2、具体内容

Java语言最大的特点在于面向对象的编程设计，并且面向对象的编程设计也在由于Java自身的发展而不断发展。同时很多当初不支持面向对象的编程也都开始转向了面向对象，但是依然有许多的开发者认为面向过程会比较好，或者说使用函数式编程会比较好。

最早并且一直到现在流行的编程语言C、C++、Java。其中C语言已经变为了面向过程开发的代表，而像C++或者是Java都是面向对象的编程语言。

所谓的面向过程指的是面对于一个问题的解决方案，更多的情况下是不会做出重用的设计思考的；而面向对象的主要形式为模块化设计，并且可以进行重用配置。在整个的面向对象的设计里面更多情况下考虑的是标准，在使用的时候根据标准进行拼装，而对于面向对象设计有三个主要的特征：

·封装性：内部的操作对外部而言不可见；当内部的操作都不可直接使用的时候才是最安全的；

·继承性：在已有结构的基础上继续进行功能的扩充；例子：手机功能的发展

·多态性：是在继承性的基础上扩充而来的概念，指的是类型的转换处理。例：变性人

在进行面向对象程序的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；

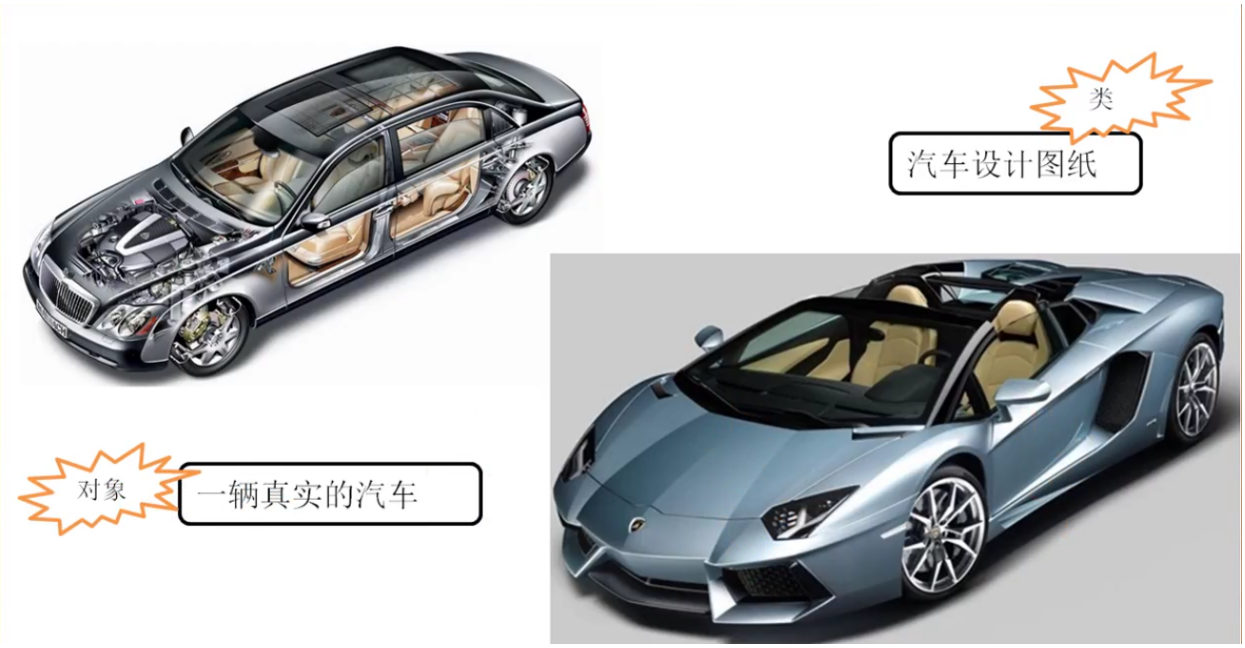
·OOP：面向对象编程；

面向对象是一个非常庞大的话题，但是任何庞大的话题都有其核心的组成：类与对象

■类与对象简简介

类是对某一类事物的共性的抽象概念，而对象描述的是一个具体的产物。例如：现在我和某一位先生进行比较的时候，你们可以立刻区分出我还有别人。

我和其他的人都一样，都是一个个具体可以使用的个体产物，但是这些个体都有一些共性的标志：中国人。人与人是不同的，所谓的人和人之间的不同依靠的是我们各自的属性，每一个属性的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。



