**3.1、面向对象（了解）**

面向对象是一种现在最为流行的程序设计方法，几乎现在的所有应用都以面向对象为主了，最早的面向对象的概念实际上是由**IBM**提出的，在70年代的Smaltalk语言之中进行了应用，后来根据面向对象的设计思路，才形成C++，而由C++产生了Java这门面向对象的编程语言。

但是在面向对象设计之前，广泛采用的是面向过程，面向过程只是针对于自己来解决问题。面向过程的操作是以程序的基本功能实现为主，实现之后就完成了，也不考虑修改的可能性，面向对象，更多的是要进行子模块化的设计，每一个模块都需要单独存在，并且可以被重复利用，所以，面向对象的开发更像是一个具备标准的开发模式。

在面向对象定义之中，也规定了其一些基本的特征：

· 封装：保护内部的操作不被破坏；

· 继承：在原本的基础之上继续进行扩充；

· 多态：在一个指定的范围之内进行概念的转换。

对于面向对象的开发来讲也分为三个过程：OOA（面向对象分析）、OOD（面向对象设计）、OOP（面向对象编程）。

## 3.2、类与对象（核心）

### 3.2.1、类与对象的基本概念

类：是抽象的概念集合，表示的是一个共性的产物，类之中定义的是属性和行为（方法）；

对象：对象是一种个性的表示，表示一个独立的个体，每个对象拥有自己独立的属性，依靠属性来区分不同对象。

那么可以依靠一句话来总结出类和对象的区别：类是对象的模板，对象是类的实例，类只有通过对象才可以使用，而在开发之中应该先产生类，之后再产生对象。

类不能直接使用，对象是可以直接使用的。

### 3.2.2、类与对象的定义

如果要在Java之中定义类的话，可以使用class关键字完成，其语法如下：

|  |
| --- |
| class 类名称 {  属性 (变量) ;  行为 (方法) ;  } |

**范例：**定义一个Person类

|  |
| --- |
| class Person { // 类名称首字母大写  String name ;  int age ;  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  } |

那么类定义完成之后，肯定无法直接使用，如果要使用，必须依靠对象，那么由于类属于引用数据类型，所以对象的产生格式如下。

**格式一：**声明并实例化对象

|  |
| --- |
| 类名称 对象名称 = new 类名称 () ; |

**格式二：**分步完成

|  |  |
| --- | --- |
| **声明对象：** | 类名称 对象名称 = null ; |
| **实例化对象：** | 对象名称 = new 类名称 () ; |

以后只要是引用数据类型的实例化操作，永远都会存在关键字new（分配空间）。当一个实例化对象产生之后，可以按照如下的方式进行类的操作：

· 对象.属性：表示调用类之中的属性；

· 对象.方法()：表示调用类之中的方法。

**范例：**使用对象操作类

|  |
| --- |
| class Person { // 类名称首字母大写  String name ;  int age ;  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person() ; // 声明并实例化对象  per.name = "张三" ;  per.age = 30 ;  per.tell() ;  }  } |

以上完成了一个基本的类和对象的操作关系，下面换另外一个操作来观察一下。

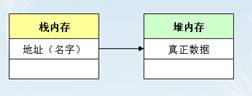
|  |
| --- |
| class Person { // 类名称首字母大写  String name ;  int age ;  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = null ; // 声明对象  per = **new** Person() ; // 实例化对象  per.name = "张三" ;  per.age = 30 ;  per.tell() ;  }  } |

**疑问？**给出的两种不同的实例化方式有什么区别呢？

如果要想解释这个问题，那么首先需要解决的就是内存的关系理解（Java是在C++之上开发的，所以本次讲解的时候还是按照C++的理论进行内存关系的讲解），首先给出两块内存空间：

· 堆内存：保存对象的真正数据，都是每一个对象的属性内容；

· 栈内存：保存的是一块堆内存的空间地址，可以把它想象成一个int型变量（每一个int型变量只能存放一个数值），所以每一块保留一块堆内存地址，但是为了方便理解，可以简单的将栈内存之中保存的数据理解为对象的名称（Person per），就假设保存的是per。



按照这样的概念理解，以上的程序就可以变成如下的内存关系图表示出来。如果要想开辟堆内存空间，只能依靠关键字new来进行开辟。**即：只要看见了关键字new不管何种情况下，都表示要开辟新的堆内存空间。**

**范例：**观察产生两个对象的操作

|  |
| --- |
| class Person { // 类名称首字母大写  String name ;  int age ;  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per1 = null ; // 声明对象  Person per2 = new Person() ; // 声明并实例化对象  per1 = new Person() ; // 实例化对象  per1.name = "张三" ;  per1.age = 30 ;  per2.name = "李四" ;  per2.age = 20 ;  per1.tell() ;  per2.tell() ;  }  } |

