类与对象

Java语言最大特点在于面向对象的编程设计，并且面向对象编程设计在于Java发展而不断扩大，很多不支持面向对象的语言也都转向了面向对象， 也有很多开发者认为使用函数编程也比较好。
并且面向对象已经以C语言为主的编程语言，面向过程指的面对于一个问题的解决方案。在更多情况下是不会做出重用的设计思考的，面向对象的主要设计模式位模块化设计。
并且可以重置配置，在整个面向对象设计里面更多情况下考虑的是标准。在使用的时候在逐一拼装。

面向对象的主要特征：
封装性：内部的操作，外部不可见；
继承性：在已有结构的基础上继续进行功能扩充；
多态性：类型的转换处理（是在基础性的基础上扩充来的概念）范围内可以变化的处理形式。

在进行面向对象的程序开发之中，一般还有三个步骤：
OOA：面向对象分析；
OOD：面向对象设计；
OOP：面向对象编程

2.类与对象简介
类是某一事物的共性抽象概念，面向对象描述的是一个具体的产物， 例如你和其他人进行比较。
就比如，每个人都是一个个体，但我们为什么没有认错，这无非是每个人都有一个个体。
和一个共有的标志：黄皮肤，骨骼等这些，人和人都是不同的，都有不同的属性和性格，每一个属性和集合就构成了一个对象，但是所有的属性都应该是群体的定义，而群体的定义就形成了一个类，比如，人类。
类是一个模板，而对象才是一个可以使用的实例。先有类，才有对象

在类之中一般会有两个属性：
成员属性（Field）：有些时候为了简化称其为属性；
一个人的年龄和性别都是不用的，所以这些整体来讲就称为属性
操作方法（method）：定义对象具有的处理行为；
这个人可以唱歌跳舞游泳；

4.类与对象的定义及使用
在Java语言之中，类是一个独立的结构体，所以需要使用class来进行定义，而在类之中，主要由属性和方法所组成。属性就是一个个具体的变量，方法就是一个重复还行的代码

范例：定义一个类
```
class person { // 定义一个类
   String name ; //人员的姓名
   int age ;  //人的年龄
   public void tell() {
      System.out.println("姓名:" + name + "、年龄：" + age) ;
}
```

在这个类之中定义有两个属性（name \ age ）和一个方法（tell()）

如果想使用类，就必须通过对象来调用，那么必须使用如下的语法格式完成：
声明并实例化对象：
类名称 对象名称 = new 类()
分步骤完成：

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()

当获取了实例化对象后，那么就需要通过对象进行类中的调用，此时有两种调用方式。
调用类中的属性：实例化对象.成员属性 ;
调用类中的方法：实例化对象.方法名称()

范例：使用对象操作类

```
class person { // 定义一个类
    String name ; //人员的姓名
    int age ;  //人的年龄
    public void tell() {
        System.out.println("姓名:" + name + "、年龄：" + age) ;
    }
}
public class JavaStudy {
    public static void main(String mage[]) {
        person per = new person() ; //生命并且实例化对象
        per.name = "张三" ;
        per.age = 18 ;
        per.tell() ; //进行方法的调用
    }
}
```

如果此时程序没有设置对象属性内容，则该数据内容为其对应数据类型的默认值。

```
class person { // 定义一个类
    String name ; //人员的姓名
    int age ;  //人的年龄
    public void tell() {
        System.out.println("姓名:" + name + "、年龄：" + age) ;
    }
}
public class JavaStudy {
    public static void main(String mage[]) {
        person per = new person() ; //生命并且实例化对象
        per.tell() ; //进行方法的调用
    }
}
```

