简答题

Java 核心

1.基本语法

1、一个".java"源文件中是否可以包括多个类（不是内部类）？有什么限制

答：可以有多个类，但只能有一个 public 的类，并且 public 的类名必须与文件名相一致。

2、Java 有没有 goto?

java 中的保留字，现在没有在 java 中使用。

3、八种基本数据类型和包装类以及大小。

八种基本数据类型	包装类	大小
byte	Byte	8 位 -2^7 ~ 2^7-1
short	Short	16 位 -2^15 ~ 2^15-1
int	Integer	32 位 -2^31 ~ 2^31-1
long	Long	64 位 -2^63 ~ 2^63-1
float	Float	32 位 -2^31 ~ 2^31-1
double	Double	64 位 -2^63 ~ 2^63-1
char	Character	16 位 \u0000 ~ \uFFFF
boolean	Boolean	1 位

4、说说**&**和**&&**的区别。

&和&&都可以用作逻辑与的运算符，表示逻辑与（ and），当运算符两边的表达式的结果都为 true 时，整个运算结果才为 true，否则，只要有一方为 false，则结果为 false。

&&还具有短路的功能，即如果第一个表达式为 false，则不再计算第二个表达式，例如，对于 if(str != null&& !str.equals(“”))表达式，当 str 为 null 时，后面的表达式不会执行，所以不会出现 NullPointerException 如果将&&改为&，则会抛出 NullPointerException 异常。

&还可以用作位运算符，当&操作符两边的表达式不是 boolean 类型时， &我们通常使用 0x0f 来与一个整数进行&运算，来获取该整数的最低 4 个 bit 位，例如， 0x31 & 0x0f 的结果为 0x01。

5、用最有效率的方法算出 2 乘以 8 等於几?

2 << 3，

因为将一个数左移 n 位，就相当于乘以了 2 的 n 次方，那么，一个数乘以 8 只要将其左移 3 位即可，而位运算 cpu 直接支持的，效率最高，所以，2 乘以 8 等於几的最效率的方法是 2 <<

3。

6.能被 Switch 的数据类型有哪些？

在 switch（ expr1）中， expr1 只能是一个整数表达式或者枚举常量，
所以能被 Switch 的数据类型有： byte,short,int,char。
在 jdk1.7 之后 String 也可以。

7、char 型变量中能不能存贮一个中文汉字?为什么?

char 型变量是用来存储 Unicode 编码的字符的,unicode 编码字符集中包含了汉字，unicode 编码占用两个字节，所以 char 类型的变量也是占用两个字节。一个 char 类型的大小为 16 位也就是两个字节所以能存储一个汉字

2.面向对象

8、Overload 和 Override(OverWrite)有什么区别？Overload 的方法是否可以改变返回值的类型?

1）重载 Overload 是方法的重载，表示同一个类中可以有多个名称相同的方法，但这些方法的参数列表各不相同，这些不同包括位置不同，个数不同，类型不同。
2）重写 Override 是方法的重写，表示子类中的方法可以与父类中的某个方法的名称和参数完全相同，通过子类创建的实例对象调用这个方法时，将调用子类中的定义方法，这相当于把父类中定义的那个完全相同的方法给覆盖了,这也是面向对象编程的多态性的一种表现。
3）Override 子类覆盖父类的方法时，只能比父类抛出更少的异常和更小的异常，或者抛出父类抛出的异常的子异常。子类方法的访问权限只能和父类相同或者比父类的更大，不能更小。如果父类的方法是 private 类型，那么，子类则不存在覆盖的限制，相当于子类中增加了一个全新的方法。 覆盖的方法的返回值必须和被覆盖的方法的返回一致；

4)Overload 与返回值类型和修饰符以及异常无关只要方法名相同形式参数不同即为重载。

9、成员变量（全局变量）和局部变量的区别？

1，定义的位置。
局部变量定义在函数中，语句内。
成员变量定义在类中。
2，内存中的位置。
局部变量存储在栈内存中。
成员变量存储在堆内存中。
3，初始化值。
局部变量没有默认初始化值，必须赋值才可以使用。
成员变量有默认初始化值。
4，生命周期。
局部变量一旦作用区域结束，就立刻释放。
成员变量也称为实例(对象)变量,随着对象的出现而出现。随着对象的被回收而释放。

10、静态变量 和实例变量的区别？

在语法定义上的区别：静态变量前要加 static 关键字，而实例变量前则不加。

在程序运行时的区别：实例变量属于某个对象的属性，必须创建了实例对象，其中的实例变量才会被分配空间，才能使用这个实例变量。静态变量不属于某个实例对象，而是属于类，所以也称为类变量，只要程序加载了类的字节码，不用创建任何实例对象，静态变量就会被分配空间，静态变量就可以被使用了。总之，实例变量必须创建对象后才可以通过这个对象来使用，静态变量则可以直接使用类名来引用。

1，生命周期。

成员变量又叫实例变量，是随着对象的出现而出现，随着对象的消失而消失。

静态变量又叫类变量，是随着类的加载而出现，随着类的消失而消失。

2，内存中的存储位置。

成员变量存在在堆内存的对象中。

静态变量存储在方法区的静态区中。

3，存储的数据特点。

成员变量存储的数据是对象的特有数据。

静态变量存储的数据是对象的共享数据。

4，调用方式。

成员变量，只能被对象所调用。

静态变量，能被对象调用，也可以被类名调用

11、请说出作用域 public， private， protected，以及不写时的区别

说明：如果在修饰的元素上面没有写任何访问修饰符，则表示 friendly。

作用域	当前类	同一包（package）	子孙类	其他包（package）
public	√	√	√	√
protected	√	√	√	×
friendly	√	√	×	×
private	√	×	×	×

12、构造器 Constructor 能否被继承？是否可被 override?能否被 overload?

