个人整理 -Java 后端面试题 - 基础篇

作者：valarchie
原文链接：https://ld246.com/article/1583069770186
来源网站：链滴

- 标★号的知识点为重要知识点

★为什么重写equals还要重写hashcode?

如果重写了equals的话，可以根据自己的规则进行判定两个对象是否相等。（比如学号姓名一样的学实体即为相等）但是，如果将实体存进与哈希有关的集合当中时，哈希地址是根据hashcode进行计的。如果没有重写hashcode的时候，默认返回的是引用值。这将会导致Hash表当中存入两个相同的生。 所以所有有关hash的方法都会出问题。

两个对象值相同(x.equals(y) == true)，但却可有不同的hashCode?

如果重写equals但未重写hashCode时的话会导致这样的问题。按照规范的话，两对象equals相同的，hashCode也必须相同。

Object默认的hashcode实现?

Object类对于hashcode并没有具体的实现，调用的是jvm底层的c++代码进行计算。该方法返回的一个int数字，底层的cpp代码中

一共有六种默认的实现方式。

Object基类有哪些方法?

类中的方法：clone()，但是使用该方法必须实现Java.lang.Cloneable接口， equals()方法判断引用否一致，指向同一对象，即相等于==，只有覆写了equals()方法之后， 才可以说不同。hashcode()对象的地址， toString()， finalize()。

==比较的是什么?

在java中==符号比较的是两个对象在内存当中的引用值。

若对一个类不重写，它的equals()方法是如何比较的?

若不重写equals的话，将直接调用基类Object的equals方法，该方法比较的还是内存当中的引用值。

★java中的四种引用类型：

- 强引用：java默认声明的就是强引用，只要强引用存在，垃圾回收器将永远不会回收被引用的对象哪怕内存不足时，JVM也会直接抛出OutOfMemoryError，不会去回收。
- 软引用：在内存足够的时候，软引用对象不会被回收，只有在内存不足时，系统则会回收软引用对，如果回收了软引用对象之后仍然没有足够的内存，才会抛出内存溢出异常。这种特性常常被用来实缓存技术，比如网页缓存，图片缓存等。
- 弱引用：弱引用的引用强度比软引用要更弱一些，无论内存是否足够，只要 JVM 开始进行垃圾回，那些被弱引用关联的对象都会被回收。（常用作集合的key，避免内存泄露）
- 虚引用：虚引用是最弱的一种引用关系，如果一个对象仅持有虚引用，那么它就和没有任何引用一，它随时可能会被回收。

java的基础类型和字节大小。

一个字节是8位

整形数： byte 1字节 short 2字节 int 4字节 long 8字节

浮点数：float 4字节 double 8字节

char：2字节

boolean：1字节

Java支持的数据类型有哪些？什么是自动拆装箱？

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

自动拆装箱是在代码编译之后，自动进行 int i = Integer.valueOf(i) 的转换操作，

最早的1.5JDK之前没有自动拆装箱，所以当时是需要手动写拆箱装箱。

什么是自动拆装箱？

Integer total = 99;这是自动装箱。int totalprim = total;这是自动拆箱。意思是将一个int基本类型到一个需要 Integer类型的地方会自动装箱，将一个Integer类型的包装类型放到需要int的地方会自拆箱。包装类型当中，有一个类似 字符串常量池的常数常量池，数值大小在-127~128之间。

int 和 Integer 有什么区别？

- int是基本类型，Integer是int的包装类型。
- int的默认值是0,Integer的默认值null。
- Integer缓存了-128~127之间的数。

java8的新特性。

- **lambda表达式支持函数式编程，简化代码。**
- **接口可有默认方法与静态方法。**
- **方法引用的简便写法。**
- **可重复注解。**
- **扩展注解的使用范围：局部变量、泛型变量、异常。**
- **可获取参数的名字。**
- Optional对于Null优雅的处理。
- **java集合中的Stream的增强处理（类似数据库），以及并行处理。**

- **Date/Time API更好的支持。（duration特别好用）**
- 内置javasrcipt引擎
- Base64包
- **JVM中的永久区被移除，换做元数据区**

lambda表达式的优缺点（待讨论）

- 优点：
  - 简洁，不再需要匿名内部类。
  - 并行计算在大数据量的情况下会比for循环更块
- 缺点：
  - 普通情况下比for循环慢
  - 调试不方便
  - 类型转换要特殊处理

一个十进制的数在内存中是怎么存的?

