目录

[第一章、javase 2](#_Toc9132)

[一、常见命名约定： 2](#_Toc20073)

[二、Java文件夹说明： 2](#_Toc29510)

[三、Person p = new Person(); 在内存中做了哪些事情： 3](#_Toc20640)

[四、显示初始化： 4](#_Toc25393)

[五、常用cmd窗口命令： 4](#_Toc16468)

[六、数据类型的内存占用和取值范围： 4](#_Toc27731)

[七、&&、||与&、|的区别： 5](#_Toc22604)

[八、位运算符>>、<<、>>> 5](#_Toc22001)

[九、算法：求N个数中前K个最大数: 6](#_Toc9222)

[十、Java和javaw的区别： 7](#_Toc431)

[十一、常用的文档注释标签： 8](#_Toc31272)

[十二、返回值是引用数据类型的时候出的错： 12](#_Toc24779)

[十三、String 可以用字符数组初始化： 13](#_Toc28525)

[十四、使用 == 做比较String： 13](#_Toc9178)

[十五、枚举（enum） 14](#_Toc11719)

[十六、构造方法私有化 16](#_Toc470)

[十七、Java堆空间和栈内存之间的区别 17](#_Toc14805)

[十八、Java内存的构成 18](#_Toc6210)

[二十、Java中的地址值 26](#_Toc16615)

[二十一、常量池 26](#_Toc13733)

[二十一\_1 静态常量池 26](#_Toc7192)

[二十一\_2 运行时常量池 27](#_Toc10415)

[二十一\_3 常量池的好处 28](#_Toc22857)

[二十二、描述符和特征签名： 28](#_Toc16841)

[二十三、方法区： 30](#_Toc24846)

[二十四、修饰符的权限： 33](#_Toc25054)

[二十五、调用子类方法和成员变量： 33](#_Toc18624)

[二十六、建包： 34](#_Toc2200)

[二十七、static 34](#_Toc6731)

[二十八、instanceof 35](#_Toc9580)

[二十九、多态 35](#_Toc26503)

[三十、编译器的常量优化 36](#_Toc27914)

[三十一、静态代码块 36](#_Toc4855)

[三十二、Java中输出二进制和十六进制以及八进制 36](#_Toc26839)

[三十三、arraylist扩容机制 37](#_Toc30384)

[三十四、快捷键 38](#_Toc11108)

[三十五、Date类 38](#_Toc3199)

[三十六、集合 38](#_Toc23993)

[三十七、关键字用来修饰什么 39](#_Toc30168)

[三十八、Ipv6 40](#_Toc3526)

[三十九、饿汉式与懒汉式有什么区别么（单例模式） 40](#_Toc23058)

[四十、静态内部类 42](#_Toc23066)

[四十一、lambda表达式 44](#_Toc1965)

[四十二、接口中的内容 45](#_Toc8353)

[四十三、反射中的泛型 46](#_Toc12456)

[四十四、重载时用integer来代替int算重载 47](#_Toc28701)

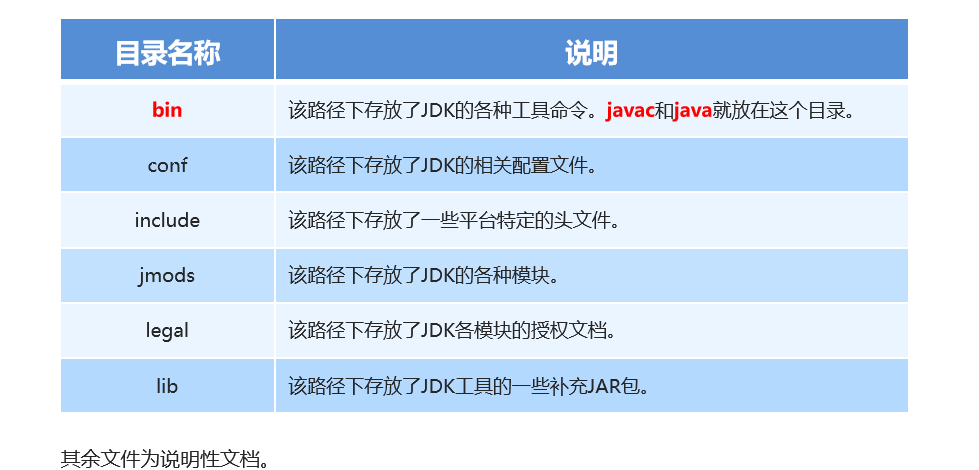
[四十五、怎么用反射获取继承的保护方法 48](#_Toc8851)

