2020 面试题总结——JAVA 基础篇

1、JAVA 中的几种数据类型是什么，各自占用多少字节。

Java 语言提供了八种基本数据类型。六种数据类型（四个整数型，两个浮点型），一种字符类型，还有一种布尔型。

- byte：1 字节，表示范围是-128~127 之间。
- short：2 字节，表示范围是-32768~32767 之间。
- int：4 字节，表示范围是负的 2 的 31 次方到正的 2 的 31 次方减 1。
- long：8 字节，表示范围为负的 2 的 63 次方到正的 2 的 63 次方减 1。
- float：4 字节，表示范围在 3.4e-45~1.4e38，直接赋值时必须在数字后加上 f 或 F。
- double：8 字节，表示范围在 4.9e-324~1.8e308，赋值时可以加 d 或 D 也可以不加。
- boolean：只有 true 和 false 两个取值。
- char：2 字节，存储 Unicode 码，用单引号赋值。

2、String 类能被继承吗，为什么。

不能被继承，因为 String 类有 final 修饰符，而 final 修饰的类是不能被继承的。

```
public final class String implements java.io.Serializable,
Comparable<String>, CharSequence {
    // 省略...
}
```

3、String，StringBuffer，StringBuilder 的区别。

String 为字符串常量（因为内部数组 value[]有 final 修饰），而 StringBuilder 和 StringBuffer 均为字符串变量，即 String 对象一旦创建之后该对象是不可更改的，但后两者的对象是变量，是可以更改的。

在线程安全上，StringBuilder 是线程不安全的，而 StringBuffer 是线程安全的，因为 StringBuffer 在 append()方法上添加了 synchronized 修饰。

String，StringBuffer，StringBuilder 都被 final 修饰，不能继承。 String 变量创建后是放入方法区的常量池（或者常量池）中，而 StringBuilder 和 StringBuffer 则是存入堆中。

构造 String 对象的方式有很多，String 的内存分配比较特殊：

- 方式一：String str1="123";

通过引号直接创建字符串对象，先会从常量池中判断是否存在"123"对象，如果不存在，则会在常量池中创建该对象，并且返回常量池中"123"对象的引用给str；如果之前常量池存在"123"的话，则直接返回常量池中"123"的对象引用。

- 方式二：String str2=new String("123");

首先"123"是一个常量字符串，因此会先在常量池创建"123"字符串对象，然后在堆中再创建一个字符串对象，将"123"的字符数组复制到堆中新创建的对象字符数组中，因此该方式不仅会在堆中，还会在常量池中创建"123"字符串对象。

- 方式三：String str3="123".intern();

该种方式通过 intern 方法返回一个字符串引用，intern 方法是一个 native 方法，当常量池中存在"123"字符串常量时，则直接返回该常量池中的字符串引用；若不存在，则会先在常量池中创建"123"字符串对象，然后返回新创建对象的引用，与方式一类似。该方法常用于将某些经常访问的字符串对象保存在常量池中，避免经常创建对象。 方式四：String str4=str2.intern();

该种方式是在方式二基础上进行的，intern 方法会先判断常量池中是否存在与 str2 相同字符串的对象，若有，则返回该引用；若无，则在常量池创建一个引用（CONSTAT_String_info）指向 str2，然后返回该引用，实际上返回的是 str2 的引用。

具体可以参考这篇文献：String 对象内存分配(常量池和堆的分配)

4、ArrayList 和 LinkedList 有什么区别。

- ArraryList 是基于动态数组的数据结构，LinkeList 是基于链表的数据结果（LinkedList 是双向链表，有 next 也有 previous）。
- 对于随机访问 get 和 set，ArrayList 优于 LinkedList，因为 LinkedList 要移动指针。
- 对于新增和删除操作 add 和 remove，LinkedList 比较占优势，因为 ArrayList 要移动数据。

5、讲讲类的实例化顺序，比如父类静态数据，构造函数，字段，子类静态数据，构造函数，字段。当 new 的时候，他们的执行顺序。

Java 程序的初始化一般遵循 3 个原则（优先级以此递减）：

- 静态对象（变量）优先于非静态对象（变量）初始化，其中，静态对象（变量）只初始化一次，而非静态对象（变量）可能会初始化多次。
- 父类优先于子类进行初始化。

- 按照成员变量的顺序进行初始化。

当 new 时，他们的执行顺序为：父类静态变量、父类静态代码块、子类静态变量、子类静态代码块、父类非静态变量、父类非静态代码块、父类构造函数、子类非静态变量、子类非静态代码块、子类构造函数。