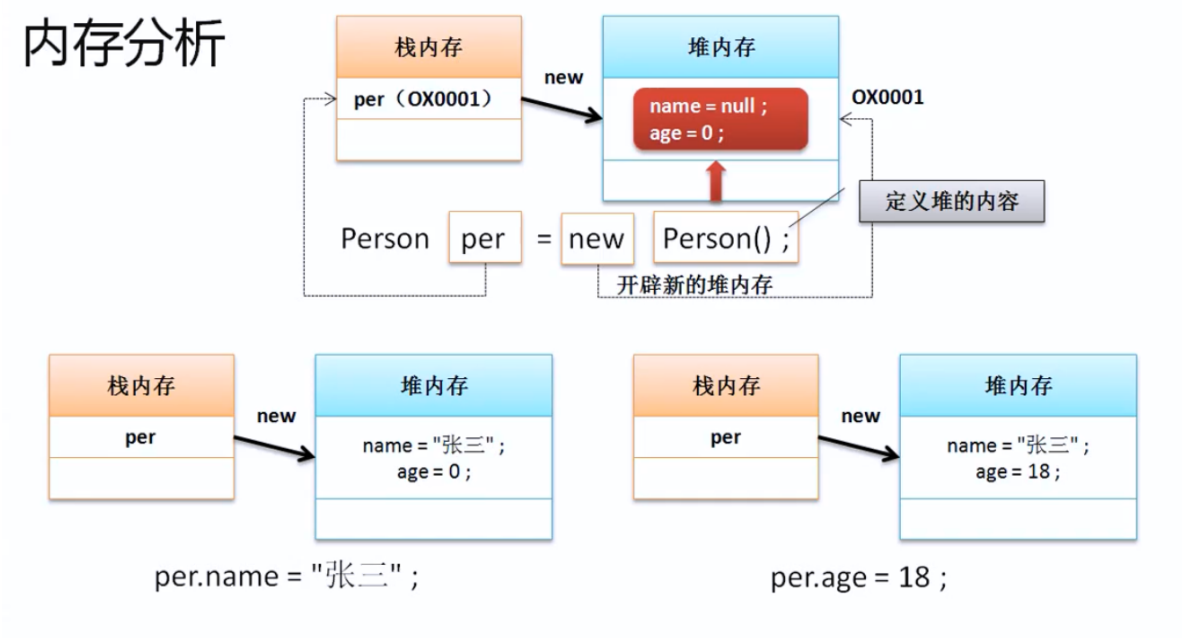
string是引用数据位null，而Int为基本类型，其默认值是0

\5. 对象内存初步分析
Java之中数据类型，应用数据类型最大的苦难之处在于要进行内存的管理，同时在进行操作的时候也会发生关系变化

范例：以下面程序为主作分析。

```
class person { // 定义一个类
    String name ; //人员的姓名
    int age ;  //人的年龄
    public void tell() {
        System.out.println("姓名:" + name + "、年龄：" + age) ;
    }
```

```
}
public class JavaStudy {
    public static void main(String mage[]) {
        person per = new person() ; //生命并且实例化对象
        per.tell() ; //进行方法的调用
    }
}
```