# 第一章、javase

一、常见命名约定：

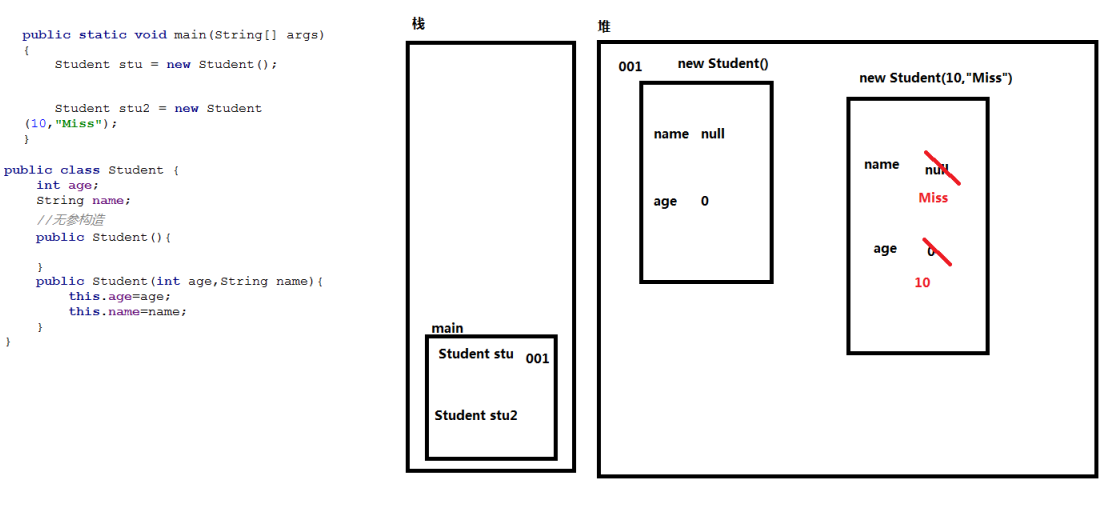


二、Java文件夹说明：



三、Person p = new Person(); 在内存中做了哪些事情：

1. 将Person.class文件加载到内存中。
2. 在堆内存中创建一个对象Person。
3. 把Person中的属性进行默认初始化。
4. 把Person中的属性进行显示初始化。
5. 调用构造代码块(如果没有，不执行这个操作)。
6. 调用构造函数进行初始化。
7. 在栈内存中声明Person类型的变量P。
8. 把堆内存的地址(引用)赋给了栈内存中P。



有参构造方法的初始化是：先默认初始化再给成员变量赋值；

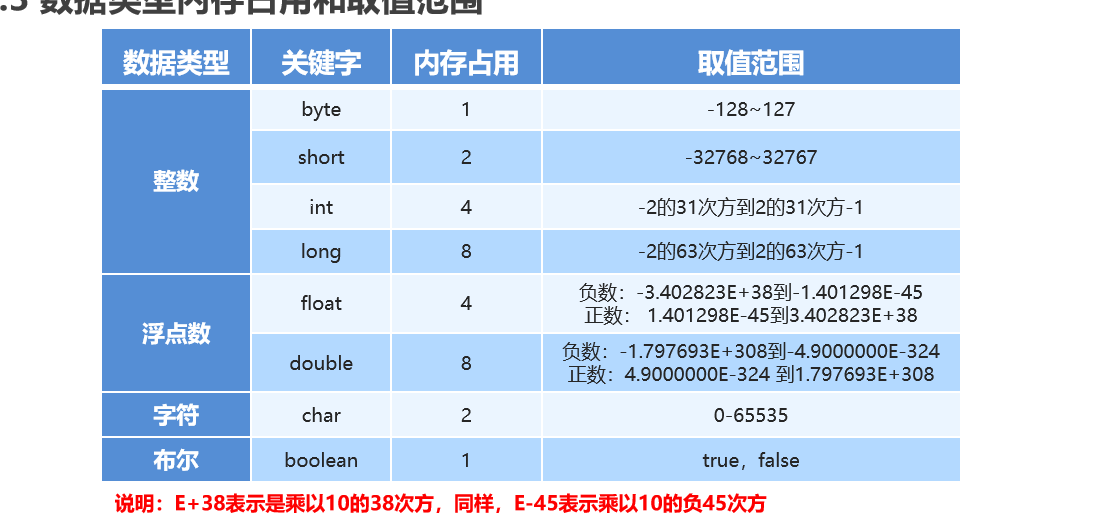
四、显示初始化：

显示初始化就是方法中的变量初始化；即局部变量初始化；（上面的初始化还包括了成员变量的常量赋值）

五、常用cmd窗口命令：



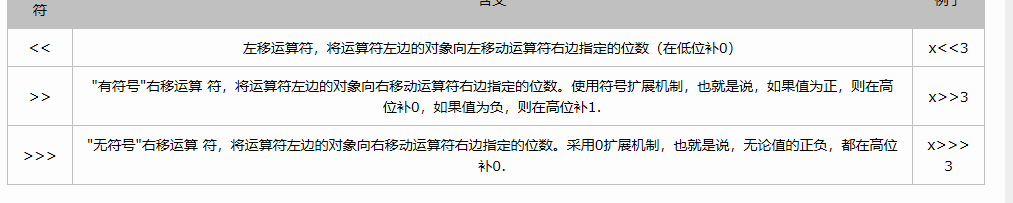
六、数据类型的内存占用和取值范围：



七、&&、||与&、|的区别：

&&，||会短路，即能判断结果时不算后面的式子；

八、位运算符>>、<<、>>>



位运算符主要针对二进制，它包括了：“与”、“非”、“或”、“异或”。从表面上看似乎有点像逻辑运算符，但逻辑运算符是针对两个关系运算符来进行逻辑运算，而位运算符主要针对两个二进制数的位进行逻辑运算。

int a=129;  
int b=128;  
System.out.println("a 和b 与的结果是："+(a&b));

运行结果  
a 和b 与的结果是：128  
下面分析这个程序：  
“a”的值是129，转换成二进制就是10000001，而“b”的值是128，转换成二进制就是10000000。根据与运算符的运算规律，只有两个位都是1，结果才是1，可以知道结果就是10000000，即128。

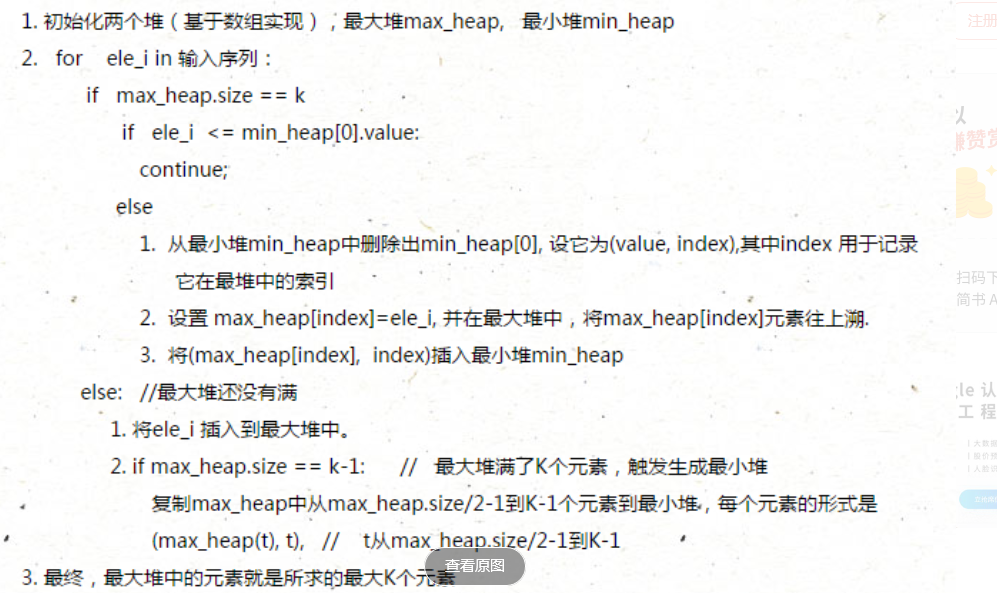
九、算法：求N个数中前K个最大数:

