# Android基本功三Java的类

搜索关键字：

内部类

匿名类

匿名对象

回调方法(回调函数)

内存回收机制

## 内部类知识点简介

【知识点】Android SDK与Java SDK之间的关系。

首先介绍一下什么叫作SDK。SDK通常指：[软件开发工具包](http://baike.baidu.com/view/592963.htm" \t "_blank)（[外语](http://baike.baidu.com/view/127315.htm)全称：Software Development Kit）一般都是一些被[软件工程师](http://baike.baidu.com/view/45743.htm" \t "_blank)用于为特定的[软件包](http://baike.baidu.com/view/600107.htm)、[软件框架](http://baike.baidu.com/view/10434053.htm)、硬件平台、[操作系统](http://baike.baidu.com/subview/880/4940471.htm)等建立[应用软件](http://baike.baidu.com/view/7886.htm)的[开发工具](http://baike.baidu.com/view/1355803.htm)的集合。

SDK通常是为了开发某一个方面的程序软件，由厂商提供的集成封装的库（library），通常比较底层，通用性强。例如，Windows的API也可以看作是一个SDK。如：Java Develop Toolkit，就是针对JAVA语言的SDK。

Android 虽然使用Java 语言作为开发工具 ，但是在实际开发中发现，还是与Java SDK 有一些不同的地方。Android SDK引用了大部分的Java SDK，少数部分被Android SDK抛弃，比如说界面部分，java.awt package除了java.awt.font被引用外，其他都被抛弃，在Android平台开发中不能使用。Android  SDK与Java SDK的具体细节区别，有兴趣的读者可查阅搜索引擎，在此就不详细展开了。

本篇并没有按照“编年体”的方式来讲解JAVA语言，因为这里并不是定位专门讲解JAVA语言的教程，而是假设读者有一定的编程基础（如C++），然后选取了JAVA相比面向对象语言比较特殊的地方，如“内部类”、“事件监听”、“多线程”和“异常处理”等方面进行讲解。

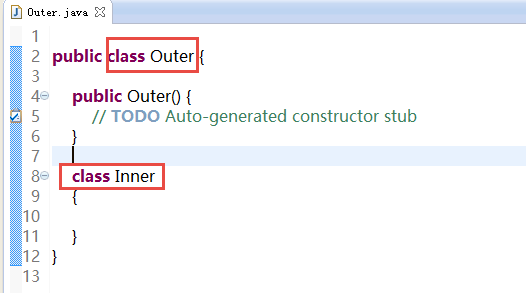
本章主要讲解JAVA中类之间的关系。

### 1.1内部类是什么

在一个类的内部定义的类称为内部类(或嵌套类)，包含内部类的类称为外部类。如Outer是外部类，Inner是内部类。内部类与外部类的称呼是相对的。

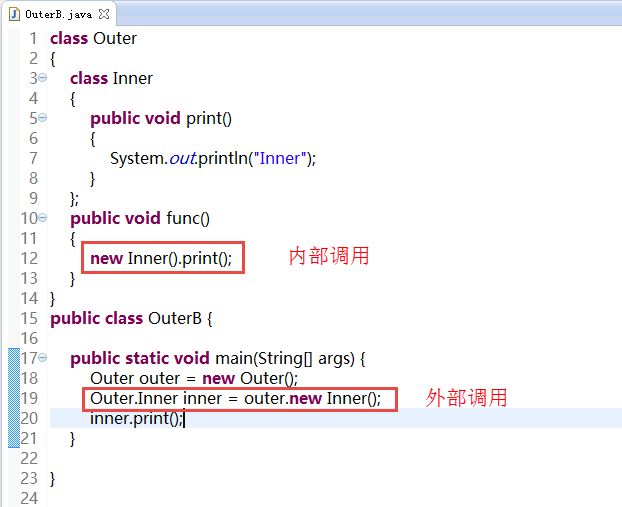
如图4A-1所示，Outer是一个外部类，Inner是一个内部类。

图 4A-1 外部类与内部类



由图4A-1可以看出，内部类是依赖于外部类而存在的，如果需要在外部类之外定义内部类的对象，则应使用 外部类名.内部类名 的格式来表示内部类。如图4A-2所示：在外部类Outer外面使用内部类Inner对象，必须以 Outer.Inner 表示内部类，假若是在内部类Outer里面使用外部类Inner对象，则不必写上外部类名和点号(.)。

