[java内部类的作用分析](http://blog.csdn.net/ilibaba/article/details/3866537)

提起Java内部类（Inner Class）可能很多人不太熟悉，实际上类似的概念在C++里也有，那就是嵌套类（Nested Class），关于这两者的区别与联系，在下文中会有对比。内部类从表面上看，就是在类中又定义了一个类（下文会看到，内部类可以在很多地方定义），而实际上并没有那么简单，乍看上去内部类似乎有些多余，它的用处对于初学者来说可能并不是那么显著，但是随着对它的深入了解，你会发现Java的设计者在内部类身上的确是用心良苦。学会使用内部类，是掌握Java高级编程的一部分，它可以让你更优雅地设计你的程序结构。下面从以下几个方面来介绍：

**第一次见面**

**[java]** [view plaincopy](http://blog.csdn.net/ilibaba/article/details/3866537)

1. **public** **interface** Contents {
2. **int** value();
3. }
4. **public** **interface** Destination {
5. String readLabel();
6. }
7. **public** **class** Goods {
8. **private** **class** Content **implements** Contents {
9. **private** **int** i = 11;
10. **public** **int** value() {
11. **return** i;
12. }
13. }
14. **protected** **class** GDestination **implements** Destination {
15. **private** String label;
16. **private** GDestination(String whereTo) {
17. label = whereTo;
18. }
19. **public** String readLabel() {
20. **return** label;
21. }
22. }
23. **public** Destination dest(String s) {
24. **return** **new** GDestination(s);
25. }
26. **public** Contents cont() {
27. **return** **new** Content();
28. }
29. }
30. **class** TestGoods {
31. **public** **static** **void** main(String[] args) {
32. Goods p = **new** Goods();
33. Contents c = p.cont();
34. Destination d = p.dest("Beijing");
35. }
36. }

在这个例子里类Content和GDestination被定义在了类Goods内部，并且分别有着protected和private修饰符来控制访问级别。Content代表着Goods的内容，而GDestination代表着Goods的目的地。它们分别实现了两个接口Content和Destination。在后面的main方法里，直接用 Contents c和Destination d进行操作，你甚至连这两个内部类的名字都没有看见！这样，内部类的第一个好处就体现出来了 隐藏你不想让别人知道的操作，也即封装性。

同时，我们也发现了在外部类作用范围之外得到内部类对象的第一个方法，那就是利用其外部类的方法创建并返回。上例中的cont()和dest()方法就是这么做的。那么还有没有别的方法呢？当然有，其语法格式如下：

outerObject=new outerClass(Constructor Parameters);

outerClass.innerClass innerObject=outerObject.new InnerClass(Constructor Parameters);

注意在创建非静态内部类对象时，一定要先创建起相应的外部类对象。至于原因，也就引出了我们下一个话题 非静态内部类对象有着指向其外部类对象的引用，对刚才的例子稍作修改：

**[java]** [view plaincopy](http://blog.csdn.net/ilibaba/article/details/3866537)

1. **public** **class** Goods {
2. **private** **int** valueRate = 2;
3. **private** **class** Content **implements** Contents {
4. **private** **int** i = 11 \* valueRate;
5. **public** **int** value() {
6. **return** i;
7. }
8. }
9. **protected** **class** GDestination **implements** Destination {
10. **private** String label;
11. **private** GDestination(String whereTo) {
12. label = whereTo;
13. }
14. **public** String readLabel() {
15. **return** label;
16. }
17. }
18. **public** Destination dest(String s) {
19. **return** **new** GDestination(s);
20. }
21. **public** Contents cont() {
22. **return** **new** Content();
23. }
24. }

在这里我们给Goods类增加了一个private成员变量valueRate，意义是货物的价值系数，在内部类Content的方法value()计算价值时把它乘上。我们发现，value()可以访问valueRate，这也是内部类的第二个好处 一个内部类对象可以访问创建它的外部类对象的内容，甚至包括私有变量！这是一个非常有用的特性，为我们在设计时提供了更多的思路和捷径。要想实现这个功能，内部类对象就必须有指向外部类对象的引用。Java编译器在创建内部类对象时，隐式的把其外部类对象的引用也传了进去并一直保存着。这样就使得内部类对象始终可以访问其外部类对象，同时这也是为什么在外部类作用范围之外向要创建内部类对象必须先创建其外部类对象的原因。

