1、比较String用“==”则比较的是内存地址。使用new都会新开辟空间。使用直接赋值的方式可以有效的节省内存。字符串的内容一旦声明则不可改变，要改变实际上变得是内存地址的引用关系。

2、静态方法不能调用非静态方法/非静态变量，因为非静态方法属于具体类，有可能还没有创建，反之可以。

3、一个类中定义了一个常量，如果在编译时就可知道是什么值，则在调用主动访问该常量时不会对该常量所在的类进行初始化；如果该常量在运行时才知道是什么值，则在调用主动访问该常量时会对该常量所在的类进行初始化。

4、java访问权限都是针对同一个命名空间（根据加载器的不同区分）而言的，不同命名空间的类只能通过反射进行访问。

5、若子区域内重新定义了和父区域的同名变量，则子区域会隐藏父区域，若没定义，则可直接操作父区域变量。

**一、内存管理原理：**

在java中，有java程序、虚拟机、操作系统三个层次，其中java程序与虚拟机交互，而虚拟机与操作系统间交互！这就保证了java程序的平台无关性！下面我们从程序运行前，程序运行中、程序运行内存溢出三个阶段来说一下内存管理原理！

1、程序运行前：JVM向操作系统请求一定的内存空间，称为初始内存空间！程序执行过程中所需的内存都是由java虚拟机从这片内存空间中划分的。

2、程序运行中：java程序一直向java虚拟机申请内存，当程序所需要的内存空间超出初始内存空间时，java虚拟机会再次向操作系统申请更多的内存供程序使用！

3、内存溢出：程序接着运行，当java虚拟机已申请的内存达到了规定的最大内存空间，但程序还需要更多的内存，这时会出现内存溢出的错误！

至此可以看出，Java 程序所使用的内存是由 Java 虚拟机进行管理、分配的。Java 虚拟机规定了 Java 程序的初始内存空间和最大内存空间，开发者只需要关心 Java 虚拟机是如何管理内存空间的，而不用关心某一种操作系统是如何管理内存的。

**二、 RUNTIME 类的使用：**

Java 给我们提供了Runtime 类得到JVM 内存的信息

|  |  |  |  |
| --- | --- | --- | --- |
| 方法名称 | 参数 | 作用 | 返回值 |
| getRuntime | 无 | 获取Runtime 对象 | Runtime 对象 |
| totalMemory | 无 | 获取JVM 分配给程序的内存数量 | long：内存数量 |
| freeMemory | 无 | 获取当前可用的内存数量 | long：内存数量 |
| maxMemory | 无 | 获取JVM 可以申请到的最大内存数量 | long：内存数量 |

**三、内存空间逻辑划分：**

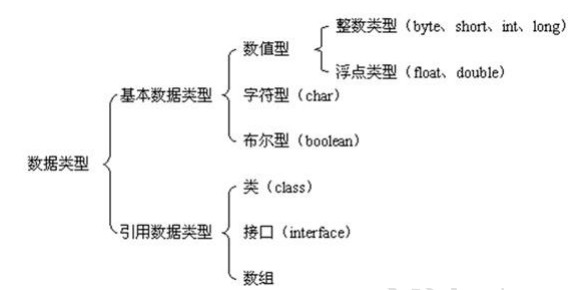
JVM 会把申请的内存从逻辑上划分为三个区域，即：方法区、堆与栈。

方法区：方法区默认最大容量为64M，Java虚拟机会将加载的java类存入方法区，保存类的结构（属性与方法），类静态成员等内容。

堆：默认最大容量为64M，堆存放对象持有的数据，同时保持对原类的引用。可以简单的理解为对象属性的值保存在堆中，对象调用的方法保存在方法区。

栈：栈默认最大容量为1M，在程序运行时，每当遇到方法调用时，Java虚拟机就会在栈中划分一块内存称为栈帧（Stack frame），栈帧中的内存供局部变量（包括基本类型与引用类型）使用，当方法调用结束后，Java虚拟机会收回此栈帧占用的内存。

**四、java数据类型**



0、Java中只有值传递。对于String类型，直接使用“”赋值，不管字符串池中有没有这个常量，返回的都是字符串池中的地址（有的话直接返回，没有的 话，创建好，放在池中，在返回）；而是用new关键字，不管字符串池中有没有这个值，都会在堆中创建一个String对象，并且返回堆中的地址，（有的话将值复制到堆中，再返回堆中的地址；没有的话会在字符串池和堆中分别创建，再返回堆中的地址）。使用“+”的（个人理解）若加号两边都是“”型，则只会产生一个或零个对象（原常量池中有则直接返回这个地址，没有就创建再返回，【可以确定】）；若加两边有一个变量则肯定会在堆中创建一个新对象，（常量池中若没有被“+”连接的常量，则在其中创建，若没有加后结果也会创建），按此种理解堆中有的常量池中肯定有，常量池中有的堆中不一定有。

在编译器创建的对象存放在字符串常量池，字符串常量区是在JVM栈中，当类被加载后常量区被搬到方法区里的运行时常量池。

String重写了equals()和hashCode()方法，只要内容相同即相同，原生的这两种方法是要求地址和内容都相同的。

1、基本数据类型：没封装指针的变量。

声明此类型变量，只会在栈中分配一块内存空间。

2、引用类型：就是底层封装指针的数据类型。

他们在内存中分配两块空间，第一块内存分配在栈中，只存放别的内存地址，不存放具体数值，我们也把它叫指针类型的变量，第二块内存分配在堆中，存放的是具体数值，如对象属性值等。