但是在这里需要提醒的是，如果在开发之中出现了以下代码：

|  |
| --- |
| class Person { // 类名称首字母大写  String name ;  int age ;  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  **Person per = null ; // 声明对象**  per.name = "张三" ;  per.age = 30 ;  per.tell() ;  }  } |

这个时候的程序发现只声明了Person对象，但是并没有实例化Person对象（只有了栈内存，并没有对应的堆内存空间），则程序在编译的时候不会出现任何的错误，但是在执行的时候出现了以下的错误信息：

|  |
| --- |
| Exception in thread "main" java.lang.NullPointerException  at TestDemo.main(TestDemo.java:11) |

这个错误信息表示的是“NullPointerException（空指向异常）”，这种异常**只会在引用数据类型上产生**，此异常会一直伴随着大家，到你不写程序的那一天。

### 3.2.3、引用传递的初步深入

下面通过若干个程序，以及程序的内存分配图，来进行代码的讲解。

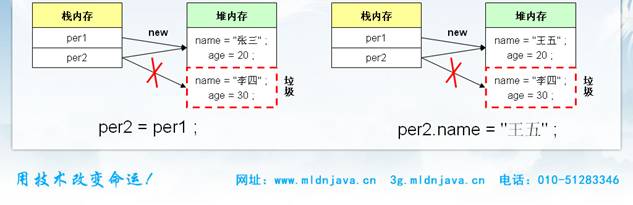
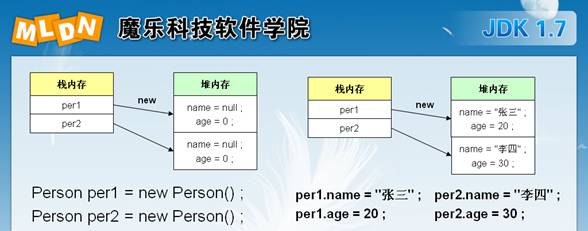
**范例：**观察以下程序的结果

|  |
| --- |
| class Person { // 类名称首字母大写  String name ;  int age ;  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per1 = new Person() ; // 声明并实例化对象  per1.name = "张三" ;  per1.age = 20 ;  **Person per2 = per1 ; // 引用传递**  per2.name = "李四" ;  per1.tell() ;  }  } |

**引用传递的精髓：同一块堆内存空间，同时被多个栈内存所指向，不同的栈可以修改同一块堆内存的内容。**

**范例：**观察以下程序的运行

|  |
| --- |
| class Person { // 类名称首字母大写  String name ;  int age ;  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per1 = new Person() ; // 声明并实例化对象  Person per2 = new Person() ;  per1.name = "张三" ;  per1.age = 20 ;  per2.name = "李四" ;  per2.age = 30 ;  per2 = per1 ;  per2.name = "王五" ;  per1.tell() ;  }  } |



垃圾：指的是在程序开发之中没有任何对象所指向的一块堆内存空间，这块空间就成为垃圾，所有的垃圾将等待GC（垃圾收集器）不定期的进行回收与空间的释放。

**范例：**观察如下的程序（恶心点的）

|  |
| --- |
| class Person { // 类名称首字母大写  String name ;  int age ;  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per1 = new Person() ; // 声明并实例化对象  Person per2 = new Person() ;  per1.name = "张三" ;  per1.age = 20 ;  per2.name = "李四" ;  per2.age = 30 ;  Person per3 = per1 ;  per3.name = "王五" ;  per3 = per2 ;  per1 = per2 ;  per2.name = per3.name ;  per3.age = per1.age ;  per2.tell() ;  }  } |

首先本程序是几乎是不可能在开发之中出现的，如果真的出现也是不可能的。

下面还是通过内存关系分析，但是强调几个代码：

|  |
| --- |
| per2.name = per3.name ;  per3.age = per1.age ; |

· “per2.name = per3.name ;”：将per3的name属性的给per2.name属性；

· “per3.age = per1.age ;”：将per1的age属性给per3.age属性。

**3.3、封装性（重点）**

封装属于面向对象的第一大特性，但是本次所讲解的封装只是针对于其中的一点进行讲解，而对于封装操作由于涉及的内容过多，以后会有完整的介绍。但是在讲解封装操作之前，首先先要来解决一个问题：为什么要有封装？

**范例：**观察没有封装操作的情况

|  |
| --- |
| class Person { // 类名称首字母大写  String name ;  int age ;  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person() ;  per.name = "张三" ;  **per.age = -30 ;**  per.tell() ;  }  } |

发现这个时候所设置的人的年龄是“-30”岁，结果从代码编译上不会有问题，但是从实际来讲，一个人的年龄不可能是-30岁，这个是属于业务逻辑出错。而造成这种错误的关键在于没有检查，用户直接操作。就好比银行，你自己能直接操作金库？而检查的第一步是需要让用户看不见操作的东西，那么在这种情况下，就可以使用private关键字，将类之中的属性进行私有化的操作。

|  |
| --- |
| class Person { // 类名称首字母大写  private String name ;  private int age ;  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person() ;  **per.name = "张三" ; // TestDemo.java:11: 错误: name可以在Person中访问private**  **per.age = -30 ; // TestDemo.java:12: 错误: age可以在Person中访问private**  per.tell() ;  }  } |