1. 用最多K个元素的最大堆max\_heap记录最终结果
2. 最大堆的max\_heap的所有叶子节点，组成最小堆组成最小堆min\_heap
3. 该思路的提出，受启发于逆波兰式算法，双数据结构解决表达式计算问题

比较优势：

1. 众所周知，用快速排序的思想，也可以解出N个数中的前K个最大数，但该算法的好坏，取决于PIVOIT点，对于该点的取舍，目前已知最好策略是取N个数中的随机某个元素。所以需要的轮次比较多。本算法一轮，就能找出最大K个元素，复杂度O(N)(视K为常数的情况下）。
2. 该算法可以对付N超级大（亿级别），K相对较小的情况，因为内存复杂度只有O（K)

算法步骤如下：



十、Java和javaw的区别：

1. java启动的程序是命令行程序或阻塞程序，如果该程序未执行完毕或未被关闭，则所打开的命令行将被阻塞，不能执行其它命令如dir等，可以通过Ctrl+C等方式关闭程序；
2. javaw启动的程序是窗口程序或非阻塞程序，在使用该命令运行程序后，可接着在命令行中执行下一命令，且启动的程序与命令行无关不依赖命令行，不能通过Ctrl+C关闭。
3. 也可以这样解释：两者的根本区别：java.exe是win32控制台应用,javaw.exe是一个win32的GUI应用

十一、常用的文档注释标签：

标签

描述

示例

@author

标识一个类的作者

@author description

@deprecated

指名一个过期的类或成员

@deprecated description

{@docRoot}

指明当前文档根目录的路径Directory Path

@exception

标志一个类抛出的异常

@exception exception-name explanation

{@inheritDoc}

从直接父类继承的注释

Inherits a comment from the immediate surperclass.

{@link}

插入一个到另一个主题的链接

{@link name text}

{@linkplain}

插入一个到另一个主题的链接，但是该链接显示纯文本字体

Inserts an in-line link to another topic.

@param

说明一个方法的参数

@param parameter-name explanation

@return说明返回值类型@return explanation

@see

指定一个到另一个主题的链接

@see anchor

@serial

说明一个序列化属性

@serial description

@serialData

说明通过writeObject( ) 和 writeExternal( )方法写的数据

@serialData description

@serialField

说明一个ObjectStreamField组件

@serialField name type description

@since

标记当引入一个特定的变化时

@since release

@throws和 @exception标签一样.The

@throws tag has the same meaning as the @exception tag.

{@value}

显示常量的值，该常量必须是static属性。

Displays the value of a constant, which must be a static field.

@version

指定类的版本

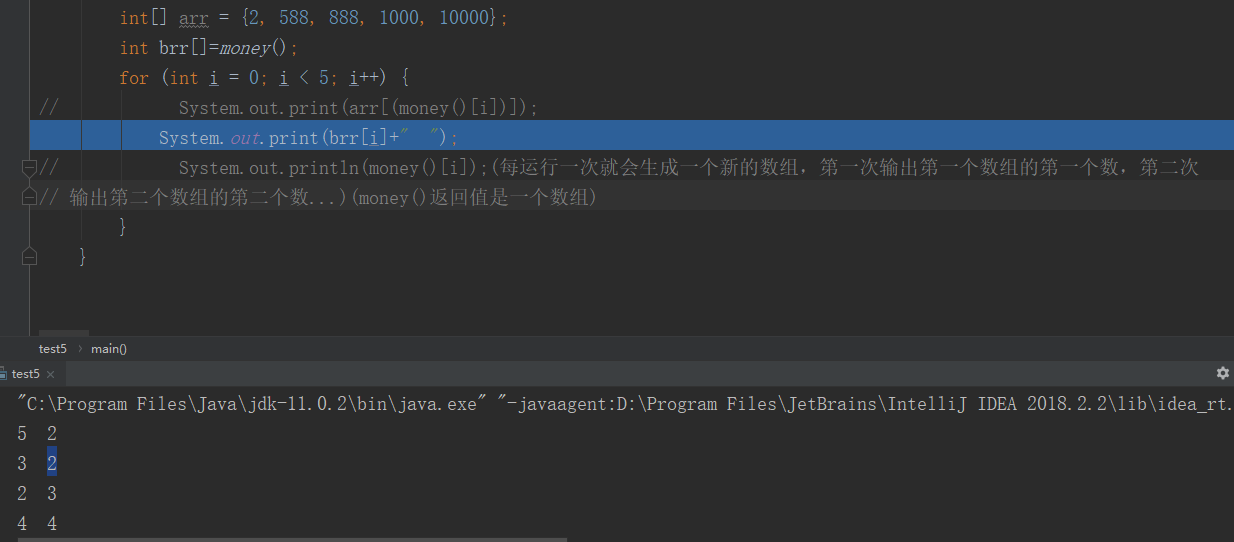
@version info



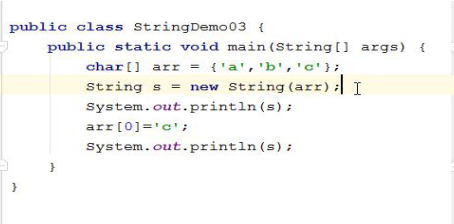


十二、返回值是引用数据类型的时候出的错：

每调用一次方法就会有一个新对象出现



十三、String 可以用字符数组初始化：



输出：abc

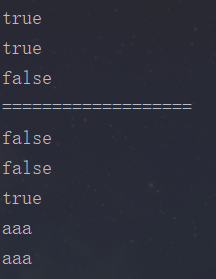
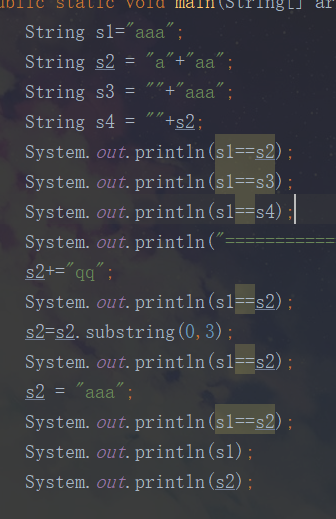
abc