构造器 Constructor 不能被继承，因此不能重写 Override，但可以被重载 Overload。

13、final, finally, finalize 的区别

final 用于声明属性，方法和类，分别表示属性不可变，方法不可覆盖，类不可继承。

内部类要访问局部变量，局部变量必须定义成 final 类型。

finally 是异常处理语句结构的一部分，表示总是执行。

finalize 是 Object 类的一个方法，在垃圾收集器执行的时候会调用被回收对象的此方法，

可以覆盖此方法提供垃圾收集时的其他资源回收，例如关闭文件等。JVM 不保证此方法总被调用。

14、"=="和 equals 方法究竟有什么区别？

==操作符专门用来比较两个变量的值是否相等，也就是用于比较变量所对应的内存中所存储的数值是否相同，要比较两个基本类型的数据或两个引用变量是否相等，只能用==操作符。如果要比较两个变量是否指向同一个对象，即要看这两个变量所对应的内存中的数值是否相等，这时候就需要用==操作符进行比较。

equals 方法是用于比较两个独立对象的内容是否相同，就好比去比较两个人的长相是否相同，它比较的两个对象是独立的。

参见笔试题第十二题

15.是否可以从一个 static 方法内部发出对非 static 方法的调用？

不可以。因为非 static 方法是要与对象关联在一起的，必须创建一个对象后，才可以在该对象上进行方法调用，而 static 方法调用时不需要创建对象，可以直接调用。也就是说，当一个 static 方法被调用时，可能还没有创建任何实例对象，如果从一个 static 方法中发出对非 static 方法的调用，那个非 static 方法是关联到哪个对象上的呢？这个逻辑无法成立，所以，一个 static 方法内部不能对非 static 方法的调用。

16、接口是否可继承接口?抽象类是否可实现(implements)接口?抽象类是否可继承具体类(concrete class)?抽象类中是否可以有静态的 main 方法？

接口可以继承接口。抽象类可以实现(implements)接口，抽象类可以继承具体类。抽象类中可以有静态的 main 方法。

17、abstract class（抽象类）和 interface（接口）有什么区别?

抽象类：

（1）含有 abstract 修饰符的类即为抽象类，abstract 类不能创建的实例对象换句话说就是不能被 new。(2)包含有 abstract 方法的类必须定义为抽象类，抽象类中的可以有普通方法。(3)抽象类的子类必须全部覆盖该抽象类的所有的抽象方法，所以，不能有抽象构造方法或抽象静态方法。如果的子类没有实现抽象父类中的所有抽象方法，那么子类也必须定义为 abstract 类型。（4）Abstract 能修饰类，修饰方法，不能修饰属性

接口：

（1）接口（interface）可以说成是抽象类的一种特例，接口为更为纯粹的抽象类，不能有任何方法的实现，只能有方法的声明。（2）接口中的方法定义默认为 public abstract 类型，接口中的成员变量类型默认为 public static final。（3）实现类必须实现所有接口中的全部方法声明；（4）接口中没有构造函数，即接口没有对象；（5）接口是多实现，即一个类可以实现多个接口，中间用逗号隔开；

两者比较语法区别：

1.抽象类可以有构造方法，接口中不能有构造方法。

2.抽象类中可以有普通成员变量，接口中没有普通成员变量

3.抽象类中可以包含非抽象的普通方法，接口中的所有方法必须都是抽象的，不能有非抽象的普通方法。

4. 抽象类中可以包含静态方法，接口中不能包含静态方法

5. 抽象类和接口中都可以包含静态成员变量，抽象类中的静态成员变量的访问类型可以任意，但接口中定义的变量只能是 public static final 类型，并且默认即为 public static final 类型。

6. 一个类可以实现多个接口，但只能继承一个抽象类。

两者在应用上的区别：

接口更多的是在系统架构设计方法发挥作用，接口是用来定义规范的。主要用于定义模块之间的通信契约。而抽象类在代码实现方面发挥作用，可以实现代码的重用，实现了一种价值叫模版策略。

18、java 中实现多态的机制是什么？

多态可分为：

1.编译多态：主要是体现在重载，系统在编译时就能确定调用重载函数的哪个版本。

2.运行多态：主要体现在 OO 设计的继承性上，子类的对象也是父类的对象，即上溯造型，所以子类对象可以作为父类对象使用，父类的对象变量可以指向子类对象。因此通过一个父类发出的方法调用可能执行的是方法在父类中的实现，也可能是某个子类中的实现，它是由运行时刻具体的对象类型决定的。

靠的是父类或接口定义的引用变量可以指向子类或具体实现类的实例对象，而程序调用的方法在运行期才动态绑定，就是引用变量所指向的具体实例对象的方法，也就是内存里正在运行的那个对象的方法，而不是引用变量的类型中定义的方法。

19、什么是内部类？Static Nested Class 和 Inner Class 的不同

(1)内部类就是在一个类的内部定义的类，内部类分为静态内部类，非静态内部类，局部内部类和匿名内部类。

(2)非静态内部类可以无条件访问外部类的成员，不可以和自己的外部类重名但是可以与外部类平级同包的类重名，非静态内部类不能有 static 修饰的变量，

(3)静态内部类不能访问外部类的非静态属性，外部类也不能访问静态内部类的成员，但是可以使用静态内部类的类名作为调用者使用。

(4)内部类可以通过 new 外部类().new 内部类()进行实例化 Inner in=new Outer().new Inner();

内部类可以被其他类继承继续它的类需要写一个构造方法并且手工调用外部类.super();

```
public class SubClass extends Out.In{
        public SubClass(Out out){
            out.super();
```

```
        }
    }
```

20、面向对象的特征有哪些方面

21、 Integer 与 int 的区别

（1）int 是 java 提供的 8 种原始数据类型之一。Java 为每个原始类型提供了封装类，Integer 是 java 为 int 提供的封装类。int 的默认值为 0，而 Integer 的默认值为 null，即 Integer 可以区分出未赋值和值为 0 的区别， int 则无法表达出未赋值的情况。
（2）在 JSP 开发中， Integer 的默认为 null，所以用 el 表达式在文本框中显示时，值为空白字符串，而 int 默认的默认值为 0，所以用 el 表达式在文本框中显示时，结果为 0，所以， int 不适合作为 web 层的表单数据的类型。
（3）在 Hibernate 中，如果将 OID 定义为 Integer 类型，那么 Hibernate 就可以根据其值是否为 null 而判断一个对象是否是临时的，如果将 OID 定义为了 int 类型，还需要在 hbm 映射文件中设置其 unsaved-value 属性为 0。
（4）另外，Integer 提供了多个与整数相关的操作方法，例如，将一个字符串转换成整数， Integer 中还定义了表示整数的最大值和最小值的常量。

22、写 clone()方法时，通常都有一行代码，是什么？

必须实现 Cloneable 接口 重写 clone 方法

```
protected Object clone() throws CloneNotSupportedException {
        return super.clone();
    }
```

23、是否可以继承String类?

