# 基础

equals和==的区别

equals

是object类的方法，通常会在自定义类中按照现实情况重新定义，用于判断两个对象是否相等。一般要求，如果equals方法判断AB两个对象相等，则两个对象的hashCode值应该也相等（hashCode也是Object的方法）；如果AB两个对象的hashCode值相同，但是不一定两个对象相等；如果hashcode不同那么对象肯定不相等。因此在重写equals方法的时候通常也要求重写hashCode方法。

==

对于基本类型，可以用来判断值是否相等。

对于引用类型，则用于判断两个对象是否引用同一块内存区域。

对于Integer在数据范围-128~127的时候，变量都是对内存中同一个的缓存的引用，因此可以用于判断相等，超出这个范围则不行。

String在使用String s = “abc”方法赋值的时候，是对字面量常量池中“abc”字符串缓存的引用，如果再次出现s1使用相同方式生成，则两者引用的内存相同；如果通过Intern方法生成的字符串同样取值自字符串常量池，因此也可以用==判断相等。

如果使用new String(“test”)生辰新字符串，则”test”不进入字符串的字面量常量池，用==判断不相等。

为什么重写equals要重写hashcode

一般要求，如果equals方法判断AB两个对象相等，则两个对象的hashCode值应该也相等（hashCode也是Object的方法）；如果AB两个对象的hashCode值相同，但是不一定两个对象相等；如果hashcode不同那么对象肯定不相等。因此在重写equals方法的时候通常也要求重写hashCode方法。

StringBuilder和StringBuffer的区别

由于String是不可变类型，因此对于经常是需要变动的情况建议使用StringBuffer或者StringBuilder。

StringBuffer和StringBuilder的方法基本类似，两者的区别在于StringBuffer是线程安全的（由于String是不可变类，所以String本身肯定是线程安全的）

StringBuffer使用了同步机制（synchronized）保证了线程安全，但是因此也带来了效率的下降，所以一般情况推荐使用StringBuilder

StringBuilder的常用api

sBuilder.append（String或者char），拼接

sBuilder.delete(0,sBuilder.length())，清空

sBuilder.length()，长度

String是如何实现它的不可变的，为什么要设置String为不可变？

String类底层是一个添加了private final关键字的char数组，并且在内部的方法中都没有允许对数组中的元素进行直接的修改。因此不可以单独的改变String的值。String在取值变化的时候是通过引用新的字符串，整体替换。

String不可变的主要原因是为了安全。String由于本身的“可读性”的特点，经常用于作为一些校验内容，例如hashCode或者key，如果设为可变，会很容易通过新变量引用相同地址在加以修改的方式，危害到值。

【注】System.arraycopy方法使用的是浅拷贝，但是在拷贝字符串数组的时候，由于指向的内容是常量池中的缓存（也不一定都是缓存，如果是new出来的则不是），所以修改拷贝出来的内容对原内容可能不造成影响。

接口和抽象类的区别

接口规定了一组行为，用于标识一组行为能力，常见的有List，Map，Serializable等。

接口通常规定了方法的描述和返回类型，方法具体的实现交给实现类去完成。1.8以后，支持了默认方法。接口中也可以含有变量，但是所有的变量都会默认带上public static final修饰（如果编写代码的时候再次加上，会报错）

抽象类通常用于表示对象在身份上的共性。抽象类中可以含有数目不定的抽象方法，由子类负责实现，也可以含有非抽象的方法。只要通过abstract关键字修饰的类都是抽象类，抽象类中可以没有抽象方法。

接口和抽象类都不能直接实例化，需要由实现类或者子类去重新实现/重写全部的抽象方法。Java中一个类只能继承自一个父类，但是可以实现多个接口。

重写和重载的区别

重载：方法名相同，但是参数不同，一个类中一个方法可以多次重载，编译器会自动给识别所对应的方法

重写：方法名相同，参数也相同。主要是用于实现抽象方法或者对于父类/接口中方法的覆盖（覆盖后无法访问原来的方法）重写时子类要比父类抛出更少的异常（抛出的异常类是父类抛出的异常类的子类）；子类的访问权限要比父类的更大。

【注】：返回值不作为方法描述的一部分，也就是说允许存在返回值不同的重写

深拷贝和浅拷贝的区别

浅拷贝：创建一个新的对象，将当前对象的非静态字段（静态字段为类所有，不需要复制）复制到该新对象；如果字段是值类型的，则复制值，如果字段是以用类型，则引用相同的对象（也就是旧对象和新对象引用相同的对象）