不是存地址，而是把内容取出来转换成bety[]存储，把bety[]的地址给字符串s；

十四、使用 == 做比较String：

基本类型：比较的是数据值是否相同

引用类型：比较的是地址值是否相同



1. 有一个常量池，使用相同的字符串常量赋值的字符串指向同一个字符串常量的地址；
2. 字符串改变后会指向的地址会改变；
3. substring()（返回字符串的子集，0开头）会改变地址值；
4. 常量运算在编译时就会处理，所以算成情况1；

十五、枚举（enum）

Enum 一般用来表示一组相同类型的常量。枚举允许用[类型化](https://www.baidu.com/s?wd=%E7%B1%BB%E5%9E%8B%E5%8C%96&tn=24004469_oem_dg&rsv_dl=gh_pl_sl_csd" \t "https://blog.csdn.net/zyhlwzy/article/details/_blank)的值替换常量并在任何地方使用这些类型；是一个特殊的类；

**十五\_1枚举和泛型的关系**

Java中的每一个单独的枚举自动继承自泛型类Enum<T>，在这里T就是枚举类型本身。Java编译器在编译时代表开发者做了这个转换，拓展一下枚举声明public enum DaysOfTheWeek 如下：

public class DaysOfTheWeek extends Enum< DaysOfTheWeek > {

// Other declarations here

}

这也就说明了为什么枚举可以实现接口但不能继承其他类：因为它隐式的继承自Enum<T>并且我们在使用对象的公共方法时已经讨论过，Java中不支持多继承。

实际上每一个继承自Enum<T>的枚举允许定义泛型类、接口和方法，通过这种方式可以让枚举类型的实例参数化或者类型参数化。比如：

public<T extends Enum< ? >> void performAction(final T instance) {

// Perform some action here

}

在上面的方法声明中，类型T被约定为任意枚举类型的实例并且Java编译器将会对其做验证。

**十五\_2枚举的方法**

基础类 Enum<T>为自动继承它的枚举实例提供了一些非常有用的方法。

String name()

返回枚举声明声明的枚举常量的名称

int ordinal()

返回枚举常量的次序（即枚举声明时的位置，初始常量分配的位置是0）

此外，Java编译器为每个枚举类型自动生成两个更有用的静态方法（让我们将这个特殊的枚举类型假设为T）。

T[] values()

返回枚举T所声明的所有常量

T valueOf(String name)

返回指定名称的枚举常量

十六、构造方法私有化

一个类的构造方法私有化之后，只能从类的内部取得实例化对象。那么，要考虑的问题是，如何把内部生成的

instance 对象拿到外部来，这样外部就可以直接实例化了。

正常情况下，instance属性只能通过singleton类的实例化对象才可以进行调用。

如果没有实例化对象的时候依然可以取得instance对象，则就需要将instance声明成static访问类型，因为使用static

声明的变量，可以直接使用类名称进行访问。



如果现在不希望一个类产生过多的对象的话，则必须采用单例设计模式。

十七、Java堆空间和栈内存之间的区别

1.堆内存由应用程序的所有部分使用，而栈内存仅由一个执行线程使用。

2.每当创建一个对象时，它总是存储在堆空间中，而栈存储器包含对它的引用。栈内存仅包含堆空间中对象的本地原始变量和引用变量。

3.存储在堆中的对象是全局可访问的，而栈内存不能被其他线程访问。

4.堆栈中的内存管理以LIFO（后进先出）方式完成，而在堆内存中则更复杂，因为它是全局使用的。堆内存分为Young-Generation，Old-Generation等。

5.栈内存是短暂的，而堆内存从应用程序执行的开始到结束都存在。

6.我们可以使用-Xms和-Xmx JVM选项来定义堆内存的启动大小和最大大小。我们可以使用-Xss来定义栈内存大小。

7.当堆栈内存已满时，Java运行时抛出，java.lang.StackOverFlowError而如果堆内存已满，则抛出java.lang.OutOfMemoryError:Java Heap Space错误。

8.与堆内存相比，栈内存大小非常小。由于内存分配（LIFO）的简单性，与堆内存相比，栈内存非常快。

十八、Java内存的构成

整块区域分为Young Generation、Tenured Generation（年轻代→老年代 ）（j在于JDK8中把存放元数据中的永久内存从堆内存中移到了本地内存(native memory)中）

**十八\_1 Young区：**

Young区又分为：Eden、Survivor Space。

Survivor Space又分为 To Survivor、 From Survivor，

Eden和Survivor的比例是8:1:1

**十八\_2 垃圾回收（GC）的产生过程**

1.新生成的对象在Eden区完成内存分配

2.当Eden区满了，再创建对象，会因为申请不到空间，触发minor GC，进行young(eden+From survivor)区的垃圾回收。（为什么是eden+From survivor：两个survivor中始终有一个survivor是空的，空的那个被标记成To Survivor）

3.minorGC时，Eden不能被回收的对象被 复制 到空的survivor（也就是放到ToSurvivor，同时Eden肯定会被清空），另一个survivor（From Survivor）里不能被GC回收的对象也会被 复制 到这个survivor（To Survivor），始终保证一个survivor是空的。（MinorGC完成之后，To Survivor 和 From Survivor的标记互换）