6、用过哪些 Map 类，都有什么区别，HashMap 时线程安全的吗，并发下使用的 Map 是什么，他们的内部原理分别是什么，比如存储方法，hashcode，扩容，默认容量等。

主要用过 HashMap，HashMap 不是线程安全的，并发下使用的 Map 是 ConcurrentHashMap，HashMap 是数组+链表+红黑树（JDK1.8 增加了红黑树部分）实现的。

HashMap 中 Node[] table 的默认长度 length 是 16，所能容纳的最大容量数据的 Node（键值对）个数为 threshold=length*Loadfactor。也就是说，在数组定义好长度之后，负载因子越大，所能容纳的键值对个数越多。

结合负载因子的定义公式可知，threshold 就是在此 Load factor 和 length(数组长度)对应下允许的最大元素数目，超过这个数目就重新 resize(扩容)，扩容后的 HashMap 容量是之前容量的两倍。默认的负载因子 0.75 是对空间和时间效率的一个平衡选择。

Hash 算法本质上就是三步：取 key 的 hashCode 值，高位运算，取模运算。

注意，一般 hashtable 桶数都会选择素数，因为素数因子最少，能减少冲突。但是，hashmap 却采用非常规方法，没有选用素数，而是选用合数，主要是为了在取模和扩容时做优化，同时为了减少冲突，HashMap 定位哈希桶索引位置时，也加入了高位参与运算的过程。

强力推荐这篇文章： Java 8 系列之重新认识 HashMap

7、Java8 的 ConcurrentHashMap 为什么放弃了分段锁，有什么问题吗，如果你来设计，你如何设计。

jdk8 放弃了分段锁而采用了 Node 锁，降低了锁的粒度，提高了性能，并使用 CAS 操作来确保 Node 的一些操作的原子性，取代了锁。

但是，ConcurrentHashMap 的一些操作使用了 synchronized 锁，而不是 ReentrantLock，虽然说 jdk8 中对 synchronized 进行了性能优化，但是我觉得使用 ReentrantLock 锁能更多的提高性能。

ReenTrantLock 的实现是一种自旋锁，通过循环调用 CAS 操作来实现加锁。它的性能比较好也是因为避免了使线程进入内核态的阻塞状态。

Synchronized 是悲观锁，在 jdk1.8 之后，加入了偏向锁，轻量级锁（自旋锁），性能得到了极大优化。

上述两种锁都是可重入锁，在锁的细粒度和灵活度方面，很明显 ReenTrantLock 优于 Synchronized。

8、有没有有顺序的 Map 实现类，如果有，他们怎么保证有序。

顺序的 Map 实现类：LinkedHashMap，TreeMap

- LinkedHashMap 是基于元素进入集合的顺序或者被访问的先后顺序排序。
- TreeMap 则是基于元素的固有顺序（由 Comparator 或者 Comarable 确定）。

9、抽象类和接口的区别，类可以继承多个类么，接口可以继承多个接口吗，类可以实现多个接口吗。

抽象类和接口的区别有：

- 抽象类可以有自己的实现方法，接口在 jdk1.8 之后才可以有自己的实现方法（用 default 修饰）。
- 抽象类的抽象方法必须有继承的子类实现，如果子类不实现，则子类也需要定义为抽象的；接口的抽象方法必须由实现类来实现，如果实现类不能实现接口中所有方法，则将实现类定位为抽象类。
- 抽象方法必须是 pulic/protected，接口中的变量隐式指定为 public static final 变量，抽象方法被隐式指定为 public abstract。
- 抽象类中可以存在普通属性、方法、静态属性和方法。如果一个类中有一个抽象方法，那么当前类肯定是抽象类。
- 子类只能继承一个父类，接口可以继承多个接口，类似于：Interface1 Extends Interface2, Interface3, Interface4……
- 类也可以实现多个接口。

主要注意的是，抽象方法不能用 synchronized 修饰。

从设计角度来看抽象类和接口：

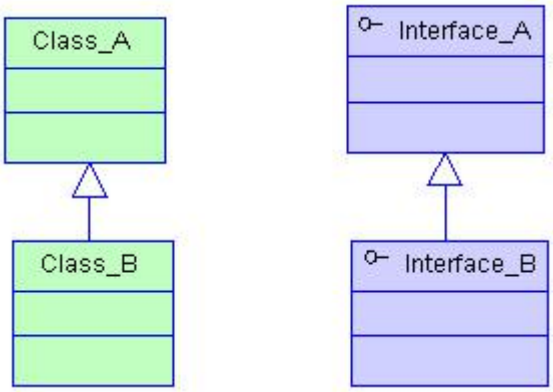
- 抽象类是 is a，是实例必须要有的，比如 Door 必须有开和关。而接口就是 has a，可以有也可以没有，比如 Door 可以有报警器，但不是必须的，是可拓展的行为。
- 抽象类强调的是同类事务的抽象，接口强调的是同类方法的抽象。
- 抽象类是从子类中发现了公共的东西，泛化出父类，然后子类继承父类；接口是根本不知道子类的存在，方法如何实现还不确认，预先定义。