图4A-2 内部类的内部调用与外部调用



### 1.2 内部类的种类

|  |  |
| --- | --- |
| 类型 | 说明 |
| 成员类 | 作为类的成员而存在在某一类的内部 |
| 内部类 | 存在于某一方法内的类 |
| 静态类 | 作为类的静态成员存在于某一类的内部，用关键字static修饰 |
| 匿名内部类 | 存在于某一类的内部，但无名称的类 |

#### 1.2.1 成员内部类（成员类）

成员类是作为外部类的成员而存在的，也就是说，成员类与外部类的成员变量和成员方法的地位是一样的。成员类是最常见的内部类，如图4A-1所示。

【注意】在内部类中可加上private、public、protected修饰符。如图4A-3所示。

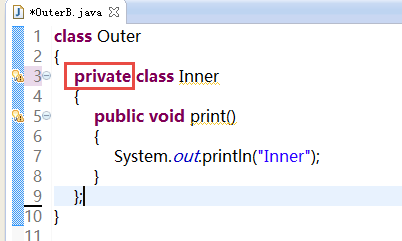


图4A-3内部类可以加上权限修饰符

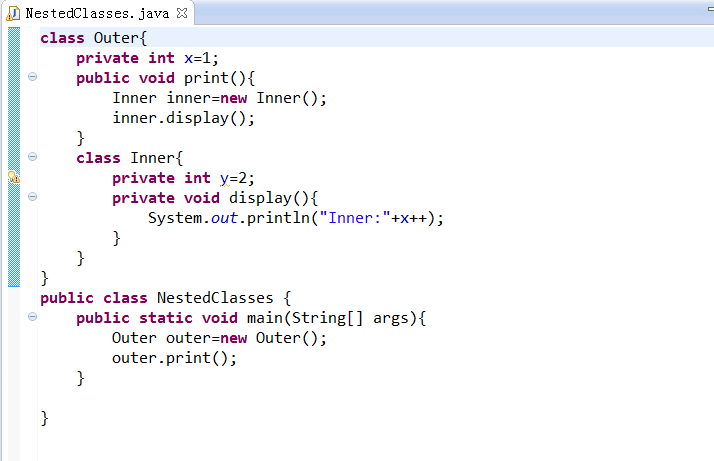
#### 1. 成员类案例01

案例设计（成员类）：

外部类Outer中包含一个内部类Inner,在主函数中创建Outer 的对象 outer,对象outer调用Outer的print()方法，在外部类Outer中又包含了一个内部类Inner。在外部类中的print()方法中我们创建了内部类Inner的对象inner，并且调用了其方法display()。

