# 1、课程名称：面向对象（基础）


# 2、知识点

## 2.1、上次课程的主要知识点

1、 Java 可移植性原理：依靠 JVM

- 所有的*.java 文件经过 javac 编译之后形成*.class 文件，之后在 JVM 上解释执行，由于对于不同的操作系统有不同的 JVM，所以程序只要有 JVM 就可以执行，那么就可以在不同的操作系统平台上运行所有的 java 程序。

2、 public class 和 class 的区别

- 使用 public class 声明的类必须与文件名称相同，如果使用 class 声明的类可以与文件名称不同，但是执行的时候一定要执行已经生成的*.class 文件。
- 在一个*.java 的文件之中，只能存在一个 public class，但是可以同时存在多个 class 的定义，而且根据不同的 class 声明，可以自动的划分成不同的*.class 文件。
- 只要是定义类则要求每个单词的首字母大写

3、 Java 数据类型划分：
- 基本数据类型：都是一个个具体的值
  - |- 数值型：表示具体的数字，所有的整数默认情况下都是 int，所有的小数都是 double 型的
    - |- 整数型：byte、short、int、long
    - |- 浮点型：float、double
  - |- 字符型：char，使用“‘’”，而且在中文环境下可以设置一个中文文字，采用的是 UNICODE 编码
  - |- 布尔型：boolean，有 true 和 false 两种取值
- 引用数据类型：是靠着内存关系存在的，例如：类、接口、数组

4、 方法：是一段可以重复调用的代码段。
- 如果一个方法要由主方法直接声明，则格式：public static 返回值类型 方法名称(参数列表)
- 重载：方法名称相同，但是参数的类型或个数不同
- 只要是定义方法，则命名规范是：第一个单词的首字母小写，之后每个单词的首字母大写，例如：printInfo()

5、 数组：数组属于引用数据类型，所以一定要划分出内存：栈和堆两块内存
- 数组的定义格式：
  - |- 格式一：数据类型 数组名称[] = new 数据类型[长度] ;
  - |- 格式二：
    - |- 声明数组：数据类型 数组名称[] = null ; à 在栈内存中开辟
    - |- 开辟数组：数组名称 = new 数据类型[长度] ; à 在堆内存中开辟指定大小的数组
- 数组的静态初始化：数据类型 数组名称[] = {值} ;
- 取得数组长度：数组名称.length
- 数组可以使用方法接收，实际上这一点与内存的关系传递是一样的。
- 数组操作相关的方法：java.util.Arrays.sort(数组名称)、System.arraycopy()数组拷贝

## 2.2、本次预计讲解的知识点

1、 面向对象的基本概念
2、 类与对象的关系、引用传递
3、 类封装性的操作
4、 构造方法的定义及使用
5、 String 类的使用
6、 this 关键字的作用

# 3、具体内容

## 3.1、面向对象的概念（了解）

面向对象算是一种比较新的软件设计的方法，在没有面向对象之前使用的是面向过程（是针对于一个问题解决问题，如果修改的话，则整个设计都要修改），面向对象是针对于一类问题来进行解决，某一局部的修改不影响其他位置的变化。

在面向对象中主要分为以下三个方面：
- OOA：面向对象分析。
- OOD：面向对象设计，根据分析的结果使用工具完成设计。

- OOP：完成具体的功能实现，代码编写。

在面向对象中，实际上还存在着三个主要的特性：

- 封装性：保证对外部不可见。
- 继承性：继续发扬广大；
- 多态性：就属于变来变去；

## 3.2、类与对象（重点）

在整个的面向对象之中，实际上最重要的就是类与对象的关系，也就是说面向对象完全是围绕类展开的。

### 3.2.1、类与对象的关系

既然类和对象是核心的基础部分，那么两者之间存在着怎样的关系呢？

类：类是一组共性的产物，是同一类事物的集中表现。

对象：是一种个性的体现，是个体的产物。

对象中的所有操作行为都由类决定，所以，在使用中只要是类定义出的操作对象都可以应用，而如果类没有定义出的操作，对象肯定无法使用。

**类是对象的模板，对象是类的实例**

### 3.2.2、类与对象的使用

在 Java 中可以使用 class 关键字来定义一个类，在类中有两大组成部分：属性（变量）、方法。

```
class Person{     // 定义类
    String name ;          // 表示一个人的姓名
    int age ;              // 表示一个人的年龄
    public void tell(){    // 表示一个功能，说话
        System.out.println("姓名：" + name + "，年龄：" + age) ;
    }
};
```

在本程序中就定义出了一个类，里面有两个属性：name、age，和一个方法：tell()。在 tell()方法将进行内容的输出。一个类定义完成之后肯定无法直接使用，需要依靠对象进行操作，那么下面给出对象定义格式：

- 类名称 对象名称 = new 类名称() ;

一旦有了对象之后就可以通过“对象.属性”或者是“对象.方法()”进行类的调用：

```
class Person{     // 定义类
    String name ;          // 表示一个人的姓名
    int age ;              // 表示一个人的年龄
    public void tell(){    // 表示一个功能，说话
        System.out.println("姓名：" + name + "，年龄：" + age) ;
    }
};
public class OODemo01{
```

```
    public static void main(String args[]){
        Person per = new Person() ;        // 产生对象
        per.name = "张三" ;            // 设置 name 属性的内容
        per.age = 30 ;             // 设置 age 属性的内容
        per.tell() ;          // 调用方法输出
    }
};
```

那么下面来进一步分析以上代码的每一步操作：

- 对象产生：Person per = new Person() ； 既然存在了关键字 new，而且类又属于引用数据类型，那么肯定现在就开辟了相应的栈及堆内存空间，而且以上的代码可以拆分成两步：

  |- 声明对象：Person per = null ; à 在栈内存中声明

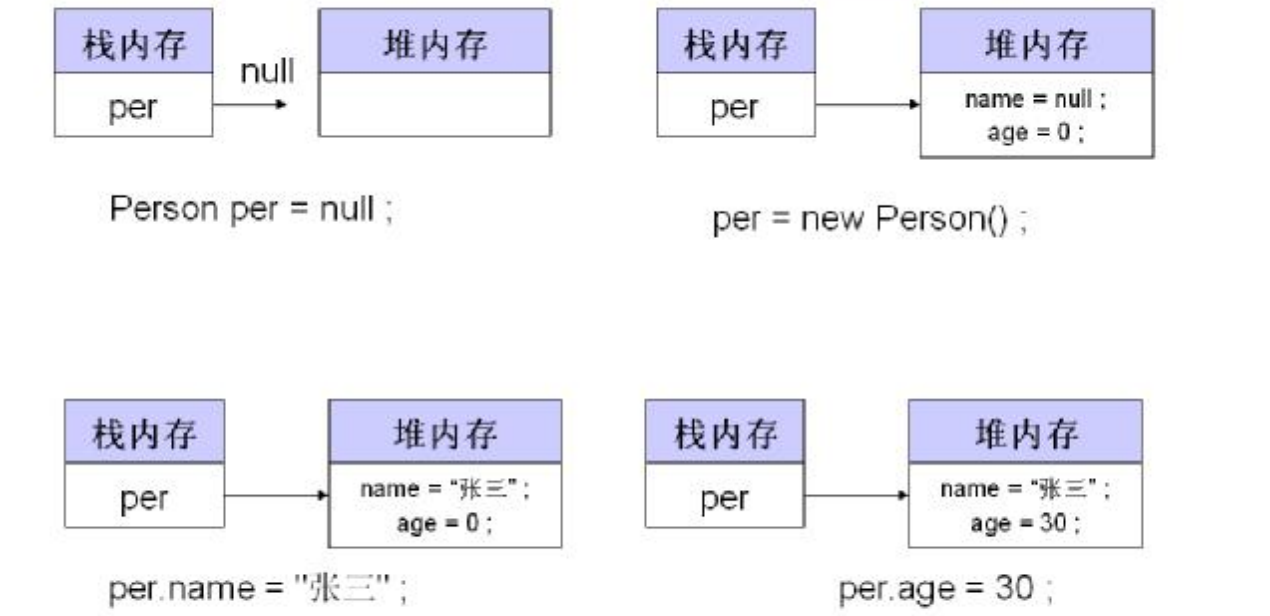
  |- 实例化对象：per = new Person() ; à 开辟对应的堆内存空间