以二进制补码形式存储，最高位是符号位，正数的补码是它的原码，负数的补码是它的反码加1，在反码时符号位不变，符号位为1，其他位取反

浮点数为什么精度会发生丢失?

2进制的小数无法精确的表达10进制小数，计算机在计算10进制小数的过程中要先转换为2进制进行算，这个过程中出现了误差。

例如0.125 二进制表示为0.001 但0.4转换为二进制的话，计算会出现无限小数。所以会导致精度丢。

★浅析Java中的final关键字?

修饰方法表示不能重写，修饰类型表示不能继承，修饰变量表示需要初始化并赋值，并且不能重新赋。

（如果这个值是一个对象，对象内的数据可改变，但此对象的引用不能改变）。

什么是值传递和引用传递?

值传递代表的是将实参的值拷贝传给形参，形参的改变并不会影响到实参。

引用传递指的是将实参的引用值传给行参，方法内基于引用值找到对象并修改对象中的值是会生效的但地址值依然是引用值的拷贝，无法生效。

Java中方法参数的传递规则?

java中是值传递的方式。意味着方法参数这个变量都不是真正要操作的变量，而是变量的拷贝。引用型的地址值改变也不会影响原来的变量，但修改引用类型中的值，可以生效，因为修改会直接作用到

中的java对象。

★当一个对象被当作参数传递到一个方法后，此方法可改变这个对象属性，并可返回变化后的结果，那么这里到底是值传递还是引用传递？

值传递指的是传递进来的对象的引用值是一份拷贝。（比如将传递进方法的参数对象的引用值设为nul的话，

在方法外部调用该对象的方法的话，是不会报空指针的。因为传递进来的是引用的拷贝值。）

String和StringBuffer的区别？

- String不可变，进行更改字符串的话相当于改了字符串的引用。
- StringBuffer可变，修改字符串是基于原来的字符串进行修改，不会创建新的字符串。

★StringBuffer和StringBuilder的区别，从源码角度分析？

StringBuffer源码中方法加了synchronized进行修饰，所以是线程同步的。而StringBuilder中的方法有用

synchronized进行修饰，所以不是线程同步的。

String，Stringbuffer，StringBuilder的区别？

String是字符串常量，编译之后其实是调用StringBuilder。

StringBuilder是字符串变量，线程非安全，效率高。在循环拼接字符串中，推荐使用StringBuider.

StringBuffer是字符串变量，线程安全，效率低。

如何输出一个某种编码的字符串？

new String(str.getBytes("ISO-8859-1"), "GBK")；通过添加字符编码进行转换。

★什么是字符串常量池？

在Java堆中，有一块专门放置字符串的池子不同JDK版本不一样，在编译期就存在的字符串将会

直接存入这个池中，或者在不同代码地方的相同的字符串将会直接引用同一个字符串，为什么能

这样引用是因为字符串的不可变性，java常量池的实现其实是享元模式的思想，可以节省创建的时间并且节省空间。

★String方法intern()的作用是？

1. 当常量池中不存在"abc"这个字符串的引用，将这个对象的引用加入常量池，返回这个对象的引用。
2. 当常量池中存在"abc"这个字符串的引用，返回这个对象的引用；

★String为什么是不可变的？

因为常量池中的字符串会被多个地方引用到，如果字符串是可变的话，更改一个字符串将导致原先引用

该字符串的所有地方都被改变了。这个不可变是指堆中常量池的字符串本身不可变，代码中的引用是变的。

String类本身设计出不可变是为了防止客户程序员继承String类，造成破坏。

为什么String不可变，为什么是final?