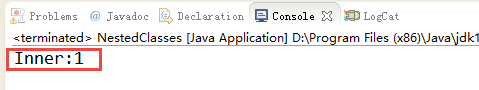
案例实现，如图4A-4所示。

图4A-4 在外部类调用内部类的方法



输出，如图4A-5所示：

图4A-5 输出结果



案例注意的问题：

1）访问权限的规则

1：内部类可以**直接访问外部类的成员**(包括私有成员)。

2：外部类**不可以直接访问内部类成员**(因为内部类的成员只有在内部类范围内是可见的),可以通过先创建外部类实例，再创建内部类实例来调用内部类的变量和方法。

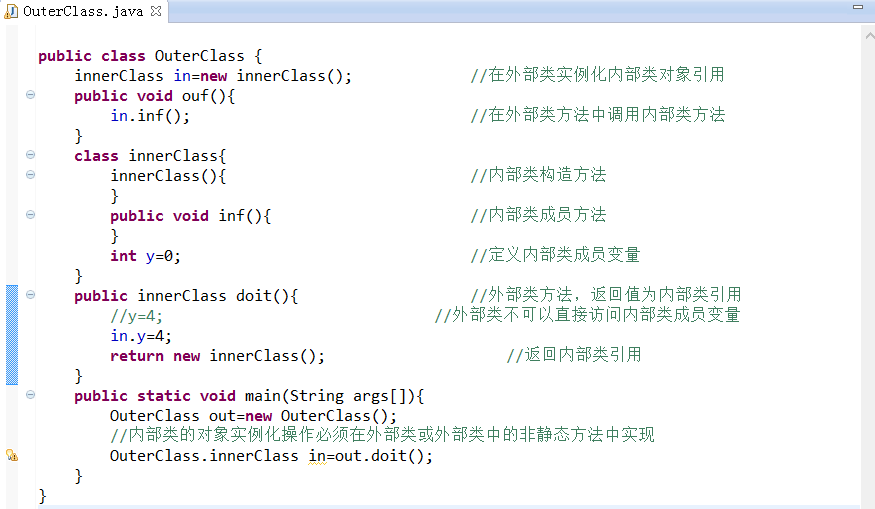
2）内部类的适用场合：一个类A的程序代码要用到另一个类B的实例，而另一个类B的程序代码又要访问第一个类A的成员，将另一个类B做成第一个类的内部类，程序代码的编写就要容易得多。这种情况在实际应用中很多(如事件处理)。

3）如果内部类与外部类的变量同名，则可以在内部类中采用如下格式进行区别：this.变量名 -> 内部类的变量、外部类名.this.变量名 -> 外部类的变量。

#### 2. 成员类案例02

案例实现：如图4A-6所示

图4A-6 在外部类调用内部类方法



案例注意的问题：

此例子的外部类创建内部类实例时与其他类创建对象引用时相同。内部类可以访问它的外部类的成员，但内部类的成员只有在内部类的范围之内是可知的，不能被外部类使用。如果将内部类的成员变量y再次赋值时将会出错，但是如果使用内部类对象引用调用成员变量y即可。

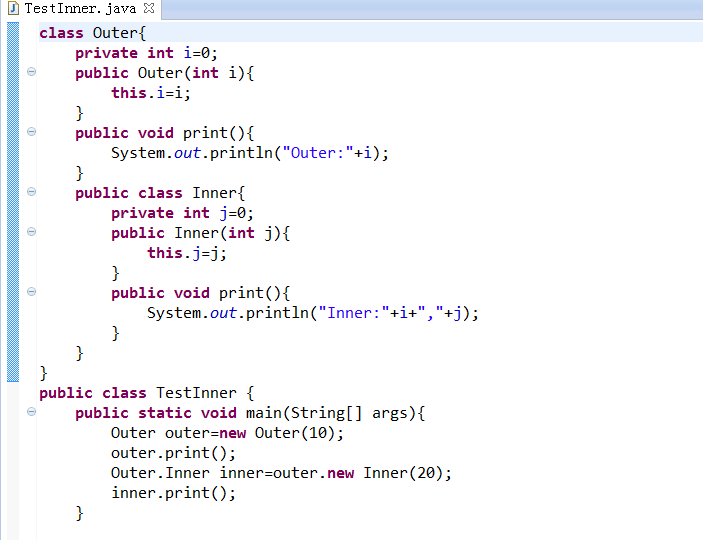
#### 3. 成员类案例03

案例设计：

主函数中创建Outer外部类的对象outer并传入形参10调用了其构造方法进行初始化，继而通过对象outer调用了其print()方法，输出属性i的值.通过类似的方法我们可以利用外部类Outer的对象outer创建出内部类Inner的对象inner，利用inner对象调用其内部类所特有的print()方法。

