## 31.java类加载过程：

1.加载时机：创建实例、访问静态变量或方法、反射、加载子类之前  
2.验证：验证文件格式、元数据、字节码、符号引用的正确性  
3.加载：根据全类名获取文件字节流、将字节流转化为静态储存结构放入方法区、生成class对象  
4.准备：在堆上为静态变量划分内存  
5.解析：将常量池中的符号引用转换为直接引用  
6.初始化：初始化静态变量  
7.书籍推荐：**深入理解java虚拟机**，博客推荐：Java/Android阿里面试JVM部分理解

90、八种基本数据类型（概率20%）

byte、short、int、long、float、double、char、boolean

122.Int Integer的区别 （概率30%）

int 是基本类型，直接存数值

1.Java 中的数据类型分为基本数据类型和复杂数据类型

int 是前者>>integer 是后者（也就是一个类）

2.初始化时>>

int i =1;

Integer i= new Integer(1);(要把integer 当做一个类看)

int 是基本数据类型（面向过程留下的痕迹，不过是对java的有益补充）

Integer 是一个类，是int的扩展，定义了很多的转换方法

类似的还有：float Float;double Double;string String等

31·&和&&的区别（概率10%）

一个位运算符，一个是逻辑运算符

具体代码中&&具有短路的作用(前面如果条件不成立,后面就不走了), &没有,

16·面向对象的特征？（概率60%）

**封装**

封装最好理解了。封装是面向对象的特征之一，是对象和类概念的主要特性。

封装，也就是把客观事物封装成抽象的类，并且类可以把自己的数据和方法只让可信的类或者对象操作，对不可信的进行信息隐藏。

**继承**

面向对象编程 (OOP) 语言的一个主要功能就是“继承”。继承是指这样一种能力：它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。

**多态**

多态性（polymorphisn）是允许你将父对象设置成为和一个或更多的他的子对象相等的技术，赋值之后，父对象就可以根据当前赋值给它的子对象的特性以不同的方式运作。简单的说，就是一句话：允许将子类类型的指针赋值给父类类型的指针。

实现多态，有二种方式，覆盖，重载。

**覆盖**，是指子类重新定义父类的虚函数的做法。

1:方法名相同,参数类型相同,子类的返回值类型大于等于父类,修饰符子类的访问权限要大于等于父类,子类抛出的异常不能大于父类.

**重载**，是指允许存在多个同名函数，而这些函数的参数表不同（或许参数个数不同，或许参数类型不同，或许两者都不同）。

1:同一方法中方法名相同,参数列表不同,参数的个数,类型,顺序(具体使用自定义adapter)

109.重载和重写的区别（概率30%）

方法的重载属于,编译时多态,方法名相同参数列表不同，返回值必须相同或都没有返回值类型。 方法的覆盖属于运行时多态，子类覆盖父类的方法,子类指向父类引用,在调用方法的时候用父类的引用调用。

49.静态变量合实例变量的区别（概率30%）

语法：静态变量前要加static来修饰，而实例变量前则不加；

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，静态变量就会被分配空间，静态变量就可以被使用了。

55.接口和抽象类的区别（概率40%）

abstract可以修饰抽象方法，而一个类只要有一个抽象方法，就必须用

abstract定义该类，即抽象类。

用interface修饰的类，里面的方法都是抽象方法，因此在定义接口的时候，可以

直接不加那些修饰，系统会默认的添上去。接口里面的字段都是公有常量，即

public static final修饰的字段。

//接口是多继承 、抽象类是单继承

//接口里面只有抽象方法

// 抽象类: 有抽象方法和普通方法 、 抽象类==父类( 抽象类是 顶级父类 )

// 比较两个对象是否相等 == 比较的是地址、 = 比较的是值

//final不能和 static 、actaret 一起使用

134.抽象类和接口的区别（概率20%）

abstract可以修饰抽象方法，而一个类只要有一个抽象方法，就必须用abstract定义该类，即抽象类。抽象类，被继承，实现它里面的方法，可以只有普通方法

用interface修饰的类，里面的方法都是抽象方法（不能实例化），因此在定义接口的时候，可以直接不加那些修饰，系统会默认的添上去。接口里面的字段都是公有常量，即public static final修

