1、哪些情况下的对象会被垃圾回收机制处理掉？

利用可达性分析算法，虚拟机会将一些对象定义为 GC Roots，从 GC Roots 出发沿着引用链向下寻找，如果某个对象不能通过 GC Roots 寻找到，虚拟机就认为该对象可以被回收掉。

1.1 哪些对象可以被看做是 GC Roots 呢？

1）虚拟机栈（栈帧中的本地变量表）中引用的对象；

2）方法区中的类静态属性引用的对象，常量引用的对象；

3）本地方法栈中 JNI(Native 方法）引用的对象；

1.2 对象不可达，一定会被垃圾收集器回收么？

即使不可达，对象也不一定会被垃圾收集器回收，1）先判断对象是否有必要执行 finalize()方法，对象必须重写 finalize()方法且没有被运行过。2）若有必要执行，会把对象放到一个队列中，JVM 会开一个线程去回收它们，这是对象最后一次可以逃逸清理的机会。

2、讲一下常见编码方式？

编码的意义：计算机中存储的最小单元是一个字节即 8bit，所能表示的字符范围是 255 个，而人类要表示的符号太多，无法用一个字节来完全表示，固需要将符号编码，将各种语言翻译成计算机能懂的语言。

1）ASCII 码：总共 128 个，用一个字节的低 7 位表示，0～31 控制字符如换回车删除等；32~126 是打印字符，可通过键盘输入并显示出来；

2）ISO-8859-1,用来扩展 ASCII 编码，256 个字符，涵盖了大多数西欧语言字符。

3）GB2312:双字节编码，总编码范围是 A1-A7,A1-A9 是符号区，包含 682 个字符，B0-B7 是汉字区，包含 6763 个汉字；

4）GBK 为了扩展 GB2312,加入了更多的汉字，编码范围是 8140~FEFE，有 23940 个码位，能表示 21003 个汉字。

5）UTF-16: ISO 试图想创建一个全新的超语言字典，世界上所有语言都可通过这本字典 Unicode 来相互翻译，而 UTF-16 定义了 Unicode 字符在计算机中存取方法，用两个字节来表示 Unicode 转化格式。不论什么字符都可用两字节表示，即 16bit，固叫 UTF-16。

6）UTF-8：UTF-16 统一采用两字节表示一个字符，但有些字符只用一个字节就可表示，浪费存储空间，而 UTF-8 采用一种变长技术，每个编码区域有不同的字码长度。 不同类型的字符可以由 1~6 个字节组成。

3、utf-8 编码中的中文占几个字节；int 型几个字节？

utf-8 是一种变长编码技术，utf-8 编码中的中文占用的字节不确定，可能 2 个、3 个、4 个，int 型占 4 个字节。

4、静态代理和动态代理的区别，什么场景使用？

代理是一种常用的设计模式，目的是：为其他对象提供一个代理以控制对某个对象的访问，将两个类的关系解耦。代理类和委托类都要实现相同的接口，因为代理真正调用的是委托类的方法。

区别：

1）静态代理：由程序员创建或是由特定工具生成，在代码编译时就确定了被代理的类是哪一个是静态代理。静态代理通常只代理一个类；

2）动态代理：在代码运行期间，运用反射机制动态创建生成。动态代理代理的是一个接口下的多个实现类；

实现步骤：a.实现 InvocationHandler 接口创建自己的调用处理器；b.给 Proxy 类提供 ClassLoader 和代理接口类型数组创建动态代理类；c.利用反射机制得到动态代理类的构造函数；d.利用动态代理类的构造函数创建动态代理类对象；

使用场景：Retrofit 中直接调用接口的方法；Spring 的 AOP 机制；

5、Java 的异常体系

Java 中 Throwable 是所有异常和错误的超类，两个直接子类是 Error（错误）和 Exception（异常）：

1）Error 是程序无法处理的错误，由 JVM 产生和抛出，如 OOM、ThreadDeath 等。这些异常发生时，JVM 一般会选择终止程序。

2）Exception 是程序本身可以处理的异常，又分为运行时异常(RuntimeException)(也叫 Checked Eception)和非运行时异常(不检查异常 Unchecked Exception)。运行时异常有 NullPointerException\IndexOutOfBoundsException 等，这些异常一般是由程序逻辑错误引起的，应尽可能避免。非运行时异常有 IOException\SQLException\FileNotFoundException 以及由用户自定义的 Exception 异常等。