深拷贝：有别于浅拷贝中，如果字段是引用类型，则新建一个引用类型，继续深拷贝该字段，而不是直接引用。

java三大特性

封装，继承，多态（面向对象的三大特性）

object的方法

clone()

equals()

getClass()，获取构造类型（实例化所使用的类）

toString()，返回@+内存地址

hashCode()

wait()

wait(long time)

notify()

notifyAll()

finalize()，用于提醒JVM进行垃圾回收（但是没有强制性）

Object类是所用类的父类，因此所有类都拥有上述方法，部分常用方法会根据实际情况重写

有些object的方法已经比较过时了，为什么还要保留

保持向下兼容的能力

作为设计类时的一个提示，可能会需要使用到哪些基本能力

static关键字，什么作用，放在哪里

用static修饰的变量，会在内存中拥有一份唯一的单独的空间，所有方法访问的时候都来访问此空间（因此可以看作为类所拥有）。如果不加static，则每一个实例中都会存在一份拷贝。

被static修饰的方法和变量，可以不需要实例化直接拷贝。（这两者在类加载过程中就已经进入内存）

被static修饰的方法块，静态方法块在main函数之前执行，可以用于初始化静态域

被static修饰的类只能是内部类。

如何高效的进行数组拷贝（System.arraycopy()的原理）

System.arraycopy（原数组，拷贝起始位置，目标数组，目标数组接收起始位置，拷贝长度）一共五个参数

arraycopy方法通过调用Native方法，加上JVM的特殊优化，实现了较高性能的拷贝。

对于基本类型的一维数组进行深拷贝，多维数组或者一维引用类型的数组进行浅拷贝

【注】由于该方法的版本早于Java1.0，所以存在命名不规范（不满足驼峰命名）的情况。此外还有NullPointerException，其实应该叫做NullReferenceException

# 集合

## List

### 【2】arraylist原理

### 扩容底层原理，访问速度，和linkedlist的区别

### 【2】linkedlist原理

## 【4】hashmap

### 你会如何设计一个hashmap

### 为什么size是2的幂

### resize的过程

### 效率受到哪些因素的影响

### 扰动函数的作用

是让高16位的值和低16位的值进行异或运算，使得高位数据参与运算，增加了hashcode的随机性，减少了碰撞的概率。

由于是高频的算法，所以采用了位运算增加效率

### hash值的计算

### 为什么使用红黑树，红黑树的特点

### 【2】1.7和1.8的改动

|  |  |
| --- | --- |
| 1.7 | 1.8 |
| 数组+链表 | 数组+链表+红黑树（防止链表过长降低查找效率） |
| 扩容时使用链表头插法（可能存在死循环） | 扩容时链表尾插法（不存在死循环） |
| 扩容时候需要重新根据hashcode取模 | 扩容直接判断高一位的值是否是1，是的话移动2^n位 |
| 先扩容，再插入 | 先插入再判断是否需要扩容 |

### put数据的过程

### 链表很长会怎样，什么情况转红黑树

链表中节点长度到达8以后就会转为红黑树

长度降低到6以后就会退回链表

### 对hashmap的resize进行优化

### concurrenthashmap

### 【2】和hashmap的比较

### 线程安全的问题

## hashtable

### 和hashmap的区别

### 理想的查找效率是多少

### 怎么解决hash碰撞

# 并发

## 多线程

### 【2】线程池核心参数

### 【2】线程池执行过程

### 常用线程池

### 拒绝策略有哪些

### 使用executor的弊端

### 【2】sleep和wait的区别

### 为什么sleep是Thread的方法，不是Object的

### 【2】volatile的作用

## ThreadLocal

### 是什么

### 底层原理

## AQS

### 框架理解

### 源码理解

## 锁

### 【2】synchronized

### 性能为什么提高了

### 锁升级过程

### 【2】和lock的区别

### 使用场景

### 常见的锁机制

### ReentrantLock的原理

### CAS底层原理

# JVM

## 【2】内存

### 每个区域的介绍

### 版本变化

### 堆和栈

### 栈的默认大小

### 字符串常量池是在堆吗？

### OOM和StackOverFlow

### 内存泄漏

### 引用类型（强软弱虚四种）

## 【2】GC算法

### 优缺点

### 使用场景

### 对应的垃圾收集器

### 分区

### 判断方法（GC触发条件）

## GC器

### 判断对象死亡的方法

### GCroot有哪些，为什么选择他们作为GCroot

### 什么时候STW

### 引用计数的循环引用的情况

## 类加载过程

### 双亲委派机制

### 原理

### 作用

# 设计模式

## 【3】单例

### 为什么双检测

### 为什么用volatile修饰

### 手写懒汉式，DLC懒汉式

### 装饰模式和代理模式的区别

## 生产者消费者

## 工厂方法模式

### 手写一个工厂方法模式