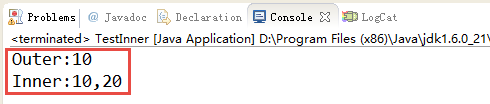
案例实现，如图4A-7所示：

图4A-7 外部类分别调用自己以及内部类的方法



输出结果，如图4A-8所示：

图4A-8 输出结果



案例注意的问题：

必须首先创建外围对象然后才可用它创建内部对象。

1. 静态成员类的优点在于 ：允许在成员类中声明静态成员。

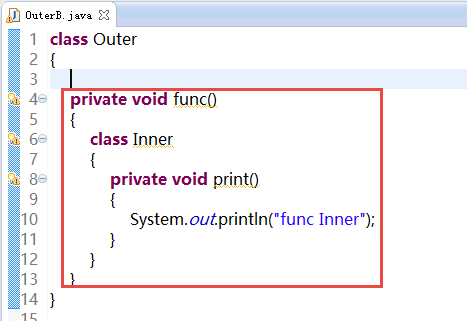
其特点如下：

1. 实例化静态类时，在new前面不需要用对象变量。
2. 静态类中只能访问其外部类的静态成员。
3. 静态方法中不能不带前缀地实例化一个非静态类。

#### 1.2.2 局部内部类(简称局部类)

局部类是包含在外部类的**某一方法**中的内部类，其作用域是在局限于该方法的范围，地位与该方法中的局部变量一样。局部类只有在方法内部才能创建对象，一旦方法执行完毕，它就会释放内存而消亡。如图4A-9所示。

图4A-9 局部内部类



1）局部类可直接访问外部类成员，但是对象的创建是在方法内进行的。

2）局部类只访问用final修饰的局部变量和形参。原因是：局部变量会随着方法的退出而消亡，通过将其定义为final变量，可以扩展其生命周期，可与访问其类实例的生命期相配合。因为类实例的生命期是由内存的回收机制决定的。

1. 局部类的作用域仅限于其直接外围块。因而局部类不可使用访问控制修饰符public 、protected 、private。

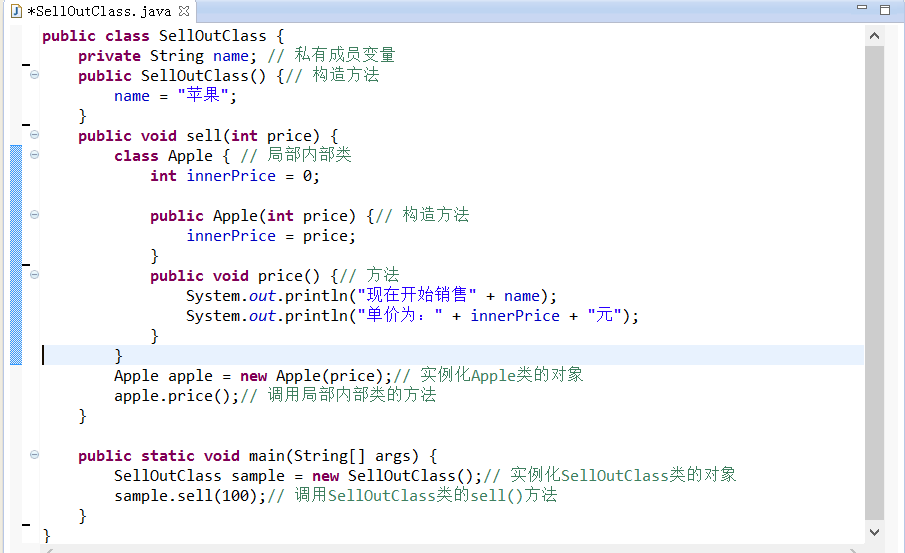
#### 1. 局部内部类案例01

案例设计：

在外部类的sell方法中创建Apple局部内部类，然后创建该内部类的实 例，并调用其定义的price()方法输出单价信息。

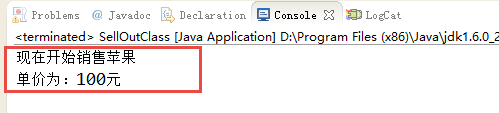
案例实现，如图4A-10所示：

图4A-10 调用局部内部类



输出，如图4A-11所示：

图4A-11 输出结果



案例注意的问题：

将内部类定义在sell()方法内部。但是有一点值得注意，内部类Apple是sell()方法的一部分，并非SellOutClass类的一部分，所以在sell()方法的外部不能访问该内部类，但是该内部类可以访问当前代码块的常量以及此外部类的所有成员。

#### 1.2.3 静态内部类(简称嵌套类)

静态类是最简单的内部类形式，与其他静态变量一样，静态内部类只需要在定义的时候加上 static 关键字，同样，静态内部类只能访问外部类中的静态成员变量。如图4A-12所示。

