**java构造方法**

2010-04-17 17:47:24|  分类： [java](http://blog.163.com/bt_q/blog/#m=0&t=1&c=fks_087069082084088071081083086095085087088065082080081065093)|举报|字号 订阅

**重载构造方法 、默认构造方法、子类调用父类的构造方法、构造方法的作用域、构造方法的访问级别**

在多数情况下，初始化一个对象的最终步骤是去调用这个对象的构造方法。构造方法负责对象的初始化工作，为实例变量赋予合适的初始值。构造方法必须满足以下语法规则：

(1) 方法名必须与类名相同。

(2)不要声明返回类型。

(3)不能被static、final、synchronized、abstract和native修饰。构造方法不能被子类继承，所以用final和

abstract修饰没有意义。构造方法用于初始化一个新建的对象，所以用static修饰没有意义。多个线程不会同时创建内存地址相同的同一个对象，因此用synchronized修饰没有必要。此外，Java语言不支持native类型的构造方法。

在以下Sample类中，具有int返回类型的Sample(int x)方法只是个普通的实例方法，不能作为构造方法：

public class Sample {

private int x;

public Sample() { // 不带参数的构造方法

this(1);

}

public Sample(int x) { //带参数的构造方法

this.x=x;

}

public int Sample(int x) { //不是构造方法

return x++;

}

}

以上例子尽管能编译通过，但是把实例方法和构造方法同名，不是好的编程习惯，容易引起混淆。例如以下Mystery类的Mystery()方法有void返回类型，因此是普通的实例方法：

public class Mystery {

private String s;

public void Mystery() { //不是构造方法

s = "constructor";

}

void go() {

System.out.println(s);

}

public static void main(String[] args) {

Mystery m = new Mystery();

m.go();

}

}

以上程序的打印结果为null。因为用new语句创建Mystery实例时，调用的是Mystery类的默认构造方法，而不是以上有void返回类型的Mystery()方法。

**重载构造方法**

当通过new语句创建一个对象时，在不同的条件下，对象可能会有不同的初始化行为。例如对于公司新进来的一个雇员，在一开始的时候，有可能他的姓名和年龄是未知的，也有可能仅仅他的姓名是已知的，也有可能姓名和年龄都是已知的。如果姓名是未知的，就暂且把姓名设为"无名氏"，如果年龄是未知的，就暂且把年龄设为-1。

可通过重载构造方法来表达对象的多种初始化行为。以下例程的Employee类的构造方法有三种重载形式。在一个类的多个构造方法中，可能会出现一些重复操作。为了提高代码的可重用性，Java语言允许在一个构造方法中，用this语句来调用另一个构造方法。

例程 Employee.java

public class Employee {

private String name;

private int age;

/\*\* 当雇员的姓名和年龄都已知，就调用此构造方法 \*/

public Employee(String name, int age) {

this.name = name;

this.age=age;

}

/\*\* 当雇员的姓名已知而年龄未知，就调用此构造方法 \*/

public Employee(String name) {

this(name, -1);

}

/\*\* 当雇员的姓名和年龄都未知，就调用此构造方法 \*/

public Employee() {

this( "无名氏" );

}

public void setName(String name){this.name=name; }

public String getName(){return name; }

public void setAge(int age){this.age=age;}

public int getAge(){return age;}

}

以下程序分别通过三个构造方法创建了三个Employee对象：

Employee zhangsan=new Employee("张三",25);

Employee lisi=new Employee("李四");

Employee someone=new Employee();

在Employee(String name)构造方法中，this(name,-1)语句用于调用Employee(String name,int age)构造方法。在Employee()构造方法中，this("无名氏")语句用于调用Employee(String name)构造方法。

用this语句来调用其他构造方法时，必须遵守以下语法规则：

（1）假如在一个构造方法中使用了this语句，那么它必须作为构造方法的第一条语句（不考虑注释语句）。

以下构造方法是非法的：

public Employee(){

String name="无名氏";

this(name); //编译错误，this语句必须作为第一条语句

}

（2）只能在一个构造方法中用this语句来调用类的其他构造方法，而不能在实例方法中用this语句来调用类的其他构造方法。

（3）只能用this语句来调用其他构造方法，而不能通过方法名来直接调用构造方法。

以下对构造方法的调用方式是非法的：

public Employee() {

String name= "无名氏";