- 若行为跨越不同类的对象，可使用接口；对于一些相似的类对象，用继承抽象类。

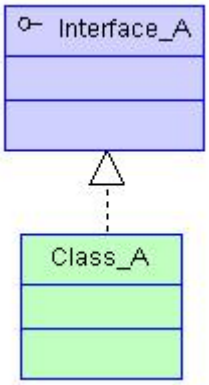
10、继承和聚合的区别在哪。

分别介绍这几种关系：

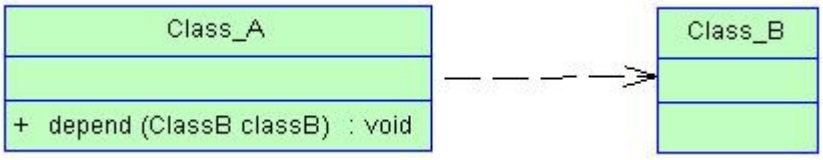
- 继承：指的是一个类继承另一个类的功能，并可以增加自己的的新功能的能力，Is-A 继承关系是类与类或者接口与接口之间最常见的关系。在 Java 类中通过关键词 extends 明确标识。



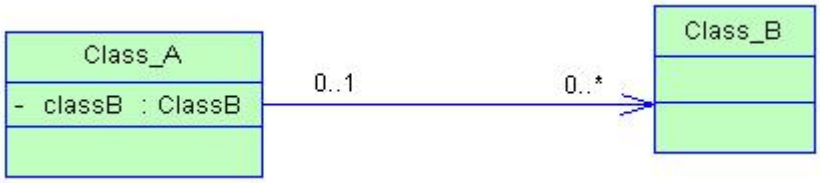
- 实现：指的是一个 class 类实现 interface 接口（可以多个）的功能，实现是类与接口之间最常见的关系。在 Java 类中通过关键词 implements 明确标识。



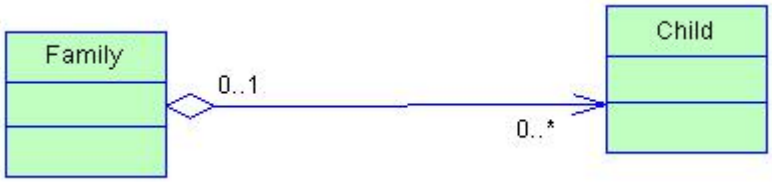
- 依赖：可以简单的理解，就是一个类 A 使用到了另一个类 B，而这种使用关系是具有偶然性的、、临时性的、非常弱的，但是 B 类的变化会影响到 A；比如某人要过河，需要借用一条船，此时人与船之间的关系就是依赖；表现在代码层面，为类 B 作为参数被类 A 在某个 method 方法中使用。



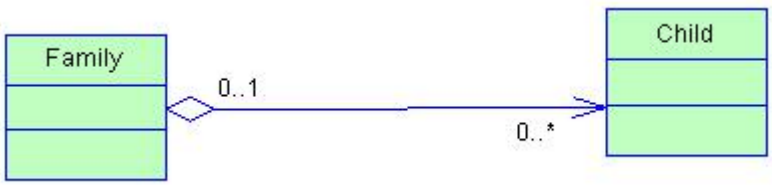
- 关联：关联关系在 java 中一般使用成员变量来实现，有时也用方法形参的形式实现。依然使用 Driver 和 Car 的例子，使用方法参数形式可以表示依赖关系，也可以表示关联关系，毕竟我们无法在程序中太准确的表达语义。



- 聚合：是关联关系的一种特例，他体现的是整体与部分、拥有的关系，即 has-a 的关系，此时整体与部分之间是可分离的，他们可以具有各自的生命周期，部分可以属于多个整体对象，也可以为多个整体对象共享；比如计算机与 CPU、公司与员工的关系等。