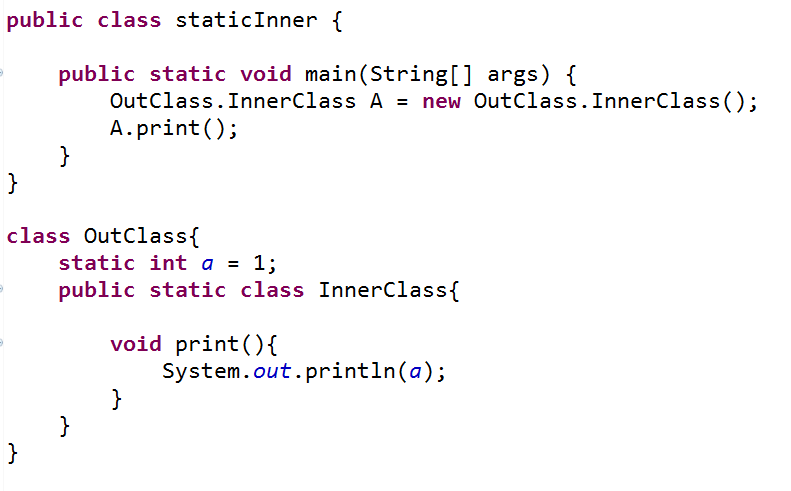


图4A-12 静态内部类

一般静态内部类都被称为嵌套类，当一个内部类是static的时候，这意味着：

1. 创建嵌套类对象并不需要外部类的对象。
2. 不能从嵌套类的对象中访问非静态类的外部类对象。

**1. 静态内部类案例 01**

案例设计：

在main()中访问使用静态内部类方法。

案例实现，如图4A-13所示：

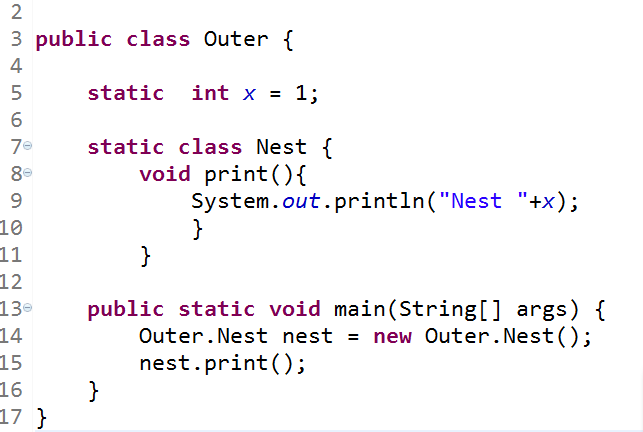


图4A-13 访问静态内部类方法

输出如图4A-14所示：

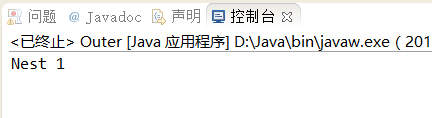


图4A-14 输出结果

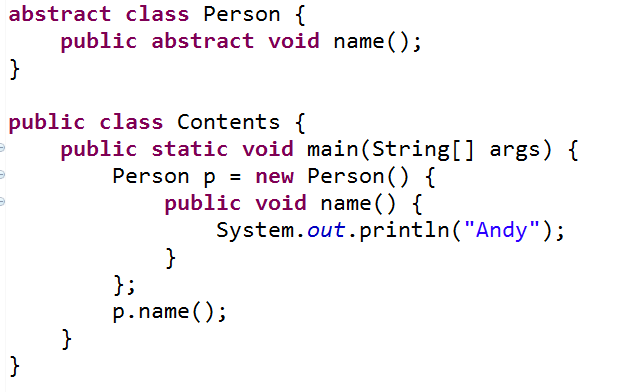
案例注意问题：

因为静态嵌套类和其他静态方法一样只能访问其它静态的成员，而不能访问实例成员。因此静态嵌套类和外部类（封装类）之间的联系就很少了，他们之间可能也就是命名空间上的一些关联。而上例需要注意的就是静态嵌套类的声明方法 new Outer.Nest() 连续写了两个类名，以至于会怀疑前面的Outer是个包名。