所以现在属性是安全了，而如果现在外部要想操作私有属性，那么按照Java的开发标准而言，现在需要按照如下形式定义操作方法：setter、getter：

· setter（private String **name**）：public void set**N**ame(String n)；

· getter（private String name）：public String getName()；

|  |
| --- |
| class Person { // 类名称首字母大写  private String name ;  private int age ;  public void setName(String n) {  name = n ;  }  public void setAge(int a) {  age = a ;  }  public String getName() {  return name ;  }  public int getAge() {  return age ;  }  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person() ;  per.setName("张三") ;  per.setAge(20) ;  per.tell() ;  }  } |

以上的代码只是可以访问了，不过没有验证，但是问题是验证在那块加？应该在setter之中增加检查操作。

|  |
| --- |
| class Person { // 类名称首字母大写  private String name ;  private int age ;  public void setName(String n) {  name = n ;  }  public void setAge(int a) {  if (a >= 0 && a <= 250) {  age = a ;  }  }  public String getName() {  return name ;  }  public int getAge() {  return age ;  }  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person() ;  per.setName("张三") ;  per.setAge(20) ;  per.tell() ;  }  } |

以后在进行开发的时候，如果有需要，则在setter上加入一些验证措施，而getter方法只是简单的将数据返回即可，不需要做任何的验证。

**疑问？**为什么现在没有在程序之中使用getter()方法？

现在类之中的getName()和getAge()两个方法虽然被定义了，但是发现并没有被使用，那么这样的定义还有意义吗？

在类之中的属性定义setter、getter操作方法目的就是为了设置和取得属性的内容，也许某一个操作暂时不使用到取得的操作，不过从开发来讲，必须全部提供。**以后在定义类的时候，所有的属性都要编写private封装，封装之后的属性如果需要被外部操作，则编写setter、getter。**

**3.4、构造方法（重点）**

在之前强调过方法和属性的区分，方法之后存在“()”，而属性之后什么都没有，如果要想清楚构造方法，则首先来观察以下的格式：

|  |
| --- |
| 类名称 对象名称 = new 类名称() ; |

这种操作格式在之前已经使用过了，那么下面可以针对于这个格式每一个出现的标记进行解释：

· 类名称（**类名称** 对象名称 = new 类名称() ;）：要定义变量的数据类型；

· 对象名称（类名称 **对象名称** = new 类名称() ;）：指的是日后进行类属性或方法操作的名称；

· new（类名称 对象名称 = **new** 类名称() ;）：开辟堆内存空间；

· 类名称()（类名称 对象名称 = new **类名称()** ;）：？？？

按照道理来讲，加上“()”都属于方法，但是这个方法稍微特殊一些，属于构造方法，所以这个时候就可以发现，构造方法和普通方法不太一样的地方；**构造方法是在实例化对象的时候使用，而普通方法是在实例化对象产生之后使用的。**

**构造方法本身的定义如下：**

**· 构造方法的名称和类名称保持一致；**

**· 构造方法不允许有返回值类型声明；**

**· 由于对象实例化操作一定需要构造方法的存在，所以如果在类之中没有明确定义构造方法的话，则会自动的生成一个无参的，无返回值的构造方法，供用户使用，如果一个类之中已经明确的定义了一个构造方法的话，则无参的什么都不做的构造方法将不会自动生成，也就是说，一个类之中至少存在一个构造方法。**

|  |
| --- |
| class Person { // 类名称首字母大写  public Person() {// 无参无返回值的方法  }  } |

**· 构造方法在对象实例化的时候完成操作，而且一个对象的构造方法只会显式调用一次。**

当然，在类之中也可以明确的定义一个构造方法，可以通过构造方法为类之中的属性初始化。

|  |
| --- |
| class Person { // 类名称首字母大写  **private String name ;**  **private int age ;**  **public Person(String n,int a) { // 通过构造方法赋值**  **name = n ;**  **setAge(a) ; // 调用本类中的setAge()方法，可以检查**  **}**  public void setName(String n) {  name = n ;  }  public void setAge(int a) {  if (a >= 0 && a <= 250) {  age = a ;  }  }  public String getName() {  return name ;  }  public int getAge() {  return age ;  }  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person("张三",-20) ;  per.tell() ;  }  } |

当然，如果现在非要强调所调用的方法是本类之中所定义方法的时候，也可以在方法前增加一个this，例如：

|  |
| --- |
| public Person(String n,int a) { // 通过构造方法赋值  this.setName(n) ;  this.setAge(a) ; // 调用本类中的setAge()方法，可以检查  } |