4.当做第3步的时候，如果发现存放对象的那个survivor满了，则这些对象被copy到old区，或者survivor区没有满，但是有些对象已经足够Old（通过XX:MaxTenuringThreshold参数来设置）默认是 15 岁，这些对象就会成为老年代。

5.但这也不是一定的，对于一些较大的对象 ( 即需要分配一块较大的连续内存空间 ) 则是直接进入到老年代。

6.当Old区被放满的之后，进行完整的垃圾回收，即 Full GC

FullGC时，整理的是Old Generation里的对象，把存活的对象放入到Permanent Generation（永久代）里。

**十八\_3 看下Minor GC、Major GC和Full GC之间的区别：**

1）Minor GC：年轻代的回收过程称为 Minor GC

1.当 JVM 无法为一个新的对象分配空间时会触发 Minor GC，比如当 Eden 区满了。所以分配率越高，越频繁执行 Minor GC。

2.内存池被填满的时候，其中的内容全部会被复制，指针会从0开始跟踪空闲内存。Eden 和 Survivor 区进行了标记和复制操作，取代了经典的标记、扫描、压缩、清理操作。所以 Eden 和 Survivor 区不存在内存碎片。写指针总是停留在所使用内存池的顶部。

3.执行 Minor GC 操作时，不会影响到永久代。从永久代到年轻代的引用被当成 GC roots，从年轻代到永久代的引用在标记阶段被直接忽略掉。

4.质疑常规的认知，所有的 Minor GC 都会触发“全世界的暂停（stop-the-world）”，停止应用程序的线程。对于大部分应用程序，停顿导致的延迟都是可以忽略不计的。其中的真相就 是，大部分 Eden 区中的对象都能被认为是垃圾，永远也不会被复制到 Survivor 区或者老年代空间。如果正好相反，Eden 区大部分新生对象不符合 GC 条件，Minor GC 执行时暂停的时间将会长很多

2）Major GC vs Full GC

Major GC 是清理老年代。

Full GC 是清理整个堆空间—包括年轻代和老年代。

首先，许多 Major GC 是由 Minor GC 触发的，所以很多情况下将这两种 GC 分离是不太可能的。另一方面，许多现代垃圾收集机制会清理部分永久代空间，所以使用“cleaning”一词只是部分正确。

**十八\_4 垃圾回收的回收器**

--串行（–XX:+UseSerialGC ）

Out ofBox算法，年轻代串行复制，年老代串行标记整理，主要用于桌面应用

--并行（–XX:+UseParallelGC ）

年轻代暂停应用程序，多个垃圾收集线程并行的复制收集，年老代暂停应用程序，与串行收集器一样，单垃圾收集线程标记整理。JDK6.0启用该算法后，默认启用了-XX:+UseParallelOldGC，性能大为提高

--并发(Concurrent Low Pause Collector)(–XX:+UseConcMarkSweepGC )

启用该参数，默认启用了-XX:+UseParNewGC；简单的说，并发是指用户线程与垃圾收集线程并发，程序在继续运行，而垃圾收集程序运行于其他CPU上。

**十九、Final关键字**

在Java中，final关键字可以用来修饰类、方法和变量（包括成员变量和局部变量。特殊的：

当final修饰一个基本数据类型时，表示该基本数据类型的值一旦在初始化后便不能发生变化；如果final修饰一个引用类型时，则在对其初始化之后便不能再让其指向其他对象了，但该引用所指向的对象的内容是可以发生变化的。本质上是一回事，因为引用的值是一个地址，final要求值，即地址的值不发生变化。

## 十九\_1 Final初始化

final修饰一个成员变量（属性），必须要显示初始化（没有默认初始化）。这里有两种初始化方式，一种是在变量声明的时候初始化；第二种方法是在声明变量的时候不赋初值，但是要在这个变量所在的类的所有的构造函数中对这个变量赋初值。

## 十九\_2 类的final变量和普通变量有什么区别

当final变量是基本数据类型以及String类型时，如果在编译期间能知道它的确切值，则编译器会把它当做编译期常量使用。也就是说在用到该final变量的地方，相当于直接访问的这个常量，不需要在运行时确定。如下：

1)、

public class Test {

public static void main(String[] args) {

String a = "hello2";

final String b = "hello";

String d = "hello";

String c = b + 2;

String e = d + 2;

System.out.println((a == c));

System.out.println((a == e));

}

}

输出结果：true、false

2)、

public class Test {

public static void main(String[] args) {

String a = "hello2";

final String b = getHello();

String c = b + 2;

System.out.println((a == c));

}

public static String getHello() {

return "hello";

}

}

这段代码的输出结果为：false

## 十九\_3 final关键字的好处

1.final关键字提高了性能。JVM和Java应用都会缓存final变量。

2.final变量可以安全的在多线程环境下进行共享，而不需要额外的同步开销。

3.使用final关键字，JVM会对方法、变量及类进行优化。

## 十九\_4 final的重要知识点

1. final关键字可以用于成员变量、本地变量、方法以及类。
2. 你不能够对final变量再次赋值。
3. 本地变量必须在声明时赋值。
4. 在匿名类中所有变量都必须是final变量。
5. final方法不能被重写。
6. final类不能被继承。
7. final关键字不同于finally关键字，后者用于异常处理
8. final关键字容易与finalize()方法搞混，后者是在Object类中定义的方法，是在垃圾回收之前被JVM调用的方法。
9. 接口中声明的所有变量本身是final的。
10. final和abstract这两个关键字是反相关的，final类就不可能是abstract的。
11. final方法在编译阶段绑定，称为静态绑定(static binding)。
12. 没有在声明时初始化final变量的称为空白final变量(blank final variable)，它们必须在构造器中初始化，或者调用this()初始化。不这么做的话，编译器会报错“final变量(变量名)需要进行初始化”。
13. 将类、方法、变量声明为final能够提高性能，这样JVM就有机会进行估计，然后优化。
14. 按照Java代码惯例，final变量就是常量，而且通常常量名要大写：
15. 对于集合对象声明为final指的是引用不能被更改，但是你可以向其中增加，删除或者改变内容