- 因为String类中持有的value数组使用final进行修饰的，所以它的数组引用就变成不可变的了。
- 这么做的原因之一是因为String有常量池这个概念。比如一个String a在常量池中被很多变量
- 引用了，如果String是可变的，当改变了a当中的字符，会造成所有引用这个字符变量的地方都跟着变。
- 但如果String是不可变的话，a原本的字符不变，将一个新的字符串赋予给a，这样就不会影响到其引用之前a字符串的地方。

String能继承吗?

- 不能，因为String类是经过final修饰的。
- 这么做的原因可能是因为String是底层的类，用final修饰的话，自然而然的方法也会被final修饰。此在调用String的任何方法的时候，都采用JVM的内嵌机制，效率会有较大的提升。
- 还有可能是出于安全的考虑，防止继承的子类改写String的方法，再将子类以String父类的形式进调用。例如使用+号操作符重载?

★String s = new String("xyz");究竟产生了几个对象，从JVM角谈谈?

当JVM执行到这一行的时候，会先去常量池中查看是否有xyz的字符串，没有的话，会新建这个字符放入

常量池，如果有的话则不创建。new 操作符将会在对象堆中创建一个字符串对象。所以这个操作可能成1

个或者2个对象。

★String拼接字符串效率低，你知道原因吗?

如果是“ab” +"cd"的话，并不会效率低，因为经过编译优化之后会自动变成“abcd”，
但如果是代码中动态拼接，或者循环拼接的话，比如a+b+c+d将会产生a,ab,abc这三个临时的字符，
创建不必要的字符串是浪费性能和空间的，所以大部分的String拼接字符串效率会低。
其实+号，编译器转换成StringBuilder的append。有点类似于C++当中的操作符重载。

String的常见API?

- indexOf 检索子串的位置
- substring 获取子串
- trim 去除字符串前后空白
- charAt 获取字符的位置

- startWith\endWith 检测是否是指定字符串开头或者结尾
- toUpperCase 转换成大写
- toLowerCase 转换成小写
- valueOf 将其他类型转为字符串
- split 分隔字符串

String.valueOf和Integer.toString的区别?

没有区别。String.valueOf(i)，方法内部调用的也是Integer.toString(i)。

★Java中的subString()真的会引起内存泄露么?

假设A字符串很大，B=A.substring，在java1.6版本以前B会持有A字符串类型内置的字符数组，AB是共享同一个字符数组，就算A被回收了，但是B还是持有A字符串内很大的字符数组，在Java1.7之后修复了这个问题，在substring中，会重新复制一份字符数组给B,性能会比1.6版本差一点，但就不会导致内存泄露。假设A="123456789",B="9",但是其实B当中还是持有完整的"123456789"组

&和&&的区别?

- 一个是按位与，一个是逻辑与。逻辑与会短路。按位与不会短路。

在Java中，如何跳出当前的多重嵌套循环?

- 定义一个标号。然后在内层循环当中break 标号；

```
ok:

for (int j = 0; j < 1000; j++) {
    for (int k = 0; k < 1000; k++) {
        System.out.println(k);
        break ok;
    }
}
```

你能比较一下Java和JavaSciprt吗?

- java是编译型语言，强类型，面向对象。由JVM执行。
- javascript是解释性语言，弱类型，一般面向过程。由浏览器引擎执行。

简述正则表达式及其用途。

- 主要用来匹配出特定规则的字符串。（如电话号码）

Java中是如何支持正则表达式操作的?

- String类中的split\match\replace 等方法都支持正则。
- Pattern类支持正则。

浅析Java中的static关键字?

修饰变量代表这是类的共享变量，修饰方法的话表示类的静态方法，可在不实例化对象的情况下进行法调用，修饰代码块的话表示类初始化会调用这些代码，还有一个非常少见的用法就是静态导入，调方法就不用使用类型.方法名的方式调用，而是直接使用方法名进行调用。

- static可以修饰内部类，但是不能修饰普通类。静态内部类的话可以直接调用静态构造器（不用对）
- static修饰方法， static 方法就是没有 this 的方法。在 static 方法内部不能调用非静态方法，反过是可以的。而且可以在没有创建任何对象的前提下，仅仅通过类本身来调用 static 方法。这实际上正 static 方法的主要用途。方便在没有创建对象的情况下来进行调用（方法/变量）。
- static修饰变量，就变成了静态变量，随类加载一次，可以被多个对象共享。
- static修饰代码块，形成静态代码块，用来优化程序性能，将需要加载一次的代码设置成随类加载静态代码块可以有多个。

