**·构造方法与匿名对象**

现在程序在使用类的时候一般都按照了如下的步骤进行:

·声明并实例化对象，这个时候实例化对象中的属性并没有任何数据存在。

·需要通过一系列的setter方法为类中的属性赋值

等于现在想要获取一个正真可以使用的实例化对象，必须经过两个步骤

范例:传统调用

public class JavaDemo {//主类

public Static void main (String args[]){

//1.对象的初始化准备

Person per = new Person();

per.name = "张三" ; //在类外部修改属性

per.age = -18 ; //在类外部修改属性

//对象的使用

per.tell() ; //方法调用

}

}

但是如果按照这样的方式来进行思考的化就会出现问题:假设说现在类中的属性有很多个（8），那么按照我们之前的做法，就需要调用8次setter方法进行内容设置，这样的调用实在是太过于繁琐。所以java中为了解决这类问题,专门提供有构造方法，即: \*\*可以通过构造方法实现属性初始化处理。只有在关键字new的时候使用构造方法，在Java程序里面构造方法的定义要求如下

·构造方法名称必须与类名保持一致;

·构造方法不允许设置任何的返回值类型; 即: 没有返回定义；

·构造方法是在使用new时自动定义的;

eg:

class Person {

private String name ; //个人姓名

private int age ; //年龄

public void tell (){

System.out.println("姓名" + name + ",年龄" + age)

}

//构造方法

public Person(String n ,int a){ //有参构造

name = n;

age = a;

}

//以下省略属性的setter与getter方法

...

}

对比之前的实例化格式:

我们能够发现，在我们没有定于构造方法时 new Person（）我们依然调用无参构造方法。 由此我们可以得知，当我们不定义构造参数时，Java为了保证程序的完整性，所有的类都会提供构造方法，也就是说当我们不定义构造参数时，Java会自动为我们定义一个默认的

无参构造方法，这个构造方法时在程序编译的时候自动创建的。如果我们定义了构造方法，

那么这个默认的构造方法将不会自动创建

疑问: 为什么构造方法上不允许设置返回值类型？

public Person(String n ,int a){}

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

既然构造方法是一个方法，那为什么不让他定义返回类型。

既然构造方法不会返回数据为什么不使用void定义呢？

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。如果在构造方法上使用 void 那么此结构就与普通方法的构造完全相同了，这样编译器会认为他是一个普通方法。

·普通方法与构造方法最大的区别: 构造方法是在类实例化时候调用

的，而普通方法实在类对象实例化产生之后调用的。既然构造方法 本身是一个方法，那么方法就具有重载的特点。

范例: （构造参数重载）

class Person {

private String name ; //个人姓名

private int age ; //年龄

//构造方法重载

public Person(){}

public void tell (){

System.out.println("姓名" + name + ",年龄" + age)

}

//构造方法

public Person(String n ,int a){ //有参构造

name = n;

age = a;

}

//以下省略属性的setter与getter方法

...

}

在进行多个构造方法定义的时候强烈建议大家有一些定义的顺序

例如:可以按照参数个数定义顺序

经过分析之后可以发现，利用构造方法可以传递属性数据，于是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()。

如果这个时候我们只是通过实例化对象来进行类的操作也是可以的，而这种形式的对象由于没有名字，我们称之为 匿名对象

范例: (观察匿名对象)

public class JavaDemo {

public static void main (String args[]){

new Person("张三" , 10).tell();

}

}

此时依然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，所以该对象在使用一次之后就将成为垃圾，而所有的垃圾将被 GC 进行回收与释放