类是一个模板，而对象才是类可以使用的实例，先有类再有对象。

在类之中一般都会有两个组成：

·成员属性(Field)：有些时候为了简化称其为属性；

|-一个人的年龄、姓名都是不同的，所以这些对于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、运动；

■类与对象的定义

在Java之中类时一个独立的结构体，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，那么属性就是一个个具体的变量，方法就是可以重复执行的代码。

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），现在有了类之后，如果想要使用类，那么就必须通过对象来完成，而如果要产生对象，那么必须使用如下的语

法格式来完成：

·声明并实例化对象：类名称 对象名 = new 类名称();

·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**之后，那么就需要通过对象进行类中的操作调用，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 JavaDemo{
        public static void main(String[] args) {
                Person per = new Person();   //声明并实例化对象
                per.name = "sanzhang";
                per.age = 18;
                per.tell(); //进行方法的调用
        }
}
```

姓名：sanzhang、年龄：18

如果此时的程序你并没有进行对象属性内容的设置，则该数据内容为其对应数据类型的默认值。

```
public class JavaDemo{
        public static void main(String[] args) {
                Person per = new Person();   //声明并实例化对象
                per.tell(); //进行方法的调用
        }
}
```

姓名：null、年龄：0

String是引用数据类型所以默认值为NULL，而int为基本类型，默认值为0.

■对象实例化操作初步分析

Java之中类属于引用数据类型，引用数据类型最大的困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内存关系的变化，所以本次针对之前的程序的内存关系进行一些简单分析。

范例：以下面的程序为主进行分析

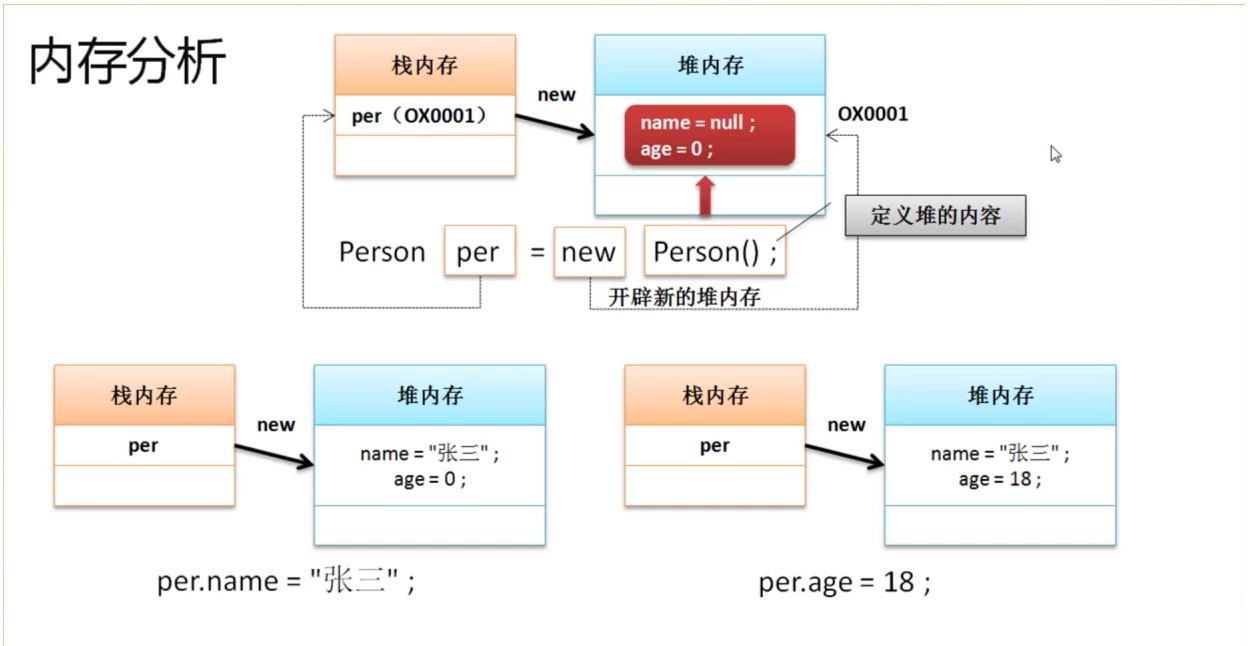
```
public class JavaDemo{
	public static void main(String[] args) {
		Person per = new Person();  //声明并实例化对象
		per.name = "sanzhang";
		per.age = 18;
		per.tell(); //进行方法的调用
	}
}
```

如果要进行内存分析，那么首先给出两块最为常用的内存空间：

·堆内存：保存的是对象的具体信息

·栈内存：保存的是一块堆内存的值，即：通过地址找到堆内存，而后找到对象内容；

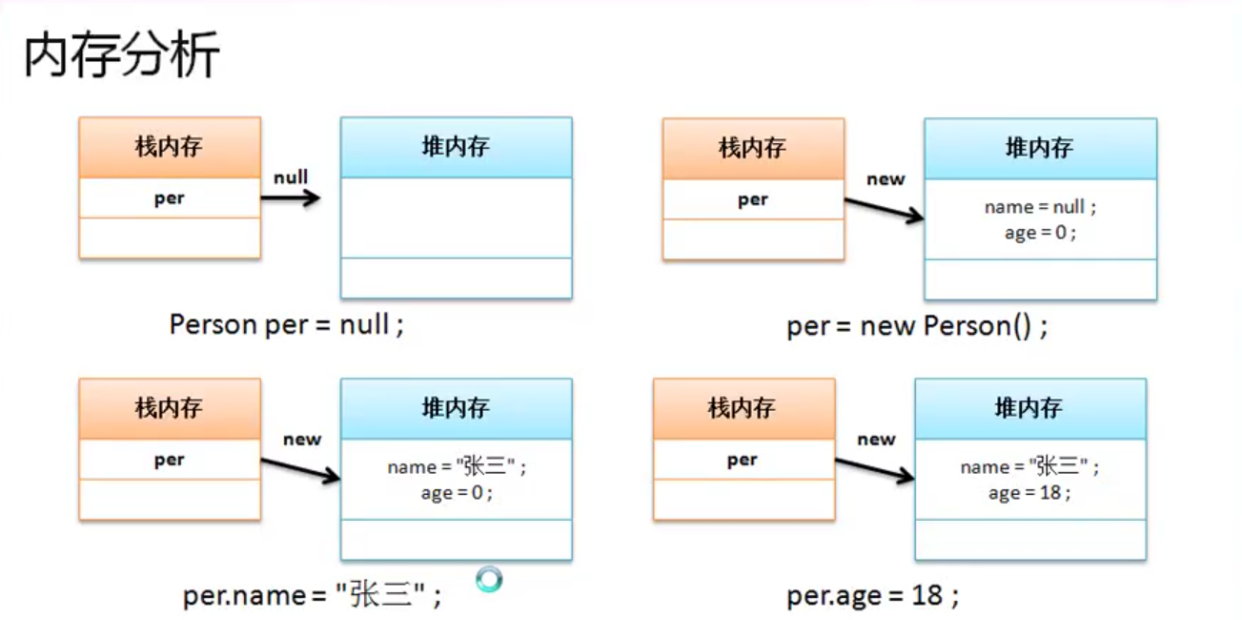
清楚了以上的对应关系之后，那么下面就针对之前的程序进行分析。



在之前进行分析的时候可以发现对象的实例化有两种用法，一种是之前使用的声明并实例化对象，另外一种就是分步完成，所以下面针对于分步的内存操作进行分析。

范例：定义程序代码

```
public class JavaDemo{
	public static void main(String[] args) {
		Person per = null;  //声明对象
			per = new Person();		//实例化对象
		per.name = "sanzhang";
		per.age = 18;
		per.tell(); //进行方法的调用
	}
}
```



需要特别引起注意的是，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：错误代码