二十、Java中的地址值

Java中的地址值被保护了可以用**.hashCode（）**获取地址的哈希值（一种算法，相当于把地址加密了），可以使用底层的库获取（sun.misc.Unsafe类），也可以通过jol工具包里的工具类实现获取对象的内存信息，使用jol工具包需要在maven里引入

二十一、常量池

分为静态常量池和运行时常量池

二十一\_1 静态常量池

即\*.class文件中的常量池，class文件中的常量池不仅仅包含字符串(数字)字面量，还包含类、方法的信息，占用class文件绝大部分空间。这种常量池主要用于存放两大类常量：字面量(Literal)和符号引用量(Symbolic References)，字面量相当于Java语言层面常量的概念，如文本字符串，声明为final的常量值等，符号引用则属于编译原理方面的概念，包括了如下三种类型的常量：

类和接口的全限定名

字段名称和描述符

方法名称和描述符

二十一\_2 运行时常量池

运行时常量池，则是jvm虚拟机在完成类装载操作后，将class文件中的常量池载入到内存中，并保存在方法区中，我们常说的常量池，就是指方法区中的运行时常量池。

运行时常量池相对于CLass文件常量池的另外一个重要特征是具备动态性，Java语言并不要求常量一定只有编译期才能产生，也就是并非预置入CLass文件中常量池的内容才能进入方法区运行时常量池，运行期间也可能将新的常量放入池中，这种特性被开发人员利用比较多的就是String类的intern()方法。

String的intern()方法会查找在常量池中是否存在一份equal相等的字符串,如果有则返回该字符串的引用,如果没有则添加自己的字符串进入常量池。

二十一\_3 常量池的好处  
常量池是为了避免频繁的创建和销毁对象而影响系统性能，其实现了对象的共享。

二十二、描述符和特征签名：

描述符描述符概念是针对class字节码的；方法描述符中包括方法的返回值类型，Java语言层面，Java方法的特征签名只包括了方法的名称、参数顺序以及参数类型，Java虚拟层面，Java虚拟机中定义的是针对字节码的，方法的特征签名包括方法的返回值以及可能抛出的受检查异常（Checked Exceptions），也就是方法描述时在throws关键字后面列举的异常，在Java语言层面，描述符与特征签名不同；在Java虚拟机规范中，描述符与特征签名是相同的。

public void test1(){} test1()V

public void test2(String str) test2(Ljava/lang/String;)V

public int test3(){} test3()I

从以上三个例子，我们就可以很简单的看出一些小小的规律：

JVM为我们提供的方法签名实际上是由方法名(上文的例子为了简单没有写出全类名)、形参列表、返回值三部分构成的，基本形式就是：

全类名.方法名(形参数据类型列表)返回值数据类型

我们知道了JVM提供的方法签名的结构之后还是不够，因为JVM是并没有特别明确的将数据类型写出来，而是提供了特殊的表示法，下表就是对特殊表示字符、字母与对应数据类型的关系：

Java方法签名中特殊字符/字母含义

特殊字符 数据类型 特殊说明

V void 一般用于表示方法的返回值

Z boolean B byte

C char S short I int

J long F float D double

[ 数组

以[开头，配合其他的特殊字符，表示对应数据类型的数组，几个[表示几维数组

L全类名; 引用类型

以L开头、;结尾，中间是引用类型的全类名

以上就是对于方法签名以及方法重载的小小解释、说明。一定要注意的是方法重载时，方法返回值没有什么意义，是由方法名和参数列表决定的。



二十三、方法区：

方法区和“PermGen space”（永久代）又有着本质的区别。前者是JVM的规范，而后者则是JVM规范的一种实现，并且只有HotSpot才有“PermGen space”，而对于其他类型的虚拟机，如JRockit（Oracle）、J9（IBM）并没有“PermGen space”。由于方法区主要存储类的相关信息，所以对于动态生成类的情况比较容易出现永久代的内存溢出。最典型的场景就是，在jsp页面比较多的情况，容易出现永久代内存溢出；元空间的本质和永久代类似，都是对JVM规范中方法区的实现。不过元空间与永久代之间最大的区别在于：元空间并不在虚拟机中，而是使用本地内存。因此，默认情况下，元空间的大小仅受本地内存限制，理论上取决于32位/64位系统可虚拟的内存大小。可见也不是无限制的，需要配置参数。

1.MetaspaceSize

初始化的Metaspace大小，控制元空间发生GC的阈值。GC后，动态增加或降低MetaspaceSize。在默认情况下，这个值大小根据不同的平台在12M到20M浮动。使用[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "https://www.cnblogs.com/yulei126/p/_blank) -XX:+PrintFlagsInitial命令查看本机的初始化参数

2.MaxMetaspaceSize

限制Metaspace增长的上限，防止因为某些情况导致Metaspace无限的使用本地内存，影响到其他程序。在本机上该参数的默认值为4294967295B（大约4096MB）。

3.MinMetaspaceFreeRatio

当进行过Metaspace GC之后，会计算当前Metaspace的空闲空间比，如果空闲比小于这个参数（即实际非空闲占比过大，内存不够用），那么虚拟机将增长Metaspace的大小。默认值为40，也就是40%。设置该参数可以控制Metaspace的增长的速度，太小的值会导致Metaspace增长的缓慢，Metaspace的使用逐渐趋于饱和，可能会影响之后类的加载。而太大的值会导致Metaspace增长的过快，浪费内存。

4.MaxMetasaceFreeRatio

当进行过Metaspace GC之后， 会计算当前Metaspace的空闲空间比，如果空闲比大于这个参数，那么虚拟机会释放Metaspace的部分空间。默认值为70，也就是70%。