String 类是 final 修饰的。因此不可以继承这个类、不能修改这个类。为了提高效率节省空间，我们应该用 StringBuffer 类

24、String s = "Hello";s = s + " world!";这两行代码执行后，原始的String对象中的内容到底变了没有？

没有。因为 String 被设计成不可变类，所以它的所有对象都是不可变对象。在这段代码中，s 原先指向一个 String 对象，内容是 "Hello"，然后我们对 s 进行了+操作，这时，s 不指向原来那个对象了，而指向了另一个 String 对象，内容为"Hello world!"，原来那个对象还存在于内存之中，只是 s 这个引用变量不再指向它了。

25、String s = new String("xyz");创建了几个String Object?二者之间有什么区别？

两个或一个，”xyz”对应一个对象，这个对象放在字符串常量缓冲区，常量”xyz”不管出现多少遍，都是缓冲区中的那一个。New String 每写一遍，就创建一个新的对象，它一句那个常量”xyz”对象的内容来创建出一个新 String 对象。如果以前就用过’xyz’，这句代表就不会创建”xyz”自己了，直接从缓冲区拿。

3.集合相关

26、String 和 StringBuffer、StringBuilder 的区别

(1) String 和 StringBuffer、StringBuilder 它们可以储存和操作字符串，即包含多个字符的字符数据。String 类型和 StringBuffer 类型的主要性能区别其实在于 String 是不可变的对象, 因此在每次对 String 类型进行改变的时候其实都等同于生成了一个新的 String 对象，然后将指针指向新的 String 对象，所以经常改变内容的字符串最好不要用 String ，因为每次生成对象都会对系统性能产生影响，特别当内存中无引用对象多了以后， JVM 的 GC 就会开始工作，那速度是一定会相当慢的。 如果是使用 StringBuffer 类则结果就不一样了，每次结果都会对 StringBuffer 对象本身进行操作，而不是生成新的对象，再改变对象引用

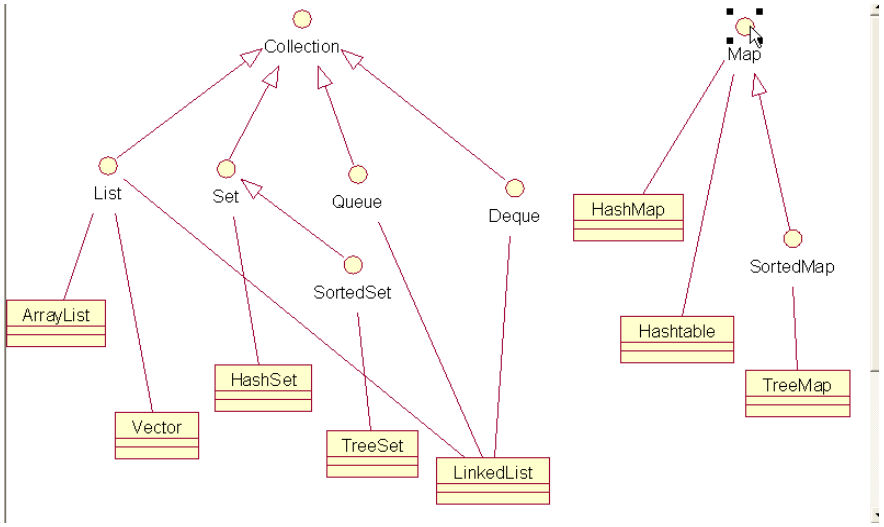
(2) StringBuffer 和 StringBuilder 中 StringBuffer 是线程安全的 StringBuilder 是线程不安全的，三者之间执行的速度是 StringBuilder>StringBuffer>String;

(3)另外，String 实现了 equals 方法，new String(“abc”).equals(newString(“abc”)的结果为 true,而 StringBuffer 没有实现 equals 方法,所以 new StringBuffer(“abc”).equals(newStringBuffer(“abc”)的结果为 false。

27、数组有没有 length()这个方法? String 有没有 length()这个方法？

数组没有 length()这个方法，有 length 的属性。String 有有 length()这个方法。

28、介绍Collection框架的结构



可以更详细的叙述每个集合。

29、ArrayList和Vector和LinkedList的区别？

1）ArrayList 和Vector底层实现是基于数组的实现 而LinkedList底层实现是基于链表的实现。
2）ArrayList和Vector 实现了List接口（List继承于Collection接口）、而LinkedList除了实现了List接口以外还实现了Deque接口。
3）他们都是有序集合，即存储在集合中的元素的位置都是有顺序的，我们以后可以按位置索引号取出某个元素，，并且其中的数据是允许重复的。
4）从同步性来说 Vector是属于线程安全的而ArrayList 和LinkedList不是线程安全的
5）从效率来说ArrayList和Vector底层属于数组 查找修改速度快 增加删除效率慢
LinkedList底层是链表 所以 增删比较快 查改 慢
6）（这个可以选择回答）从数据增长来说：ArrayList与Vector都有一个初始的容量大小，当存储进它们里面的元素的个数超过了容量时。在jdk1.7中 ArrayList底层实现增长的方式是int newCapacity = oldCapacity + (oldCapacity >> 1); Vector实现增长的方式是int newCapacity = oldCapacity + ((capacityIncrement > 0) ? capacityIncrement : oldCapacity);

30、HashMap和Hashtable的区别

1）继承不同。
public class Hashtable extends Dictionary implements Map
public class HashMap extends AbstractMap implements Map
2）Hashtable 中的方法是同步的(线程安全的)，而 HashMap 中的方法在缺省情况下是非同步的（线程不安全的）。在多线程并发的环境下，可以直接使用 Hashtable，但是要使用 HashMap 的话就要自己增加同步处理了
3）Hashtable不允许使用null 作为key和value，否则会引发空指针异常，HashMap允许空（null）作为键值（key）和Value,但最多只能有一项key-value中key为null，但是key-value中可以有多个value为null

31、Collection和 Collections的区别

Collection 是集合类的上级接口，继承与他的接口主要有 Set 和 List.

Collections 是针对集合类的一个帮助类，他提供一系列静态方法实现对各种集合的搜索、排序、线程安全化等操作。

32、Set 里的元素是不能重复的，那么用什么方法来区分重复与否呢?是用==还是 equals()?它们有何区别?

Set 里的元素是不能重复的，元素重复与否是使用 equals()方法进行判断的。equals()和==方法决定引用值是否指向同一对象 equals()在类中被覆盖，为的是当两个分离的对象的内容和类型相配的话，返回真值。

33、象值相同(x.equals(y) == true)，但却可有不同的 hash code，这句话对不对?