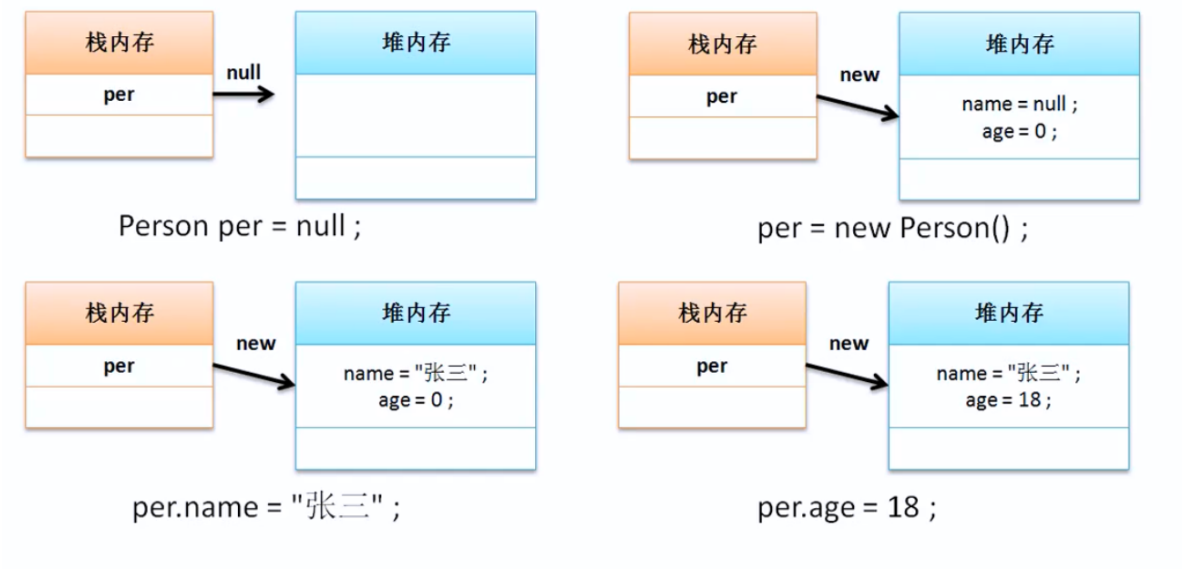


如果要进行内存分析，那么先给出两块最为常用的内存分析：
堆内存：保存的是对象具体信息
在程序之中，内存的开辟是通过nev完成的。（nev是开辟内存的最高指令）
栈内存：保存的是一块堆内存，（通过地址找到堆内存）
可以简单的理解为堆内存内容保存到栈内存里面

可以发现对象的实例化有两种方法，一种是之前使用到的声明并实例化对象，还有一种分步的内存操作进行分析

```
public class JavaStudy {
    public static void main(String mage[]) {
        person per = null ;
        per = new person() ;   //实例化对象
        per.name = "zhangsan" ;
        per.age = 18 ;
        per.tell() ;
    }
}
```

所有的对象在调用类中的属性或方法的时候必须要实例化完成后才可以执行



范例：错误的代码

```
public class JavaStudy {
   public static void main(String mage[]) {
      person per = null ;
      per.name = "zhangsan" ;
      per.age = 18 ;
      per.tell() ;
   }
}
```

代码之中只是声明了对象，但是并没有实例化，所以无法执行。

```
Exception in thread "main" java.lang.NullPointerException
      at JavaStudy.main(JavaStudy.java:11)
```