6、谈谈你对解析与分派的认识。

解析指方法在运行前，即编译期间就可知的，有一个确定的版本，运行期间也不会改变。解析是静态的，在类加载的解析阶段就可将符号引用转变成直接引用。

分派可分为静态分派和动态分派，重载属于静态分派，覆盖属于动态分派。静态分派是指在重载时通过参数的静态类型而非实际类型作为判断依据，在编译阶段，编译器可根据参数的静态类型决定使用哪一个重载版本。动态分派则需要根据实际类型来调用相应的方法。

7、修改对象 A 的 equals 方法的签名，那么使用 HashMap 存放这个对象实例的时候，会调

用哪个 equals 方法？

会调用对象的 equals 方法，如果对象的 equals 方法没有被重写，equals 方法和==都是比较栈内局部变量表中指向堆内存地址值是否相等。

8、Java 中实现多态的机制是什么？

多态是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编译时不确定，在运行期间才确定，一个引用变量到底会指向哪个类的实例。这样就可以不用修改源程序，就可以让引用变量绑定到各种不同的类实现上。Java 实现多态有三个必要条件：继承、重定、向上转型，在多态中需要将子类的引用赋值给父类对象，只有这样该引用才能够具备调用父类方法和子类的方法。

9、如何将一个 Java 对象序列化到文件里？

ObjectOutputStream.writeObject()负责将指定的流写入，ObjectInputStream.readObject()从指定流读取序列化数据。

```
//写入
try {
    ObjectOutputStream os = new ObjectOutputStream(new FileOutputStream("D:/student.txt"));
    os.writeObject(studentList);
    os.close();
} catch(FileNotFoundException e) {
    e.printStackTrace();
} catch(IOException e) {
    e.printStackTrace();
}
```

10、说说你对 Java 反射的理解

在运行状态中，对任意一个类，都能知道这个类的所有属性和方法，对任意一个对象，都能调用它的任意一个方法和属性。这种能动态获取信息及动态调用对象方法的功能称为 java 语言的反射机制。

反射的作用：开发过程中，经常会遇到某个类的某个成员变量、方法或属性是私有的，或只对系统应用开放，这里就可以利用 java 的反射机制通过反射来获取所需的私有成员或是方法。

1) 获取类的 Class 对象实例 Class clz = Class.forName("com.zhenai.api.Apple");

2) 根据 Class 对象实例获取 Constructor 对象 Constructor appConstructor = clz.getConstructor();

3) 使用 Constructor 对象的 newInstance 方法获取反射类对象 Object appleObj = appConstructor.newInstance();

4) 获取方法的 Method 对象 Method setPriceMethod = clz.getMethod("setPrice", int.class);

5) 利用 invoke 方法调用方法 setPriceMethod.invoke(appleObj, 14);

6) 通过 getFields()可以获取 Class 类的属性，但无法获取私有属性，而 getDeclaredFields()可以获取到包括私有属性在内的所有属性。带有 Declared 修饰的方法可以反射到私有的方法，没有 Declared 修饰的只能用来反射公有的方法，其他如 Annotation\Field\Constructor 也是如此。

11、说说你对 Java 注解的理解

注解是通过@interface 关键字来进行定义的，形式和接口差不多，只是前面多了一个@

```
public @interface TestAnnotation {

}
```

使用时@TestAnnotation 来引用，要使注解能正常工作，还需要使用元注解，它是可以注解到注解上的注解。元标签有@Retention @Documented @Target @Inherited @Repeatable 五种

@Retention 说明注解的存活时间，取值有 RetentionPolicy.SOURCE 注解只在源码阶段保留，在编译器进行编译时被丢弃；RetentionPolicy.CLASS 注解只保留到编译进行的时候，并不会被加载到 JVM 中。RetentionPolicy.RUNTIME 可以留到程序运行的时候，它会被加载进入到 JVM 中，所以在程序运行时可以获取到它们。

@Documented 注解中的元素包含到 javadoc 中去

@Target 限定注解的应用场景，ElementType.FIELD 给属性进行注解；ElementType.LOCAL_VARIABLE 可以给局部变量进行注解；ElementType.METHOD 可以给方法进行注解；ElementType.PACKAGE 可以给一个包进行注解 ElementType.TYPE 可以给一个类型进行注解，如类、接口、枚举