- 组合(a 拥有 b，a 没了 b 也就没了，实心)：也是关联关系的一种特例，他体现的是一种 contains-a 的关系，这种关系比聚合更强，也称为强聚合；他同样体现整体与部分间的关系，但此时整体与部分是不可分的，整体的生命周期结束也就意味着部分的生命周期结束；比如你和你的大脑；表现在代码层面，和关联关系是一致的，只能从语义级别来区分；



具体代码层次理解可以看这篇文章： java--依赖、关联、聚合和组合之间区别的理解

11、讲讲你理解的 nio，他和 bio 的区别是什么，谈谈 reactor 模型。

- BIO：同步阻塞式 IO，服务器实现模式为一个连接一个线程，即客户端有连接请求时服务器就需要启动一个线程进行处理，如果这个线程不做任何事情会造成不必要的线程开销，当然可以通过线程池来改善。

- NIO：同步非阻塞式 IO，服务器实现模式为一个请求一个线程，即客户端发送的连接请求都会注册到多路复用器上，多路复用器轮询到连接有 I/O 请求时才启动一个线程进行处理。是基于事件驱动思想完成的。

- AIO：异步非阻塞式 IO，服务器实现模式为一个有效请求一个线程，客户端的 I/O 请求都是由 OS 先完成了再通知服务器应用去启动线程进行处理。reactor 模型：反应器模式（事件驱动模式）：当一个主体发生改变时，所有的属性都得到通知，类似于观察者模式。

解释一下同步与异步：

- 同步 IO，是一种用户空间与内核空间的调用发起方式。同步 IO 是指用户空间线程是主动发起 IO 请求的一方，内核空间是被动接受方。

- 异步 IO 则反过来，是指内核 kernel 是主动发起 IO 请求的一方，用户线程是被动接受方。

再解释一下阻塞和非阻塞：

- 阻塞是指用户空间（调用线程）一直在等待，而且别的事情什么都不做；

- 非阻塞是指用户空间（调用线程）拿到状态就返回，IO 操作可以干就干，不可以干，就去干的事情。

I/O 多路复用是指内核一旦发现进程中指定的一个或者多个 IO 条件准备读取，它就通知该进程。也可以理解为，使用一个线程来检查多个文件描述符(Socket)的就绪状态，比如调用 select 和 poll 函数，传入多个文件描述符，如果有一个文件描述符就绪，则返回，否则阻塞直到超时。得到就绪状态后进行真正的操作可以在同一个线程里执行，也可以启动线程执行(比如使用线程池)。

具体 reactor 可以看这篇文章： Reactor 模式详解

在 Java 中，Selector 这个类是 select/epoll/poll 的外包类。

12、反射的原理，反射创建类实例的三种方式是什么。

反射机制：Java 反射机制是在运行状态中，对于任意一个类，如果知道一个类的名称，都能够知道这个类的所有属性和方法；对于任意一个对象，如果知道一个实例对象，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。

反射获取 Class 对象有三种方式：使用 Class.forName("类路径名称")静态方法。

- 使用类的.class 方法。
- 使用实例对象的 getClass()方法。

根据 Class 获取实例对象有两种方式：

- 直接使用字节码文件获取对应实例，Object o=clazz.newInstance();
- 对带参数的构造函数的类，先获取到其构造对象，再通过该构造方法类获取实例，如下。

```
/ /获取构造函数类的对象
Constroctor constroctor = clazz.getConstructor(String.class,Integer.class);
// 使用构造器对象的 newInstance 方法初始化对象
Object obj = constroctor.newInstance("龙哥", 29);
```

13、反射中，Class.forName 和 ClassLoader 区别。

ClassLoader 是类加载器，通过一个类的全限定名来获取描述此类的二进制字节流，遵循双亲委派模型，将.class 文件加载到 jvm 中，不会执行 static 中的内容（会先赋值为零值），只有在 newInstance 才会执行 static 块。

class.forName()方法内部实际上也是调用的 ClassLoader 来实现，但会对类进行初始化，执行类中的静态代码块，以及对静态变量的赋值等操作。

14、描述动态代理的几种实现方式，分别说出相应的优缺点。

动态代理有两种实现方式，分别是：jdk 动态代理和 cglib 动态代理。

- jdk 动态代理的前提是目标类必须实现一个接口，代理对象跟目标类实现一个接口。

- cglib 动态类是继承并重写了目标类（enhancer.create()方法返回的就是一个继承目标类的子类），所以目标类和方法不能被声明为 final。

具体可以看我的另一篇文章，详细讲解了，记动态代理

CGLib 创建的动态代理对象性能比 JDK 创建的动态代理对象的性能高不少，但是 CGLib 在创建代理对象时所花费的时间却比 JDK 多得多，所以对于单例的对象，因为无需频繁创建对象，用 CGLib 合适，反之，使用 JDK 方式要更为合适一些。