而且一定要记住的是，如果一个对象没有被实例化而直接使用的话，将出现以下的问题：

```
Exception in thread "main" java.lang.NullPointerException
```

表示空指向异常，因为没有对应的堆内存空间，而且此种异常肯定会经常出现，那么以上的代码实际上可以表示成以下的内存关系。







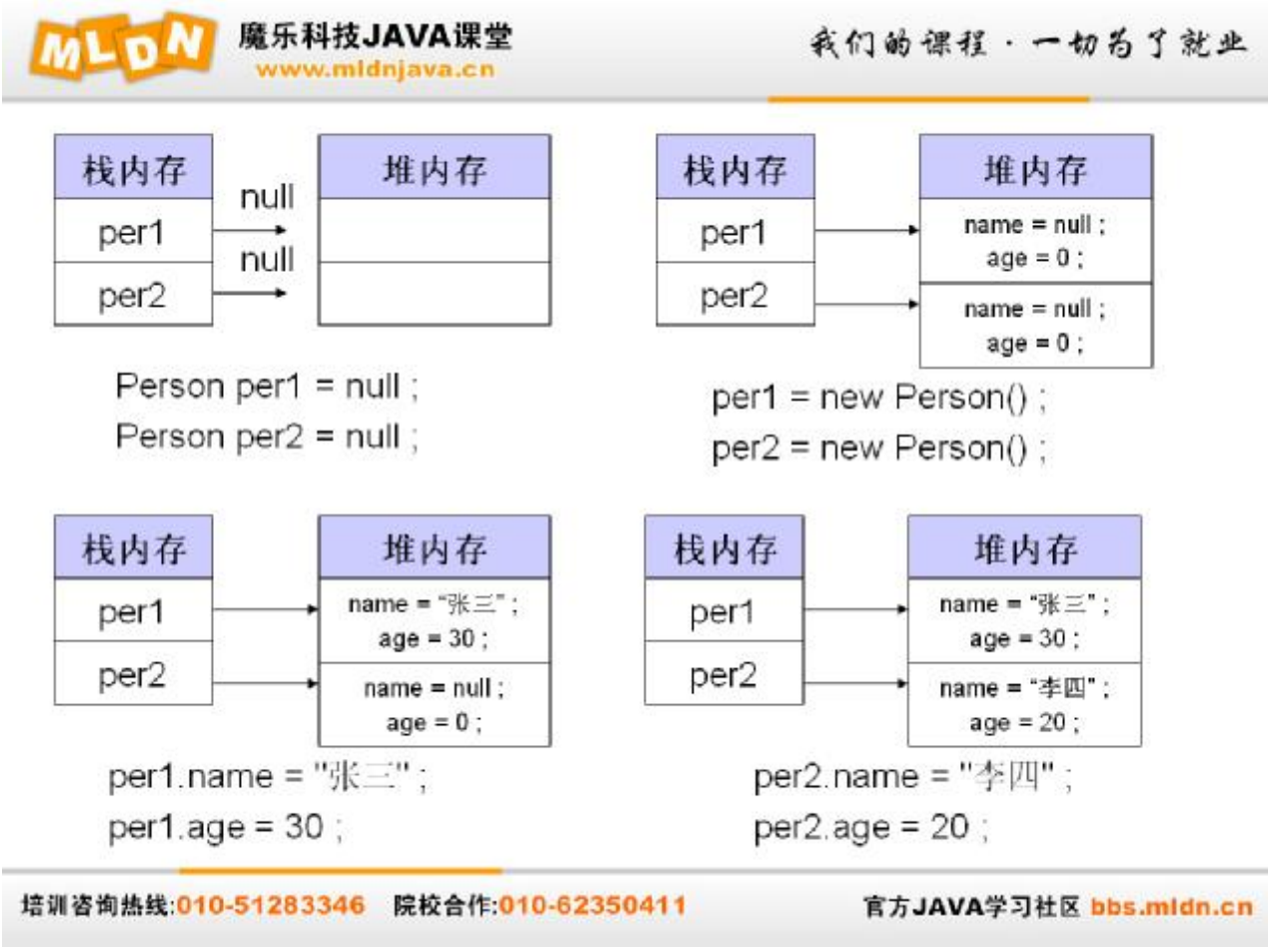

当通过“对象.属性”实际上表示的就是每一个对应的堆内存空间的属性的操作。

那么，如果现在按照以上的规律产生第二个对象肯定是不会互相影响的，因为只要存在了关键字 new 就表示永远会开辟新的内存空间。

```
class Person{     // 定义类
    String name ;          // 表示一个人的姓名
```

```
    int age ;                   // 表示一个人的年龄
    public void tell(){         // 表示一个功能，说话
        System.out.println("姓名：" + name + "，年龄：" + age) ;
    }
};
public class OODemo03{
    public static void main(String args[]){
        Person per1 = null ;
        Person per2 = null ;
        per1 = new Person() ;  // 实例化
        per2 = new Person() ;  // 实例化
        per1.name = "张三" ;
        per1.age = 30 ;
        per2.name = "李四" ;
        per2.age = 20 ;
        per1.tell() ;
        per2.tell() ;
    }
};
```

将以上的代码再次形成内存关系图：




每一个对象拥有各自的内存空间，所以不会互相影响，而且可以发现一点，每一个对象中实际上保存的只是属性，

并没有保存方法，因为所有的方法都是每个对象所共同拥有的，保存在全局代码区之中。

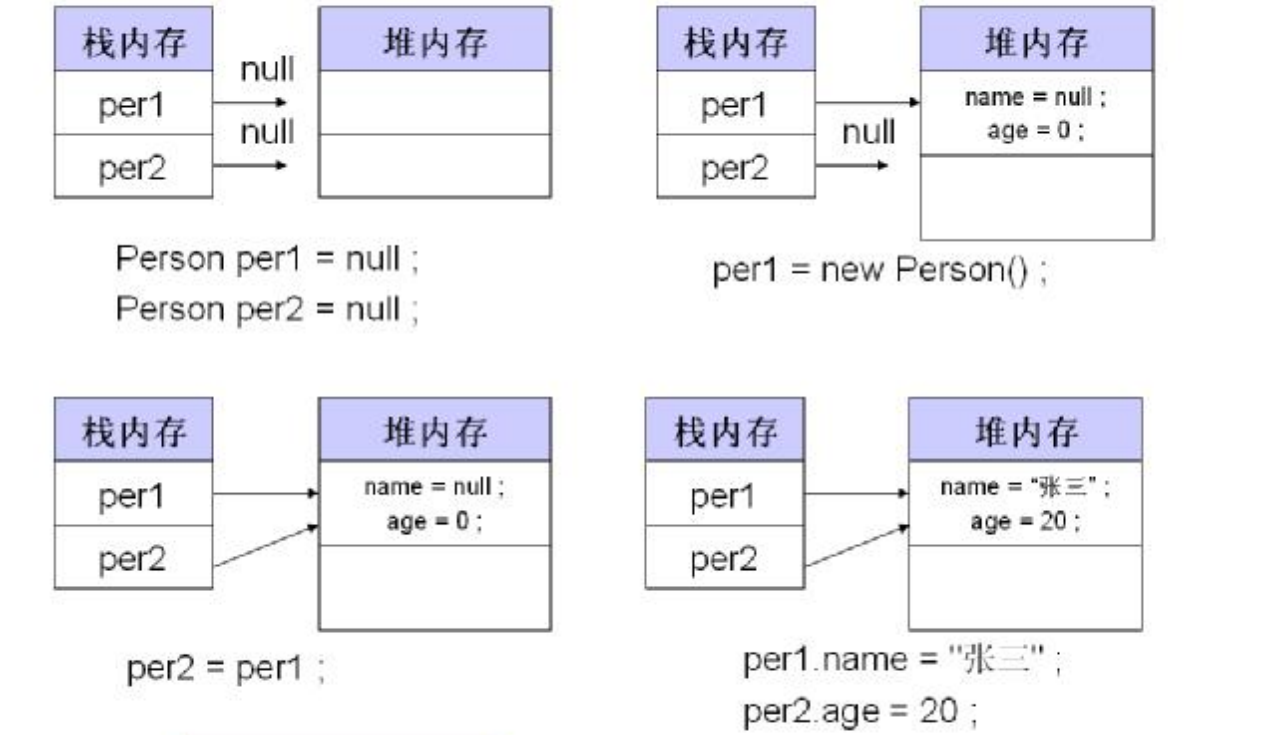
栈内存保存的是对堆内存的引用地址，而堆内存中保存的是每一个对象所拥有的属性，而全局代码区之中保存的是所有类中的方法，表示公共调用。