3、下面我们从一个例子来看一看：

public class Student {

  String stuId;

  String stuName;

  int stuAge;

}

public class TestStudent {

  public static void main(String[] args) {

    Student zhouxingxing = new Student();

    String name = new String("旺旺");

    int a = 10;

    char b = 'm';

    zhouxingxing.stuId = "9527";

    zhouxingxing.stuName = "周星星";

    zhouxingxing.stuAge = 25;

  }

}

（1）类当然是存放在方法区里面的。

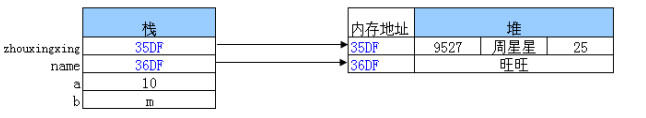
（2）Student zhouxingxing = new Student();

这行代码就创建了两块内存空间，第一个在栈中，名字叫zhouxingxing，它就相当于指针类型的变量，我们看到它并不存放学生的姓名、年龄等具体的数值，而是存放堆中第二块内存的地址，第二块才存放具体的数值，如学生的编号、姓名、年龄等信息。

（3）int a = 10;

这是 基本数据类型 变量，具体的值就存放在栈中，并没有只指针的概念！

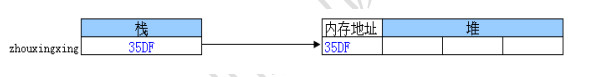
下图就是本例的内存布置图：



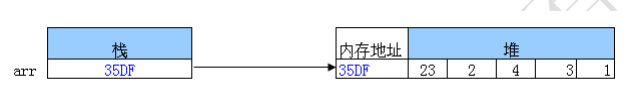
此外我们还要知道Student zhouxingxing = new Student(); 包括了声明和创建，即：Student zhouxingxing;和zhouxingxing = new Student();其中声明只是在栈中声明一个空间，但还没有具体的值，声明后的情况如下图所示：



创建后的情况如下图所示：



（4）引用类型中的数组也封装了指针，即便是基本数据类型的数组也封装了指针，数组也是引用类型。比如代码int[] arr = new int[]{23，2，4，3，1};如下图所示：



**五、java值传参与引用参数**

（1）参数根据调用后的效果不同，即是否改变参数的原始数值，又可以分为两种:按值传递的参数与按引用传递的参数。

按值传递的参数原始数值不改变，按引用传递的参数原始数值改变！这是为什么呢？其实相当简单：

我们知道基本数据类型的变量存放在栈里面，变量名处存放的就是变量的值，那么当基本数据类型的变量作为参数时，传递的就是这个值，只是把变量的值传递了过去，不管对这个值如何操作，都不会改变变量的原始值。而对引用数据类型的变量来说，变量名处存放的地址，所以引用数据类型的变量作为传参时，传递的实际上是地址，对地址处的内容进行操作，当然会改变变量的值了！

（2）特例：string

public class TestString {

  public static void main(String[] args) {

    String name = "wangwang";

    TestString testString = new TestString();

    System.out.println("方法调用前：" + name);

    testString.change(name);

    System.out.println("方法调用后：" + name);

  }

  void change(String str) {

    str = "旺旺老师";

    System.out.println("方法体内修改值后：" + str);

  }

}

结果：

方法调用前：wangwang

方法体内修改值后：旺旺老师

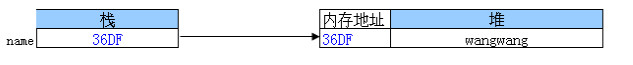
方法调用后：wangwang

分析：

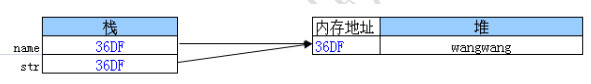
上例中，虽然参数String 是引用数据类型，但其值没有发生改变，这是因为String 类

是final 的，它是定长，我们看初始情况，即String name = "wangwang";这行代码运行

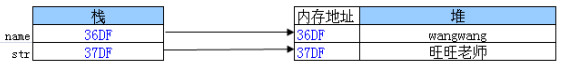
完，如下图：



 当调用方法时testString.change(name)，内存变化为：

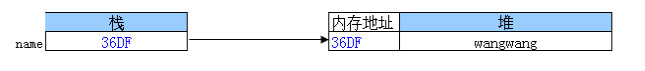


在方法体内，参数str赋予一个新值，str = "旺旺老师"。因为String是定长，系统就会在堆中分配一块新的内存空间37DF，这样str指向了新的内存空间37DF，而name还是指向36DF， 37DF的改变对它已没影响：



最后，方法调用结束，str与37DF的内存空间消亡。Name的值依然为wangwang,并没有改变。

所以String虽然是引用类型参数，但值依然不变：



 （3）无法交换的例子：

public class TestChange {

  void change(Student stu1, Student stu2) {

    stu1.stuAge ++;

    stu2.stuAge ++;

    Student stu = stu1;

    stu1 = stu2;

    stu2 = stu;

  }

  public static void main(String[] args) {

    Student furong = new Student();

    furong.stuName = "芙蓉姐姐";

    furong.stuAge = 30;

    Student fengjie = new Student();

    fengjie.stuName = "凤姐";

    fengjie.stuAge = 26;

    TestChange testChange = new TestChange();

    testChange.change(furong, fengjie);

    System.out.println(furong.stuName);

    System.out.println(furong.stuAge);

    System.out.println(fengjie.stuName);

    System.out.println(fengjie.stuAge);

  }

}

运行结果：

芙蓉姐姐

31

凤姐

27

分析：