15、动态代理与 cglib 实现的区别。

- JDK 动态代理是通过接口中的方法名，在动态生成的代理类中调用业务实现类的同名方法；

- CGlib 动态代理是通过继承业务类，生成的动态代理类是业务类的子类，通过重写业务方法进行代理；

16、为什么 CGlib 方式可以对接口实现代理。

cglib 动态代理是继承并重写目标类，所以目标类和方法不能被声明为 final。而接口时可以被继承的。

17、final 的用途。

- final 修饰的对象不能被修改。
- final 修饰的类不能被继承。
- final 修饰的方法不能被重写（但可以被重载）。

18、写出三种单例模式实现。

单例模式的意思就是只有一个实例。单例模式确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例，这个类成为单例类。

单例模式有三种：懒汉式单例，饿汉式单例，登记式单例（最好的）。

具体可以看我的其他文章。

19、如何在父类中为子类自动完成所有的 hashcode 和 equals 实现，这样做有何优劣。

父类的 equals 不一定满足子类的 equals 需求。比如所有的对象都继承于 Object，默认使用的是 Object 的 equals 方法，在比较两个对象的时候，是看他们是否指向同一个地址。

但我们的需求时对象的某个属性想通了，就相等了，而默认的 equals 方法满足不了当前的需求，所以要重写 equals 方法，加入自定义逻辑。

重写 equals 方法就必须重写 hashcode 方法，否则就会降低 map 等集合的索引速度。

20、请结合 OO 设计理念，谈谈访问修饰符 public、private、protected、default 在应用设计中的作用。

OO 面向对象编程的设计理念是：

- 抽象，先不考虑细节
- 封装，隐藏内部实现
- 继承，复用现有代码
- 多态，改写对象行为

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变量和方法只让可信的类或者对象操作，对不可信的进行隐藏，所以我们可以通过修饰符 public、private、protected、default 来进行访问控制。

修饰符	类内部	本包	子类
public	√	√	√
protected	√	√	√
default	√	√	×
private	√	×	×

21、深拷贝和浅拷贝的区别。

- 浅拷贝：对基本数据类型进行值传递，对引用数据类型进行引用传递般的拷贝。
- 深拷贝：对基本数据类型进行值传递，对引用数据类型，创建一个新的对象，并复制其内容。

22、数组和链表数据结构描述，各自的时间复杂度。

- 数组是将元素在内存中连续存放，由于每个元素占用内存空间相同，可以通过下标迅速访问数组中任何元素。
- 链表恰好相反，链表中元素在内存中不是顺序存储的，而是通过存在元素中的指针联系到一起。
- 数组利用下标定位，时间复杂度为 O(1)，链表定位元素时间复杂度 O(n)；
- 数组插入或删除元素的时间复杂度 O(n)，链表的时间复杂度 O(1)。
- 数组和栈都存放在堆中，在虚拟机栈中只需要定义引用变量即可。

23、error 和 exception 的区别，CheckedException，RuntimeException 的区别。

- Exception 和 Error 都是继承于 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。
- error（错误）表示系统级的错误和程序不必处理的异常，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，除了推出运行外别无选择，它是由 java 虚拟机抛出的，是不可预料的异常情况。

- Exception（违例）表示需要捕捉或者需要程序进行处理的异常，他处理的是因为程序设计的瑕疵而引起的问题或者在外的输入等引起的一般性问题，是程序必须处理的，是可预料的异常情况。

Exception 又分为运行时异常，受检查异常。

- CheckedException（检查性异常）必须在编写代码时，使用 try catch 捕获（比如：IOException 异常）。

- RuntimeException（运行时异常）在代码编写时可以忽略捕获操作（比如：ArrayIndexOutOfBoundsException），这种异常是在代码编写或者使用过程中通过规范可以避免发生的。

24、请列出 5 个运行时异常。

- NullPointerException(空指针)
- IndexOutOfBoundsException(数组越界)
- ClassCastException(类转换异常)
- ArrayStoreException(数据存储异常，操作数组时类型不一致)
- IllegalArgumentException（非法参数异常）

25、在自己的代码中，如果创建一个 java.lang.String 类，这个类是否可以被类加载器加载？为什么。

不可以，双亲委派模式会保证父类加载器先加载类，就是 BootStrap（启动类）加载器加载 jdk 里面的 java.lang.String 类，而自定义的 java.lang.String 类永远不会被加载到。