这里的this就表示本类之中的方法调用，加与不加结果完全相同。

**在实际之中构造方法的主要作用只有一点：在对象实例化的时候为类之中的属性初始化。**

当然，对于构造方法而言，也需要注意，那么可以进行重载，不过构造方法重载的时候只需要考虑参数的类型及个数即可，而方法名称肯定是相同的。

**范例：**观察构造方法重载

|  |
| --- |
| class Person { // 类名称首字母大写  private String name ;  private int age ;  **public Person(){}**  **public Person(String name) {**  **this.setName(name) ;**  **}**  **public Person(String n,int a) { // 通过构造方法赋值**  **this.setName(n) ;**  **this.setAge(a) ; // 调用本类中的setAge()方法，可以检查**  **}**  public void setName(String n) {  name = n ;  }  public void setAge(int a) {  if (a >= 0 && a <= 250) {  age = a ;  }  }  public String getName() {  return name ;  }  public int getAge() {  return age ;  }  public void tell() { // 没有static  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person("张三") ;  per.tell() ;  }  } |

可是对于构造方法重载强调一点，编写顺序：所有的重载的方法按照参数的个数由多到少，或者是由少到多排列。

**额外提醒（暂时无法验证）：**构造方法实际上严格来讲是属于整个对象构造过程的最后一步，对象的构造需要为其分配空间，之后为其设置默认值，最后留给构造方法进行其他操作，所以可以定义的构造方法是对象实例化的最后一步，而其他的几步用户根本就看不见，也无法操作，那么对于如下的代码：

|  |
| --- |
| class Person { // 类名称首字母大写  **private String name = "张三" ;**  public Person() { // 是构造的最后一步  System.out.println(name) ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per = new Person() ;  }  } |

这个类之中的name属性只有在构造完成之后，才可以为其赋予“张三”的内容，如果构造没有执行，则name就是其对应数据类型的默认值：null。

**3.5、匿名对象（重点）**

没名字的对象称为匿名对象，对象的名字按照之前的内存关系来讲，在栈内存之中，而对象的具体内容在堆内存之中保存，这样一来，没有栈内存指向堆内存空间，就是一个匿名对象。

|  |
| --- |
| class Person { // 类名称首字母大写  private String name ;  private int age ;  public Person(String n,int a) {  name = n ;  age = a ;  }  public void tell() {  System.out.println("姓名：" + name + "，年龄：" + age) ;  }  }  public class TestDemo {  public static void main(String args[]) {  **new Person("张三",20)**.tell() ; // 匿名对象  }  } |

如果不习惯使用匿名对象，以后的对象就都起名字。匿名对象由于没有对应的栈内存指向，所以只能使用一次，一次之后就将成为垃圾，并且等待被GC回收释放。

**3.6、思考题（核心）**

现在要求定义一个表示雇员的操作类，这个类之中包含雇员编号、姓名、职位、基本工资、 佣金，并且可以计算出一个雇员的月薪和年薪，可以返回一个雇员的完整信息（把所有的属性内容都返回）。

参考：之前讲解的Person程序。

**简单Java类：一个类之中只包含基本的属性、setter、getter方法，这种类称为简单java类，对于简单Java类的开发原则有如下几点，必须严格遵守：**

**· 类名称必须可以明确的表示出一类的定义，例如：Person、Emp、Dept；**

**· 类之中的所有属性必须使用private进行封装；**

**· 类之中的所有属性都必须定义相应的setter、getter；**

**· 类之中可以提供构造方法，为属性初始化，但是不管提供了多少个构造方法，一定要保留有一个无参构造；**

**· 类之中不允许直接使用System.out.println()输出，所有的内容要返回给被调用处输出。**

在一个类之中编写的时候，应该按照如下顺序：属性、构造、普通方法。

|  |
| --- |
| class Emp { // 雇员类  private int empno ;  private String ename ;  private String job ;  private double sal ;  private double comm ;  public Emp(){}  public Emp(int eno,String ena,String j,double s,double c){  empno = eno ;  ename = ena ;  job = j ;  sal = s ;  comm = c ;  }  public void setEmpno(int eno) {  empno = eno ;  }  public void setEname(String ena) {  ename = ena ;  }  public void setJob(String j) {  job = j ;  }  public void setSal(double s) {  sal = s ;  }  public void setComm(double c) {  comm = c ;  }  public int getEmpno() {  return empno ;  }  public String getEname() {  return ename ;  }  public String getJob() {  return job ;  }  public double getSal() {  return sal ;  }  public double getComm() {  return comm ;  }  public double salary() {  return sal + comm ;  }  public double income() {  return this.salary() \* 12 ;  }  public String getInfo() {  return "雇员信息：" + "\n" +  "\t|- 编号：" + this.getEmpno() + "\n" +  "\t|- 姓名：" + this.getEname() + "\n" +  "\t|- 职位：" + this.getJob() + "\n" +  "\t|- 工资：" + this.getSal() + "\n" +  "\t|- 佣金：" + this.getComm() + "\n" +  "\t|- 月薪：" + this.salary() + "\n" +  "\t|- 年薪：" + this.income() ;  }  }  public class TestDemo {  public static void main(String args[]) {  Emp emp = new Emp(7369,"SMITH","CLERK",800.0,0.0) ;  System.out.println(emp.getInfo()) ;  }  } |