#### 1.2.4 匿名内部类（简称内部类）

匿名类是指没有自己名字的内部类，而且必须是非静态类。匿名类是不能有名称的类，所以没办法引用它们。必须在创建时，作为new语句的一部分来声明它们。如图4A-15所示。

图4A-15 匿名内部类



在使用匿名内部类时，要记住以下几个原则：

　 1）匿名内部类不能有构造方法。

　 2）匿名内部类不能定义任何静态成员、方法和类。

　 3）**匿名内部类不能是public,protected,private,static**。

　 4）**只能创建匿名内部类的一个实例**。

5）一个匿名内部类一定是在new的后面，用其隐含实现一个接口或实现一个类。

　 6）因匿名内部类为局部内部类，所以局部内部类的所有限制都对其生效。

【注意】匿名类和内部类中的中的this :

有时候，我们会用到一些内部类和匿名类。当在匿名类中用this时，这个this则指的是匿名类或内部类本身。这时如果我们要使用外部类的方法和变量的话，则应该加上外部类的类名。

#### 1. 匿名内部类案例01

案例设计：

在main()方法中编写匿名内部类去除字符串中的全部空格。首先声明ISringDeal接口，在接口中又声明了一个过滤字符串中的空格的方法，在主函数中我们实现了ISringDeal接口并且重写了filterBlankChar()方法。

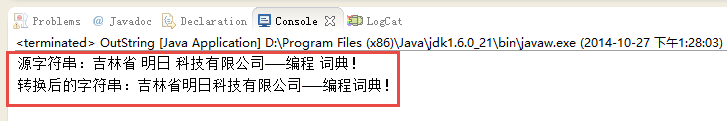
案例实现，如图4A-16所示：



图4A-16 调用匿名内部类

输出，图4A-17：

图4A-17 输出结果



案例注意的问题：

匿名类因为没有名字，所以匿名类不能有自己的构造方法。所以在初始化问题上，一般匿名类利用局部变量或形式参数完成初始化。在GUI事件编程中，大量使用匿名内部类。

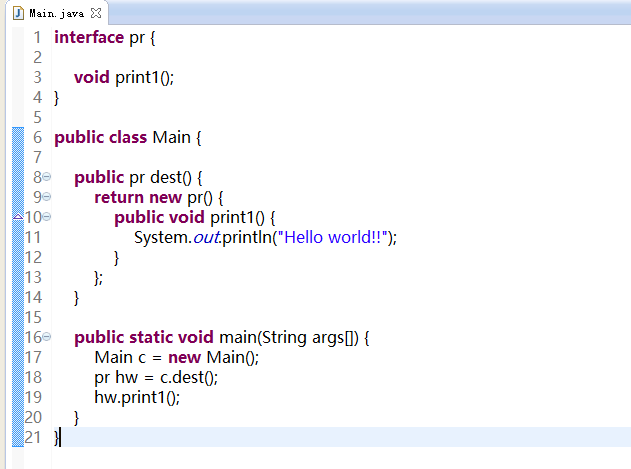
#### 2. 匿名内部类案例02

案例设计：

匿名类继承事件类重写回调方法，生成匿名对象，再调用匿名对象的方法。

案例实现，如图4A-18所示。

图4A-18 重写回调方法调用匿名内部类



输出字符串，如图4A-19所示。

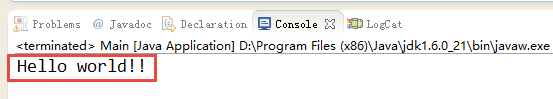


图4A-19 输出结果

案例需要注意的问题：

**使用匿名内部类必须继承一个父类或实现一个接口**。

**使用匿名内部类必须继承一个父类或实现一个接口**。

**使用匿名内部类必须继承一个父类或实现一个接口**。

### 1.3 为什么要使用内部类

结束了基础内部类的学习之后，相信还有不少读者不知道在JAVA中，内部类到底起着什么样的作用。这是因为内部类是JAVA中高级编程的应用，那么我们就来总结一下内部类的一些特点。

1. 内部类只能依靠其外部类存在，在内部类的定义中我们可以声明public、protected、private等访问权限，但是**内部类可以访问其外部类中的所有成员，无论是否private。这样，就可以起到一个隐藏代码的作用，同时也提供了一个进入外部类的窗口**。