饰的字段。可以没有函数原型，里面可以是空的。

51.什么是匿名内部类，在什么时候调用（概率30%）

内部类：内部类可以是static的或者非static的，static内部类只能包含静态方法和静态类变量，只能访问外部类的静态元素，内部类可以实例化，多次使用。

匿名内部类：它只能使用一次，不区分static和非static。如果用到外部类的变量的话，必须是类变量或者实例变量，就是必须是类定义的变量，或者final的局部变量。匿名内部类如果是继承某个类的话是可以重写那个类的方法的，这个和普通内部类一样。

实现事件监听器的时候

用匿名内部类编码非常简洁，也容易读懂，不重复利用时使用。

## 43.Java 中内部类为什么可以访问外部类

1.因为内部类创建的时候，需要外部类的对象，在内部类对象创建的时候会把外部类的引用传递进去

94 public，private，protect权限（概率40%）

public在任何情况下都可用；

protect可在当前类，同一个包中和子孙类中使用；

default：可在当前类和同一个包中使用

private只能在当前类中使用

19·Final，finally，finalized，区别；（概率30%）

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。(geitview内部类访问外部类的时候需要加final)

finally是异常处理语句结构的一部分，表示总是执行。(捕获异常的时候用过)

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。

136. Final和Finally的区别（概率30%）

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。

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。

115.什么方法不能被重写（概率30%）

被final修饰的方法，构造方法，

61.Throws与Throw的区别（概率20%）

1.throw则是指抛出的一个具体的异常类型。

2.通常在一个方法（类）的声明处通过throws声明方法（类）可能抛出的异常信息，而在方法（类）内部通过throw声明一个具体的异常信息。

3.throws通常不用显示的捕获异常，可由系统自动将所有捕获的异常信息抛给上级方法；

throw则需要用户自己捕获相关的异常，而后在对其进行相关包装，最后在将包装后的异常信息抛出。

如果在方法中会有异常被抛出而你又不希望在这个方法体内对此异常进行处理，可以使用throws在声明方法的时候同时声明他可能会跑出的异常。

1. 两者位置不同.

92、异常（概率40%）

答：编译时异常：程序正确，但因为外在的环境条件不满足引发。例如：用户错

误及I/O问题----程序试图打开一个并不存在的远程Socket端口。这不是程序本

身的逻辑错误，而很可能是远程机器名字错误(用户拼写错误)。对商用软件系统，

程序开发者必须考虑并处理这个问题。Java编译器强制要求处理这类异常，如果

不捕获这类异常，程序将不能被编译。

运行期异常：这意味着程序存在bug，如数组越界，0被除，入参不满足规范.....

这类异常需要更改程序来避免，Java编译器强制要求处理这类异常。

116.编译时异常和运行时异常的区别（概率30%）

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，但是并不要求必须声明抛出未被捕获的运行时异常。

93、==equals区别（概率40%）

1、 ==是判断两个变量或实例是不是指向同一个内存空间

equals是判断两个变量或实例所指向的内存空间的值是不是相同

2、==是指对内存地址进行比较

equals()是对字符串的内容进行比较

3、==指引用是否相同

equals（）指的是值是否相同

126.定义数组的几种方式（概率30%）

int a[]; //声明未初始化

a = new int [10]; // 定义占用空间大小（10个int）

int a[] = new int [10]; //声明并定义大小（即分配了指定大小的空间）

int a[] = {1,2,3}; // 声明并初始化，占用空间大小是3个int。

79、集合的实现类（概率40%）

HashSet：元素无需、不可重复；ArrayList：元素有序，可重复

HashMap以键值对的形式存取数据，key值不可重复，value值可重复

118.Collection 和 Collections的区别（概率30%）

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。

Native是啥

114.集合和数组的区别（概率30%）

1）：数组在定义时必须定义长度，而集合在定义时不必定义长度。

2）：数组在定义时必须要声明数组的数据类型，而集合不必。但是一般情况下我们都是存储统一数据类型的数据，我们可以使用泛型的写法来限制集合里面的数据类型。

## 38.HashMap的实现过程：Capacity就是buckets的数目，Load factor就是buckets填满程度的最大比例。如果对迭代性能要求很高的话不要把capacity设置过大，也不要把load factor设置过小。

1.简单来说HashMap就是一个会自动扩容的**数组链表**  
2.put过程

* 1.对key的hashCode()做hash，然后再计算index;
* 2.如果没碰撞直接放到bucket里；
* 3.如果碰撞了，以链表的形式存在buckets后；
* 4.如果碰撞导致链表过长(大于等于TREEIFY\_THRESHOLD)，就把链表转换成红黑树；
* 5.如果节点已经存在就替换old value(保证key的唯一性)
* 6.如果bucket满了(超过load factor\*current capacity)，就要resize。  
  3.resize：当put时，如果发现目前的bucket占用程度已经超过了Load Factor所希望的比例，那么就会发生resize。在resize的过程，简单的说就是把bucket扩充为2倍，之后重新计算index，把节点再放到新的bucket中  
  4.get过程
* 1.根据key的hash算出数组下表
* 2.使用equals遍历链表进行比较