对。

如果对象要保存在 HashSet 或 HashMap 中，它们的 equals 相等，那么，它们的 hashcode 值就必须相等。

如果不是要保存在 HashSet 或 HashMap，则与 hashcode 没有什么关系了，这时候 hashcode 不等是可以的，例如 ArrayList 存储的对象就不用实现 hashcode，但是通常都会去实现的。

34、TreeSet 里面放对象，如果同时放入了父类和子类的实例对象，那比较时使用的是父类的 compareTo 方法，还是使用的子类的 compareTo 方法，还是抛异常！

当前的 add 方法放入的是哪个对象，就调用哪个对象的 compareTo 方法，至于这个 compareTo 方法怎么做，就看当前这个对象的类中是如何编写这个方法的。

```
public class Parent implements Comparable {
    private int age = 0;

    public Parent(int age) {
        this.age = age;
    }

    public int compareTo(Object o) {
        System.out.println("method of parent");
        Parent o1 = (Parent) o;
        return age > o1.age ? 1 : age < o1.age ? -1 : 0;
    }

}
```

```
public class Child extends Parent {
    public Child() {
        super(3);
    }
    public int compareTo(Object o) {

        System.out.println("methodof child");
        return 1;
    }
}

public class TreeSetTest {
    public static void main(String[] args) {
        TreeSet set = new TreeSet();
        set.add(new Parent(3));
        set.add(new Child());
        set.add(new Parent(4));
        System.out.println(set.size());
    }
}
```

执行结果：

```
method of parent
methodof child
method of parent
method of parent
3
```

4.异常相关

35、运行时异常与一般异常有何异同？

异常表示程序运行过程中可能出现的非正常状态，运行时异常表示虚拟机的通常操作中可能遇到的异常，是一种常见运行错误。java 编译器要求方法必须声明抛出可能发生的非运行时异常或者捕获，但是并不要求必须声明抛出未被捕获的运行时异常。

36、error和exception有什么区别?

error 表示恢复不是不可能但很困难的情况下的一种严重问题。是 java 运行环境中的内部错误或者硬件问题。比如：内存资源不足等。对于这种错误，程序基本无能为力比如说内存溢出。 exception 表示一种设计或实现问题。它处理的是因为程序设计的瑕疵而

引起的问题或者外在的输入等引起的一般性问题，是程序必须处理的。

37、请写出你最常见到的5个runtime exception。

NullPointerException、ArrayIndexOutOfBoundsException、ClassCastException。

ArithmeticException 、IndexOutOfBoundsException、NumberFormatException、ConcurrentModificationException、EmptyStackException

38.捕获到异常后如何处理？异常如何处理？

第一看异常的类型

第二根据异常的类型进行不同类别的处理 可以使用 try catch 捕获 然后还可以使用 throws 和 throw 抛出异常

第三打印一下异常信息查阅是否可以正常处理

第四 记录日志信息 把错误的时间 错误发生的代码所在处于 错误的原因都记录到日志中

第五查阅日志相关的内容 提出不同解决方案

5.线程相关

38、什么是线程？

线程是操作系统能够进行运算调度的最小单位，它被包含在进程之中，是进程中的实际运作单位。程序员可以通过它进行多处理器编程，你可以使用多线程对运算密集型任务提速。比如，如果一个线程完成一个任务要 100 毫秒，那么用十个线程完成改任务只需 10 毫秒。Java 在语言层面对多线程提供了卓越的支持。

39、线程和进程有什么区别？

进程是处于运行过程中的程序，并且具有一定独立功能，进程是系统进行资源分配和调度的一个独立单位。进程一般有三个特性：独立性，动态性，并发性。

线程是进程的子集，一个进程可以有很多线程，每条线程并行执行不同的任务。不同的进程使用不同的内存空间，而所有的线程共享一片相同的内存空间。每个线程都拥有单独的栈内存用来存储本地数据。所以进程之间不能共享内存但是线程之间可以；系统创建进程需要为该进程分配系统资源，创建线程代价比较小，因此使用多线程来实现多任务并发比进程效率高。

40、线程有几种创建方式?这几种方式对比有和优缺点如何选择？

在语言层面有两种方式。java.lang.Thread 类的实例就是一个线程但是它需要调用 java.lang.Runnable 接口来执行，由于线程类本身就是调用的 Runnable 接口所以可以继承 java.lang.Thread 类或者直接调用 Runnable 接口来重写 run()方法实现线程。
两种方式比较：
采用实现 runnable 接口：
优势：a、线程继承了 Runnable 接口还可以继承其他类。b、这种方式下可以多个线程共享一个 target 对象,所以非常适合多个相同的线程来处理同一份资源，从而将 cpu，代码和数据分开，形成清晰的模型，较好的体现了面向对象的思想。
劣势：编程稍微复杂，如果需要访问当前线程必须使用 Thread.currentThread()方法。
采用继承 Thread 类：
优势：编程相对简单，如果需要访问当前线程无须使用 Thread.currentThread()方法，直接使用 this 即可获得当前线程。
劣势：因为线程已经继承了 Thread 类，所以不能在继承其他父类。
所以综合推荐使用 Runnable 接口。

41、Thread 类中的 start() 和 run() 方法有什么区别？

start()方法被用来启动新创建的线程，而且 start()内部调用了 run()方法，这和直接调用 run()方法的效果不一样。当你调用 run()方法的时候，只会是在原来的线程中调用，没有新的线程启动，start()方法才会启动新线程。

42、sleep()和 wait()有什么区别?

sleep 是线程类（Thread）的方法，导致此线程暂停执行指定时间，给执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用 sleep 不会释放对象锁。 wait 是 Object 类的方法，对此对象调用 wait 方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出 notify 方法（或 notifyAll）后本线程才进入对象锁定池准备获得对象锁进入运行状态。

1)sleep 必须指定时间，wait 可以指定可以不指定。
2)sleep 和 wait 都可以让线程处于冻结状态，释放执行权。(相同点)
3)持有锁的线程执行 sleep，不释放锁，持有锁的线程执行到 wait 释放锁。
4)sleep 到时间会自动醒，wait 没有指定时间，只能被其他线程通过 notify 唤醒。

43、简单介绍一下线程的生命周期

当线程被创建并且启动后，它经历了 5 种状态：新建、就绪、运行、阻塞和死亡状态。
当线程对象被创建出来是进入了新建状态，和其他的 java 对象一样，仅仅是由 java 虚拟机为其分配了内存。当调用了 start 方法后，线程进入就绪状态。其实线程内部还

是依赖 JVM 的调度，当调用了 start 方法后，JVM 会认为这个线程可以执行，至于什么时候执行取决于 JVM 的内部调度。

当线程在运行的时候，不能一直占有 CPU 时间片，CPU 会在多个线程之间进行调度，线程的状态也会多次切换于阻塞和运行状态。

44、线程在哪些情况下进入阻塞状态？通过什么方式又进入运行状态？

如果就绪状态的线程获取了 CPU，那么这个线程处于运行状态，当这个线程运行时，不会一直霸占 CPU，线程在执行的过程中会被在 CPU 上调度下来，以便其他线程能够获取执行机会。

线程进入阻塞状态的情况：

- 线程调用一个阻塞方法，方法返回前该线程一直阻塞。
- 线程调用 sleep 方法进入阻塞。
- 线程尝试获取同步监视器，但该同步监视器被其他线程持有。
- 线程调用了 suspend 方法挂起。

线程解除阻塞，重新进入就绪状态的情况：

- 调用的阻塞方法返回。
- 调用的 sleep 到期。
- 线程成功获取同步监视器。
- 被 suspend 的方法挂起的线程被调用了 resume 方法恢复。

45、Java 中 Runnable 和 Callable 有什么不同

Runnable 和 Callable 都代表那些要在不同的线程中执行的任务。Runnable 从 JDK1.0 开始就有了，Callable 是在 JDK1.5 增加的。它们的主要区别是 Callable 的 call() 方法可以返回值和抛出异常，而 Runnable 的 run()方法没有这些功能。Callable 可以返回装载有计算结果的 Future 对象。

不同之处总结：

1.Callable 可以返回一个类型 V，而 Runnable 不可以

2.Callable 能够抛出 checked exception,而 Runnable 不可以。

3.Callable 和 Runnable 都可以应用于 executors。而 Thread 类只支持 Runnable.