6.对象引用分析

类本身属于引用数据类型，既然是引用数据类型，那么将牵扯到内存的引用传递，

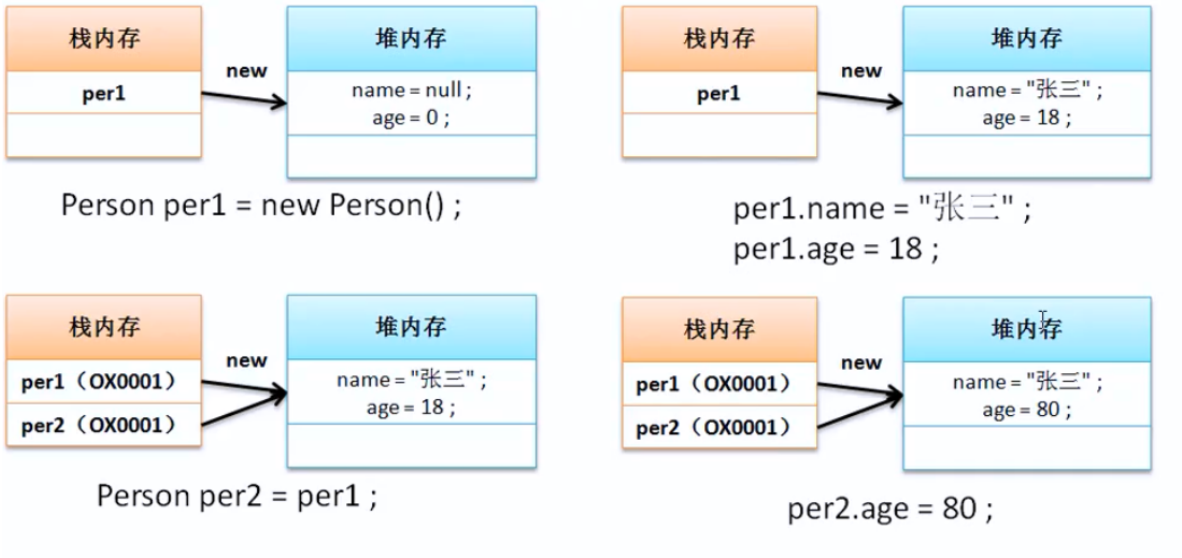
引用传递的本质：

同一块堆内存空间可以被不同的栈内存所指向，也可以更换指向

范例：定义一个引用传递的分析程序

```
public class JavaStudy {
   public static void main(String mage[]) {
      person per1 = new person() ; //声明并使并实例化对象
      per1.name = "张三" ;
      per1.age = 18 ;
      person per2 = per1 ;//引用传递，同类型接收同类型
      per2.age = 80 ;
      per1.tell () ; //进行方法的调用
   }
}
```

这个时候引用传递是直接在主方法之中定义的，也可以使用方法使用引用传递处理



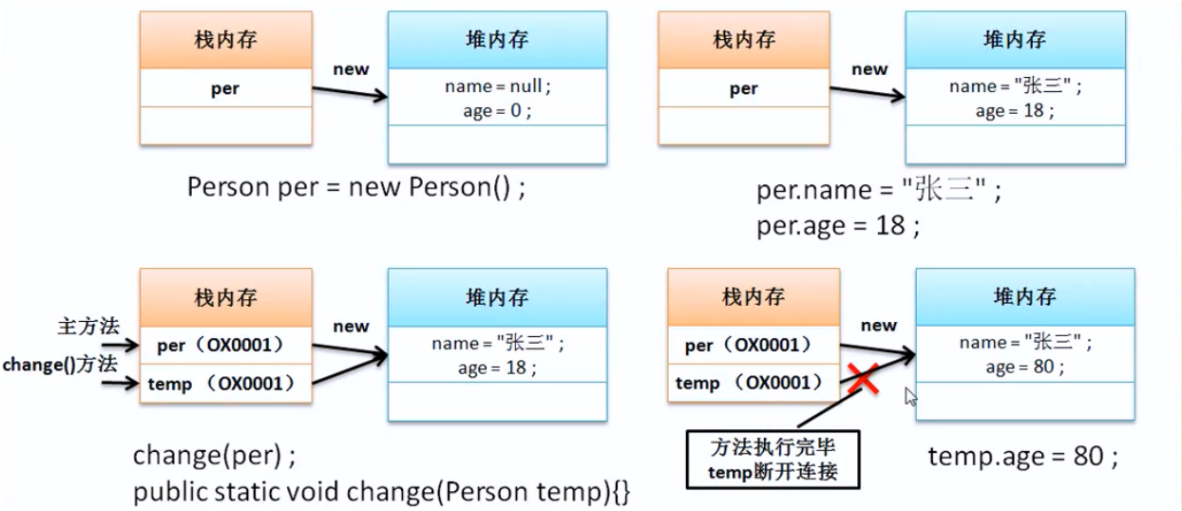
范例：利用方法实现引用传递处理

```
class person { // 定义一个类
   String name ; //人员的姓名
   int age ;  //人的年龄
   public void tell() {
      System.out.println("姓名:" + name + "、年龄：" + age) ;
   }
}
public class JavaStudy {
   public static void main(String mage[]) {
      person per = new person() ; //声名实例化对象
      per.name = "zhangsan" ;
      per.age = 18 ;
      change(per) ;
      per.tell() ;//进行方法调用
   }
   public static void change(person temp) {
      temp.age = 80 ;
   }
}
```

与之前的差别最大的地方在于，此时的程序是将person类的实例化对象（内存地址、数值）传递到了change() 方法之中，由于传递的是person了类型，那么change()方法接受的也是person类型

```
      change(per) ;   //等价于：Person temp = per ;
```