有人会问，如果内部类里的一个成员变量与外部类的一个成员变量同名，也即外部类的同名成员变量被屏蔽了，怎么办？没事，Java里用如下格式表达外部类的引用：

outerClass.this

有了它，我们就不怕这种屏蔽的情况了。

**静态内部类**

和普通的类一样，内部类也可以有静态的。不过和非静态内部类相比，区别就在于静态内部类没有了指向外部的引用。这实际上和C++中的嵌套类很相像了，Java内部类与C++嵌套类最大的不同就在于是否有指向外部的引用这一点上，当然从设计的角度以及以它一些细节来讲还有区别。

除此之外，在任何非静态内部类中，都不能有静态数据，静态方法或者又一个静态内部类（内部类的嵌套可以不止一层）。不过静态内部类中却可以拥有这一切。这也算是两者的第二个区别吧。

**局部内部类**

是的，Java内部类也可以是局部的，它可以定义在一个方法甚至一个代码块之内。

**[java]** [view plaincopy](http://blog.csdn.net/ilibaba/article/details/3866537)

1. **public** **class** Goods1 {
2. **public** Destination dest(String s) {
3. **class** GDestination **implements** Destination {
4. **private** String label;
5. **private** GDestination(String whereTo) {
6. label = whereTo;
7. }
8. **public** String readLabel() {
9. **return** label;
10. }
11. }
12. **return** **new** GDestination(s);
13. }
14. **public** **static** **void** main(String[] args) {
15. Goods1 g = **new** Goods1();
16. Destination d = g.dest("Beijing");
17. }
18. }

上面就是这样一个例子。在方法dest中我们定义了一个内部类，最后由这个方法返回这个内部类的对象。如果我们在用一个内部类的时候仅需要创建它的一个对象并创给外部，就可以这样做。当然，定义在方法中的内部类可以使设计多样化，用途绝不仅仅在这一点。

下面有一个更怪的例子：

**[java]** [view plaincopy](http://blog.csdn.net/ilibaba/article/details/3866537)

1. **public** **class** Goods2 {
2. **private** **void** internalTracking(**boolean** b) {
3. **if** (b) {
4. **class** TrackingSlip {
5. **private** String id;
6. TrackingSlip(String s) {
7. id = s;
8. }
9. String getSlip() {
10. **return** id;
11. }
12. }
13. TrackingSlip ts = **new** TrackingSlip("slip");
14. String s = ts.getSlip();
15. }
16. }
17. **public** **void** track() {
18. internalTracking(**true**);
19. }
20. **public** **static** **void** main(String[] args) {
21. Goods2 g = **new** Goods2();
22. g.track();
23. }
24. }

你不能在if之外创建这个内部类的对象，因为这已经超出了它的作用域。不过在编译的时候，内部类TrackingSlip和其他类一样同时被编译，只不过它由它自己的作用域，超出了这个范围就无效，除此之外它和其他内部类并没有区别。

**匿名内部类**

java的匿名内部类的语法规则看上去有些古怪，不过如同匿名数组一样，当你只需要创建一个类的对象而且用不上它的名字时，使用内部类可以使代码看上去简洁清楚。它的语法规则是这样的：

new interfacename(){......}; 或 new superclassname(){......};

下面接着前面继续举例子：

**[java]** [view plaincopy](http://blog.csdn.net/ilibaba/article/details/3866537)

1. **public** **class** Goods3 {
2. **public** Contents cont() {
3. **return** **new** Contents() {
4. **private** **int** i = 11;
5. **public** **int** value() {
6. **return** i;
7. }
8. };
9. }
10. }

这里方法cont()使用匿名内部类直接返回了一个实现了接口Contents的类的对象，看上去的确十分简洁。

在java的事件处理的匿名适配器中，匿名内部类被大量的使用。例如在想关闭窗口时加上这样一句代码：

**[java]** [view plaincopy](http://blog.csdn.net/ilibaba/article/details/3866537)

1. frame.addWindowListener(**new** WindowAdapter(){
2. **public** **void** windowClosing(WindowEvent e){
3. System.exit(0);
4. }
5. });

有一点需要注意的是，匿名内部类由于没有名字，所以它没有构造函数（但是如果这个匿名内部类继承了一个只含有带参数构造函数的父类，创建它的时候必须带上这些参数，并在实现的过程中使用super关键字调用相应的内容）。如果你想要初始化它的成员变量，有下面几种方法：

如果是在一个方法的匿名内部类，可以利用这个方法传进你想要的参数，不过记住，这些参数必须被声明为final。

将匿名内部类改造成有名字的局部内部类，这样它就可以拥有构造函数了。

在这个匿名内部类中使用初始化代码块。

**为什么需要内部类？**

**java内部类有什么好处？为什么需要内部类？**

**首先举一个简单的例子，如果你想实现一个接口，但是这个接口中的一个方法和你构想的这个类中的一个方法的名称，参数相同，你应该怎么办？这时候，你可以建一个内部类实现这个接口。由于内部类对外部类的所有内容都是可访问的，所以这样做可以完成所有你直接实现这个接口的功能。**