如果打破双亲委派模式，自己写一个 classLoader 来加载自己写的 java.lang.String 类，但是也会发现也不会加载成功，具体就是因为针对 java.*开头的类，jvm 的实现中已经保证了必须由 bootstrp 来加载。

26、说一说你对 java.lang.Object 对象中的 hashCode 和 equals 方法的理解。在什么场景下需要重新实现这两个方法。

Object 类中的 equals 方法和“==”是一样的，没有区别，即俩个对象的比较是比较他们的栈内存中存储的内存地址。而 String 类，Integer 类等等一些类，是重写了 equals 方法，才使得 equals 和“==不同”，他们比较的是值是不是相等。所以，当自己创建类时，自动继承了 Object 的 equals 方法，要想实现不同的等于比较，必须重写 equals 方法。比如我们的需求是对象的某个属性相同，就

相等了，而默认的 equals 方法满足不了当前的需求，所以我们要重写 equals 方法。

如果重写了 equals 方法就必须重写 hashcode 方法，否则就会降低 map 等集合的索引速度。

27、在 jdk1.5 中，引入了泛型，泛型的存在是用来解决什么问题。

泛型的好处是在编译的时候检查类型安全，减少运行时由于对象类型不匹配引发的异常，并且所有的强制转换都是自动和隐士的，提高代码的复用率。

28、这样的 a.hashcode()有什么用，与 a.equels(b)有什么关系。

hashCode()方法是得到相应对象的 hash 值，它常用于基于 hash 的集合类，比如 Hashtable，HashMap，LinkedHashMap 等等。它与 equals()方法关系特别密切，根据 java 规范，两个通过 equals()方法来判断为相等的对象，必须拥有相同的 hashcode。

29、有没有可能 2 个不相等的对象有相同的 hashcode。

有可能，最简单的方法是在自己实现类中重写 hashcode()方法。

30、Java 中的 HashSet 内部是如何工作的？

HashSet 内部默认是通过一个 HashMap 来实现的。如下

```
public HashSet() {
    map = new HashMap<>();
}
private static final Object PRESENT = new Object();
public boolean add(E e) {
    return map.put(e, PRESENT)==null;//其中 PRESENT 是一个虚拟的对象。
}
```

31、什么是序列化，怎么序列化，为什么序列化，反序列化会遇到什么问题，如何解决。

序列化是一种用户处理对象流的机制，所谓对象流就是将对象的内容进行流化。

- 序列化：把对象转换为字节序列的过程称为对象的序列化。
- 反序列化：把字节序列恢复为对象的过程称为对象的反序列化。

对象实现序列化只需要实现 Serializable 接口即可。

32、java8 的新特性。

- Lambda 表达式和函数式接口
- 接口的默认方法和静态方法，允许在接口内添加新的方法。

- 重复注解，允许在同一个地方多次使用同一个注解。

33、Java 序列化和反序列化相关。

Java 序列化的的是对象的非静态字段及其值。

Transient 关键字修饰的成员变量在序列化过程中会被自动忽略。

而 static 静态成员属于类变量，也无法被序列化。

34、二叉树、平衡二叉树、红黑树的性质。

普通的二叉查找树的性质是：

- 若它的左子树不空，则左子树上所有结点的值均小于它的根节点的值；
- 若它的右子树上所有结点的值均大于它的根节点的值；
- 它的左、右子树也分别为二叉排序树。

平衡二叉树，又被称为 AVL 树，它的左右两个子树的高度差不能超过 1。平衡二叉树的目的是为了减少二叉查找树层次，提高查找速度。

- 红黑树，也是一种近似的平衡二叉树。红黑树的每个节点上都有存储位表示节点的颜色，可能是红或黑。红黑树的特性是：
- 每个节点或者是黑色，或者是红色。
- 根节点是黑色。
- 每个叶子节点（NIL）是黑色。 [注意：这里叶子节点，是指为空(NIL 或 NULL)的叶子节点！] 如果一个节点是红色的，则它的子节点必须是黑色的。
- 从一个节点到该节点的子孙节点的所有路径上包含相同数目的黑节点。

