深入分析类与对象

1.成员属性封装处理
在类之中组成的方法就是属性与方法，一般而言方法都是对外提供服务的，所以是不会进行封装处理的，而相对于属性其较高的安全性，所以需要封装性进行保护。
在默认的情况下，在类中的属性是可以通过其他类利用对象进行调用的。

范例：属性不分装的情况下的问题
```
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 { //主类
    public static void main(String mage[]) {
        person per = new person() ; //声明并实例化对象
        per.name = "张三" ;  //在类外部修改属性
        per.age = 18 ; //在类外部修改属性
        per.tell() ; //进行方法调用
    }
}
```

此时在person类中提供的name与age两个属性未进行封装处理，这样外部就可以直接进行调用。

如果想要解决这样的问题，那就可以利用private关键字对属性进行封装处理

```
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 { //主类
    public static void main(String mage[]) {
        person per = new person() ; //声明并实例化对象
        per.name = "张三" ;  //在类外部修改属性
        per.age = -18 ; //在类外部修改属性
        per.tell() ; //进行方法调用
    }
}
```

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

```
    }
}
```

而属性一旦封装之后外部将不能够直接访问，外部不可见，但是对于内部是可见的。

如果想 让外部可见，则需要在Java开发标准中提供有以下要求：
（ setter、getter ）设置或取得属性可以使用：setXxx() 、getXxx()方法、以private String name为例
设置属性方法：public void setName(String n) ;
获取属性方法：public String getName()

范例：实现封装

```
class person { // 定义一个类
    private String name ; //人员的姓名
    private int age ;    //人的年龄
    public void tell() {
        System.out.println("姓名:" + name + "、年龄：" + 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 class JavaStudy { //主类
    public static void main(String mage[]) {
        person per = new person() ; //声明并实例化对象
        per.setName ("张三") ; //在类外部修改属性
        per.setAge (-18) ; //在类外部修改属性
        per.tell() ; //进行方法调用
    }
}
```

在以后进行任何类定义的时候一定要记住，类中的所有属性都必须使用private封装。（ 98% ）

属性如果要进行访问，必须要提供有setter 、getter方法。

2.构造方法与匿名对象
现在程序在使用类的时候一般都按你找了如下步骤进行：
声明并实例化对象，这个时候实例化对象中的属性并没有数据的存在，都是其对应数据的默认值
 需要用过一系列的setter方法为类中的属性设置内容
如果要想真正获得一个正常使用的实例化对象，必须经过两个步骤才可以实现

范例：传统调用

```
public class JavaStudy { //主类
    public static void main(String mage[]) {
        //1，对象的初始化准备
```

```
        person per = new person() ; //声明并实例化对象
        per.setName ("张三") ; //在类外部修改属性
        per.setAge (-18) ; //在类外部修改属性
        //2，对象的使用
        per.tell() ; //进行方法调用
    }
}
```

但是如果按照这样的方式进行思考就会发现一个问题，假设现在类中的属性很多个（8个）那么按照这个做法，此时需要调用 8次setter方法进行内容设置，所以在Java里面，为了考虑到了初始化的问题，专门提供了专有构造方法，

即：可以通过构造方法实现实例化对象中的属性初始化处理*（只有在关键字new的时候构造方法）

在Java程序里面构造方法定义如下：
构造方法必须与类名称保持一致；
构造方法不允许设置任何的返回值类型 。即：没有返回值定义
构造方法是是在使用关键字new实例化对象的时候自动调用的

范例：定义构造方法

```
public class JavaStudy { //主类
    public static void main(String mage[]) {
        //1，对象的初始化准备
        person per = new person ("张三",18) ; //在类外部修改属性
        //2，对象的使用
        per.tell() ; //进行方法调用
    }
}
```

针对当前的对象实例化格式与之前的对象实例化格式做一个比较：
之前的对象实例化格式 Person per = new person() ;
①Person ②per = ③new ④Person() ;
当前的对象实例化格式 Person per = new person("张三", 18) ;
①Person ②per = ③new ④Person("张三",18) ;
①Person:主要是定义对象的所属类型，类型决定了你调用的方法
②per =：实例化对象的名称，所有的操作通过对象来机械能访问;
③new：开辟一块新内容空格键
④Person("张三",18) ; 定义一个有参构造
| ④Person() 调用一个无参构造