引用传递可以发生方法上，这个时候一定要观察方法的参数类型，同时也要观察方法的执行过程



7.引用与垃圾产生分析

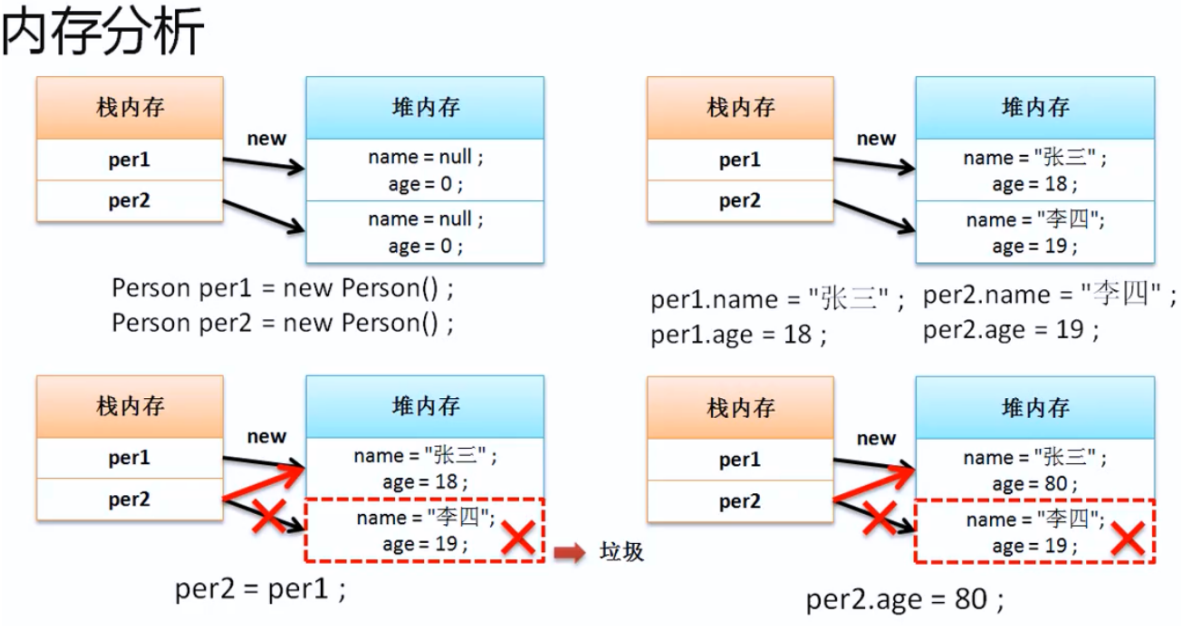
经过一系列的分析之后发现，所有的引用传递的本质就是一场堆内存的调戏游戏。但是引用传递在使用的过程中，如果使用不当，则会有垃圾产生。

范例：定义一段代码

```
class person { // 定义一个类
   String name ; //人员的姓名
   int age ;  //人的年龄
   public void tell() {
      System.out.println("姓名:" + name + "、年龄：" + age) ;
   }
}
public class JavaStudy {
   public static void main(String mage[]) {
      person per1 = new person() ; //声明并实例化对象
      person per2 = new person() ;
      per1.name = "zhangsan" ;
      per1.age = 18 ;
      per2.name = "lisi" ;
      per2.age = 19 ;
      per2 = per1 ; //引用传递
      per2.age = 80 ;
      per1.tell() ; //进行方法调用
   }
}
```

此时已经明确的发生了引用传递，并且也成功的完成了引用传递的处理操作，

观察一下内存的分配于处理



所谓的垃圾空间指的就是没有任何栈内存所指向的堆内存空间，所有的垃圾将会被GC（Garbage Collector 垃圾搜集机器）不定期进行回收并释放无用内存空间。但是垃圾过多则会降低程序的 处理性能。

一个栈内存只能够保存有一个堆内存的地址数据，如果发生更改，如果发生更改，则之前的地址数据将从此在栈内存中彻底消失