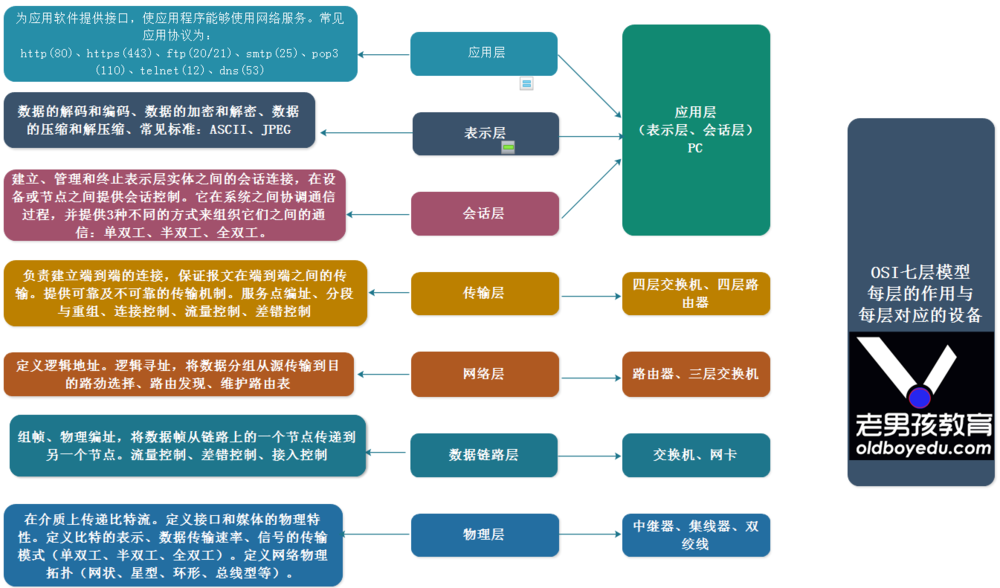




77.OSI7层模型中各层模型的主要协议，运行设备，端口号有哪些？







78．常用的网络通信协议？同上

79．TCP和UDP的区别？以及网络编程的实现方式？

（1： 1.基于连接与无连接；

2.对系统资源的要求（TCP较多，UDP少）；

3.UDP程序结构较简单；

4.流模式与数据报模式 ；

5.TCP保证数据正确性，UDP可能丢包，TCP保证数据顺序，UDP不保证。

（2：

TCP:

TCP编程的服务器端一般步骤是： TCP包头的最小长度，为20字节。

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt(); \* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();

　　4、开启监听，用函数listen()；

　　5、接收客户端上来的连接，用函数accept()；

　　6、收发数据，用函数send()和recv()，或者read()和write();

　　7、关闭网络连接；

　　8、关闭监听；

TCP编程的客户端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

　　4、设置要连接的对方的IP地址和端口等属性；

　　5、连接服务器，用函数connect()；

　　6、收发数据，用函数send()和recv()，或者read()和write();

　　7、关闭网络连接；UDP:与之对应的UDP编程步骤要简单许多，分别如下：

UDP编程的服务器端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();

　　4、循环接收数据，用函数recvfrom();

　　5、关闭网络连接；

UDP编程的客户端一般步骤是：

　　1、创建一个socket，用函数socket()；

　　2、设置socket属性，用函数setsockopt();\* 可选

　　3、绑定IP地址、端口等信息到socket上，用函数bind();\* 可选

　　4、设置对方的IP地址和端口等属性;

　　5、发送数据，用函数sendto();

　　6、关闭网络连接；

80.如何获取一个类的描类Class类的途径？（三种）

1:.class

2:.getClass()

3.Class.forName()

81.使用java在内存中创建一个新对象的方式？

new创建

对象流创建

反射创建

克隆创建

82.什么是反射？反射的作用和意义？什么是Class类？作用是什么？

(1: JAVA反射机制是在运行状态中，对于任意一个实体类，都能够获取这个类的所有属性和方法；对于任意一个对象，都能够调用它的任意方法和属性；这种动态获取类的构造信息以及动态调用实例对象的行为方法的功能称为java语言的反射机制。

(2:java的反射是动态的，在程序运行时才能获取的一些信息和操作；

java的反射机制是对类在运行时的操作，是第三方（外部）的操作，不是类或者实例对象本身的操作；

java的反射机制是采用一种解剖的视角去分析和控制一个类或实例对象的行为动作；不是这个类或实例本身的行为动作；

（3:Class类：

Class类是一个抽象封装类的类；

jvm在反射机制中将一个类的构成信息进行抽象封装，使用Class类来进行描述；

Class是对某一个类的抽象封装，Class是来自于一个具体的类在jvm执行时所获取信息的抽象封装；

Class 没有公共构造方法。Class 对象是在加载类时由 Java 虚拟机以及通过调用类加载器中的 defineClass 方法自动构造的。

（4: 反射机制的使用，就是通过获取某一个类或实例对象的Class对象，进而通过Class对象进行操作处理；

反射就是操作一个类或实例对象的Class；

注意带有Declared和不带有的get方法之间的区别：

不带有Declared的get方法只获取公共的内容；

带有Declared的get方法获取所有的内容，包括 public、protected、默认的、private；

83.什么是注解？注解和注释的区别？几种常用元注解？各有什么作用？

（1：注解是java程序中使用元数据对功能代码进行补充说明的一种编程方式；

注解是在jdk1.5以后才支持的；

注解使用使用在类、属性、方法、构造方法、局部变量、参数、枚举类型上，之前用的注解有 @override @Test

（2：注解与注释的区别：

注释：

是对程序的解释说明的内容，是给编程人员阅读理解代码使用；

注释不参与程序的编译运行；

注解：

是对程序的补充说明，是给jvm在编译运行程序时使用的“注释说明”

注解是参与程序的编译运行的；

★

注释是给编程人员看的，不参与程序的编译运行；

注解是给jvm看的，参与程序的编译运行；

★

（3：@Target

用于指定注解所可以修饰的范围

范围从枚举类 ElementType中获取

如果不写，默认应用范围是所有的可以使用注解的范围；

不写默认是应用于全部；

@Retention

用于指定自定义注解的生命周期长度；

通过RetentionPolicy类中的静态变量指定：

RetentionPolicy.SOURCE

源码级，jvm编译时会直接丢弃，不参与编译；

RetentionPolicy.CLASS 默认值

编译级，注解存活到编译期间，运行期间在被jvm丢弃；

RetentionPolicy.RUNTIME

运行级，注解存活到运行时，从源码、编译、运行注解都存活；

默认值是Class源码级别

@Documented

决定在使用javadoc自动生成的api文档中包不包含该注解信息；

默认不包含；使用该注解表示api中包含该注解的内容；

@Inherited

决定父类的注解可不可以被子类继承到；

默认是不继承；使用该注解表示子类可以继承；

84.常用的排序算法以及稳定性？对应的时间复杂度和空间复杂度分析？