## 3.2.3、对象的引用传递

在数组中本身也属于引用传递，而且发现所谓的引用传递指的就是一个堆内存空间，可以同时被多个栈内存所指向，那么类本身也是一样的，即：一块堆内存可以同时被多个对象所同时指向。

```
public class OODemo04 {
    public static void main(String args[]){
        Person per1 = null ;
        Person per2 = null ;
        per1 = new Person() ;  // 实例化
        per2 = per1 ;                    // 引用传递
        per1.name = "张三" ;
        per2.age = 20 ;
        per1.tell() ;
    }
};
```





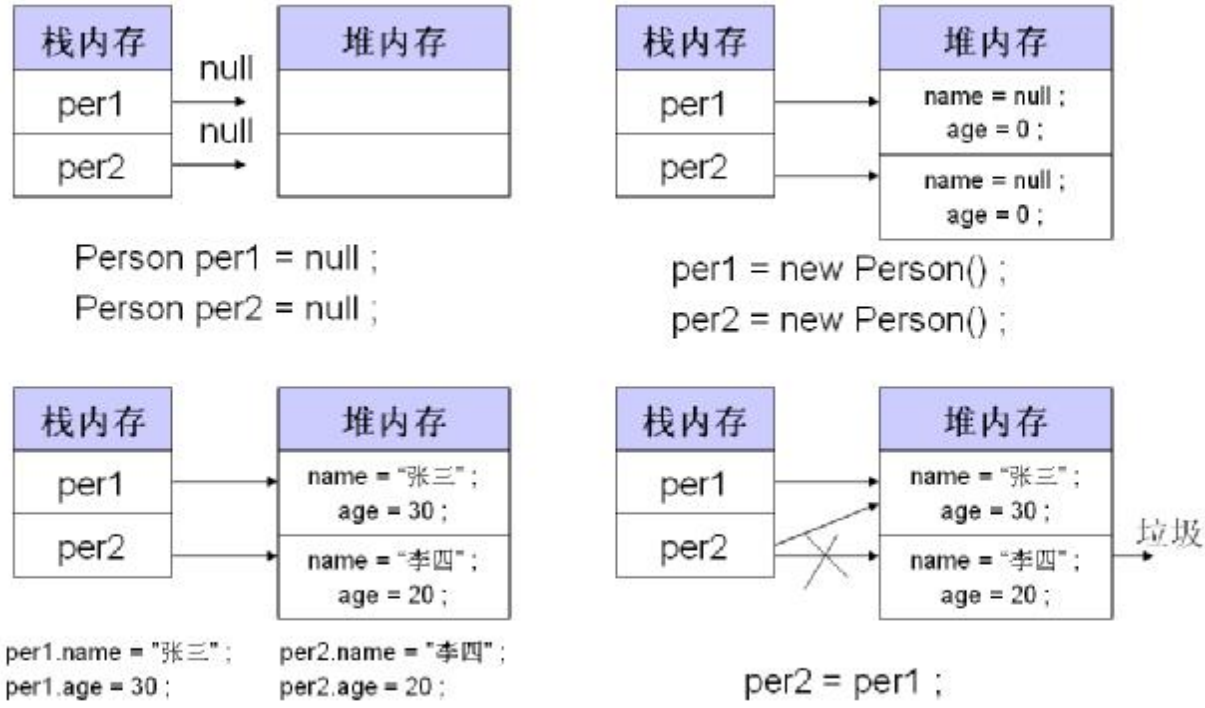



此就称为对象的引用传递，可以发现就是使用内存表示出关系。

但是，在使用引用传递的过程中也会存在一种问题，如下代码：

```
public class OODemo05 {
    public static void main(String args[]){
        Person per1 = null ;
        Person per2 = null ;
        per1 = new Person() ;  // 实例化
        per2 = new Person() ;  // 实例化
        per1.name = "张三" ;
        per1.age = 30 ;
        per2.name = "李四" ;
        per2.age = 20 ;
        per2 = per1 ;
        per2.tell() ;
    }
};
```








现在的某些空间已经不再使用了，所以就成为了垃圾，只要是垃圾就要等待系统进行垃圾收集，在 java 中使用 GC 完成垃圾收集的操作，由于垃圾在程序中会占用系统的资源，所以在开发中一定要尽量避免过多的垃圾产生。

## 3.3、封装性（重点）

封装性属于面向对象的第一大特性，所谓的封装性就是指类内部的定义对外部不可见。

**范例：**观察以下程序的问题

```
class Person{     // 定义类
        String name ;           // 表示一个人的姓名
        int age ;               // 表示一个人的年龄
        public void tell(){     // 表示一个功能，说话
                System.out.println("姓名：" + name + "，年龄：" + age) ;
        }
};
public class EncDemo01 {
        public static void main(String args[]){
                Person per = new Person() ;
                per.name = "张三" ;
                per.age = -30 ;
                per.tell() ;
        }
};
```

此时，将年龄设置成-30 岁发现没有任何的错误，因为在整型的表示范围之中，是允许设置成负数的，那么在这种情况下，语法没有错，但是从实际来看肯定是错误的，因为没有任何一个人的年龄是-30 岁。

那么，会造成这种问题主要就是因为现在类中的属性可以被外部直接访问，那么**如果希望属性或方法不希望被外部所访问的话，则可以使用 private 关键字声明**。

```
class Person{     // 定义类
        private String name ;       // 表示一个人的姓名
        private int age ;           // 表示一个人的年龄
        public void tell(){     // 表示一个功能，说话
                System.out.println("姓名：" + name + "，年龄：" + age) ;
        }
};
public class EncDemo01 {
        public static void main(String args[]){
                Person per = new Person() ;
                per.name = "张三" ;
                per.age = -30 ;
                per.tell() ;
        }
};
```

以上在属性的声明中加入了 private 关键字，所以此时，程序编译的时候：

```
EncDemo01.java:11: name has private access in Person
                        per.name = "张三" ;
                            ^
EncDemo01.java:12: age has private access in Person
                        per.age = -30 ;
                            ^
2 errors
```