★你对Java中的volatile关键字了解多少?

volatile关键字的作用在于内存可见性（更改变量导致各个线程的变量缓存失效，使其需要从主内存取新的变量值）、 禁止指令重排序。volatile关键字并不能保证原子性。适合于修饰状态值，相当于个轻量级的锁

★i++是线程安全的吗？如何解决线程安全性?

i++操作包括三步，读取i，对i+1，写入i,所以这并不是原子性操作，并不能保证线程安全。可以对i用olatile关键字进行修饰在对i的加操作用synchronized 或者 lock 或者 CAS 方式(AtomicInteger)进操作。

★请谈谈什么是CAS?

Compare And Swap 比较并更换

内存值 预期值 想修改的值 如果内存值等于预期的值那么修改为想修改的值。

底层是调用cpu指令集的cmpxchg指令来实现。

AtomicInteger的原理就是使用volatile保证可见性，利用CAS保证原子性。

★从字节码角度深度解析 i++ 和 ++i 线程安全性原理?

i++操作包括三步，读取i，对i+1，写入i，所以在各自的本地线程中i+1操作可能会丢失

i++ 先将本地变量i读取到操作数栈，再进行++写会本地变量，所以操作数栈里的是原值。

++i 先将本地变量进行++,再读取到操作数栈，所以操作数栈里的值是+1后的值。

Java有哪些特性，举个多态的例子。

- 例如Animal类型都有move方法，Cat和Bird都集成了Animal类。但是Cat的move是用四肢跑。Bir

的move使用翅膀飞。

- 多态是一种动态方法绑定的机制。

类和对象的区别?

- 好比猫属于生物中的一个物种（类）。而家里养了一只叫kitty的猫，kitty就是猫这个物种中一个具并真实存在的对象。

Object当中的主要方法?

```
- getClass()：  获取对象所属类型
- hashCode()：  获取对象的
- equals()   ：  判断对象是否相等
- toString()：  将对象转换为字符串
- clone()    ：  克隆一个对象
- wait()...  ：  暂停线程让出锁，进入锁池
- notify()   ：  唤醒线程
- notifyAll()：  唤醒所有线程
- finalize()：  在对象回收时会执行的方法
```

重载和重写的区别？相同参数不同返回值能重载吗?

- 重载是在同一个类中，方法名一样，但参数列表不一样的静态多态。虚拟机无法根据返回值的不同判断使用哪个重载方法。
- 重写是子类重写了父类的方法，方法签名一样，但实现不一样。
- 重载其实是一种静态多态，在编译成字节码的时候已经完成绑定。
- 重写其实是一种动态多态，在程序运行期间才完成绑定。

Java中是否可以覆盖(override)一个private或者是static的方法?

- Java中static方法不能被覆盖，因为方法覆盖是基于运行时动态绑定的，而static方法是编译时静态定的。
- 还有私有的方法不能被继承，子类就没有访问权限，肯定也是不能被覆盖。

类加载机制的双亲委派模型，好处是什么?

- 双亲委派模型是每次收到类加载请求时，先将请求委派给父类加载器完成，如果父类加载器无法完加载，那么子类尝试自己加载。
- 双亲委派机制可以避免加载子类自定义的Object类、String类等一些跟jdk命名相同的类。使得加的类都是同一个。这样才安全。

当一个类收到了类加载请求，他首先不会尝试自己去加载这个类，而是把这个请求委派给父类去完成每一个层次类加载器都是如此，因此所有的加载请求都应该传送到启动类加载其中，只有当父类加载反馈自己无法完成这个请求的时候（在它的加载路径下没有找到所需加载的Class），子类加载器才尝试自己去加载。

静态变量存在哪里?

- 存在方法区（永久代，在1.8之后是元数据区）当中。

什么是泛型?