2. 说起内部类，最吸引人的就是内部类可以直接继承一个接口，而并不用管外部类是否继承这个接口。这样，内部类使得**JAVA不支持的“多重继承”的解决方案变得完整**。（见参考文献）

## 2.匿名对象

### 2.1 匿名对象概念

概念：匿名对象是在**一个对象被创建之后**，调用对象的方法或属性时可以**不定义对象的引用变量**。

### 2.2 匿名对象案例

例一：

如图4A-20所示。

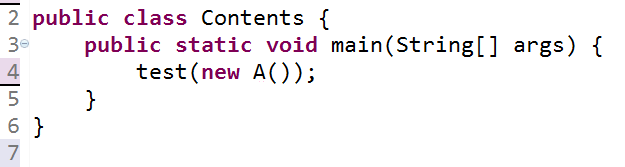


图4A-20 调用匿名对象

其中图4A-20中第4行代码的test(new A());语句等价于图4A-21中的第5行到第6行所示。而在图4A-20中的new A()就是一个匿名对象。

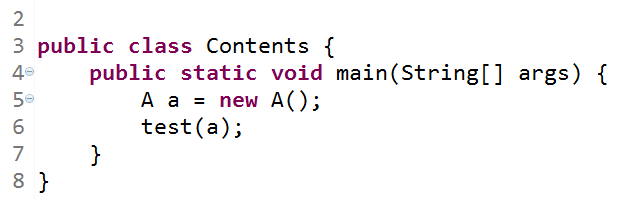


图4A-21 平时的调用方法

例二：

如图4A-22所示。”abc”.equals(str) 一个字符串能够调用一个函数，我们就可以看出来：一个字符串是String的匿名对象。

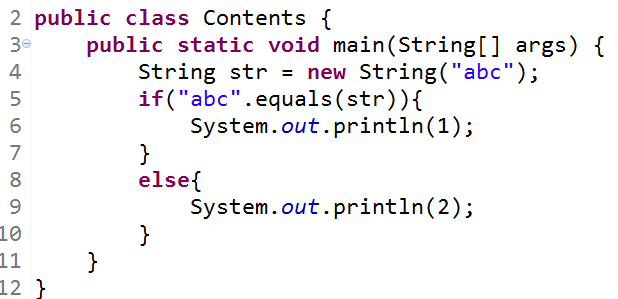


图4A-22 字符串调用方法

在JAVA中使用匿名对象能够使代码简洁并提高代码的可阅读性。

## 3.回调函数

所谓回调，就是客户程序**甲**调用服务程序**乙**中的某个函数A，然后**乙**又在某个时候反过来调用**甲**中的某个函数B，对于**甲**来说，这个B便叫做回调函数。

一般说来，**甲**不会自己调用B，**甲**提供B的目的就是让**乙**来调用它，而且是**甲**不得不提供。由于**乙**并不知道**甲**提供的B姓甚名谁，所以**乙**会约定B的接口规范（函数原型），然后由**甲**提前通过**乙**的一个函数R告诉**乙**自己将要使用B函数，这个过程称为回调函数的注册，R称为注册函数。Web Service以及Java的RMI都用到回调机制，可以访问远程服务器程序。

### 3.1回调原理图

如图4A-23所示。

图4A-23 回调原理

回调实现类

调用类

注册回调实例

被调用类

触发回调函数

实

回调

实现接口

定义接口引用

调用

### 3.2回调函数案例

案例设计：

Test（乙）是一个用于测试的调用者类，它通过main方法中实例化一个FooBar（甲），并用实现的ICallBack的匿名类作为参数传递给FooBar的被调用方法setCallBack，而在这个虚拟方法中，FooBar调用了匿名类的匿名类的postExec方法的动作，这个动作就是回调（Callback）。