4.Callable 与 executors 联合在一起，在任务完成时可立刻获得一个更新了的 Future。而 Runable 却要自己处理。

46、Java 中 CyclicBarrier 和 CountDownLatch 有什么不同？

CountDownLatch 类位于 java.util.concurrent 包下，利用它可以实现类似计数器的功能。比如有一个任务 A，它要等待其他 4 个任务执行完毕之后才能执行，此时就可以利用 CountDownLatch 来实现这种功能了。

CyclicBarrier 通过它可以实现让一组线程等待至某个状态之后再全部同时执行。叫做回环是因为当所有等待线程都被释放以后，CyclicBarrier 可以被重用。我们暂且把这个状态就叫做 barrier，当调用 await()方法之后，线程就处于 barrier 了。

CyclicBarrier 和 CountDownLatch 都可以用来让一组线程等待其它线程。不同的是，CyclicBarrier 可以多次使用，CountdownLatch 不能重新使用，只能用一次（为 0 后不可变）

47、Java 中的 volatile 变量是什么？

volatile 是一个特殊的修饰符，只有成员变量才能使用它。在 Java 并发程序缺少同步类的情况下，多线程对成员变量的操作对其它线程是透明的。volatile 变量可以保证下一个读取操作会在前一个写操作之后发生。volatile 是轻量级的锁，它只具备可见性，但没有原子特性。用 volatile 声明的变量，它的同步特性，简单来讲就是对该变量的单个读/写是同步的。

48、 Java 中如何停止一个线程？

Java 提供了很丰富的 API 但没有为停止线程提供 API。JDK 1.0 具有 stop(), suspend()和 resume()的控制方法但是由于潜在的死锁威胁因此在后续的 JDK 版本中他们被弃用了，之后 Java API 的设计者就没有提供一个兼容且线程安全的方法来停止一个线程。当 run() 或者 call() 方法执行完的时候线程会自动结束,如果要手动结束一个线程，可以用 volatile 布尔变量来退出 run()方法的循环或者是取消任务来中断线程。

49、一个线程运行时发生异常会怎样？

如果异常没有被捕获该线程将会停止执行。Thread.UncaughtExceptionHandler 是用于处理未捕获异常造成线程突然中断情况的一个内嵌接口。当一个未捕获异常将造成线程中断的时候 JVM 会使用 Thread.getUncaughtExceptionHandler()来查询线程的 UncaughtExceptionHandler 并将线程和异常作为参数传递给 handlerUncaughtException()方法进行处理。

50、 Java 中 notify 和 notifyAll 有什么区别？

为多线程可以等待单监控锁，Java API 的设计人员提供了一些方法当等待条件改变的时候通知它们，但是这些方法没有完全实现。notify()方法不能唤醒某个具体的线程，所以只有一个线程在等待的时候它才有用武之地。而 notifyAll()唤醒所有线程并允许他们争夺锁确保了至少有一个线程能继续运行。

51、什么是 ThreadLocal 变量？

ThreadLocal 是 Java 里一种特殊的变量。每个线程都有一个 ThreadLocal 就是每个线程都拥有了自己独立的一个变量，竞争条件被彻底消除了。它是为创建代价高昂的对象获取线程安全的好方法，比如你可以用 ThreadLocal 让 SimpleDateFormat 变成线程安全的，因为那个类创建代价高昂且每次调用都需要创建不同的实例所以不值得在局部范围使用它，如果为每个线程提供一个自己独有的变量拷贝，将大大提高效率。首先，通过复用减少了代价高昂的对象的创建个数。其次，你在没有使用高代价的同步或者不变性的情况下获得了线程安全。线程局部变量的另一个不错的例子是 ThreadLocalRandom 类，它在多线程环境中减少了创建代价高昂的 Random 对象的个数。

52、 什么是 FutureTask？

在 Java 并发程序中 FutureTask 表示一个可以取消的异步运算。它有启动和取消运算、查询运算是否完成和取回运算结果等方法。只有当运算完成的时候结果才能取回，如果运算尚未完成 get 方法将会阻塞。一个 FutureTask 对象可以对调用了 Callable 和 Runnable 的对象进行包装，由于 FutureTask 也是调用了 Runnable 接口所以它可以提交给 Executor 来执行。

53、Java 中 interrupted 和 isInterruptedd 方法的区别？

interrupted() 和 isInterrupted()的主要区别是前者会将中断状态清除而后者不会。Java 多线程的中断机制是用内部标识来实现的，调用 Thread.interrupt()来中断一个线程就会设置中断标识为 true。当中断线程调用静态方法 Thread.interrupted()来检查中断状态时，中断状态会被清零。而非静态方法 isInterrupted()用来查询其它线程的中断状态且不会改变中断状态标识。简单的说就是任何抛出 InterruptedException 异常的方法都会将中断状态清零。无论如何，一个线程的中断状态有有可能被其它线程调用中断来改变。

54、什么是线程池？ 为什么要使用它？

创建线程要花费昂贵的资源和时间，如果任务来了才创建线程那么响应时间会变长，而且一个进程能创建的线程数有限。为了避免这些问题，在程序启动的时候就创建若干线程来响应处理，它们被称为线程池，里面的线程叫工作线程。从 JDK1.5 开始，Java API 提供了 Executor 框架让你可以创建不同的线程池。比如单线程池，每次处理一个任务；数目固定的线程池或者是缓存线程池（一个适合很多生存期短的任务的程序的可扩展线程池）。

55、 如何避免死锁？

Java 多线程中的死锁 死锁是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。这是一个严重的问题，因为死锁会让你的程序挂起无法完成任务，死锁的发生必须满足以下四个条件：