- 泛型是参数化类型。避免为不同类型参数的方法进行多次重载，方便开发。并且编译器会进行泛型检查，在开发上更安全。
- 但是泛型信息在运行时其实是擦除的。
- 简单的说就是类型的参数化，就是可以把类型像方法的参数那样传递。
- 泛型使编译器可以在编译期间对类型进行检查以提高类型安全，减少运行时由于对象类型不匹配引的异常。
- 并且所有的强制转换都是自动和隐式的，提高代码的重用率和性能。

★extends 和super 泛型限定符的区别?

- extends表示的是所需类型是本类或其子类。 表示的是限定上界。
- super表示的是所需类型是本类或其父类。 表示的是限定下界。

★是否可以在static环境中访问非static变量?

- 因为静态的成员属于类，随着类的加载而加载到静态方法区内存，当类加载时，此时不一定有实例建，没有实例，就不可以访问非静态的成员。

通过反射创建对象的方法?

- 1.通过类对象的.newInstance方法。
- 2.通过类对象的构造器对象的.newInstance方法。

如何通过反射获取和设置对象私有字段的值?

- 先通过getDeclaredFields();方法获取属性列表。
- 再通过field.setAccessible(true);打开访问权限。
- 再通过field.set(obj, value);将值填入。

★反射的实现与作用?

在java中实现反射是通过Class对象，得到该对象后可通过该对象调用相应的方法获取该类中的属性或方法，给属性赋值或
调用该类中的方法。

反射的用途我个人的理解主要是为了弥补Java静态语言不灵活的缺陷。而实现的一种动态编程的方法

动态决定调用某些类，属性，方法。

作用如下：
1、java集成开发环境，每当我们敲入点号时，IDE便会根据点号前的内容，动态展示可以访问的字段方法。
2、java调试器，它能够在调试过程中枚举某一对象所有字段的值。
3、web开发中，我们经常接触到各种配置的通用框架。为保证框架的可扩展性，他往往借助java的射机制。
例如Spring框架的依赖反转（IOC）便是依赖于反射机制。

一般来说反射是用来做框架的，或者说可以做一些抽象度比较高的底层代码。

java反射机制。

可以在运行时判断一个对象所属的类，构造一个类的对象，判断类具有的成员变量和方法，调用1个象的方法。

4个关键的类：Class，Constructor，Field，Method。

- getConstructor获得构造函数/getDeclardConstructor；
- getField/getFields/getDeclardFields获得类所生命的所有字段；
- getMethod/getMethods/getDeclardMethod获得类声明的所有方法，

正常方法是一个类创建对象，而反射是1个对象找到1个类。

java支持多继承吗？

- java不支持类的多继承，只支持接口的多继承。

接口和抽象类的区别是什么？

- 抽象类可以有构造方法（可让子类调用），接口中不能有构造方法。
- 抽象类中可以有普通成员变量，接口中没有普通成员变量。
- 抽象类中可以包含非抽象普通方法（模板方法模式），接口中的所有方法必须都是抽象的。
- 抽象类中的抽象方法的访问权限可以是 public、protected 和(默认类型子类不能继承)，接口中的象方法只能是 public 类型的，并且默认即为 public abstract 类型
- 抽象类中可以包含静态方法，在 JDK1.8 之前接口中不能不包含静态方法，JDK1.8 以后可以包含。
- 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权限可以是任意的，但口中定义的变量只能是 public static final 类型的，并且默认即为 public static final 类型。
- 一个类可以实现多个接口，用逗号隔开，但只能继承一个抽象类，接口不可以实现接口，但可以继接口，并且可以继承多个接口，用逗号隔开。

Comparable和Comparator的区别？

- 简而言之
- Comparable是一个接口比较器。类自己实现了排序规则，可直接调用集合类的sort
- Comparator是一个外部比较器。通过定义一个独立的比较器，在集合进行排序的时候传入一个比器。

- 个人偏向于Comparator的做法，比较低耦合，可以应用不用的comparator进行不用的比较。

面向对象的特征有哪些方面？

- 抽象、继承、封装、多态

final、finally、finalize的区别？

- final是修饰符。修饰变量、方法、类。
- finally是异常处理块当中最后执行的语句。
- finalize是在垃圾收集器将对象从内存中清除出去之前调用的方法。