Employee(name); //编译错误，不能通过方法名来直接调用构造方法

}

**默认构造方法**

默认构造方法是没有参数的构造方法，可分为两种：（1）隐含的默认构造方法（2）程序显式定义的默认构造方法。

在Java语言中，每个类至少有一个构造方法。为了保证这一点，如果用户定义的类中没有提供任何构造方法，那么Java语言将自动提供一个隐含的默认构造方法。该构造方法没有参数，用public 修饰，而且方法体为空，格式如下：

public ClassName(){} //隐含的默认构造方法

在程序中也可以显式的定义默认构造方法，它可以是任意的访问级别。例如：

protected Employee() { //程序显式定义的默认构造方法

this("无名氏");

}

如果类中显式定义了一个或多个构造方法，并且所有的构造方法都带参数，那么这个类就失去了默认构造方法。在以下程序中，Sample1类有一个隐含的默认构造方法，Sample2类没有默认构造方法，Sample3类有一个显式定义的默认构造方法：

public class Sample1{}

public class Sample2{

public Sample2(int a){System.out.println("My Constructor");}

}

public class Sample3{

public Sample3(){System.out.println("My Default Constructor");}

}

可以调用Sample1类的默认构造方法来创建Sample1对象：

Sample1 s=new Sample1(); //合法

Sample2类没有默认构造方法，因此以下语句会导致编译错误：

Sample2 s=new Sample2(); //编译出错

Sample3类显式定义了默认构造方法，因此以下语句是合法的。

Sample3 s=new Sample3();

**子类调用父类的构造方法**

父类的构造方法不能被子类继承。以下MyException类继承了java.lang.Exception类：

public class MyException extends Exception{} // MyException类只有一个隐含的默认构造方法

尽管在Exception类中定义了如下形式的构造方法：

public Exception(String msg)

但MyException类不会继承以上Exception类的构造方法，因此以下代码是不合法的：

//编译出错，MyException类不存在这样的构造方法

Exception e=new MyException("Something is error");

在子类的构造方法中，可以通过super语句调用父类的构造方法。例如：

public class MyException extends Exception{

public MyException(){

//调用Exception父类的Exception(String msg)构造方法

super("Something is error");

}

public MyException(String msg){

//调用Exception父类的Exception(String msg)构造方法

super(msg);

}

}

用super语句来调用父类的构造方法时，必须遵守以下语法规则。

（1）在子类的构造方法中，不能直接通过父类方法名调用父类的构造方法，而是要使用super语句。

以下代码是非法的：

public MyException(String msg){

Exception(msg); //编译错误

}

（2）使用super语句时，必须放在最前面。

以下代码是非法的：

public MyException(){

String msg= "Something wrong";

super(msg); //编译错误，super语句必须作为构造方法的第一条语句

}

在创建子类的对象时，Java虚拟机首先执行父类的构造方法，然后再执行子类的构造方法。在多级继承的情况下，将从继承树的最上层的父类开始，依次执行各个类的构造方法，这可以保证子类对象从所有直接或间接父类中继承的实例变量都被正确的初始化。例如以下Base父类和Sub子类分别有一个实例变量 a和b，当构造Sub实例时，这两个实例变量都会被初始化。

public class Base{

private int a;

public Base(int a){ this.a=a;}

public int getA(){return a;}

}

public class Sub extends Base{

private int b;

public Base(int a,int b){super(a); this.b=b;}

public int getB(){return b;}

public static void main(String args[]){

Sub sub=new Sub(1,2);

System.out.println("a="+sub.getA()+" b="+sub.getB()); //打印a=1 b=2

}

}

在以下例程（Son.java）中，Son类继承Father类，Father类继承Grandpa类。这三个类都显式定义了默认的构造方法，此外还定义了一个带参数的构造方法。

例程Son.java

class Grandpa{

protected Grandpa(){

System.out.println("default Grandpa");

}

public Grandpa(String name){

System.out.println(name);

}

}

class Father extends Grandpa{

protected Father(){

System.out.println("default Father");

}

public Father(String grandpaName,String fatherName){

super(grandpaName);

System.out.println(fatherName);

}

}

public class Son extends Father{

public Son(){

System.out.println("default Son");

}

public Son(String grandpaName,String fatherName,String sonName){

super(grandpaName,fatherName);

System.out.println(sonName);

}

public static void main(String args[]){

Son s1= new Son("My Grandpa", "My Father", "My Son"); //①

Son s2=new Son(); //②

}

}

执行以上main()方法的第①条语句，打印结果如下：

My Grandpa

My Father

My Son

当子类的构造方法没有用super语句显式调用父类的构造方法，那么通过这个构造方法创建子类对象时，Java虚拟机会自动先调用父类的默认构造方法。执行以上Son类的main()方法的第②条语句，打印结果如下：

default Grandpa

default Father

default Son

当子类的构造方法没有用super语句显式调用父类的构造方法，而父类又没有提供默认构造方法时，将会出现编译错误。例如把例程Son.java做适当修改，删除Grandpa类中显式定义的默认构造方法：