@Inherited 若一个超类被@Inherited 注解过的注解进行注解，它的子类没有被任何注解应用的话，该子类就可继承超类的注解；

注解的作用：

1）提供信息给编译器：编译器可利用注解来探测错误和警告信息

2）编译阶段：软件工具可以利用注解信息来生成代码、html 文档或做其它相应处理；

3）运行阶段：程序运行时可利用注解提取代码

注解是通过反射获取的，可以通过 Class 对象的 isAnnotationPresent()方法判断它是否应用了某个注解，再通过 getAnnotation()方法获取 Annotation 对象

12、说一下泛型原理，并举例说明

泛型就是将类型变成参数传入，使得可以使用的类型多样化，从而实现解耦。Java 泛型是在 Java1.5 以后出现的，为保持对以前版本的兼容，使用了擦除的方法实现泛型。擦除是指在一定程度无视类型参数 T，直接从 T 所在的类开始向上 T 的父类去擦除，如调用泛型方法，传入类型参数 T 进入方法内部，若没在声明时做类似 public T methodName(T extends Father t){}，Java 就进行了向上类型的擦除，直接把参数 t 当做 Object 类来处理，而不是传进去的 T。即在有泛型的任何类和方法内部，它都无法知道自己的泛型参数，擦除和转型都是在边界上发生，即传进去的参在进入类或方法时被擦除掉，但传出来的时候又被转成了我们设置的 T。在泛型类或方法内，任何涉及到具体类型（即擦除后的类型的子类）操作都不能进行，如 new T()，或者 T.play()（play 为某子类的方法而不是擦除后的类的方法）

13、Java 中 String 的了解

1）String 类是 final 型，固 String 类不能被继承，它的成员方法也都默认为 final 方法。String 对象一旦创建就固定不变了，对 String 对象的任何改变都不影响到原对象，相关的任何改变操作都会生成新的 String 对象。

2）String 类是通过 char 数组来保存字符串的，String 对 equals 方法进行了重定，比较的是值相等。

String a = "test"; String b = "test"; String c = new String("test");

a、b 和字面上的 test 都是指向 JVM 字符串常量池中的"test"对象，他们指向同一个对象。而 new 关键字一定会产生一个对象 test，该对象存储在堆中。所以 new String("test")产生了两个对象，保存在栈中的 c 和保存在堆中的 test。而在 java 中根本就不存在两个完全一模一样的字符串对象，故在堆中的 test 应该是引用字符串常量池中的 test。

例：

String str1 = "abc"; //栈中开辟一块空间存放引用 str1，str1 指向池中 String 常量"abc"
String str2 = "def"; //栈中开辟一块空间存放引用 str2，str2 指向池中 String 常量"def"
String str3 = str1 + str2;//栈中开辟一块空间存放引用 str3
//str1+str2 通过 StringBuilder 的最后一步 toString()方法返回一个新的 String 对象"abcdef"
//会在堆中开辟一块空间存放此对象，引用 str3 指向堆中的(str1+str2)所返回的新 String 对象。
System.out.println(str3 == "abcdef");//返回 false
因为 str3 指向堆中的"abcdef"对象，而"abcdef"是字符池中的对象，所以结果为 false。JVM 对 String str="abc"对象放在常量池是在编译时做的，而 String str3=str1+str2 是在运行时才知道的，new 对象也是在运行时才做的。

14、String 为什么要设计成不可变的？

1）字符串常量池需要 String 不可变。因为 String 设计成不可变，当创建一个 String 对象时，若此字符串值已经存在于常量池中，则不会创建一个新的对象，而是引用已经存在的对象。如果字符串变量允许必变，会导致各种逻辑错误，如改变一个对象会影响到另一个独立对象。

2）String 对象可以缓存 hashCode。字符串的不可变性保证了 hash 码的唯一性，因此可以缓存 String 的 hashCode，这样不用每次去重新计算哈希码。在进行字符串比较时，可以直接

比较 hashCode，提高了比较性能；

3）安全性。String 被许多 java 类用来当作参数，如 url 地址，文件 path 路径，反射机制所需的 Strign 参数等，若 String 可变，将会引起各种安全隐患。