静态内部类和普通内部类的区别？

内部类：

- 内部类中的变量和方法不能声明为静态的。（非静态内部类只有在实例化外部类的时候才会加载，果内部类有静态变量的话，导致可以通过Out.Inner.static 访问到该内部类的静态变量，但此时内部还未被加载）
- 内部类实例化：B是A的内部类，实例化B：A.B b = new A().new B()。
- 内部类可以引用外部类的静态或者非静态属性及方法。（因为内部类实例化的话，外部类必然已经例化）

静态内部类：

- 静态内部类属性和方法可以声明为静态的或者非静态的。
- 实例化静态内部类：B是A的静态内部类，A.B b = new A.B()。
- 静态内部类只能引用外部类的静态的属性及方法。（因为引用外部类中的属性时，外部类未必实例了）

内部类可以引用外部类的成员吗？

- 普通内部类的话可以引用。静态内部类的话只能引用外部类的静态变量。

Java的接口和C++的虚类的相同和不同处。

- 相当于接口和抽象类的区别。

JAVA语言如何进行异常处理，关键字：throws,throw,try,catch,finally分别代表什么意义？在try块中可以抛出异常吗？

- throw 直接抛出一个异常。
- throws 声明将要抛出的异常。
- try包围可能出现异常的范围。
- catch在异常出现时，执行的代码

- finally不管异常有没有出现，都会执行的一段代码。

在try块中可以抛出异常。

Java中throw和throws的区别是什么?

throw是new 一个异常，要么自己捕获，要么声明继续传给调用者。

throws是声明一个异常但是不处理，将异常的处理交给调用者。

finally语句块你踩过哪些坑?

比较少，因为关于流的关闭操作都有放在finally中进行关闭，return操作很少会放入finally中。

程序最终执行到哪个块就在哪个有return的块中进行return。

- finally块一定会执行.
- finally前有return，会先执行return语句，并保存下来，再执行finally块，最后return。不会覆盖前的返回值。
- finally前有return、finally块中也有return，先执行前面的return，保存下来，再执行finally的retur
，

覆盖之前的结果，并返回

return在finally中则会覆盖。但尽量不要乱写多个return。

谈一下面向对象的"六原则一法则"。

- 单一职责原则
- 开闭原则
- 依赖倒置原则
- 里氏替换原则
- 接口隔离原则
- 合成聚合复用原则
- 迪米特法则

请问JDK和JRE的区别是什么?

JRE是Java运行时环境。它是运行编译好的Java程序所必需的一切包，包括Java虚拟机(JVM)、Java库、java命令和其他基础设施。
如果您只关心在计算机上运行Java程序，那么您将只安装JRE。
JDK是Java开发工具包，功能齐全的SDKforJava。它是JRE的超集，包含编译器(javac)和一些开发工具如javadoc和jdb)。

他们两个的明显区别就是一个用来运行程序。一个用来开发程序（开发程序当然需要能运行程序的环）。

Java中的LongAdder和AtomicLong的区别

LongAdder类与AtomicLong类的区别在于高并发时前者将对单一变量的CAS操作分散为对数组cells多个元素的CAS操作，取值时进行求和；而在并发较低时仅对base变量进行CAS操作，与AtomicLon类原理相同。

error和exception有什么区别?

● Error类一般是指与虚拟机相关的问题，如系统崩溃，虚拟机错误，内存空间不足，方法调用栈溢等。

对于这类错误的导致的应用程序中断，仅靠程序本身无法恢复和和预防，遇到这样的错误，建议让程终止。

● Exception类表示程序可以处理的异常，可以捕获且可能恢复。遇到这类异常，应该尽可能处理异
，

使程序恢复运行，而不应该随意终止异常。

什么是java序列化，如何实现java序列化?