```
public class JavaDemo{
	public static void main(String[] args) {
		Person per = null;  //声明对象
		per.name = "sanzhang";
		per.age = 18;
		per.tell(); //进行方法的调用
	}
}
```

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

at JavaDemo.main(**JavaDemo.java:11**)

代码之中只是声明了对象，但是并没有为对象进行实例化，所以此时无法调用。而此时程序中出现的NullPointerException(空指向异常)就是在没有堆内存开辟后时所产生的问题，并且只有引用数据类型存在有此问题。

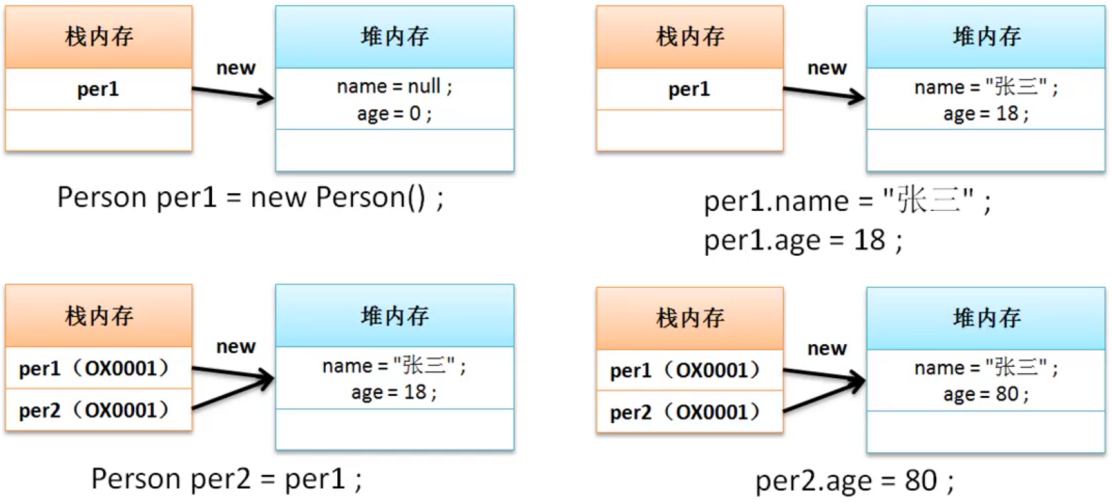
■引用传递分析

类本身属于引用数据类型，既然是引用数据类型，那么就牵引到内存的引用传递，所谓的引用传递的本质：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 JavaDemo{
	public static void main(String[] args) {
		Person per1 = new Person(); //声明并实例化对象
		per1.name = "sanzhang";
		per1.age = 18;
		Person per2 = per1;	//引用传递
		per2.age = 80;
		per1.tell();	//进行方法的调用
	}
}
```