8.ArrayList和LinkedList区别？（概率30%）

存数据，ArrayList**数组存储数据**，索引值以下标来搜索，**查询比较方便，删除增加比较麻烦**，

但是linkedList以**链表式存储数据**，对于增删比较方便。

可以这样说：当操作是在一列数据的后面添加数据而不是在前面或中间,并且需要随机地访问其中的元素时,使用ArrayList会提供比较好的性能；

当你的操作是在一列数据的前面或中间添加或删除数据,并且按照顺序访问其中的元素时,就应该使用LinkedList了。

ArrayList和LinkedList在性能上各 有优缺点，都有各自所适用的地方，总的说来可以描述如下：

对ArrayList和LinkedList而言，在列表末尾增加一个元素所花的开销都是固定的。对 ArrayList而言，主要是在内部数组中增加一项，指向所添加的元素，偶尔可能会导致对数组重新进行分配；而对LinkedList而言，这个开销是 统一的，分配一个内部Entry对象。

在ArrayList的 中间插入或删除一个元素意味着这个列表中剩余的元素都会被移动；而在LinkedList的中间插入或删除一个元素的开销是固定的。

LinkedList不支持高效的随机元素访问。

ArrayList的空间浪费主要体现在在list列表的结尾预留一定的容量空间，而LinkedList的空间花费则体现在它的每一个元素都需要消耗相当的空间

110.Set list map集合的特点（概率30%）

List接口 元素有序可重复.

实现类有：ArrayList 数组实现轻量级，运行快，线程不安全。JDK1.2 查询快

Vector 数组实现重量级，运行慢，线程安全。JDK1.0保证元素的无序唯一，自定义对象存进HashSet为了保证元素内容不重复需要覆盖hashCode()与equals()方法。

SortedSet(不重要) 元素有序（Unicode升序）唯一

LinkedList 链表实现 常用语堆栈与队列的实现 增删操作快

Set 接口 元素无序不可重复

实现类有：HashSet，底层用hashCode()算法实现，

TreeSet要求元素有序，自定义的对象需要实

现Comparable接口的 compareTo（object o）方法

Map(接口): 与Collection接口无关,有一个子接口SortedMap特点: 元素是key-value, key

唯一,无序; value可重复

实现类: HashMap 轻量级 线程不安全的,允许key或value为null JDK1.2

HashTable 重量级 线程安全的 不允许key或value为null JDK1.0

Properties是HashTable的子类,主键和值都是字符串

SortedMap:(不重要)

特点: key唯一,有序(Unicode升序)

实现类:TreeMap

## 41.ArrayList 如何删除重复的元素或者指定的元素；

1.删除重复：Set  
2.删除指定：迭代器

## 47.ConcurrentHashMap内部实现，HashTable的实现被废弃的原因:

1.HashTable容器在竞争激烈的并发环境下表现出效率低下的原因，是因为所有访问HashTable的线程都必须竞争同一把锁，那假如容器里有多把锁，每一把锁用于锁容器其中一部分数据，那么当多线程访问容器里不同数据段的数据时，线程间就不会存在锁竞争，从而可以有效的提高并发访问效率，这就是ConcurrentHashMap所使用的锁分段技术，首先将数据分成一段一段的存储，然后给每一段数据配一把锁，当一个线程占用锁访问其中一个段数据的时候，其他段的数据也能被其他线程访问。  
2.ConcurrentHashMap是由Segment数组结构和HashEntry数组结构组成。Segment是一种可重入锁ReentrantLock，在ConcurrentHashMap里扮演锁的角色，HashEntry则用于存储键值对数据。一个ConcurrentHashMap里包含一个Segment数组，Segment的结构和HashMap类似，是一种数组和链表结构， 一个Segment里包含一个HashEntry数组，每个HashEntry是一个链表结构的元素，每个Segment守护者一个HashEntry数组里的元素,当对HashEntry数组的数据进行修改时，必须首先获得它对应的Segment锁。