序列化：把Java对象转换为字节序列的过程。

反序列化：把字节序列恢复为Java对象的过程。

用途：把对象的字节序列持久化到硬盘上或者在网络上传输。

序列化的实现：将需要被序列化的类实现Serializable接口，

该接口没有需要实现的方法。Serializable只是标注该对象是可被序列化的，

然后使用一个输出流(如：FileOutputStream或者ByteArrayOutputStream)来构造一个ObjectOutpu Stream(对象流)对象，接着，使用ObjectOutputStream对象的writeObject(Object obj)方法就可以参数为obj的对象写出(即保存其状态)，要恢复的话则用输入流。

其好处一是实现了数据的持久化，通过序列化可以把数据永久地保存到硬盘上（通常存放在文件里），

二是，利用序列化实现远程通信，即在网络上传送对象的字节序列。

当序列化ID不一致时，会导致序列化失败。

★对象克隆和实现方式

克隆的对象可能包含一些已经修改过的属性，而new1个对象属性都还是初始化时候的值，被复制克的类要实现Clonable接口，覆盖clone()方法，访问修饰符为public，方法中调用super.clone()得到需要的复制方法，类中的属性类也需要实现Clonable接口，覆写clone()方法，并在super中也调用子性类的clone()复制，才可以实现深拷贝。

或者写到流中序列化的方式来实现，不必考虑引用类型中还包含引用类型，直接用序列化来实现对象深复制拷贝，即将对象写到流，再从流中读出来，需要实现seriazation接口。

反射中，Class.forName和classloader的区别?

区别：

● Class.forName除了将类的.class文件加载到jvm中之外，还会对类进行解释，执行类中的static块。

● 而classloader只干一件事情，就是将.class文件加载到jvm中，不会执行static中的内容，只有在new nstance才会去执行static块。

ClassLoader就是遵循双亲委派模型最终调用启动类加载器的类加载器，实现的功能是“通过一个类全限定名来获取描述此类的二进制字节流”， 获取到二进制流后放到JVM中。Class.forName()方法际上也是调用的CLassLoader来实现的。

java 判断对象是否是某个类的类型方法?

● instanceof 运算符是用来在运行时指出对象是否是特定类的一个实例。instanceof通过返回一个布值来指出，这个对象是否是这个特定类或者是它的子类的一个实例。

● getClass判断，如o.getClass().equals(ClassA.class)。（使用instanceof来判断一个对象是不是属某个类，但是有时候这个类是继承于一个父类的，所以，不能严格判断出是不是自己的类，而不是自的父类。）

★Java内存泄露的问题调查定位：jmap，jstack的使用等等

jstack 观测进程状态，可以查看死锁、阻塞、等待资源的异常进程。

先使用jps查看当前运行的java进程，然后使用 jstack id是观测进程内的死锁、阻塞、等待资源等异情况。

jmap 观测虚拟机内的内存使用情况以及检测频繁gc，jmap -histo:live 进程号 查看哪些数量异常高实例进行分析。或者jmap -dump:format=b,file=d:/heap.hprof <pid> 分析内存快照。如果是字串的话可以分析其内的内容。

一般来说都是业务相关的数据。

Arrays.sort实现原理和Collections.sort实现原理

事实上Collections.sort方法底层就是调用的array.sort方法，其中涉及legacyMergeSort(a)：归并排，

ComparableTimSort.sort()：Timsort排序。Timsort排序是结合了合并排序（merge sort）和插入序（insertion sort）而得出的排序算法。

核心过程：

TimSort 算法为了减少对升序部分的回溯和对降序部分的性能倒退，将输入按其升序和降序特点进行分区。排序的输入的单位不是一个个单独的数字，而是一个个的块-分区。其中每一个分区叫一个run针对这些 run 序列，每次拿一个 run 出来按规则进行合并。每次合并会将两个 run合并成一个 run。并的结果保存到栈中。合并直到消耗掉所有的 run，这时将栈上剩余的 run合并到只剩一个 run 为止这时这个仅剩的 run 便是排好序的结果。

（0）如何数组长度小于某个值，直接用二分插入排序算法

（1）找到各个run，并入栈

（2）按规则合并run

foreach和while的区别(编译之后)