提示的错误是 name 和 age 两个属性在 Person 中属于私有的访问，所以外部无法直接调用。

但是，一旦属性被封装之后，访问就成为了一个问题，所以在 JAVA 开发的标准规定中，只要是属性封装，则设置和

取得就要依靠 setter 及 getter 方法完成操作。

例如：现在有一个 String name 的属性

- setter：public void setName(String n){} à 设置的时候可以进行检查
- getter：public String getName(){} à 取得时候只是简单的返回

```
class Person{     // 定义类
    private String name ;          // 表示一个人的姓名
    private int age ;              // 表示一个人的年龄
    public void setName(String n){     // 设置姓名
        name = n ;
    }
    public void setAge(int a){     // 设置年龄
        if(a>=0 && a<=150){ // 合法年龄
            age = a ;
        }
    }
    public String getName(){     // 取得姓名
        return name ;
    }
    public int getAge(){     // 取得年龄
        return age ;
    }
    public void tell(){       // 表示一个功能，说话
        System.out.println("姓名：" + name + "，年龄：" + age) ;
    }
};
public class EncDemo02 {
    public static void main(String args[]){
        Person per = new Person() ;
        per.setName("张三") ;
        per.setAge(30) ;
        per.tell() ;
    }
};
```

**以后只要是属性就必须进行封装，封装之后的属性必须通过 setter 和 getter 设置和取得。**

```
class Person{     // 定义类
    private String name ;          // 表示一个人的姓名
    private int age ;              // 表示一个人的年龄
    public void setName(String n){     // 设置姓名
        name = n ;
    }
    public void setAge(int a){     // 设置年龄
        if(a>=0 && a<=150){ // 合法年龄
            age = a ;
        }
```

```
    }
    public String getName(){    // 取得姓名
        return name ;
    }
    public int getAge(){    // 取得年龄
        return age ;
    }
    public void tell(){     // 表示一个功能，说话
        System.out.println("姓名：" + this.getName() + "，年龄：" + this.getAge()) ;
    }
};
```

强调：在一个类中所有的方法是允许互相调用的，如果非要强调是本类中的方法的话，则在调用的方法前增加一个“this”关键字，表示的是本类中的方法。

## 3.4、构造方法（重点）

在讲解构造方法之前，先来观察以下的代码：

```
Person per = new Person() ;
```

以上的 Person()就表示的是一个构造方法，此构造方法属于默认的构造方法。

构造方法的定义：在一个类中定义的方法名称与类名称相同，且无返回值声明的方法，称为构造方法。

但是，在一个类中如果没有明确的定义一个构造方法的话，则会自动生成一个无参的，什么都不做的构造方法。

```
class Person{    // 定义类
    private String name ;        // 表示一个人的姓名
    private int age ;            // 表示一个人的年龄
    public Person(){        // 如果一个类没有定义构造，则自动生成此代码
        System.out.println("一个新的 Person 对象产生了。") ;
    }
    public void setName(String n){    // 设置姓名
        name = n ;
    }
    public void setAge(int a){    // 设置年龄
        if(a>=0 && a<=150){ // 合法年龄
            age = a ;
        }
    }
    public String getName(){    // 取得姓名
        return name ;
    }
    public int getAge(){    // 取得年龄
        return age ;
    }
    public void tell(){     // 表示一个功能，说话
        System.out.println("姓名：" + this.getName() + "，年龄：" + this.getAge()) ;
```

```
    }
};
public class ConDemo01 {
    public static void main(String args[]){
        Person per = null ;
        per = new Person() ;  // 实例化
        per = new Person() ;  // 实例化
        per = new Person() ;  // 实例化
    }
};
```

可以发现，所有的构造方法是在对象使用关键字 new 进行实例化的时候调用的，而且每次使用关键字 new 开辟新的堆内存空间时，构造方法都要被调用。

**实际上构造方法的主要作用就是为一个类中的属性初始化的。**

```
class Person{    // 定义类
    private String name = "测试" ;          // 表示一个人的姓名
    private int age = 10 ;                  // 表示一个人的年龄
    public Person(){        // 如果一个类没有定义构造，则自动生成此代码
        System.out.println("一个新的 Person 对象产生了。") ;
    }
}
```

在以上的类中定义属性的时候，为每一个属性都设置了内容，但是这些设置的内容是只有在构造方法执行完毕之后才会为属性赋值的。

可以发现，构造方法在对象实例化的时候调用，所以一般构造方法的主要作用是在一个对象实例化时，向类中的属性传递一些初始化内容使用的。

**当一个类中已经明确的定义了一个构造方法的时候，则无参构造方法将不再自动生成，也就是说一个类永远都会保证至少有一个构造方法。**

```
class Person{    // 定义类
    private String name   ;      // 表示一个人的姓名
    private int age ;            // 表示一个人的年龄
    public Person(String n,int a){// 通过构造方法为属性自动初始化
        this.setName(n) ;
        this.setAge(a) ;
    }
    public void setName(String n){    // 设置姓名
        name = n ;
    }
    public void setAge(int a){    // 设置年龄
        if(a>=0 && a<=150){ // 合法年龄
            age = a ;
        }
    }
    public String getName(){    // 取得姓名
        return name ;
    }
```

```
    public int getAge(){    // 取得年龄
        return age ;
    }
    public void tell(){     // 表示一个功能，说话
        System.out.println("姓名：" + this.getName() + "，年龄：" + this.getAge()) ;
    }
};
public class ConDemo02 {
    public static void main(String args[]){
        Person per = new Person("张三",-30) ;  // 实例化
        per.tell() ;
    }
};
```

当在开发中，需要通过构造方法设置内容的时候，而且需要检验的时候，需要在构造方法中明确的调用 setter 方法进行内容的设置。

构造方法本身既然是方法，方法就允许重载，所以构造方法本身也可以进行重载的操作，重载的原则：方法名称相同，参数的类型或个数不同。

```
class Person{    // 定义类
    private String name   ;      // 表示一个人的姓名
    private int age ;            // 表示一个人的年龄
    public Person(){
        System.out.println("无参构造。") ;
    }
    public Person(String name){
        System.out.println("有一个参数的构造") ;
    }
    public Person(String name,int age){
        System.out.println("有两个参数的构造") ;
    }
    public void tell(){
        System.out.println("姓名：" + name + "，年龄：" + age) ;
    }
};
public class ConDemo03 {
    public static void main(String args[]){
        Person per1 = new Person() ;// 实例化
        Person per2 = new Person("张三") ;    // 实例化
        Person per3 = new Person("张三",-30) ; // 实例化
    }
};
```

## 3.5、匿名对象（重点）

匿名对象就是表示没有名字的对象。那么什么叫对象的名字呢？

```
class Person{     // 定义类
        private String name    ;        // 表示一个人的姓名
        private int age ;                // 表示一个人的年龄
        public Person(String n,int a){// 通过构造方法为属性自动初始化
                this.setName(n) ;
                this.setAge(a) ;
        }
        public void setName(String n){     // 设置姓名
                name = n ;
        }
        public void setAge(int a){     // 设置年龄
                if(a>=0 && a<=150){ // 合法年龄
                        age = a ;
                }
        }
        public String getName(){     // 取得姓名
                return name ;
        }
        public int getAge(){     // 取得年龄
                return age ;
        }
        public void tell(){       // 表示一个功能，说话
                System.out.println("姓名：" + this.getName() + "，年龄：" + this.getAge()) ;
        }
};
public class NonameDemo {
        public static void main(String args[]){
                new Person("张三",30).tell() ;
        }
};
```

但是由于匿名对象没有对应的栈内存所指向，所以使用一次之后就等待被垃圾回收了。

# 3.6、题目（重点）

现在要求实现一个雇员的操作类，里面包含雇员编号、姓名、基本工资、奖金，要求可以输出一个雇员的完整信息，可以求出一个雇员的年薪、月薪、日平均工资，计算工资的时候要加上奖金。