案例实现：如图4A-24所示。

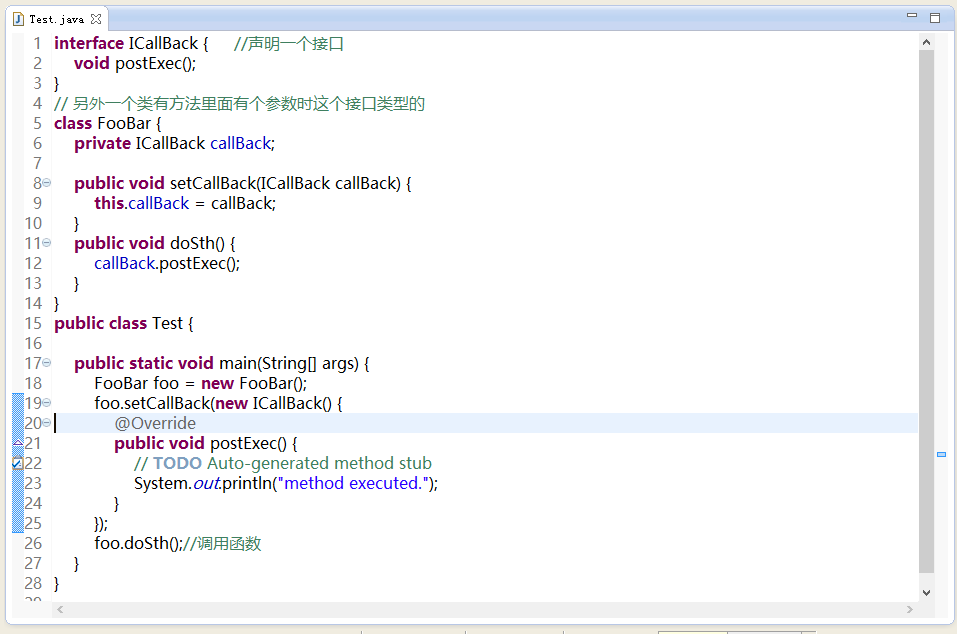


图4A-24 使用回调函数

## 输出结果如图4A-25所示：

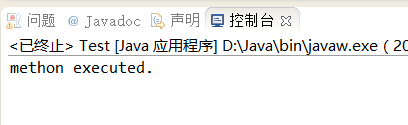


图4A-25 回调函数输出结果

## 4.回收机制

回收机制概念：在Java程序运行过程中，垃圾回收器是以后台线程方式运行，它会被不定时地被唤醒以检查是否有不再被使用的对象，以释放它们所占的内存空间。

1. finalize()是Object类的一个方法，它可以被其它类继承或改写。finalize()的作用：在对象被当做垃圾从内存中释放前调用，并不是在对象变成垃圾前被调用。由于垃圾回收器的启用是不定时的，因此，finalize()方法的调用并不可靠。
2. System.gc()作用:强制启动垃圾回收器来回收垃圾(指不再使用的对象)。
3. 在Java语言中，采用new为对象申请内存空间，至于内存空间的释放和回收，是由Java运行系统来完成的，用户可以完全不管，这样可以避免内存泄漏和无用内存的调用两类错误的产生，这种机制被称为垃圾回收机制。C++程序使用内存空间的策略：用new申请空间，用delete归还空间，否则，会产生内存泄漏。
4. 垃圾回收器的启动不用程序员控制，也无规律可循。并不会一产生了垃圾，它就被唤醒，甚至可能程序终止，它都没有启动的机会。当内存空间严重不足或调用相关外部命令时，它会被唤醒。

## 5. 项目心得

在本章学习中，主要讲述了Java中的内部类、匿名对象、回调函数以及垃圾回收机制。

本章的重点是内部类的学习。下面来总结下四种的内部类的使用场景：

静态内部类：作为类的静态成员存在于某一类的内部，用关键字static修饰。最简单的内部类，主要用于**访问静态对象**。

成员内部类：作为类的成员而存在在某一类的内部。最常用的**内部类种类**。

匿名内部类：存在于某一类的内部，但无名称的类。常用于需要**隐藏的类**。

局部内部类：存在于某一方法内的类。常用于用完就需要丢弃的类，会**随着方法的消逝而消逝**。

通过学习本章的内容，希望能加深读者对Android中对Java语法代码以及系统运行机制的理解。

参考文献：

http://www.cnblogs.com/nerxious/archive/2013/01/25/2876489.html