**作业：**写出10个简单Java类，自己随便去定义，类之中的属性不能少于5个，多多的。

## 3.7、数组（重点）

数组属于引用型数据，所以数组的操作过程之中，也一定会牵扯到内存的分配问题。

### 3.7.1、数组的基本概念

例如，现在要求定义100个整型变量，那么按照之前所学的概念来讲，要定义成：int i1,i2 … i100，表示100个整型变量，这样的确是可以定义，但是这些变量不方便管理，可以通过数组管理，数组：**就是一组相关变量的集合。**

数组的定义格式如下（动态操作格式）：

**格式一：声明并开辟（实例化）数组**

|  |
| --- |
| 数据类型 数组名称 [] = new 数据类型 [长度] ; |
| 数据类型 [] 数组名称 = new 数据类型 [长度] ; |

**格式二：**分步完成

|  |  |
| --- | --- |
| **声明数组：** | 数据类型 数组名称 [] = null ; |
| **开辟数组：** | **数组名称** = new 数据类型 [长度] ; |

当数组开辟完成空间之后，可以利用“数组名称[索引]”的方式来操作数组，而索引的范围为：0 ~ 数组长度-1，例如：开辟了三个大小的数组，则索引的取值是：0、1、2，一共三个内容。

在默认情况下，以上的格式为数组的动态初始化，动态初始化的最大特点，在于开辟空间之后，里面的所有数据的内容都是其对应数据类型的默认值，如果是int，默认值是0。

由于数组是通过一个名称统一的进行管理，所以数组在输出的时候往往可以利用for循环完成，for循环需要知道明确的循环次数，那么数组数据的输出循环次数就是数组长度，而数组长度的取得“**数组名称.length**”。

**范例：**定义并使用一个数组

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = **new** int [3] ; // 开辟3个空间的数组  data [0] = 10 ;  data [1] = 20 ;  data [2] = 30 ;  for (int x = 0 ; x < data.length ; x ++) {  System.out.print(data[x] + "、") ;  }  }  } |

以上程序的内存分析图操作如下。

整个的操作流程和对象操作几乎是一样的，无外乎在对象之中保存的是属性，而在数组之中保存的是各个索引元素。

**范例：**数组的引用传递

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [3] ; // 开辟3个空间的数组  data [0] = 10 ;  data [1] = 20 ;  data [2] = 30 ;  int temp [] = data ; // 类型统一，起了别名  temp [0] = 100 ;  for (int x = 0 ; x < data.length ; x ++) {  System.out.print(data[x] + "、") ;  }  }  } |

以上的是采用了声明并开辟空间的数组初始化方式完成的，下面采用分步的方式完成。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = null ; // 声明数组  data = new int [3] ; // 开辟3个空间的数组  data [0] = 10 ;  data [1] = 20 ;  data [2] = 30 ;  for (int x = 0 ; x < data.length ; x ++) {  System.out.print(data[x] + "、") ;  }  }  } |

如果这个时候在使用数组的过程之中，没有进行实例化操作？

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  **int data [] = null ; // 声明数组**  data [0] = 10 ;  data [1] = 20 ;  data [2] = 30 ;  for (int x = 0 ; x < data.length ; x ++) {  System.out.print(data[x] + "、") ;  }  }  } |
| Exception in thread "main" java.lang.NullPointerException  at TestDemo.main(TestDemo.java:4) |

如果在操作的过程之中，超出了数组本身的访问界限。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [3] ; // 开辟3个空间的数组  data [5] = 10 ;  }  } |
| Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5  at TestDemo.main(TestDemo.java:5) |

从现在的开发来讲，可以很负责任的说，数组一定会出现，但是真心的讲不多。因为数组最麻烦的问题在于它的长度限制上。

### 3.7.2、数组的静态初始化

数组的动态初始化在开辟空间之后，里面的所有内容都是其对应数据类型的默认值，那么也可以使用静态初始化的方式，为数组在开辟空间时，设置好指定的内容，而数组的静态初始化的语法格式如下：

**格式一：**简写格式

|  |
| --- |
| 数据类型 数组名称 [] = {值,值,...} ; |
| 数据类型 [] 数组名称= {值,值,...} ; |

**格式二：**完整格式（**推荐使用**）

|  |
| --- |
| 数据类型 数组名称 [] = new 数据类型 [] {值,值,...} ; |
| 数据类型 [] 数组名称 = new 数据类型 [] {值,值,...} ; |