78、String,StringBuffer区别（概率30%）

JAVA平台提供了两个类：String和StringBuffer，它们可以储存和操作字符串，

即包含多个字符的字符数据。这个String类提供了数值不可改变的字符串。而这个

StringBuffer类提供的字符串进行修改。

StringBuilder与StringBuffer的区别 。

StringBuffer

1，一个类似于 String 的字符串缓冲区，对它的修改不会像String那样重创建对象。

2，使用append()方法修改Stringbuffer的值，使用toString()方法转换为字符串。

Stringbuild是jdk1.5后用来替换stringBuffer的一个类，大多数时候可以替换

StringBuffer。和StringBuffer的区别在于Stringbuild是一个单线程使用的类，不值

执行线程同步所以比StringBuffer的速度快，效率高。

82.io流读取输出方法（概率20%）

文件读取：

public static void main(String[] args) throws IOException

{

//创建一个文件读取流对象，并初始化时指定名称的文件相关联。

//要保证此文件是已存在，如不存在会发生异常：FileNotFoundException

FileReader fr = new FileReader("demo.txt");

//调用读取流对象的read方法。

//read方法一次只读一个字符，而且会自动向下读取。当读取完后会返回-1.

int ch = 0;

while((ch = fr.read())!=-1)

System.out.print((char)ch);

fr.close();

}

## 40.java的线程如何实现

1.Thread继承  
2.Runnale  
3.Future  
4.线程池

225.同步 异步的区别（概率80%）

发出一个功能调用时，在没有得到结果之前，该调用就不返回。也就是必须一件一件事做,等前一件做完了才能做下一件事

当一个异步过程调用发出后，调用者不能立刻得到结果。实际处理这个调用的部件在完成后，通过状态、通知和回调来通知调用者。

**25.线程同步的问题，常用的线程同步**

1.sycn：保证了原子性、可见性、有序性  
2.锁：保证了原子性、可见性、有序性

* 1.自旋锁:可以使线程在没有取得锁的时候，不被挂起，而转去执行一个空循环。
  + 1.优点:线程被挂起的几率减少，线程执行的连贯性加强。用于对于锁竞争不是很激烈，锁占用时间很短的并发线程。
  + 2.缺点:过多浪费CPU时间，有一个线程连续两次试图获得自旋锁引起死锁
* 2.阻塞锁:没得到锁的线程等待或者挂起，Sycn、Lock
* 3.可重入锁:一个线程可多次获取该锁，Sycn、Lock
* 4.悲观锁:每次去拿数据的时候都认为别人会修改，所以会阻塞全部其他线程 Sycn、Lock
* 5.乐观锁: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，可以使用版本号等机制。cas
* 6.显示锁和内置锁:显示锁用Lock来定义、内置锁用synchronized。
* 7.读-写锁:为了提高性能，Java提供了读  
  3.volatile
* 1.只能保证可见性，不能保证原子性
* 2.自增操作有三步，此时多线程写会出现问题  
  4.cas
* 1.操作:内存值V、旧的预期值A、要修改的值B，当且仅当预期值A和内存值V相同时，将内存值修改为B并返回true，否则什么都不做并返回false。
* 2.解释:本地副本为A，共享内存为V，线程A要把V修改成B。某个时刻线程A要把V修改成B，如果A和V不同那么就表示有其他线程在修改V，此时就表示修改失败，否则表示没有其他线程修改，那么把V改成B。
* 3.局限:如果V被修改成V1然后又被改成V，此时cas识别不出变化，还是认为没有其他线程在修改V，此时就会有问题
* 4.局限解决:将V带上版本。  
  5.线程不安全到底是怎么回事：
* 1.一个线程写，多个线程读的时候，会造成写了一半就去读
* 2.多线程写，会造成脏数据

53.什么是线程（概率60%）

线程，有时被称为[轻量级进程](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=65949654)(Lightweight Process，LWP)，是程序执行流的最小[单元](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=275431)。一个标准的线程由线程[ID](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=265569)，当前指令指针(PC)，[寄存器](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=35071)集合和[堆栈](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=331232)组成。另外，线程是[进程](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=288622)中的一个实体，是被系统独立调度和分派的基本单位，线程自己不拥有[系统](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=349837)资源，只拥有一点在运行中必不可少的资源，但它可与同属一个进程的其它线程共享进程所拥有的全部资源。一个线程可以[创建](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=100768)和[撤消](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=126328)另一个线程，同一进程中的多个线程之间可以并发执行。由于线程之间的相互制约，致使线程在运行中呈现出间断性。线程也有[就绪](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=425221)、[阻塞](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=646488)和[运行](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=493544)三种基本状态。每一个程序都至少有一个线程，那就是程序本身。