// protected Grandpa(){

// System.out.println("default GrandPa");

// }

这样，Grandpa类就失去了默认构造方法，这时，在编译Father类的默认构造方法时，因为找不到Grandpa类的默认构造方法而编译出错。如果把Grandpa类的默认构造方法的protected访问级别改为private访问级别，也会导致编译错误，因为Father类的默认构造方法无法访问Grandpa类的私有默认构造方法。

在以下例子中，子类Sub的默认构造方法没有通过super语句调用父类的构造方法，而是通过this语句调用了自身的另一个构造方法Sub(int i)，而在Sub(int i)中通过super语句调用了父类Base的Base(int i)构造方法。这样，无论通过Sub类的哪个构造方法来创建Sub实例，都会先调用父类Base的Base(int i)构造方法。

class Base{

Base(int i){System.out.println("call Base(int i)");}

}

public class Sub extends Base{

Sub(){this(0); System.out.println("call Sub()");}

Sub(int i){super(i); System.out.println("call Sub(int i)");}

public static void main(String args[]){

Sub sub=new Sub();

}

}

执行以上Sub类的main()方法的new Sub()语句，打印结果如下：

call Base(int i)

call Sub(int i)

call Sub()

在下面的例子中，Base类中没有定义任何构造方法，它实际上有一个隐含的默认构造方法：

Base(){}

Sub类的Sub(int i)构造方法没有用super语句显式调用父类的构造方法，因此当创建Sub实例时，会先调用Base父类的隐含默认构造方法。

class Base{} //具有隐含默认构造方法

public class Sub extends Base{

Sub(int i){System.out.println(i);}

public static void main(String args[]){

System.out.println(new Sub(1)); //打印1

}

}

**构造方法的作用域**

构造方法只能通过以下方式被调用：

(1) 当前类的其他构造方法通过this语句调用它。

(2) 当前类的子类的构造方法通过super语句调用它。

(3)在程序中通过new语句调用它。

对于例程（Sub.java）的代码，请读者自己分析某些语句编译出错的原因。

例程 Sub.java

class Base{

public Base(int i,int j){}

public Base(int i){

this(i,0); //合法

Base(i,0); //编译出错

}

}

class Sub extends Base{

public Sub(int i,int j){

super(i,0); //合法

}

void method1(int i,int j){

this(i,j); //编译出错

Sub(i,j); //编译出错

}

void method2(int i,int j){

super(i,j); //编译出错

}

void method3(int i,int j){

Base s=new Base(0,0); //合法

s.Base(0,0); //编译出错

}

}

**构造方法的访问级别**

构造方法可以处于public、protected、默认和private这四种访问级别之一。本节着重介绍构造方法处于private级别的意义。

当构造方法为private级别，意味着只能在当前类中访问它：在当前类的其他构造方法中可以通过this语句调用它，此外还可以在当前类的成员方法中通过new语句调用它。

在以下场合之一，可以把类的所有构造方法都声明为private类型。

（1）在这个类中仅仅包含了一些供其他程序调用的静态方法，没有任何实例方法。其他程序无需创建该类的实例，就能访问类的静态方法。例如 java.lang.Math类就符合这种情况，在Math类中提供了一系列用于数学运算的公共静态方法，为了禁止外部程序创建Math类的实例， Math类的惟一的构造方法是private类型的：

private Math(){}

abstract类型的类也不允许实例化。也许你会问，把Math类定义为如下abstract类型，不是也能禁止Math类被实例化吗？

public abstract class Math{…}

如果一个类是抽象类，意味着它是专门用于被继承的类，可以拥有子类，而且可以创建具体子类的实例。而JDK并不希望用户创建Math类的子类，在这种情况下，把类的构造方法定义为private类型更合适。

（2）禁止这个类被继承。当一个类的所有构造方法都是private类型，假如定义了它的子类，那么子类的构造方法无法调用父类的任何构造方法，因此会导致编译错误。把一个类声明为final类型，也能禁止这个类被继承。这两者的区别是：

1)如果一个类允许其他程序用new语句构造它的实例，但不允许拥有子类，那就把类声明为final类型。

2)如果一个类既不允许其他程序用new语句构造它的实例，又不允许拥有子类，那就把类的所有构造方法声明为private类型。

由于大多数类都允许其他程序用new语句构造它的实例，因此用final修饰符来禁止类被继承的做法更常见。

（3）这个类需要把构造自身实例的细节封装起来，不允许其他程序通过new语句创建这个类的实例，这个类向其他程序提供了获得自身实例的静态方法，这种方法称为静态工厂方法。

http://blog.163.com/bt\_q/blog/static/119266729201031754724496/