## 3.6.1、参考步骤

以后拿到一个编写类的程序，按照以下的步骤编写代码：

1、 根据要求写出类的名称；
2、 根据要求定义出类中所应该包含的属性；
3、 所有的属性必须封装；

4、 封装之后的属性必须通过 setter 和 getter 设置和取得；

5、 根据其他的要求编写出相应的方法；

6、 所有的输出不能在类中直接完成，而是交给被调用处输出；

7、 根据实际的情况添加构造方法，但是不管类中有多少个构造，一定要保留一个无参构造方法。

## 3.6.2、代码实现

```
class Emp {
    private int empno ;
    private String ename ;
    private float sal ;
    private float comm ;
    public Emp(){}  // 无参构造必须保留
    public Emp(int empno,String ename,float sal,float comm){
        this.setEmpno(empno) ;
        this.setEname(ename) ;
        this.setSal(sal) ;
        this.setComm(comm) ;
    }
    public String getInfo(){
        return      "雇员信息：\n" +
                    "\t|- 雇员编号：" + this.getEmpno() + "\n" +
                    "\t|- 雇员姓名：" + this.getEname() + "\n" +
                    "\t|- 基本工资：" + this.getSal() + "\n" +
                    "\t|- 奖金：" + this.getComm() +   "\n" +
                    "\t|- 日薪：" + this.daily() +   "\n" +
                    "\t|- 月薪：" + this.salary() +   "\n" +
                    "\t|- 年薪：" + this.income();
    }
    public float salary(){
        return sal + comm ;
    }
    public float income(){
        return this.salary() * 12 ;
    }
    public float daily(){
        return this.salary() / 30 ;
    }
    public void setEmpno(int e){
        empno = e ;
    }
    public void setEname(String e){
        ename = e;
```

```
    }
    public void setSal(float s){
        sal = s ;
    }
    public void setComm(float c){
        comm = c ;
    }
    public int getEmpno(){
        return empno ;
    }
    public String getEname(){
        return ename ;
    }
    public float getSal(){
        return sal ;
    }
    public float getComm(){
        return comm ;
    }
};
public class EmpDemo {
    public static void main(String args[]){
        Emp emp = new Emp(1001,"张三",3000.0f,500.0f) ;
        System.out.println(emp.getInfo()) ;
    }
};
```

# 3.7、String 类（重点）

String 类本身由于其命名规范的要求肯定是一个类，但是却发现，此类在使用上较为特殊，而且发现在 String 类的操作中依然可以使用“+”或“+=”进行字符串的连接，还可以发现，各个数据类型的数据只要是碰到了 String 类的内容，则都统一向 String 进行转换，所以 String 类的使用及特点就非常的重要。

## 3.7.1、String 类的对象实例化

String 在使用中有两种实例化的方法，一种是采用直接赋值（将字符串赋 String 的对象），第二种方法是通过构造方法完成实例化的操作，方法如下：

| No. | 方法名称 | 类型 | 描述 |
|---|---|---|---|
| 1 | public String(String original) | 普通 | 接收 String 对象，实例化 String 类 |

**范例：**方法一进行对象实例化

```
public class StringDemo01 {
    public static void main(String args[]){
        String str = "Hello" ;  // 定义一个字符串
```

```
        str += " World" ;       // 字符串连接
        System.out.println(str) ;
    }
};
```

**范例：**方法二进行对象实例化

```
public class StringDemo02 {
    public static void main(String args[]){
        String str = new String("Hello") ;  // 定义一个字符串
        str += " World" ;       // 字符串连接
        System.out.println(str) ;
    }
};
```

现在可以发现，String 类的实例化方式有两种，具体有何区别，之后再讲解。

## 3.7.2、字符串的比较（重点）

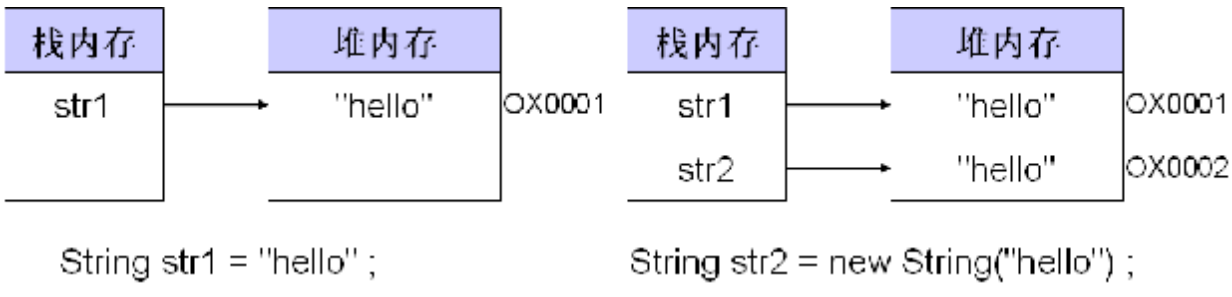
如果现在假设有两个整数，直接通过“==”判断两个内容是否相等。

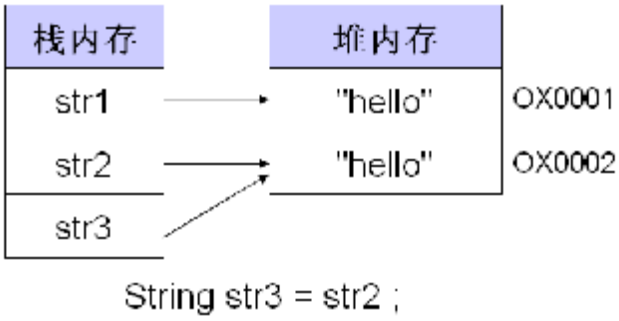
```
public class StringDemo03 {
    public static void main(String args[]){
        int x = 10 ;
        int y = 10 ;
        System.out.println(x==y) ;
    }
};
```

两个数字之间直接使用==进行判断即可，但是，==也同样可以应用在 String 上。

```
public class StringDemo04 {
    public static void main(String args[]){
        String str1 = "hello" ;
        String str2 = new String("hello") ;
        String str3 = str2 ;
        System.out.println("str1 == str2 --> " + (str1==str2)) ;     // false
        System.out.println("str1 == str3 --> " + (str1==str3)) ;     // false
        System.out.println("str2 == str3 --> " + (str2==str3)) ;     // false
    }
};
```

以上的内容确实相等，但是使用了“==”之后发现有的相等，有的不等，为什么呢？观察内存关系决定。






从以上的图形中可以清晰的发现，str1 和 str2、str3 的地址完全不一样，但是 str2 和 str3 的地址是一样的，如果按照==比较的结果来看，实际上在使用==操作的时候比较的是两个对象的内存地址，与内容无关。

**所以“==”用在数值上表示判断是否相等，如果是用在了对象上，则表示的是判断两个对象的地址是否相等。**

但是，如果现在要是想进行内容比较的话，则就只能靠 String 类提供的方法完成了。

| No. | 方法名称 | 类型 | 描述 |
|---|---|---|---|
| 1 | public boolean equals(String anObject) | 普通 | 判断两个字符串的内容是否相等 |

**范例：**验证 equals()方法

```
public class StringDemo05 {
    public static void main(String args[]){
        String str1 = "hello" ;
        String str2 = new String("hello") ;
        String str3 = str2 ;
        System.out.println("str1 equals str2 --> " + (str1.equals(str2))) ; // true
        System.out.println("str1 equals str3 --> " + (str1.equals(str3))) ; // true
        System.out.println("str2 equals str3 --> " + (str2.equals(str3))) ; // true
    }
};
```

通过比较可以发现，虽然现在各个对象的地址不一样，但是通过了 equals()方法进行比较之后，发现比较的是内容，所以，以后只要是字符串的比较操作就统一都使用 equals()方法完成。