互斥条件：一个资源每次只能被一个进程使用。

请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

不剥夺条件：进程已获得的资源，在末使用完之前，不能强行剥夺。

循环等待条件：若干进程之间形成一种头尾相接的循环等待资源关系。

避免死锁最简单的方法就是阻止循环等待条件，将系统中所有的资源设置标志位、排序，规定所有的进程申请资源必须以一定的顺序（升序或降序）做操作来避免死锁

56、Java 中 Semaphore 是什么？

Java 中的 Semaphore 是一种新的同步类，它是一个计数信号。从概念上讲，从概念上讲，信号量维护了一个许可集合。如有必要，在许可可用前会阻塞每一个 acquire()，然后再获取该许可。每个 release()添加一个许可，从而可能释放一个正在阻塞的获取者。但是，不使用实际的许可对象， Semaphore 只对可用许可的号码进行计数，并采取相应的行动。信号量常常用于多线程的代码中，比如数据库连接池。

57、Java 线程池中 submit() 和 execute()方法有什么区别？

两个方法都可以向线程池提交任务， execute()方法的返回类型是 void，它定义在 Executor 接口中，而 submit()方法可以返回持有计算结果的 Future 对象，它定义在 ExecutorService 接口中，它扩展了 Executor 接口，其它线程池类像 ThreadPoolExecutor 和 ScheduledThreadPoolExecutor 都有这些方法。

6、IO 和 GC

58、java 中有几种类型的流？JDK 为每种类型的流提供了一些抽象类以供继承，请说出他们分别是哪些类？

字节流，字符流。字节流继承于 InputStream OutputStream，字符流继承于 InputStreamReader

和 OutputStreamWriter。在 java.io 包中还有许多其他的流，主要是为了提高性能和使用方便。

59、字符流和字节流的区别

字节流与字符流

字节流：主要由 InputStream 和 OutStream 作为基类

字符流：主要由 Reader 和 Writer 作为基类

字节流和字符流的用法几乎一样，区别在于字节流和字符流所操作的数据单元不同，字节流操作的数据单元是 8 位的字节，而字符流操作的数据单元是 16 位的字符。

60、什么是 java 序列化，如何实现 java 序列化？或者请解释 Serializable 接口的作用。

我们有时候将一个 java 对象变成字节流的形式传出去或者从一个字节流中恢复成一个 java 对象，例如，要将 java 对象存储到硬盘或者传送给网络上的其他计算机，这个过程我们可以自己写代码去把一个 java 对象变成某个格式的字节流再传输，但是，jre 本身就提供了这种支持，我们可以调用 OutputStream 的 writeObject 方法来做，如果要让 java 帮我们做，要被传输的对象必须实现 serializable 接口，这样，javac 编译时就会进行特殊处理，编译的类才可以被 writeObject 方法操作，这就是所谓的序列化。需要被序列化的类必须实现 Serializable 接口，该接口是一个 mini 接口，其中没有需要实现的方法，implements Serializable 只是为了标注该对象是可被序列化的。

例如，在 web 开发中，如果对象被保存在了 Session 中，tomcat 在重启时要把 Session 对象序列化到硬盘，这个对象就必须实现 Serializable 接口。如果对象要经过分布式系统进行网络传输或通过 rmi 等远程调用，这就需要在网络上传输对象，被传输的对象就必须实现 Serializable 接口。

61、GC 是什么?为什么要有 GC?

GC 是垃圾收集的意思（Gabage Collection）,内存处理是编程人员容易出现问题的地方，忘记或者错误的内存回收会导致程序或系统的不稳定甚至崩溃，Java 提供的 GC 功能可以自动监测对象是否超过作用域从而达到自动回收内存的目的，Java 语言没有提供释放已分配内存的显示操作方法。

62、垃圾回收的优点和原理。并考虑2种回收机制。

Java 语言中一个显著的特点就是引入了垃圾回收机制，它使得 Java 程序员在编写程序的时候不再需要考虑内存管理。由于有个垃圾回收机制，Java 中的对象不再有"作用域"的概念，只有对象的引用才有"作用域"。垃圾回收可以有效的防止内存泄露，有效的使用可以使用的内存。垃圾回收器通常是作为一个单独的低级别的线程运行,不可预知的情况下对内存堆中已经死亡的或者长时间没有使用的对象进行清楚和回收,程序员不能实时的调用垃圾回收器对某个对象或所有对象进行垃圾回收。回收机制有分代复制垃圾回收和标记垃圾回收，增量垃圾回收。

63、垃圾回收器的基本原理是什么？垃圾回收器可以马上回收内存吗？有什么办法主动通

知虚拟机进行垃圾回收？

对于 GC 来说，当程序员创建对象时，GC 就开始监控这个对象的地址、大小以及使用情况。通常，GC 采用有向图的方式记录和管理堆(heap)中的所有对象。通过这种方式确定哪些对象是"可达的"，哪些对象是"不可达的"。当 GC 确定一些对象为"不可达"时，GC 就有责任回收这些内存空间。可以。程序员可以手动执行 System.gc()，通知 GC 运行，但是 Java 语言规范并不保证 GC 一定会执行。

64、java 中会存在内存泄漏吗，请简单描述。

所谓内存泄露就是指一个不再被程序使用的对象或变量一直被占据在内存中。java 中有垃圾回收机制，它可以保证一对象不再被引用的时候，即对象编程了孤儿的时候，对象将自动被垃圾回收器从内存中清除掉。由于 Java 使用有向图的方式进行垃圾回收管理，可以消除引用循环的问题，例如有两个对象，相互引用，只要它们和根进程不可达的，那么 GC 也是可以回收它们的。

7、高级

1.Java 内存模型是什么？

Java 内存模型规定和指引 Java 程序在不同的内存架构、CPU 和操作系统间有确定性地行为。它在多线程的情况下尤其重要。Java 内存模型对一个线程所做的变动能被其它线程可见提供了保证，它们之间是先行发生关系。这个关系定义了一些规则让程序员在并发编程时思路更清晰。比如，先行发生关系确保了：

- 线程内的代码能够按先后顺序执行，这被称为程序次序规则。
- 对于同一个锁，一个解锁操作一定要发生在时间上后发生的另一个锁定操作之前，也叫做管程锁定规则。
- 前一个对 volatile 的写操作在后一个 volatile 的读操作之前，也叫 volatile 变量规则。
- 一个线程内的任何操作必需在这个线程的 start()调用之后，也叫作线程启动规则。
- 一个线程的所有操作都会在线程终止之前，线程终止规则。
- 一个对象的终结操作必需在这个对象构造完成之后，也叫对象终结规则。
- 可传递性

2. J2EE 的 13 个规范

什么是 J2EE?

在企业级应用中,都有一些通用企业需求模块,如数据库连接,邮件服务,事务处理等.既然很多企业级应用都需要这些模块,一些大公司便开发了自己的通用模块服务,即中间件.这样一来,

就避免了重复开发,开发周期长和代码可靠性差等问题.但是,各公司的中间件不兼容的问题就出现了,用户无法将它们组装在一起为自己服务.于是,"标准"就应运而生了.J2EE 就是基于 Java 技术的一系列标准.