增强for循环只是java提供的一个语法糖，在编译之后其实是利用Iterator和While进行遍历。如果foreach中进行增加移出操作之后又紧跟hasNext()操作会触发异常。这其中涉及到fail-fast（同步修改失败和fail-safe（对副本进行修改）机制。

https://www.cnblogs.com/luyu1993/p/7148765.html

cusor比size大，导致认为还有next元素

★cloneable接口实现原理，浅拷贝or深拷贝

cloneable接口实际上只是标记该类是否可以克隆，具体操作需要重写，一般在重写的方法内调用clon方法。如果在拷贝这个对象的时候，只对基本数据类型进行了拷贝，而对引用数据类型只是进行了引的传递，而没有真实的创建一个新的对象，则认为是浅拷贝。反之，在对引用数据类型进行拷贝的时，创建了一个新的对象，并且复制其内的成员变量，则认为是深拷贝。

深拷贝有三种方式：

- Cloneable接口实现深拷贝，实现clone方法，并且在clone方法内部，把该对象引用的其他对象也clone一份，这就要求这个被引用的对象必须也要实现Cloneable接口并且实现clone方法。
- 序列化实现深拷贝。
- FastJSON或者利用反射依次复制值。

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

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（父类实例成员变量）、

父类构造函数、

子类非静态变量（子类实例成员变量）、

子类构造函数。

★描述动态代理的几种实现方式，分别说出相应的优缺点。

JDK动态代理：利用反射机制生成一个实现代理接口的Proxy类，在调用具体方法前调用InvokeHandlr来处理。

CGlib动态代理：利用ASM（开源的Java字节码编辑库，操作字节码）开源包，将代理对象类的class件加载进来，通过修改其字节码生成子类来处理。

区别：JDK代理只能对实现接口的类生成代理；CGlib是针对类实现代理，对指定的类生成一个子类并覆盖其中的方法，这种通过继承类的实现方式，不能代理final修饰的类。

JDK动态代理

1、因为利用JDKProxy生成的代理类实现了接口，所以目标类中所有的方法在代理类中都有。

2、生成的代理类的所有的方法都拦截了目标类的所有的方法。而拦截器中invoke方法的内容正好就代理类的各个方法的组成体。

3、利用JDKProxy方式必须有接口的存在。

4、invoke方法中的三个参数可以访问目标类的被调用方法的API、被调用方法的参数、被调用方法的回类型。

cglib动态代理

1、 CGlib是一个强大的,高性能,高质量的Code生成类库。它可以在运行期扩展Java类与实现Java接。
2、 用CGlib生成代理类是目标类的子类。
3、 用CGlib生成 代理类不需要接口
4、 用CGLib生成的代理类重写了父类的各个方法。
5、 拦截器中的intercept方法内容正好就是代理类中的方法体

CheckedException，RuntimeException的区别。

Throwable有两个子类，一个Error，一个Exception。直接继承Exception的是受检异常，继承Exception的子类RuntimeException的是非受检异常。

受检异常：编译器强制必须捕获。不是程序本身的错误，只是外部因素如文件读写异常、数据库异常为了保障程序的健壮性，必须要捕获。属于不可控的异常。但是为了对异常进行抛出、捕获和处理异需要增加较多代码，会降低代码的可读性。

非受检查异常：不必须捕获，抛给虚拟机。一般是由于程序不严谨导致的错误。比如空指针异常或者数异常等。这种属于一种程序的bug，按照道理来讲是程序员编程不当导致的，这种bug一旦发生应消除。

请列出5个运行时异常。

ClassCastException(类转换异常)

IndexOutOfBoundsException(数组越界)

NullPointerException(空指针)

ArithmeticException (算数异常，除0)

NumberFormatException (字符串转数字错误)

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

不会，因为类加载是双亲委托机制，String类会从顶层类加载器加载，轮不到自定义的加载器加载。

tomcat结构，类加载器流程

tomcat的WebClassLoader先尝试自己加载，加载不到的话，再交给父类进行双亲委托机制。这个加器不会加载java核心类。

什么情况下会发生栈内存溢出。

栈溢出有两种，一种是stackoverflow，另一种是outofmemory，前者一般是因为方法递归没终止条，后者一般是方法中线程启动过多。

转自我的github

技术讨论群QQ:1398880