在Java程序里面考虑到程序结构的完整性，所以所有的类都会提供有工造方法，也就是说你的类中没有构造方法，那么一定会提供一个无参的，什么都没有的不做的构造方法，这个构造方法是在程序编译的时候构造的，如果你已经在类中已经明确的定义方法，那么这个就不会被创建。

结论，一个类中至少存在有一个构造方法，永恒存在

疑问：为什么构造方法上不允许设置返回值类型？
既然构造方法是一个方法，那么为什么不让他定义返回值类型
呢？
既然构造方法不会返回数据，为什么不使用void呢？

```
public Person(String n,int a) {}
public void  Person(String n,int a) {}
```

分析：程序编译器是根据代码结构来进行编译处理的，执行的时候也是根据代码结构来处理的。

```
public Person(String n,int a) {}
public void  Person(String n,int a) {}
```

如果构造方法上使用了void，那么此结构与普通方法的结构完全相同。这样编译器会认为这个方法是普通方法

普通方法与构造方法最大区别在于，构造方法是在类对象实例化的时候调用的，而普通方法是在类对象实例化产生过后调用的。

既然构造方法本身是一个方法，那么方法就居用重载的忒但，而构造方法重载的时候只需要考虑参数的类型。

```
class person { // 定义一个类
   private String name ; //人员的姓名
   private int age ;    //人的年龄
   public person() {
      name = "无名氏" ;
      age = -1 ;
   }
   public person(String n ) {
      name = n ;
   }
   //方法名称与类名称相同，并无返回值定义
   public person(String n,int a) { //定义有参构造
      name = n ; //为类中的属性赋值（初始化）
      age = a ; //为类中的属性赋值(初始化)
   }
   public void tell() {
      System.out.println("姓名:" + name + "、年龄：" + age) ;
   }
}
public class JavaStudy { //主类
   public static void main(String mage[]) {
      //1，对象的初始化准备
      person per = new person() ;
      //2，对象的使用
      per.tell() ; //进行方法调用
   }
}
```

在进行多个构造方法的定义中，需要保持一种代码数据和习惯，好的代码结构，会提升整段代码的高端性

方法的确可以进行数据的设置，而对setter也可以进行数据的设置，构造方法是在对象是劣化的时候为属性设置内容的初始化内容，而setter不仅具有数据功能之外，也具有修改数据的功能。

范例：使用setter修改数据

```
public class JavaStudy { //主类
   public static void main(String mage[]) {
      //1，对象的初始化准备
      person per = new person("张三",10) ;
      //2，对象的使用
```

```
        per.setAge(18) ; //修改了属性内容
        per.tell() ; //进行方法调用
    }
}
```

利用构造方法可以传递属性数据，进一步分析 对象产生格式
定义对象的名称：类名称 对象名称 = null ;
实例化对象：对象名称 = new 类名称()

如果使用实例化对象进行操作也是可以的，而这种形式的对象，由于没有名字，所以叫匿名对象

范例：观察一个匿名对象

```
public class JavaStudy { //主类
    public static void main(String mage[]) {
        new person("张三",10).tell() ; //进行方法调用
    }
}
```

此方法通过了对象进行类中tell()方法的调用，但是由于此对象没有任何的引用名称，所以该对象使用一次之后就会成为垃圾，而所有的垃圾就会被GC所回收。

利用构造方法进行一次内存分析

范例：编写一个分析程序

```
class Message {
    private String title ;
    public Message(String t) {
        title = t ;
    }
    public String getTitle() {
        return title ;
    }
    public void setTirle(String t) { //具有修改功能
        title = t ;
    }
}

class person { // 定义一个类
    private String name ; //人员的姓名
    private int age ;    //人的年龄
    public person(Message msg,int a) {
        name = msg.getTitle() ;
        age = a ;
    }
    public Message getInfo() {
        return new Message (name + " : " + age) ;
    }
    public void tell() {
        System.out.println("姓名:" + name + "、年龄：" + age) ;
    }
}
public class JavaStudy { //主类
    public static void main(String mage[]) {
        Message msg = new Message("mldn") ;
        person per = new person(msg,20) ;
```

```
        msg = per.getInfo() ;
        System.out.println(msg.getTitle()) ;
    }
}
```

内存分析