**范例：**数组的静态初始化

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {209,201,2,2,3,6,7} ;  for (int x = 0 ; x < data.length ; x ++) {  System.out.println(data[x]) ;  }  }  } |

到底在开发之中使用何种数组初始化的方式并没有一个明确的定义，还是看功能，如果你已经知道了所有的操作数据，那么使用静态合适，如果有一些数据需要单独配置，那么用动态合适。

### 3.7.3、数组与方法（难点）

既然数组内容可以进行引用传递，那么就可以把数组给方法之中的参数，如果一个方法要想接收参数，则对应的参数类型必须是数组，下面来观察如下的一道程序。

**范例：**使用方法接收数组

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {209,201,2,2,3,6,7} ;  print(data) ;// 引用传递，int temp [] = data ;  }  public static void print(int temp []) {  for (int x = 0 ; x < temp.length ; x ++) {  System.out.print(temp[x] + "、") ;  }  System.out.println() ;  }  } |

**范例：**判断某一个数据是否在指定的数组之中

基本操作原理：使用数组的静态初始化，声明一个数组内容，而后随便设置一个数据，判断这个数据是否在指定的数组之中存在，如果存在则提示存在，不存在，则提示不存在。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {209,201,2,2,3,6,7} ;  int searchData = 3 ; // 要查找的内容  boolean flag = false ; // 查找标记  // flag = true：表示查找到内容  // flag = false：表示没有查找到内容  for (int x = 0 ; x < data.length ; x ++) {  if (searchData == data[x]) { // 找到了  flag = true ; // 找到  break ; // 退出循环  }  }  if (flag) { // 为true  System.out.println("已经查找到了内容。") ;  } else {  System.out.println("没有找到内容。") ;  }  }  } |

可是，在以上的代码之中，发现主方法里面的程序太多了，我们的目标是：让主方法的代码变的越少越好，所以应该定义一个查找方法，那么这个查找方法应该返回boolean型数据。这个方法要接收两个参数：要查找的数据、数组。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {209,201,2,2,3,6,7} ;  int searchData = 3 ; // 要查找的内容  if (isExists(data,searchData)) { // 为true  System.out.println("已经查找到了内容。") ;  } else {  System.out.println("没有找到内容。") ;  }  }  public static boolean isExists(int temp [],int search) { // 存在  for (int x = 0 ; x < temp.length ; x ++) {  if (temp[x] == search) {  return true ; // 查找到了，后面的循环不做了  }  }  return false ; // 没有查找到  }  } |

以上的两个程序，只是完成了方法接收数组的操作，但是并没有在方法之中对数组的内容进行修改，而由于是引用传递，方法也是可以对数组进行修改的。下面首先通过一个简单的代码来观察。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {1,2,3,4,5} ; // 5个元素  inc(data) ;  print(data) ;  }  // 数组之中的每一个元素扩大两倍  public static void inc(int temp []) { // 接收数组  for (int x = 0 ; x < temp.length ; x ++) {  temp[x] \*= 2 ;  }  }  public static void print(int temp []) {  for (int x = 0 ; x < temp.length ; x ++) {  System.out.print(temp[x] + "、") ;  }  System.out.println() ;  }  } |

**范例：**数组排序的操作

数组排序是在笔试之中经常会见到的一个最基本的程序。下面就随便讲解一个冒泡程序的实现。

下面按照这个思路，进行代码实现。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {1,3,2,6,10,0,5,8} ;  for (int x = 0 ; x < data.length ; x ++) {  for (int y = 0 ; y < data.length - 1 ; y ++) {  if (data[y] > data[y + 1]) { // 后面的小  int t = data[y] ; // 第三方接收  data[y] = data[y + 1] ;  data[y + 1] = t ;  }  }  }  print(data) ;  }  public static void print(int temp []) {  for (int x = 0 ; x < temp.length ; x ++) {  System.out.print(temp[x] + "、") ;  }  System.out.println() ;  }  } |

但是现在主方法之中的代码过多了，可以编写一个排序方法。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {1,3,2,6,10,0,5,8} ;  sort(data) ;  print(data) ;  }  public static void sort(int temp []) {  for (int x = 0 ; x < temp.length ; x ++) {  for (int y = 0 ; y < temp.length - 1 ; y ++) {  if (temp[y] > temp[y + 1]) { // 后面的小  int t = temp[y] ; // 第三方接收  temp[y] = temp[y + 1] ;  temp[y + 1] = t ;  }  }  }  }  public static void print(int temp []) {  for (int x = 0 ; x < temp.length ; x ++) {  System.out.print(temp[x] + "、") ;  }  System.out.println() ;  }  } |