线程是程序中一个单一的顺序控制流程。在单个程序中同时运行多个线程完成不同的工作，称为[多线程](http://baike.sogou.com/lemma/ShowInnerLink.htm?lemmaId=452880)。

54.线程和进程的区别（概率60%）

// 线程与进程的区别:

定义:

1. 进程是系统进行资源分配和调度的一个独立单位
2. 线程是进程的一个实体，是CPU调度和分派的基本单位，线程自己基本上不拥有系统资源，只拥有一点在系统运行中必不可少的资源，但是它可与同属一个进程的其他线程共享进程所拥有的全部资源

关系:

1. 一个线程可以创建和撤销另一个线程，同一个进程中的多个线程可以同时并发执行，他可以与同进程中的其他线程共享数据，但拥有自己的栈空间，拥有独立的执行顺序

区别:

进程和线程区别: 他们是不同的操作系统资源管理方式。

进程有独立的地址空间，一个进程崩溃后，不会对其他进程产生影响

线程只是一个进程中不同的执行路径，线程有自己的堆栈和局部变量，线程之间没有单独的地址空间，一个线程死掉就等于整个进程死掉， 所以多线程的操作要比多进程的程序健壮。 但在进程切换时耗费资源较大，效率要差一些

对于一些要求同时进行并且又要共享某些变量的并发操作，只能用线程，不能用进程，（因为进程在切换时耗费的资源要大一些，效率差一些）

进程是表示资源分配的基本单位，又是调度运行的基本单位。

线程是进程中执行运算的最小单位，亦即执行处理机调度的基本单位。

线程的优点：

（1）易于调度。

（2）提高并发性。通过线程可方便有效地实现并发性。进程可创建多个线程来执行同一程序的不同部分。

（3）开销少。创建线程比创建进程要快，所需开销很少。。

（4）利于充分发挥多处理器的功能。通过创建多线程进程（即一个进程可具有两个或更多个线程），每个线程在一个处理器上运行，从而实现应用程序的并发性，使每个处理器都得到充分运行。

进程和线程的关系：

（1）一个线程只能属于一个进程，而一个进程可以有多个线程，但至少有一个线程。 （2）资源分配给进程，同一进程的所有线程共享该进程的所有资源。

（3）处理机分给线程，即真正在处理机上运行的是线程。

（4）线程在执行过程中，需要协作同步。不同进程的线程间要利用消息通信的办法实现同步。

## 37.操作系统进程和线程的区别

1.简而言之,一个程序至少有一个进程,一个进程至少有一个线程.  
2.线程的划分尺度小于进程，使得多线程程序的并发性高。  
3.另外，进程在执行过程中拥有独立的内存单元，而多个线程共享内存，从而极大地提高了程序的运行效率。  
4.多线程的意义在于一个应用程序中，有多个执行部分可以同时执行。有将多个线程看做多个独立的应用，来实现进程的调度和管理以及资源分配

47.Sleep和wait的区别（概率30%）

1.这两个方法来自不同的类分别是，sleep来自Thread类，和wait来自Object

类。

2.最主要是sleep方法没有释放锁，而wait方法释放了锁，使得其他线程可

以使用同步控制块或者方法。sleep不出让系统资源；wait是进入线程等待池

等待，出让系统资源，其他线程可以占用CPU。一般wait不会加时间限制，

因为如果wait线程的运行资源不够，再出来也没用，要等待其他线程调用

notify/notifyAll唤醒等待池中的所有线程，才会进入就绪队列等待OS分配系

统资源。sleep(milliseconds)可以用时间指定使它自动唤醒过来，如果时间不到

只能调用interrupt()强行打断。

3.wait，notify和notifyAll只能在同步控制方法或者同步控制块里面使用，而

sleep可以在任何地方使用

Sleep需要捕获异常,而wait不需要

139. 线程的几种状态（概率30%）

1)新建状态(New)：新创建了一个线程对象。

2)就绪状态(Runnable)：线程对象创建后，其他线程调用了该对象的start()方法。该状态的线程位于可运行线程池中，变得可运行，等待获取[CPU](http://product.it168.com/list/b/0217_1.shtml)的使用权。