J2EE 的 13 种规范是什么?

1.JDBC(JavaDatabase Connectivity)

JDBC 是以统一方式访问数据库的 API.它提供了独立于平台的数据库访问,也就是说,有了 JDBC API,我们就不必为访问 Oracle 数据库专门写一个程序,为访问 Sybase 数据库又专门写一个程序等等,只需要用 JDBC API 写一个程序就够了,它可以向相应数据库发送 SQL 调用.JDBC 是 Java 应用程序与各种不同数据库之间进行对话的方法的机制.简单地说,它做了三件事:与数据库建立连接--发送操作数据库的语句--处理结果.

2.JNDI(JavaName and Directory Interface)

JNDI 是一组在 Java 应用中访问命名和目录服务的 API. 它提供了标准的独立于命名系统的 API,这些 API 构建在命名系统之上.这一层有助于将应用与实际数据源分离,因此不管是访问的 LDAP,RMI 还是 DNS.也就是说,JNDI 独立于目录服务的具体实现,只要有目录的服务提供接口或驱动,就可以使用目录.

3.EJB(EnterpriseJavaBean)

J2EE 将业务逻辑从客户端软件中抽取出来，封装在一个组件中。这个组件运行在一个独立的服务器上，客户端软件通过网络调用组件提供的服务以实现业务逻辑，而客户端软件的功能只是负责发送调用请求和显示处理结果。在 J2EE 中，这个运行在一个独立的服务器上，并封装了业务逻辑的组件就是 EJB 组件。其实就是把原来放到客户端实现的代码放到服务器端

4.RMI(Remote MethodInvoke)

是一组用户开发分布式应用程序的 API.这一协议调用远程对象上的方法使用了序列化的方式在客户端和服务器之间传递数据,使得原先的程序在同一操作系统的方法调用,变成了不同操作系统之间程序的方法调用,即 RMI 机制实现了程序组件在不同操作系统之间的通信.它是一种被 EJB 使用的更底层的协议.，并依靠 RMI 进行通信。

5.JavaIDL/CORBA(Common Object Request BrokerArchitecture)

Java 接口定义语言/公用对象请求代理程序体系结构在 JavaIDL 的支持下，开发人员可以将 Java 和 CORBA 集成在一起。他们可以创建 Java 对象并使之可在 CORBA ORB 中展开,或者他们还可以创建 Java 类并作为和其它 ORB 一起展开的 CORBA 对象的客户

6.JSP(Java Server Pages)

JSP 页面=HTML+Java,其根本是一个简化的 Servlet 设计.服务器在页面被客户端请求后,对这些 Java 代码进行处理,然后将执行结果连同原 HTML 代码生成的新 HTML 页面返回给客户端浏览器.

7.Java Servlet

Servlet 是一种小型的 Java 程序,扩展了 Web 服务器的功能,作为一种服务器的应用,当被请求时开始执行.Servlet 提供的功能大多和 JSP 类似,不过,JSP 通常是大多数的 HTML 代码中嵌入少量的 Java 代码,而 Servlet 全部由 Java 写成并生成 HTML.

8.XML

XML 是一个用来定义其它标记语言的语言,可用作数据共享。XML 的发展和 Java 是相互独立的。不过，它和 Java 具有的相同目标就是跨平台。通过将 Java 与 XML 结合，我们可以得到一个完全与平台无关的解决方案。

9.JMS(JavaMessage Service)

它是一种与厂商无关的 API,用来访问消息收发系统消息.它类似于 JDBC.JDBC 是可以用来访问不同关系数据库的 API,而 JMS 则提供同样与厂商无关的访问消息收发服务的方法,这样就可以通过消息收发服务实现从一个 JMS 客户机向另一个 JMS 客户机发送消息,所需要的是厂商支持 JMS.换句话说,JMS 是 Java 平台上有关面向消息中间件的技术规范

10.JTA(JavaTransaction API)

定义了一种标准 API,应用程序由此可以访问各种事务监控.它允许应用程序执行分布式事务处理--在两个或多个网络计算机资源上访问并且更新数据.JTA 和 JTS 为 J2EE 平台提供了分布式事务服务.

JTA 事务比 JDBC 事务更强大,一个 JTA 事务可以有多个参与者,而一个 JDBC 事务则被限定在一个单一的数据库连接.

11.JTS(JavaTransaction Service)

JTS 是 CORBA OTS 事务监控器的一个基本实现。JTS 指定了一个事务管理器的实现（Transaction Manager），这个管理器在一个高级别上支持 JTA 规范，并且在一个低级别上实现了 OMGOTS 规范的 Java 映射。一个 JTS 事务管理器为应用服务器、资源管理器、standalone 应用和通信资源管理器提供事务服务。

12.JavaMail

用于访问邮件服务器的 API,提供了一套邮件服务器的抽象类.

13.JAF(JavaBeansActivation Framework)

JAF 是一个专用的数据处理框架,它用于封装数据,并为应用程序提供访问和操作数据的接口.也就是说,JAF 让 Java 程序知道怎么对一个数据源进行查看,编辑,打印等.

笔试题

1.下面代码执行的结果是多少

```
System.out.println('a'+1);
System.out.println("a"+1);
```

答： 98 和 a1

2. 下列代码①处和②处正确吗？为什么

```
byte b,b1 = 5,b2 = 2;
 b = 5 + 2;①
b = b1 + b2;②
```

答： 一处正确 二处 错误 因为两个 byte 数据类型相加是不确定的所以需要类型转换

3. 下列代码执行的结果是多少？

```
    int z=0;
    z=z++;
    z=z++;
    z=z++;
System.out.println(z);
```

结果：0

解析：z = z++;//temp = z=0 z++= 0 + 1; z = temp=0;

4. 下面 ①和②能否编译通过为什么？

```
short s = 4;
```

```
s = s + 5;①
s += 5;②
```

解析：一处编译失败，整形的默认类型是 int,一个 short 类型和 int 类型相加得到是 int 所以一处需要强制类型转换，二出因为是赋值运算符能自动完成类型转换所以正确。

5.下面代码执行完毕的结果是多少？

```
boolean b1= true?false:true==true?false:true;
boolean b2= false?true:false==false?true:false;
boolean b3= false?false:true==true?false:true;
boolean flag= b1==b2?b2:b3;
System.out.println(b1);
System.out.println(b2);
System.out.println(b3);
System.out.println(flag);
```

答案：

false true false false

解析：

(1)==优先与三目运算符？： (2)有多个三目运算符从右侧执行

所以b1 可以按照以下步骤

(a)先执行== boolean b1= true?false:true?false:true;