**两种比较的区别：==和 equals()**

- ==：用于数值比较，比较的是两个字符串的地址值
- equals()：用于内容的比较，比较两个字符串的内容是否相等

## 3.7.3、一个字符串就是一个 String 的匿名对象

在程序中使用“""”括起来的内容就属于字符串，那么对于一个普通的字符串来讲，一个字符串就是 String 类的匿名对象可以直接使用。

```
public class StringDemo06 {
    public static void main(String args[]){
        System.out.println("hello".equals("hello")) ;  // true
    }
};
```

通过本程序可以清楚的进行验证，所以之前使用直接赋值的方式为 String 类进行实例化的操作，就属于将一个匿名对象起了一个名字。

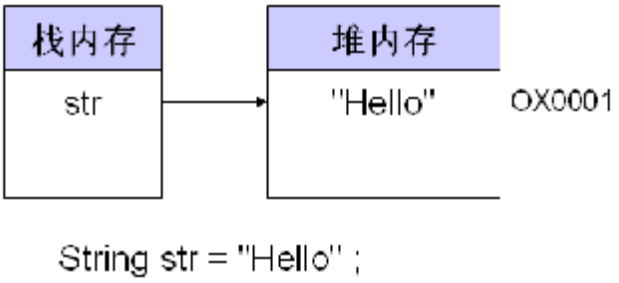
# 3.7.4、两种实例化的区别

String 类的对象可以有两种实例化方式，那么这两种到底有什么区别，以及在开发中到底该使用那种呢？

**范例：**第一种

```
String str = "Hello" ;
```

按照之前的理解，现在肯定是将一个字符串的堆空间的地址给 str，所以图形如下：

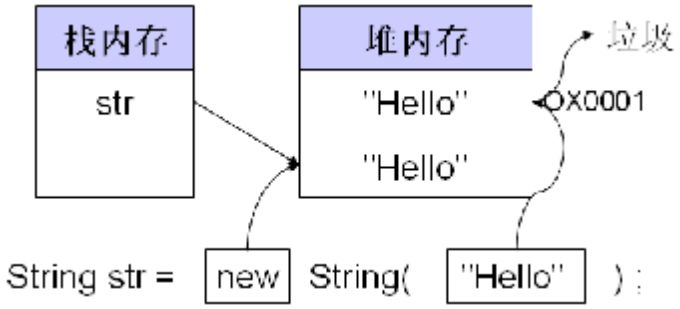



String str = "Hello" ;

此时的代码中，只在堆内存中开辟了一块空间，但是如果现在是第二种方式：

```
String str = new String("Hello") ;
```

使用构造方法进行对象实例化的时候，由于内部的字符串属于匿名对象的原因，所以将产生两块堆内存空间，其中有一块将成为垃圾。




除了内存空间的少之外，如果使用了直接赋值现在还有另外一种好处，就是如果以后如果再声明了与之一样的字符串的话，则不会再重新开辟新的内存空间。

```
public class StringDemo07 {
    public static void main(String args[]){
        String str1 = "hello" ;
        String str2 = "hello" ;
        String str3 = "hello" ;
        System.out.println("str1 == str2 --> " + (str1==str2)) ;      // true
        System.out.println("str1 == str3 --> " + (str1==str3)) ;      // true
        System.out.println("str2 == str3 --> " + (str2==str3)) ;      // true
    }
};
```

此时返回的结果都是 true，所以很明显，以上的三个 String 的对象都同时指向同一个堆内存空间的“hello”字符串，那么如果使用的是构造方法赋值的话，根本不可能实现此效果。

之所以内容会相等，是因为在 JAVA 底层中存在了一个字符串的对象池，每次的内容都会从池中取出，如果内容已经存在了，则使用已经存在的，如果不存在，则会生成一个新的，放入池中，属于**共享设计模式**。

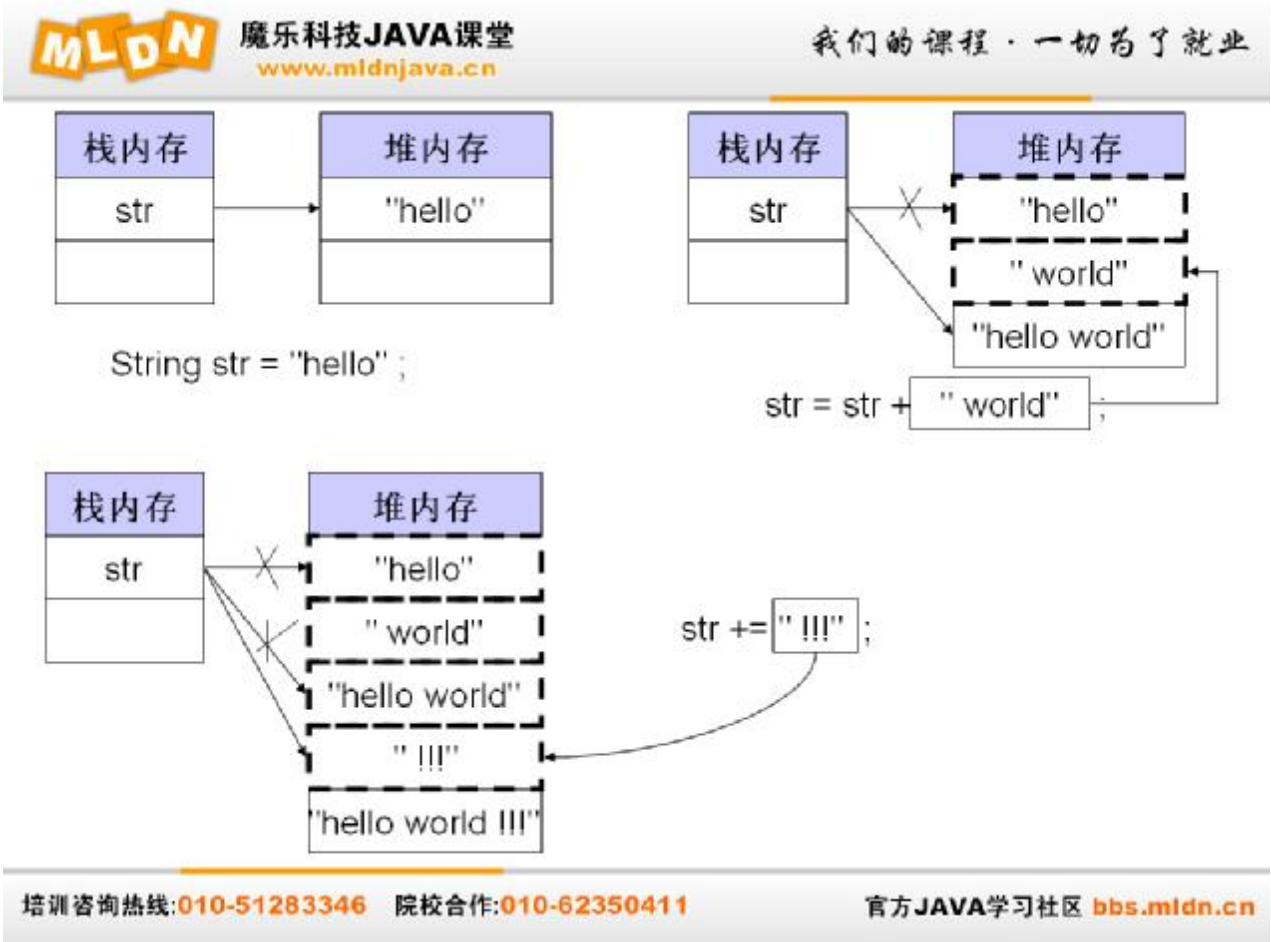
由于这种原因，只要是使用 String 类的话，永远都要采用直接赋值的方式完成。