红黑树的时间复杂度为 O(logN)，效率很高。

有一个比较有意思的小公式，树中节点数=总分叉数+1。（这里的分叉数就是所有节点的度之和）

后续的 B-树或者 B+树都是多路搜索树，不一定是二叉的了。在 MySQL 那篇我会仔细说明。

35、Java 的 IO 流用了什么设计模式。

ava IO 流的设计是基于装饰者模式&适配模式，面对 IO 流庞大的包装类体系，核心是要抓住其功能所对应的装饰类。 装饰模式又名包装（Wrapper）模式。装饰模式以对客户端透明的方式扩展对象的功能，是继承关系的一个替代方案。装饰模式通过创建一个包装对象，也就是装饰，来包裹真实的对象。装饰模式以对客户端透明的方式动态地给一个对象附加上更多的责任。换言之，客户端

并不会觉得对象在装饰前和装饰后有什么不同。装饰模式可以在不创造更多子类的情况下，将对象的功能加以扩展。装饰模式把客户端的调用委派到被装饰类。装饰模式的关键在于这种扩展是完全透明的。

例如，InputStream 就是装饰者模式中的超类，ByteArrayInputStream，FileInputStream 相当于被装饰者，FilterInputStream 就是装饰者。

36、Java 中对象的引用。

Java 语言中，除了原始的八个基本数据类型外，其他都是引用类型，指向各种不同的对象。不同的引用类型，不同之处在于对象不同的可达性及对垃圾回收的影响。主要分为：

- 强引用：

只要引用存在，垃圾回收器永远不会回收。例如：

```
Object obj = new Object(); //其中 obj 就是强引用。通过关键字 new 创建的对象所关联的引用是强引用。
```

特点：JVM 内存空间不足时，JVM 宁愿抛出 OutOfMemoryError（OOM）运行时错误，使程序异常终止，也不会靠随意回收具有强引用的“存活”对象来解决内存不足的问题。对于一个普通对象，如果没有其他的引用关系，只要超过了引用的作用域（如超出局部变量作用范围）或者显示将相应（强）引用赋值为 null，就可以根据具体的垃圾回收机制被回收。

- 软引用：

软引用通过 SoftReference 实现，非必须引用，生命周期比强引用短一些，内存溢出之前进行回收。

应用场景： 软引用通常用来实现内存敏感的缓存。如果还有空闲内存，就可以暂时保留缓存，当内存不足时清理掉，这样就保证了使用缓存的同时，不会耗尽内存。

- 弱引用：

弱引用通过 WeakReference 类实现，弱引用的生命周期比软引用短，被弱引用关联的对象只能生存到下一次垃圾回收之前，。在垃圾回收器线程扫描它所管辖的内存区域的过程中，一旦发现具有弱引用的对象，不管当前的内存空间足够与否，都会回收他的内存。（由于垃圾回收器县城是一个优先级很低的线程，因此不一定会很快回收弱引用的对象）。

应用场景：弱引用同样是很多缓存实现的选择。

- 幻象引用（虚引用）：

通过 PhantomReference 类来实现。无法通过幻象引用访问对象的任何属性或函数。幻象引用仅仅是提供一种确保对象被 finalize 以后，做某些事情的机制。如果一个对象仅持有幻象引用，那么他就和没有任何引用一样，在任何时候都可能被垃圾回收器回收。

37、select、poll 和 epoll 之间的区别。

目前支持 I/O 多路复用的系统调用有 select、poll 和 epoll，I/O 多路复用就是通过一种机制，一个进程可以监视多个描述符，一旦某个描述符就绪（一般是读就绪或者写就绪），能够通知程序进行相应的读写操作。但 select、poll 和 epoll 本质上都是同步 I/O，因为它们都需要在读写事件就绪后自己负责进行读写，也就是说这个读写过程是阻塞的，而异步 I/O 则无需自己负责进行读写，异步 I/O 的实现会负责把数据从内核拷贝到用户空间。

与多进程和多线程技术相比，I/O 多路复用技术的最大优势是系统开销小，系统不必创建进程/线程，也不必维护这些进程/线程，从而大大减小了系统的开销。

- select

select 函数监视的文件描述符分 3 类，分别是 writefds、readfds、和 exceptfds。调用后 select 函数会阻塞，直到有描述符就绪（有数据 可读、可写、或者有 except），或者超时（timeout 指定等待时间，如果立即返回设为 null 即可），函数返回。当 select 函数返回后，可以通过遍历 fdset，来找到就绪的描述符。

它的优点是跨平台性能好，几乎在所有的平台上都支持。

select 的不足在三个方面，第一个是单个进程能够监视的文件描述符的数量存在最大限制，在 Linux 上一般为 1024。第二个是对 socket 进行扫描时是线性扫描，即采用轮询的方式，效率较低。第三个是需要维护一个用来存放大量 fd 的数据结构，这样会使用户空间和内核空间在传递该结构时复制开销大。