以上的所有操作都是指方法接收数组进行操作，而方法也可以返回数组，那么这个时候只需要将返回值类型定义为数组即可。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = init() ; // 返回数组接收  print(data) ;  }  public static int [] init() {  int temp [] = new int [] {1,3,2,6,10,0,5,8} ;  return temp ;  }  public static void print(int temp []) {  for (int x = 0 ; x < temp.length ; x ++) {  System.out.print(temp[x] + "、") ;  }  System.out.println() ;  }  } |

**思考题：**要求定义一个方法，这个方法可以统计出数组的最大值、最小值、总和、平均值（忽略小数）

下面首先分着进行操作。

· 统计出数组最大值：假设第一个数据是最大值，如果后面有比这个数据还要大的，修改最大值

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {1,3,2,6,10,0,5,8} ;  int max = data[0] ; // 假设第一个为最大值  for (int x = 0 ; x < data.length ; x ++) {  if (max < data[x]) { // 后面的数据大  max = data[x] ; // 修改max的内容  }  }  System.out.println(max) ;  }  } |

· 最小值：

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {1,3,2,6,10,0,5,8} ;  int min = data[0] ; // 假设第一个为最大值  for (int x = 0 ; x < data.length ; x ++) {  if (min > data[x]) { // 后面的数据大  min = data[x] ; // 修改max的内容  }  }  System.out.println(min) ;  }  } |

· 总和：所有数据累加：

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {1,3,2,6,10,0,5,8} ;  int sum = 0 ; // 保存结果  for (int x = 0 ; x < data.length ; x ++) {  sum += data[x] ; // 累加  }  System.out.println(sum) ;  }  } |

· 平均值：

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {1,3,2,6,10,0,5,8} ;  int sum = 0 ; // 保存结果  for (int x = 0 ; x < data.length ; x ++) {  sum += data[x] ; // 累加  }  System.out.println(sum / data.length) ;  }  } |

· 综合操作，以上的程序是分步完成了数据的计算，可是现在要求的是通过一个方法，取得统计结果。既然是需要方法的统计，那么这个统计的结果就需要返回给调用处，可是一个方法只能返回一种数据类型，现在需要多种数据？可以将返回值定义为数组，而数组之中一共包含四个元素：数组[0] = 最大值、数组[1] = 最小值、数组[2] = 总和、数组[3] = 平均值。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {1,3,2,6,10,0,5,8} ;  int st [] = stat(data) ;  System.out.println("最大值：" + st[0]) ;  System.out.println("最小值：" + st[1]) ;  System.out.println("总 和：" + st[2]) ;  System.out.println("平均值：" + st[3]) ;  }  public static int [] stat(int temp[]) { // 统计  int result [] = new int [4] ; // 长度为4个数组  result[0] = temp[0] ; // 假设第一个元素为最大值  result[1] = temp[0] ; // 假设第一个元素为最小值  for (int x = 0 ; x < temp.length ; x ++) {  if (result[0] < temp[x]) { // 需要更改最大值  result[0] = temp[x] ;  }  if (result[1] > temp[x]) { // 需要更改最小值  result[1] = temp[x] ;  }  result[2] += temp[x] ; // 总和  }  result[3] = result[2] / temp.length ; // 平均值  return result ;  }  } |

以后可能这种复杂的操作不会出现的很多，但是数组是作为逻辑训练的一个有效手段。

### 3.7.4、与数组有关的操作方法

如果要想确定一门语言的优秀，那么一定要看这门语言对开发者的支持度有多好，对于数组的操作，在Java之中提供了两种操作方法：

**1、 数组排序：java.util.Arrays.sort(数组名称)**

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {1,3,2,6,10,0,5,8} ;  **java.util.Arrays.sort(data) ; // 排序**  print(data) ;  }  public static void print(int temp []) {  for (int x = 0 ; x < temp.length ; x ++) {  System.out.print(temp[x] + "、") ;  }  System.out.println() ;  }  } |

这种操作是在工作之中使用的（工作之中也真没见过使）。

**面试题：**请编写一个数组排序操作

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [] = new int [] {1,3,2,6,10,0,5,8} ;  sort(data) ;  print(data) ;  }  **public static void sort(int temp []) {**  **for (int x = 0 ; x < temp.length ; x ++) {**  **for (int y = 0 ; y < temp.length - 1 ; y ++) {**  **if (temp[y] > temp[y + 1]) { // 后面的小**  **int t = temp[y] ; // 第三方接收**  **temp[y] = temp[y + 1] ;**  **temp[y + 1] = t ;**  **}**  **}**  **}**  **}**  public static void print(int temp []) {  for (int x = 0 ; x < temp.length ; x ++) {  System.out.print(temp[x] + "、") ;  }  System.out.println() ;  }  } |