3)运行状态(Running)：就绪状态的线程获取了CPU，执行run()方法。

4)阻塞状态(Blocked)：阻塞状态是线程因为某种原因放弃CPU使用权，暂时停止运行。直到线程进入就绪状态，才有机会转到运行状态。阻塞的情况分三种：

等待阻塞：运行的线程执行wait()方法，JVM会把该线程放入等待池中。

同步阻塞：运行的线程在获取对象的同步锁时，若该同步锁被别的线程占用，则JVM会把该线程放入锁池中。

其他阻塞：运行的线程执行sleep()或join()方法，或者发出了I/O请求时，JVM会把该线程置为阻塞状态。当sleep()状态超时、join()等待线程终止或者超时、或者I/O处理完毕时，线程重新转入就绪状态。

5)死亡状态(Dead)：线程执行完了或者因异常退出了run()方法，该线程结束生命周期。

当调用start方法的时候，该线程就进入就绪状态。等待CPU进行调度执行，此时还没有真正执行线程。

72、说说java的反射机制原理以及应用场合（概率40%）

JAVA反射机制是在运行状态中，对于任意一个类，都能够知道这个类的所有

属性和方法；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态

获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

类对象: 封装了类的描述信息的对象,类加载的产物,由jvm创建java.lang.Class

应用场景: Gson 序列化反序列化

获取类对象的方式:1. 类名.class2. 类的对象.getClass()3. Class.forName("

包名.类名")

74、java有几种引用类型（概率40%）

强引用，软引用、弱引用、幽灵引用、

68.gc是什么（概率70%）

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。

97、什么时候用递归（概率30%）

树用递归方便，用其它方法好像跟本不可能．

递归用在特定的函数如 f(x)= f(x-1) + 2

像这种情况，你想算出f(x)就必需算出f(x-1)，而f(x)和f(x-1)实际上都是用共一个方法，只是参数相差一．这种时候用递归就很快．

98、jdbc中怎么处理事物（概率50%）

事务，也是数据库事务，指的是作为单个逻辑工作单元执行的一系列操作。正常的情况下，操作应该顺利进行，与操作相关的所有数据库信息也成功地更新；

但是，如果在这一系列过程中任何一个环节出了差错，导致操作失败了，数据库中所有信息都必须保持操作前的状态不变。否则，数据库的信息将会一片混乱而不可预测。

一个逻辑工作单元要称为事务，必须满足ACID（原子性，一致性，隔离性和持久性）

事务的结束只能有两种形式：提交和回滚。操作完全成功则提交，产生永久性的修改；操作不完全成功则回滚，恢复到事务开始前的状态。它们将结束一个事务。

（1）关闭自动提交事务。通过设置连接的自动提交事务属性为false，

130.常遇到的异常、如何解决（概率80%）

异常的继承结构：

基类为Throwable，Error和Exception继承Throwable，RuntimeException和IOException等继承Exception，具体的RuntimeException继承RuntimeException。

Error和RuntimeException及其子类成为未检查异常（unchecked），其它异常成为已检查异常（checked）。

1、 SQLException：操作数据库异常类。

问题：当Java应用程序与数据库进行沟通时如果产生了错误，就会触发这个类。同时会将数据库的错误信息通过这个类显示给用户。当用户插入数据的时候，如果这个字段的值跟现有的纪录重复了，违反了数据库的唯一性约束，此时数据库就会跑出一个异常信息。这个信息一般用户可能看不到，因为其发生在数据库层面的。此时这个操作数据库异常类就会捕捉到数据库的这个异常信息，并将这个异常信息传递到前台。如此的话，前台用户就可以根据这个异常信息来分析发生错误的原因。

解决方案：检查插入的数据是否重复。

2、 ClassCastException：数据类型转换异常。

问题：在Java应用程序中，有时候需要对数据类型进行转换。这个转换包括显示的转换与隐式的转换。不过无论怎么转换，都必须要符合一个前提的条件，即数据类型的兼容性。在数据类型进行转换之前，就保证数据类型的兼容性。如此的话，就不容易造成数据类型的转换异常。如在只允许数值类型的字段中，可以设置不允许用户输入数值以外的字符。