内存分析

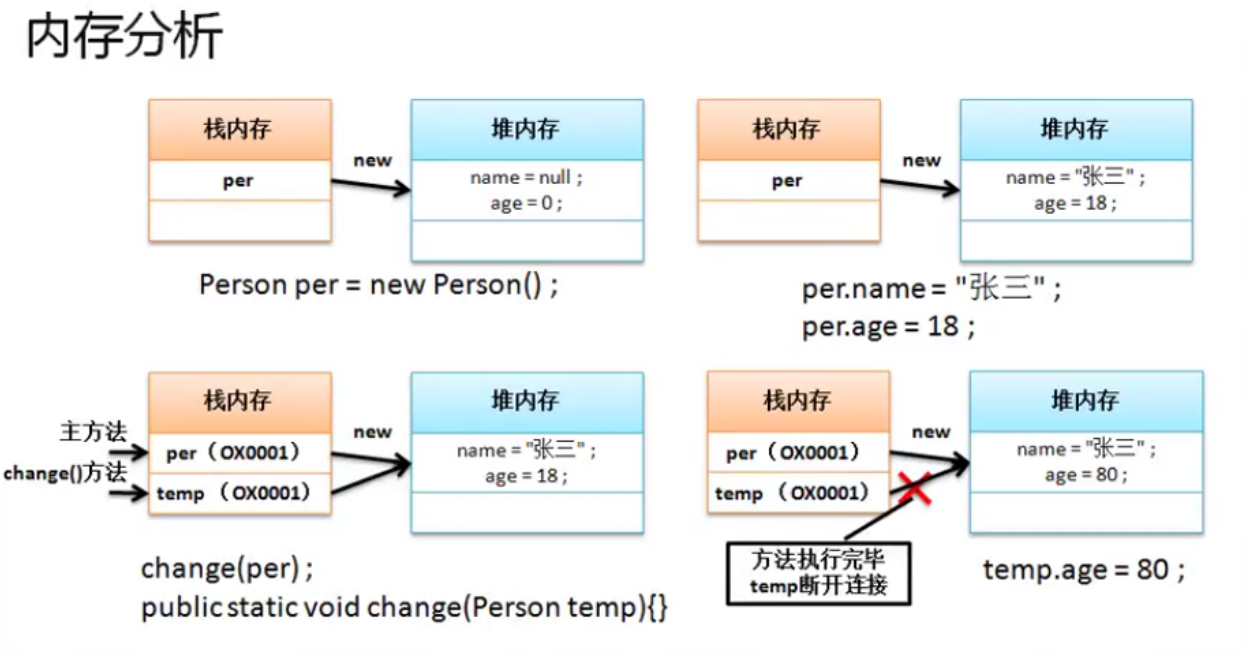


这个时候的引用传递就是直接在主方法之中定义的，也可以通过方法实现引用传递处理。

范例：利用方法实现引用传递处理

```
public class JavaDemo {
	public static void main(String args[]) {
		Person per = new Person() ;  // 声明并实例化对象
		per.name = "张三" ;
		per.age = 18 ;
		change(per) ;
		per.tell() ; // 进行方法的调用
	}
	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，由于传递的是一个Person类型，那么change()方法接收的也是Person类型。



引用传递可以发生在方法上，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。

■引用传递与垃圾产生分析

经过了一系列分析之后已经确认，所有的引用传递的本质就是一场堆内存的调用游戏。但是对于引用传递如果处理不当，那么也会造成垃圾的产生，那么本次将针对于垃圾产生原因进行简单分析。

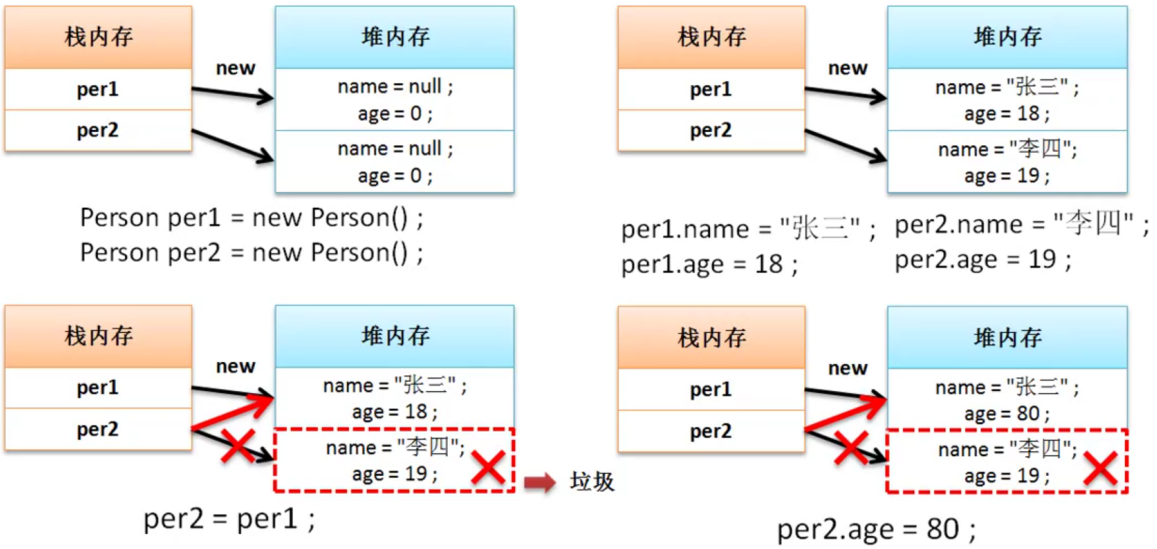
范例：定义一个要分析的程序

```
public class JavaDemo{
	public static void main(String[] args) {
		Person per1 = new Person(); //声明并实例化对象
		Person per2 = new Person();
		per1.name = "sanzhang";
		per1.age = 18;
		per2.name = "lisi";
		per2.age = 19;
		per2 = per1;  //引用传递
		per2.age = 80;
		per1.tell();	//进行方法的调用
	}
}
```

此时已经明确发生了引用传递，并且也成功的完成了引用传递的处理操作，但是下面观察一下其内存的分配与处理流程。

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

内存分析



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，所有的垃圾将被GC(GarbageCollector、垃圾收集器)不定期进行回收并且释放无用内存空间，但是如果垃圾过多，一定将影响到GC的处理性能，从而降低整体的程序性能，那么在实际的开发之中，对于垃圾的产生应该越少越好。