使用java.util.Arrays.sort()也可以排序。

**2、 数组拷贝，从一个数组之中拷贝部分内容到另外一个数组之中**

**· 方法：System.arraycopy(源数组名称，源数组开始点，目标数组名称，目标数组开始点，拷贝长度) ;**

例如，现在给定两个数组：

· 数组A：1、2、**3、4、5**、6、7、8、9；

· 数组B：11、22、33、44、55、**66、77、88**、99。

将数组B的部分内容替换到数组A中，数组A的最终结果：1、2、66、77、88、6、7、8、9。

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int dataA [] = new int [] {1,2,3,4,5,6,7,8,9} ;  int dataB [] = new int [] {11,22,33,44,55,66,77,88,99} ;  System.arraycopy(dataB,5,dataA,2,3) ;  print(dataA) ;  }  public static void print(int temp []) {  for (int x = 0 ; x < temp.length ; x ++) {  System.out.print(temp[x] + "、") ;  }  System.out.println() ;  }  } |

这些操作都属于固定的功能，写熟练即可。

### 3.7.5、二维数组

之前所定义的数组只有一个“[]”，就表示一维数组，如果有两个“[]”就表示二维数组，但是下面通过一个简单的表格，来区分出一维和二维数组的区别。

对于二维数组，本身也存在着两种实例化格式：

**格式一：**动态初始化

|  |
| --- |
| 数据类型 数组名称 [][] = new 数组名称 [行数] [列数] ; |

**格式二：**静态初始化

|  |
| --- |
| 数据类型 数组名称 [][] = new 数组名称 [] [] {  {值,值,...},{值,值,...},{值,值,...},...} ; |

**范例：**观察二维数组

|  |
| --- |
| public class TestDemo {  public static void main(String args[]) {  int data [][] = new int [][] {  {1,2,3},{54,6},{8,9,10,16}} ;  for (int x = 0 ; x < data.length ; x ++) {  for (int y = 0 ; y < data[x].length ; y ++) {  System.out.print(data[x][y] + "、") ;  }  System.out.println() ;  }  }} |

### 3.7.6、对象数组

之前所讲解的全部数组，都属于基本数据类型的数组，但是如果现在要想表示出多个对象，那么就需要对象数组的概念，而对象数组的定义格式和之前是完全一样，只是把数据类型换成类即可。

**格式一：**对象数组的动态初始化

|  |
| --- |
| 类名称 对象数组名称 = new 类名称 [长度] ; |

如果使用了对象数组的动态初始化，则默认情况下，里面的每一个元素都是其对应的默认值null，都需要分别进行对象的实例化操作。

**格式二：**对象数组的静态初始化

|  |
| --- |
| 类名称 对象数组名称 = new 类名称 [] {实例化对象,实例化对象,...} ; |

**范例：**对象数组的动态初始化

|  |
| --- |
| class Person {  private String name ;  private int age ;  public Person(String n,int a) {  name = n ;  age = a ;  }  public String getInfo() {  return "姓名：" + name + "，年龄：" + age ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per [] = new Person[3] ; // 对象数组  per[0] = new Person("张三",20) ;  per[1] = new Person("李四",21) ;  per[2] = new Person("王五",22) ;  for (int x = 0 ; x < per.length ; x ++) {  System.out.println(per[x].getInfo()) ;  }  }  } |

**范例：**对象数组的静态初始化

|  |
| --- |
| class Person {  private String name ;  private int age ;  public Person(String n,int a) {  name = n ;  age = a ;  }  public String getInfo() {  return "姓名：" + name + "，年龄：" + age ;  }  }  public class TestDemo {  public static void main(String args[]) {  Person per [] = new Person[] {  new Person("张三",20),new Person("李四",21),  new Person("王五",22)} ; // 对象数组  for (int x = 0 ; x < per.length ; x ++) {  System.out.println(per[x].getInfo()) ;  }  }  } |

从概念上讲，对象数组的唯一优势，就是可以包含多个对象进行操作。

# 4、总结

1、 类和对象的定义、内存分配、引用传递；

2、 构造方法的定义结构，基本概念；

3、 使用private实现的封装性和setter、getter方法定义要求；

4、 简单java类的定义原则；

5、 数组的基本概念，熟悉即可。

# 5、预习任务

String类的特点及常用方法、this关键字。

# 6、作业

1、 将一个给定的一维整型数组转置输出，

例如： 源数组，1 2 3 4 5 6

转置之后的数组，6 5 4 3 2 1

按照同理，完成一个二维数组的行列转换并输出。

2、 现在有如下的一个数组：

int oldArr[]={1,3,4,5,0,0,6,6,0,5,4,7,6,7,0,5} ;

要求将以上数组中值为0的项去掉，将不为0的值存入一个新的数组，生成的新数组为：

int newArr[]={1,3,4,5,6,6,5,4,7,6,7,5} ;

3、 现在给出两个数组：

• 数组A：“1，7，9，11，13，15，17，19：；

• 数组b：“2，4，6，8，10”

两个数组合并为数组c，按升序排列。