5.MaxMetaspaceExpansion

Metaspace增长时的最大幅度。在参考资料的人的机器上该参数的默认值为5452592B（大约为5MB）。

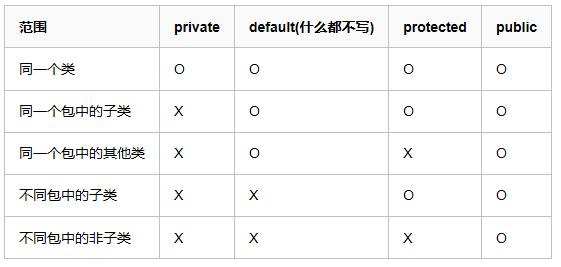
6.MinMetaspaceExpansion

Metaspace增长时的最小幅度。在参考资料的人的机器上该参数的默认值为340784B（大约330KB为）。

转换的好处：

1. 字符串存在永久代中，容易出现性能问题和内存溢出。（字符串1.7后放在堆中，但PermGen永久代1.8才被Metaspace元空间替代）
2. 类及方法的信息等比较难确定其大小，因此对于永久代的大小指定比较困难，太小容易出现永久代溢出，太大则容易导致老年代溢出。
3. 永久代会为 GC 带来不必要的复杂度，并且回收效率偏低。
4. Oracle可能会将HotSpot与JRockit合二为一。

二十四、修饰符的权限：



子类（父类的外部包）中访问父类的protetcted属性或者方法，是不可以通过创建父类对象调用的。

Class A{ Class B extends A{

Protetcted fun(){} A a = new A();

} a.fun();}

这样是不行的

二十五、调用子类方法和成员变量：

调用子类方法和成员变量时如果没有，会去找的父类，还没有会继续找父类的父类，直到找到或者object也没有。子类的构造方法会默认调父类的无参构造方法。

二十六、建包：



二十七、static

局部变量不能用static修饰；static属于类先于对象的构造，即先于构造函数执行；static修饰的方法不能被继承，它独属于类；static变量在编译时已经[分配内存](https://www.baidu.com/s?wd=%E5%88%86%E9%85%8D%E5%86%85%E5%AD%98&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)；同是static就按照顺序结构执行，显示赋值在后面赋值；同public一样不能修饰局部变量，因为局部变量是属于方法的不能属于对象，所以被禁掉了。

Mian函数要用static修饰：

假设没有static关键字,那意味着需要用生成一个实例后才可以调用这个Ｍain方法,而Ｍain方法是程序入口点，你没有进入Ｍain方法,自然无法生成一个实例,既然没有实例，那就无法调用Ｍain函数，岂不矛盾?所以Ｍain函数被设置为static.

二十八、instanceof

Java 中的instanceof 运算符是用来在运行时指出对象是否是特定类（接口）的一个实例

Scanner sc =new Scanner();

if(sc instanceof Scanner){

}

<https://blog.csdn.net/qq_38663729/article/details/78022990>

二十九、多态

多态的前提条件:

1.继承或实现

2.子类重写了方法

3.父类引用指向子类对象

多态中成员方法的调用:编译看左边,运行看右边

三十、编译器的常量优化

如果赋值运算符右边的全是常量且没有超过左边的取值范围，那么在编译时就会把运算结果直接赋给左边。如：

byte b = 5+8；实际上编译完是byte b = 13；

但超过取值范围就不行：

Byte b = 120+8；就会报错

三十一、静态代码块

会在首次使用类时调用唯一的一次；

三十二、Java中输出二进制和十六进制以及八进制

0b（零） 二进制输出成十进制

十进制输出成二进制：Integer.toBinaryString()

0（零） 八进制输出和成十进制

十进制输出成二进制：Integer.toOctalString()

0x 十六进制输出成十进制

十进制输出成二进制：Integer.toHexString()

三十三、arraylist扩容机制

每次不够就\*1.5，1.6前（含）初始是10，1.7后初始是0，输入到第十一个时扩大成15，下次在输入到16时扩大到22；

三十三、内部类（非静态）如何可以访问外部类成员

内部类逻辑上和外部类的关系：

1. 内部类的创建依赖于外部类；
2. 内部类持有外部类的引用。

如何做到的：

1. 编译器自动为内部类添加了一个成员变量，这个成员变量的类型和外部类的类型相同，这个成员变量就是指向外部类的引用；
2. 编译器自动为内部类的构造方法添加一个参数，参数的类型就是外部类的类型，在构造方法内部使用这个参数为1中添加的成员变量赋值；
3. 在调用内部类的构造函数初始化内部类对象时，会默认传入外部类的引用。

三十四、快捷键

Shift+f6 快速改变标识符

Alt+7 打开本类所有方法、类等的目录

CapsLock+j left CapsLock+l right

CapsLock+k down CapsLock+i up

CapsLock+h backspace CapsLock+u home

CapsLock+o end CapsLock+； enter

CapsLock+p delete

Ctrl+alt+t try{}catah(){}快捷键

Ctrl+alt+m 抽取方法

三十五、Date类

Cate(long l)中是long类型，直接输入数字是int型会有最大值限制，要在最后加L；

三十六、集合

集合中的类型不能写基本类型，只能写它们的包装类；

三十七、关键字用来修饰什么

类 属性 方法 构造方法

private OK OK OK

默认 OK Ok Ok OK

protected Ok Ok OK

public Ok Ok Ok OK

static Ok Ok

final Ok Ok Ok

abstract Ok Ok

特殊：private和static可以修饰内部类。不用掌握。

static可以代码块。静态代码块。掌握。

A：权限修饰符任意时刻只能使用一种。

B：static,private,final不能和abstract并用

三十八、Ipv6

Ipv6地址：fe80::a4ea:2919:54c1:ef1b%3中的

：：（表示全为0）

%3（表示windows系统的链接号，即从哪张网卡出去，linux需要加网卡号(“-I eth0”)）

三十九、饿汉式与懒汉式有什么区别么（单例模式）

饿汉式：如果长时间不用这个对象的话，占用内存，线程安全（看static）

class Fun {  
 private Fun() {  
 }  
 private static Fun *fun* = new Fun();  
 public static Fun getFun(){  
 return *fun*;  
 }  
}

懒汉式：初始版线程不安全，要加锁，加锁后又效率低，还要再判断一次是否为空

class Fun1 {  
 private Fun1 fun;  
 private Fun1() {  
 }  
 public Fun1 fun() {  
 if (fun == null) {  
 synchronized (Fun1.class) {  
 if (fun == null) {  
 fun = new Fun1();  
 }  
 }  
 }  
 return fun;  
 }  
}

四十、静态内部类

根据Oracle官方的说法：

Nested classes are divided into two categories: static and non-static. Nested classes that are declared static are called static nested classes. Non-static nested classes are called inner classes.