解决方案：注意相应的组件在初始化的时候，是否相对应。

3、 NumberFormatException：字符串转换为数字类型时抛出的异常。

在数据类型转换过程中，如果是字符型转换为数字型过程中出现的问题，对于这个异常在Java程序中采用了一个独立的异常，即NumberFormatException.如现在讲字符型的数据“123456”转换为数值型数据时，是允许的。但是如果字符型数据中包含了非数字型的字符，如123#56，此时转换为数值型时就会出现异常。系统就会捕捉到这个异常，并进行处理。

解决方案：检查数据是否正确。

4. java.lang.NullPointerException

(1).如果使用第三包的jar包，jar包中有错误

(2).没有实例化组件

(3).没有把Adapter和值绑定；

(4)．当页面中某个组件赋值为null时activity会出现错误，程序崩溃后其他页面的数据就会被重新初始化

(5).用Intent传值时，获取的key不存在

(6).没有赋初始值

5. java.lang.ClassNotFoundException　　异常的解释是"指定的类不存在"。

6. java.lang.ArithmeticException　　这个异常的解释是"数学运算异常"，比如程序中出现了除以零这样的运算就会出这样的异常。

7. java.lang.ArrayIndexOutOfBoundsException

　　异常的解释是"数组下标越界"，现在程序中大多都有对数组的操作，因此在调用数组的时候一定要认真检查，看自己调用的下标是不是超出了数组的范围，一般来说，显示（即直接用常数当下标）调用不太容易出这样的错，但隐式（即用变量表示下标）调用就经常出错了，还有一种情况，是程序中定义的数组的长度是通过某些特定方法决定的，不是事先声明的，这个时候，最好先查看一下数组的length，以免出现这个异常。

8. java.lang.IllegalArgumentException

这个异常的解释是"方法的参数错误"，比如g.setColor(int red,int green,int blue)这个方法中的三个值，如果有超过２５５的也会出现这个异常，因此一旦发现这个异常，我们要做的，就是赶紧去检查一下方法调用中的参数传递是不是出现了错误。

9. java.lang.IllegalAccessException

　　这个异常的解释是"没有访问权限"，当应用程序要调用一个类，但当前的方法即没有对该类的访问权限便会出现这个异常。对程序中用了Package的情况下要注意这个异常

10.安全异常：产生的原因是由于当前的设备不支持当前程序，这个是由于机型产生的问题，我们应该换一个设备进行测试，检测当前程序存在什么样的异常；另一种原因是获取激活的网络信息实体类，需要添加权限，否则会出现此异常

解决办法：在配置文件中，添加android\_permission\_Access\_NetWork\_State的权限

11. RejectExcuteException：一个异步任务只能执行一次，否则会报异常

一个线程最多能执行5个线程，超出则等待，否则会报拒绝执行的异常

解决办法：使用子线程+handler来代替AsyncTask，或者一个线程中少添加几个异步操作。

12. Out of memory：内存溢出是存放的数据过多导致内存不足的溢出事件

13. Memory leak：内存泄露是本应该释放的资源没有充分得到释放，就继续添加数据导致内存泄露事件；

Java应用程序中常见的异常类还有很多。如未找到相应类异常、不允许访问某些类异常、文件已经结束异常、文件未找到异常、字段未找到异常等等。一般系统开发人员都可以根据这个异常名来判断当前异常的类型。

ArithmeticException（除数为0的异常）, BufferOverflowException（缓冲区上溢异常）, BufferUnderflowException（缓冲区下溢异常）, IndexOutOfBoundsException（出界异常）, NullPointerException（空指针异常）, EmptyStackException（空栈异常）, IllegalArgumentException（不合法的参数异常）, NegativeArraySizeException, NoSuchElementException, SecurityException, SystemException, UndeclaredThrowableException

## 61.面试常考的算法

1.快排、堆排序为首的各种排序算法  
2.链表的各种操作：判断成环、判断相交、合并链表、倒数K个节点、寻找成环节点  
3.二叉树、红黑树、B树定义以及时间复杂度计算方式  
4.动态规划、贪心算法、简单的图论  
5.推荐书籍：**算法导论**，将图论之前的例子写一遍

## 44.设计移动端的联系人存储与查询的功能，要求快速搜索联系人，可以用到哪些数据结构？数据库索引，平衡二叉树(B树、红黑树)

## 45.红黑树特点

1.root节点和叶子节点是黑色  
2.红色节点后必须为黑色节点  
3.从root到叶子每条路径的黑节点数量相同