# 3.7.5、字符串的内容一旦声明则不可改变

一个字符串肯定是保存在堆内存之中的，所以一旦一个字符串声明之后，则无法改变。

```
public class StringDemo08 {
    public static void main(String args[]){
        String str = "hello" ;
        str = str + " world" ;
        str += " !!!" ;
        System.out.println(str) ;
    }
};
```

现在 str 的内容确实改变了，但是字符串呢？通过程序分析一下。




通过内存图可以发现一点，每一个字符串的内容原本并没有发生任何的改变，改变的是只一个字符串的内存地址的指向，而且操作完成之后会发现有很多的垃圾产生，所以，以后对于频繁修改字符串的代码应该尽可能的回避，所以以下的代码是极其垃圾的：

```
public class StringDemo09 {
    public static void main(String args[]){
        String str = "hello" ;
        for(int x=0;x<10000;x++){
            str += x ;
```

```
        }
        System.out.println(str) ;
    }
};
```

此种操作的的代码要不断的修改 String 的指向 1000 次，所以肯定将产生极大的垃圾空间，那么这种代码必须回避。

# 3.8、String 类的相关操作（重点）

在实际的开发中，String 类的使用是最频繁的，而且在 String 类中提供了大量的操作方法，这些操作方法对于程序的开发都是非常有帮助的，所以必须把该背的都背下来。

所有的方法都是在 JDK 的 doc 文档之中存在的。

## 3.8.1、字符串与字符数组

在许多的编程语言中实际上都是一直在强调一个字符串就是一个字符数组，所以在 Java 的 String 类为了体现此概念也提供了以下的一些操作方法：

| No. | 方法名称 | 类型 | 描述 |
|---|---|---|---|
| 1 | public String(char[] value) | 构造 | 将一个字符数组全部变为字符串 |
| 2 | public String(char[] value,int offset,int count) | 构造 | 将一个字符数组的指定范围的内容变为字符串 |
| 3 | public char[] toCharArray() | 普通 | 将字符串变为字符数组 |
| 4 | public char charAt(int index) | 普通 | 返回指定位置的字符 |

**范例：**观察字符串和字符数组的转换

```
public class StringAPIDemo01 {
    public static void main(String args[]){
        String str = "helloworld" ;
        char c[] = str.toCharArray() ;// 将字符串变为字符数组
        for(int x=0;x<c.length;x++){
            System.out.print(c[x] + "、") ;
        }
        System.out.println("\n" + new String(c)) ;
        System.out.println(new String(c,0,5)) ;
    }
};
```

**范例：**取出指定位置的字符

```
public class StringAPIDemo02 {
    public static void main(String args[]){
        String str = "helloworld" ;
        System.out.println(str.charAt(0)) ;
        System.out.println(str.charAt(3)) ;
    }
};
```

字符串和字符数组的关系就非常的明确了，当然也可以通过循环的方式，利用 charAt()取出每一个字符的内容，如果要想循环，则肯定要确定字符串的长度，长度通过以下方法求得：

| No. | 方法名称 | 类型 | 描述 |
|---|---|---|---|
| 1 | public int length() | 普通 | 求出字符串的长度 |

但是一定要注意的是，在数组的操作中，可以通过“数组名称.length”求得长度。

```
public class StringAPIDemo03 {
    public static void main(String args[]){
        String str = "helloworld" ;
        System.out.println("字符串的长度是：" + str.length()) ;
        for(int x=0;x<str.length();x++){
            System.out.print(str.charAt(x) + "、") ;
        }
    }
};
```

## 3.8.2、字符串与字节数组

在程序中 byte 表示每一个字节，一般在以后进行 IO 操作的时候往往会使用到字节数组的功能，现在先来观察一些方法的支持：

| No. | 方法名称 | 类型 | 描述 |
|---|---|---|---|
| 1 | public String(byte[] bytes) | 普通 | 将字节数组变为字符串 |
| 2 | public String(byte[] bytes,int offset,int length) | 普通 | 将指定范围的字节数组变为字符串 |
| 3 | public byte[] getBytes() | 普通 | 将字符串变为字节数组 |

一定要再次强调的是，以后只要是操作二进制数据的时候才会考虑使用字节，现阶段暂时不使用。

**范例：**观察转换的操作

```
public class StringAPIDemo04 {
    public static void main(String args[]){
        String str = "helloworld" ;
        byte b[] = str.getBytes() ;     // 将字符串变为字节数组
        System.out.println(new String(b)) ;
        System.out.println(new String(b,0,5)) ;
    }
};
```

## 3.8.3、字符串比较

之前已经讲解过了字符串的比较方法使用的是 equals()但是，这种比较的方法本身是区分大小写的，所以如果不区分大小写，可以使用如下的方法完成：

| No. | 方法名称 | 类型 | 描述 |
|---|---|---|---|
| 1 | public boolean equalsIgnoreCase(String anotherString) | 普通 | 忽略大小写的比较 |

**范例：**观察比较的操作

```
public class StringAPIDemo05 {
    public static void main(String args[]){
        String str = "hello" ;
```

```
        System.out.println(str.equals("HELLO")) ;
        System.out.println(str.equalsIgnoreCase("HELLO")) ;
    }
};
```

## 3.8.4、字符串截取

在字符串的操作中可以使用如下的两个方法进行字符串的截取操作：

| No. | 方法名称 | 类型 | 描述 |
|---|---|---|---|
| 1 | public String substring(int beginIndex) | 普通 | 从指定位置截取到末尾 |
| 2 | public String substring(int beginIndex,int endIndex) | 普通 | 截取指定范围的内容 |

**范例：**观察截取

```
public class StringAPIDemo06 {
    public static void main(String args[]){
        String str = "helloworld" ;
        System.out.println(str.substring(5)) ;
        System.out.println(str.substring(0,5)) ;
    }
};
```

## 3.8.5、判断字符串是否存在

可以在一个字符串中判断指定的内容是否存在，使用如下的方法：

| No. | 方法名称 | 类型 | 描述 |
|---|---|---|---|
| 1 | public int indexOf(String str) | 普通 | 从头开始判断，如果找到则返回位置，找不到返回-1 |
| 2 | public int indexOf(String str,int fromIndex) | 普通 | 从指定位置开始查找，如果找到则返回位置，找不到返回-1 |
| 3 | public boolean contains(String s) | 普通 | 直接判断是否存在 |

一般而言已经开发施加较长的人员都会使用 indexOf()进行判断，通过判断其是否是-1，来决定是否有内容。

**范例：**使用 indexOf()完成功能

```
public class StringAPIDemo07 {
    public static void main(String args[]){
        String str = "helloworld" ;
        if(str.indexOf("hello") != -1){
            System.out.println("已经查找到了内容，位置是：" + str.indexOf("hello")) ;
        }
    }
};
```

但是，在新的 JDK 里面增加了一个 contains()方法，可以直接判断。

```
public class StringAPIDemo08 {
    public static void main(String args[]){
```

```
        String str = "helloworld" ;
        if(str.contains("hello")){
            System.out.println("已经查找到了内容，位置是：" + str.indexOf("hello")) ;
        }
    }
};
```

## 3.8.6、字符串拆分

按照一个指定的字符串将一个大的字符串进行拆分，使用方法如下：

| No. | 方法名称 | 类型 | 描述 |
|---|---|---|---|
| 1 | Public String[] split(String regex) | 普通 | 按照指定字符串拆分 |

**范例：**观察拆分的操作

```
public class StringAPIDemo09 {
    public static void main(String args[]){
        String str = "hello world !!!" ;
        String s[] = str.split(" ") ;
        for(int x=0;x<s.length;x++){
            System.out.print(s[x] + "、") ;
        }
    }
};
```