从字面上看，一个被称为静态嵌套类，一个被称为内部类。从字面的角度解释是这样的：什么是嵌套？嵌套就是我跟你没关系，自己可以完全独立存在，但是我就想借你的壳用一下，来隐藏一下我自己（真TM猥琐）。什么是内部？内部就是我是你的一部分，我了解你，我知道你的全部，没有你就没有我。（所以内部类对象是以外部类对象存在为前提的）至于具体的使用场景，我就不当翻译工了，有兴趣的直接去[官网](https://link.jianshu.com?t=http://docs.oracle.com/javase/tutorial/java/javaOO/nested.html" \t "https://www.jianshu.com/p/_blank)看吧。

静态内部类就是个独立的类。之所以要弄这么个东西，只是起到一个注释的效果，而且这个注释可静态检查。比如有A，B两个类，B有点特殊，虽然可以独立存在，但只被A使用。这时候怎么办？如果把B并入A里，复杂度提高，搞得A违反单一职责。如果B独立，又可能被其他类（比如同一个包下的C）依赖，不符合设计的本意。所以不如将其变成A.B，等于添加个注释，告诉其他类别使用B了，它只跟A玩。非静态的才是真正的内部类，对其外部类有个引用。

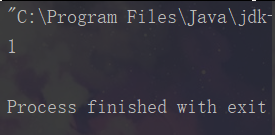
静态内部类的加载过程：

静态内部类的加载不需要依附外部类，在使用时才加载。不过在加载静态内部类的过程中也会加载外部类

静态内部类不持有外部类的引用

1. 静态内部类可以有静态成员(方法，属性)，而非静态内部类则不能有静态成员(方法，属性)（静态常量可以）。

public class Demo {  
 public static void main(String[] args) {  
 System.*out*.println(Demo.a.*i*);  
 }  
 class a{  
 static final int *i* = 1;  
 }  
}

  
 2.静态内部类只能够访问外部类的静态成员,而非静态内部类则可以访问外部类的所有成员(方法，属性)。  
 3.实例化一个非静态的内部类的方法：  
 a.先生成一个外部类对象实例  
OutClassTest oc1 = new OutClassTest();  
 b.通过外部类的对象实例生成内部类对象  
OutClassTest.InnerClass no\_static\_inner = oc1.new InnerClass();  
4.实例化一个静态内部类的方法：  
a.不依赖于外部类的实例,直接实例化内部类对象  
OutClassTest.InnerStaticClass inner = new OutClassTest.InnerStaticClass();  
b.调用内部静态类的方法或静态变量,通过类名直接调用  
 OutClassTest.InnerStaticClass.static\_value  
 OutClassTest.InnerStaticClass.getMessage()

四十一、lambda表达式

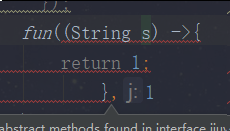
用接口的地方（参数列表）->{重写抽象方法；}

只有一句话时（包括return语句）可省略{}和；和return，不是一个语句（ifelse语句switch语句不行）

（）只有一个参数可省略

（）{}；可省略 可推导，可省略

Jdk8才有，可推导，可省略，特殊接口不能多抽象方法，不管是不是相同；interface T {  
 public int fun(String s);  
 public char fun1(int s);  
}





四十二、接口中的内容

Jdk7之前

常量 public static final

抽象方法 public abstract

Jdk8加入：

静态方法（静态方法只能通过接口名调用，不能通过实现类名或者对象名调用），默认方法（重写的时候去掉default关键字）

（都有方法体）

Jdk9加入：

私有方法

四十三、反射中的泛型

？Extend \*\*\* 指最多是\*\*\*类

？Supper \*\*\*\* 不能这么写，没有意义

ArrayList<Integer>

的集合用反射传入String类型的数据，会转成 Integer，可以用.Var观察，

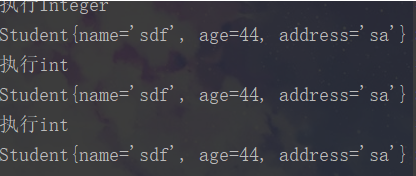
1）、如何在子类的构造方法中获得父类的泛型

class A<T>{}  
class B extends A<String >{  
 public B(){  
 //如何在子类的构造方法中获得父类的泛型  
 //虽然返回的是Type类型但实际上返回的实际上是Type的子接口ParameterizedType所以可以强转  
// Type fclass = B.class.getGenericSuperclass();  
 ParameterizedType fclass = (ParameterizedType)B.class.getGenericSuperclass();  
 //有很多泛型所以用数组表示  
 Type[] fan = fclass.getActualTypeArguments();  
 System.*out*.println(fan[0]);  
 //或者获得带泛型的名字(getTypeName是Type中的方法)  
 System.*out*.println(fclass.getTypeName());  
 }  
}

四十四、重载时用integer来代替int算重载

public Student(String name, int age, String address) {  
 System.*out*.println("执行int");  
}  
public Student(String name, Integer age, String address){  
 System.*out*.println("执行Integer");  
}

Integer I = 44;  
int i = 44;  
System.*out*.println(new Student("sdf", I, "sa"));  
System.*out*.println(new Student("sdf", i, "sa"));  
System.*out*.println(new Student("sdf", 44, "sa"));



四十五、怎么用反射获取继承的保护方法

获取父类的字节码对象，才能获取方法。（getsupperClass()）

# Web

1. having语句不能直接用么？

可以，但是跟where一样不能跟聚合语句；