- poll

poll 和 select 没有区别，它的优点是没有最大连接数的限制，原因是它是基于链表来存储的，但是同样有一个缺点，则是大量的 fd 数组被整体复制于用户态和内核地址之间。

- epoll

epoll 是 select 和 poll 的增强版本，相对于前两者来说，它更加灵活，没有描述符的数量限制。epoll 使用一个文件描述符管理多个描述符，将用户关系的文件描述符的事件存放到内核的一个事件表中，这样在用户空间和内核空间的 copy 只需一次。所有的 fd 都存入红黑树中。

epoll 支持水平触发和边缘触发，最大的特点是边缘触发，它只告诉线程哪些 fd 刚刚变为就绪态，并且只会通知一次。还有一个特点是，epoll 通过“事件”的就绪通知方式，通过 epoll_ctl 注册 fd，一旦该 fd 就绪，内核就会采用类似 callback 的回调机制来把该 fd 放入到双向链表（保存已就绪的事件），epoll_wait 便可以收到通知。

epoll 的优点主要是三个方面：第一是没有最大并发连接的限制，能打开的 FD 的上限远大于 1024；第二是效率提升，不是轮询的方式，不会随着 FD 数目的增加效率下降，这时候用到了回调；第三是内存拷贝，使用 mmap 减少了复制开销。

综上：

- 表面上看 epoll 的性能最好，但是在连接数少并且连接都十分活跃的情况下，select 和 poll 的性能可能比 epoll 好，毕竟 epoll 的通知机制需要很多函数回调。
- select 低效是因为每次它都需要轮询。但低效也是相对的，视情况而定，也可通过良好的设计改善。

38、final 修饰的变量是引用不可变，还是引用的对象不能改变。

final 修饰的变量是引用不可变，但是引用的对象还是可以发生改变。

如果 final 修饰的是一个基本数据类型的变量，那么这个变量就确定了，不能变了。

而如果 final 修饰的是一个引用变量，那么该变量存的是一个内存地址，该地址就不能变了，但是该地址所指向的那个对象还是可以变的。

举个例子

```
final StringBuilder sb=new StringBuilder();
//sb=new StringBuilder();//程序报错。
sb.append("a");//成功执行，sb 对象内部可以改变。
```

39、Java 中 Comparable 和 Comparator 接口的区别。

Comparable 是排序接口，若一个类实现了 Comparable 接口，就意味着这个类支持排序，不需要再去指定比较器。接口中通过 x.compareTo(y)来比较 x 和 y 的大小。若返回负数，意味着 x 比 y 小；返回零，意味着 x 等于 y；返回正数，意味着 x 大于 y。

Comparator 是比较器接口。我们若需对某个类集合进行排序，而该类本身不支持排序（即没有实现 Comparable 接口），那么，我们就可以建立一个“该类的比较器”来进行排序。这个比较器只需要实现 Comparator 接口即可。

两者的联系则为：Comparable 相当于内部比较器，而 Comparator 相当于外部比较器。

40、如何通过反射获取和设置对象私有字段的值。

可以通过类对象的 getDeclaredField()方法字段（Field）对象，然后再通过字段对象的 setAccessible(true)将其设置为可以访问，接下来就可以通过 get/set 方法来获取/设置字段的值了。代码如下所示。

```
public class Test {
    static class User {
        private String name;
        private int age;
    }
    public static void main(String[] args) {
        Class<User> clz = User.class;
        try {
            Field age = clz.getDeclaredField("age");
            age.setAccessible(true);
            Object obj = clz.newInstance();
            age.setInt(obj, 18);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

41、sleep()方法和 yield()方法的区别。

- sleep()方法给其他线程运行机会时不考虑线程的优先级，因此会给低优先级的线程以运行的机会；yield()方法只会给相同优先级或更高优先级的线程以运行的机会。
- 线程执行 sleep()方法后转入阻塞（blocked）状态，而执行 yield()方法后转入就绪（ready）状态。
- sleep()方法声明抛出 InterruptedExecption，而 yield()方法没有声明任何异常。
- sleep()方法比 yield()方法（跟操作系统 CPU 调度相关）具有更好的可移植性。

需要注意的是：

如果线程 A 希望立即结束线程 B，则可以对线程 B 对应的 Thread 实例调用 interrupt 方法。如果此刻线程 B 正在 wait/sleep/join，则线程 B 会立刻抛出 InterruptedException，在 catch() {} 中直接 return 即可安全地结束线程。