但是，以上属于正常的拆分，因为可以按照“ ”进行拆分，但是如果现在有一个 IP 地址要求拆分。

```
public class StringAPIDemo10 {
    public static void main(String args[]){
        String str = "192.168.1.3" ;
        String s[] = str.split("\\.") ;
        for(int x=0;x<s.length;x++){
            System.out.print(s[x] + "、") ;
        }
    }
};
```

**在进行字符串拆分的时候，如果发现有某些字符串无法拆分，则使用转义。**

## 3.8.7、字符串替换

可以将一个字符串中的指定内容替换成其他的显示内容，方法如下：

| No. | 方法名称 | 类型 | 描述 |
|---|---|---|---|
| 1 | public String replaceAll(String regex,String replacement) | 普通 | 全部替换 |
| 2 | public String replaceFirst(String regex,String replacement) | 普通 | 替换第一个满足的内容 |

**范例：**观察替换的操作

```
public class StringAPIDemo11 {
    public static void main(String args[]){
        String str = "helloworld" ;
        System.out.println(str.replaceAll("l","⊙")) ;
        System.out.println(str.replaceFirst("l","⊙")) ;
    }
};
```

## 3.8.8、其他操作

在 String 类中还有一些方法无法归类，统一称为其他操作，方法有如下几个：

| No. | 方法名称 | 类型 | 描述 |
|---|---|---|---|
| 1 | public String toLowerCase() | 普通 | 转小写 |
| 2 | public String toUpperCase() | 普通 | 转大写 |
| 3 | public String trim() | 普通 | 去掉左右空格 |
| 4 | public **boolean** startsWith(String prefix) | 普通 | 判断是否以指定的字符串开头 |
| 5 | public **boolean** endsWith(String suffix) | 普通 | 判断是否以指定的字符串结尾 |

**范例：**观察以上的操作代码

```
public class StringAPIDemo12 {
    public static void main(String args[]){
        String str = "                helloworld                ".trim()  ;
        System.out.println(str) ;
        System.out.println(str.toUpperCase()) ;
        System.out.println(str.startsWith("hello")) ;
        System.out.println(str.endsWith("d")) ;
    }
};
```

## 3.8.9、题目

1、 要求将以下的内容进行拆分显示

- 原始内容：“张三：30 | 李四：20 | 王五：25”
- 显示结果： 姓名：张三，年龄：20

```
public class StringAPIDemo13 {
    public static void main(String args[]){
        String str = "张三:30|李四:20|王五:25" ;
        String s1[] = str.split("\\|") ;
        for(int x=0;x<s1.length;x++){
            String s2[] = s1[x].trim().split(":") ;
            System.out.println("姓名：" + s2[0] + "，年龄：" + s2[1]) ;
        }
    }
```

```
};
```

2、 判断一个 email 地址是否合法

- 提示：只需要判断@和.即可

```
public class StringAPIDemo14 {
    public static void main(String args[]){
        String email = "aa@aa.com" ;
        if(check(email)){
            System.out.println("EMAIL 合法！") ;
        } else {
            System.out.println("EMAIL 不合法！") ;
        }
    }
    public static boolean check(String mail){
        boolean flag = true ;   // 假设认为传进来的就是合法的 email 地址
        if(mail == null){ // 没有任何的内容
            flag = false ;
        } else {
            if(mail.indexOf("@")==-1 || mail.indexOf(".")==-1){      // 判断@和.是否存在
                flag = false ;
            } else {
                if(mail.indexOf("@") > mail.indexOf(".")){   // 判断@是否在.之后
                    flag = false ;
                } else {
                    if(mail.startsWith("@") || mail.endsWith(".")){
                        flag = false ;
                    }
                }
            }
        }
        return flag ;
    }
};
```

# 3.9、对象数组（重点）

对象数组就是一组对象，对象数组的定义格式如下：

```
类名称 对象数组名称 [] = new 类名称[对象数组长度] ;
```

但是由于类是引用数据类型，所以对象数组中的每一个内容都是 null，在使用的时候肯定是需要进行分别实例化的。

```
class Person {
    private String name ;
    private int age ;
    public Person(String n,int a){
        name = n ;
```

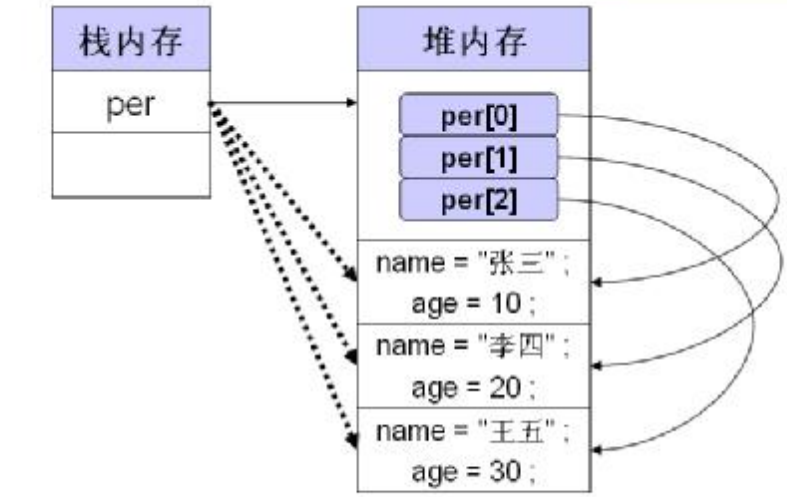
```
            age = a ;
        }
        public String getInfo(){
            return "姓名：" + name + "，年龄：" + age ;
        }
};
public class ObjectArrayDemo01 {
        public static void main(String args[]){
            Person per[] = new Person[3] ;     // 开辟三个大小的对象数组
            per[0] = new Person("张三",10) ;
            per[1] = new Person("李四",20) ;
            per[2] = new Person("王五",30) ;
            for(int x=0;x<per.length;x++){
                System.out.println(per[x].getInfo()) ;
            }
        }
};
```

以上的操作是属于一个对象数组的动态初始化的，但是回顾一下，之前曾经使用过“String s[]”，这个实际上也是对象数组，因为 String 本身就属于一个类，只是这个类是由系统提供的操作类。

与数组的概念一样，对象数组也可以使用静态初始化的方式完成。

```
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 ObjectArrayDemo02 {
        public static void main(String args[]){
            Person per[] = {new Person("张三",10),
                new Person("李四",20),new Person("王五",30)} ;
            for(int x=0;x<per.length;x++){
                System.out.println(per[x].getInfo()) ;
            }
        }
};
```

内存关系图如下：




```
Person per[] = {new Person("张三",10),
        new Person("李四",20),new Person("王五",30)};
```



从实际意义上讲，Java 本身是很难画出内存关系图的，包括所谓的栈、读音堆之类的内存实际上都是引入了 C++的内存分配机制进行讲解的，Java 本身就一块内存。

# 4、总结

1、 面向对象的三大特性的概念必须理解；

2、 类与对象的关系，类的组成、对象的使用、引用传递、垃圾的产生；

3、 使用 private 可以对属性或方法进行封装，只要是属性就必须封装，封装之后的属性必须使用 setter 和 getter；

4、 当一个新对象实例化的时候要调用构造方法，构造方法名称与类名称相同，无返回值声明，而且一个类中至少保留一个构造方法，如果任何一个构造方法都没有定义的话，则会自动生成一个无参的，什么都不做的构造；

5、 String 类的各个特点及常用的方法（名称、参数的类型及含义、作用）；

6、 对象数组。

# 5、预习任务

面向对象基础部分，深入引用传递的使用、this 关键字、static 关键字、内部类。