(b)在执行右侧 boolean b1=true?false:false;

b2同样可以分解为以下步骤

boolean b2= false?true:false?true:false;

boolean b2=false?true:true;

b3 分解为

boolean b3 = false?false:true?false:true;

boolean b3=false?false:false;

flag =false==true?true:false

6.下列代码执行的结果是多少？

```
public class Test {
    public static void main(String[] args)
    {
        int i = 0;
        for (foo('A'); foo('B') && (i < 2); foo('C'))
        {
            i++;
            foo('D');
        }
```

```
    }
    public static boolean foo(char c)
    {
        System.out.print(c + " ");
        return true;
    }
}
```

答案： A B D C B D C B
理解for循环表达式的执行顺序即可。

7.下列代码执行的结果是？

```
public class GrandFather {

    public GrandFather()
    {
        System.out.println("1");
    }

    static
    {

        System.out.println("2");
    }

    {
        System.out.println("3");
    }

}


public class Father extends GrandFather {
    public Father()
    {
        System.out.println("4");
    }

    static
    {

        System.out.println("5");
```

```
    }

    {
        System.out.println("6");
    }

}


public class Child extends Father {

    public Test3Child()
    {
        System.out.println("7");
    }
    static
    {
        System.out.println("8");
    }

    {
        System.out.println("9");
    }
}


public class TestGo {

    public static void main(String[] args)
    {
        Child test3Child = new Child();
    }
}
```

执行结果：

```
2
5
8
3
1
6
4
```

9
7
Static代码块属于类 类优先于对象存在，代码块优先于构造函数执行。

8.下列代码执行的结果是？

```
public class Test {
    public static void main(String[] args)
    {
        Test t = new Test();
    }
    public Test()
    {
        System.out.println("1");
    }
    static
    {
        System.err.println("2");
    }
    {
        System.out.println("3");
    }
}
```

执行结果2不确定，但是3一定在1的前方 。因为2是以err输出，err是带有缓冲的输出。

9.下列代码执行的结果是？

```
public class Test {
    public static void main(String[] args)
    {
        Calendar c = Calendar.getInstance();
        c.set(Calendar.YEAR, 2008);
        c.set(Calendar.MONTH, 1);
        c.set(Calendar.DATE, 32);
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd");
        System.out.println(sdf.format(c.getTime()));
    }
}
```

答案：2008/03/03
月份是从0月开始的，超过日期自动递增一个月，2008年瑞年2月有29天

10.下列代码执行的结果是？

```
public class Test {
    String str = new String("hangzhou");

    char[] ch = { 'a', 'b', 'c' };

    public static void main(String[] args)
    {
        Test t = new Test();
        t.changeStr(t.str, t.ch);
        System.out.print(t.str + " and ");
        System.out.println(t.ch);
    }

    public void changeStr(String str, char ch[])
    {
        str = "guigu";
        ch[0] = 'q';
    }
}
```

执行结果：

hangzhou and qbc

记住一个规律：类似这种 基本数据类型（8种）都不改变，抽象数据类型除了String之外都改变。数据也是属于抽象数据类型。

11.下列代码执行的结果是？

```
public class Test {

    public static void main(String[] args)
    {
        Map<String, String> map = new HashMap<String, String>();
        map.put(String.valueOf(System.currentTimeMillis()) + "a", "1");
        map.put(String.valueOf(System.currentTimeMillis()) + "b", "2");
        map.put(String.valueOf(System.currentTimeMillis()) + "c", "3");
        for (Map.Entry<String, String> entry : map.entrySet())
        {
            System.out.println(entry.getValue());
        }

    }
```

```
}
```

执行结果： 不确定的
明确 HashMap存储原理。 HashMap取出来的值是不确定的。

12、下列代码执行的结果是？

```
public class TestEq {

    public static void main(String[] args) {
        // boolean equals(Object anObject)
        //将此字符串与指定的对象比较。
        String s1="hello";
        String s2="hello";
        String s3= new String("hello");
        String s4=new String("hello");

        System.out.println(s1==s2);
        System.out.println(s1==s3);
        System.out.println(s3==s4);

        System.out.println(s1.equals(s2));
        System.out.println(s1.equals(s3));
        System.out.println(s3.equals(s4));
    }
}
```

答案：

```
true
false
false
true
true
true
```

13、下列代码执行的结果是？

```
importjava.util.Date;
public class Test  extends Date{
    public static voidmain(String[] args) {
       new Test().test();
    }
    public void test(){
```

```
        System.out.println(super.getClass().getName());
    }
}
```

结果：Test
在test方法中，直接调用getClass().getName()方法，返回的是Test类名
由于getClass()在Object类中定义成了final，子类不能覆盖该方法，所以，在test方法中调用getClass().getName()方法，其实就是在调用从父类继承的getClass()方法，等效于调用super.getClass().getName()方法，所以，super.getClass().getName()方法返回的也应该是Test。
如果想得到父类的名称，应该用如下代码：
getClass().getSuperClass().getName();

设计模式相关

简述 MVC 设计模式

Model 模型：应用程序的主体部分，用于表示业务逻辑。 View 视图：应用程序中用户界面相关的部分，是用户看到并与之交互的界面。 Controller 控制器：用于根据用户的输入，控制用户界面数据显示，更新 Model 对象状态。

MVC 模式的出现不仅实现了功能模块和显示模块的分离，同时还提够了应用系统的可维护、可扩展性、可移植性、和组建的可复用性。

简述你熟悉的设计模式（23 种设计模式大家自己查阅）

答：Java 中的 23 种设计模式：

Factory（工厂模式），Builder（建造模式），Factory Method（工厂方法模式），
Prototype（原始模型模式），Singleton（单例模式），Facade（门面模式），
Adapter（适配器模式），Bridge（桥梁模式），Composite（合成模式），
Decorator（装饰模式），Flyweight（享元模式），Proxy（代理模式），
Command（命令模式），Interpreter（解释器模式），Visitor（访问者模式），
Iterator（迭代子模式），Mediator（调停者模式），Memento（备忘录模式），
Observer（观察者模式），State（状态模式），Strategy（策略模式），
Template Method（模板方法模式），Chain Of Responsibleity（责任链模式）

IO 流使用的是装饰者模式
按钮监听 观察者模式
forEach 遍历 List
TreeSet 使用策略模式

编程题

1、 请写出两种单例模式

饿汉式：

```
public class TestSingle {
    //构造函数私有化
    private TestSingle(){
    }
    //提供一个静态的全局变量
    private static TestSingle instance = new TestSingle();
    // 3.提供一个get方法获取全局变量
    public  static TestSingle getInstance(){
        return instance;
    }
}
```

懒汉式：

```
public class TestSingleton {
    private TestSingleton(){
    }
    private static TestSingleton instance=null;
    public static  TestSingleton getInstance(){
         if(instance==null){
             instance =new TestSingleton();
         }
         return instance;
        }
}
```