**不过你可能要质疑，更改一下方法的不就行了吗？**

**的确，以此作为设计内部类的理由，实在没有说服力。**

**真正的原因是这样的，java中的内部类和接口加在一起，可以的解决常被C++程序员抱怨java中存在的一个问题 没有多继承。实际上，C++的多继承设计起来很复杂，而java通过内部类加上接口，可以很好的实现多继承的效果。**

Java 内部类

分四种：成员内部类、局部内部类、静态内部类和匿名内部类。   
1、成员内部类： 即作为外部类的一个成员存在，与外部类的属性、方法并列。  
注意：成员内部类中不能定义[静态变量](http://zhidao.baidu.com/search?word=%E9%9D%99%E6%80%81%E5%8F%98%E9%87%8F&fr=qb_search_exp&ie=utf8),但可以访问外部类的所有成员。  
public class Outer{  
private static int i = 1;  
private int j=10;  
private int k=20;  
public static void outer\_f1(){  
 //do more something  
}  
public void out\_f2(){  
 //do more something  
}  
  
//成员内部类  
class Inner{  
//static int inner\_i =100; //内部类中不允[许定](http://zhidao.baidu.com/search?word=%E8%AE%B8%E5%AE%9A&fr=qb_search_exp&ie=utf8)义[静态变量](http://zhidao.baidu.com/search?word=%E9%9D%99%E6%80%81%E5%8F%98%E9%87%8F&fr=qb_search_exp&ie=utf8)  
int j=100;//内部类中外部类的[实例变量](http://zhidao.baidu.com/search?word=%E5%AE%9E%E4%BE%8B%E5%8F%98%E9%87%8F&fr=qb_search_exp&ie=utf8)可以共存  
int inner\_i=1;  
void inner\_f1(){  
 System.out.println(i);//外部类的变量如果和内部类的变量没有同名的，则可以直接用变量名访问外部类的变量  
 System.out.println(j);//在内部类中访问内部类自己的变量直接用变量名  
 System.out.println(this.j);//也可以在内部类中用"this.变量名"来访问内部类变量  
 //访问外部类中与内部类同名的[实例变量](http://zhidao.baidu.com/search?word=%E5%AE%9E%E4%BE%8B%E5%8F%98%E9%87%8F&fr=qb_search_exp&ie=utf8)可用"外部类名.this.变量名"。  
 System.out.println(k);//外部类的变量如果和内部类的变量没有同名的，则可以直接用变量名访问外部类的变量  
 outer\_f1();  
 outer\_f2();  
}  
}  
//外部类的非[静态方法](http://zhidao.baidu.com/search?word=%E9%9D%99%E6%80%81%E6%96%B9%E6%B3%95&fr=qb_search_exp&ie=utf8)访问成员内部类  
public void outer\_f3(){  
 Inner inner = new Inner();  
 inner.inner\_f1();  
}  
  
//外部类的[静态方法](http://zhidao.baidu.com/search?word=%E9%9D%99%E6%80%81%E6%96%B9%E6%B3%95&fr=qb_search_exp&ie=utf8)访问成员内部类，与在外部类外部访问成员内部类一样  
public static void outer\_f4(){  
 //step1 建立外部类对象  
 Outer out = new Outer();  
 //\*\*\*step2 根据外部类对象建立内部类对象\*\*\*  
 Inner inner=out.new Inner();  
 //step3 访问内部类的方法  
 inner.inner\_f1();  
}  
  
public static void main(String[] args){  
 outer\_f4();  
}  
}  
成员内部类的优点：  
⑴ 内部类作为外部类的成员，可以访问外部类的私有成员或属性。（即使将外部类声明为PRIVATE，但是对于处于其内部的内部类还是可见的。）  
⑵ 用内部类定义在外部类中不可访问的属性。这样就在外部类中实现了比外部类的private还要小的访问权限。  
注意：内部类是一个编译时的概念，一旦编译成功，就会成为完全不同的两类。对于一个名为outer的外部类和其内部定义的名为inner的内部类。编译完成后出现outer.class和outer$inner.class两类。  
2、局部内部类： 即在方法中定义的内部类，与[局部变量](http://zhidao.baidu.com/search?word=%E5%B1%80%E9%83%A8%E5%8F%98%E9%87%8F&fr=qb_search_exp&ie=utf8)类似，在局部内部类前不加修饰符public或private，其范围为定义它的代码块。  
注意：局部内部类中不可定义[静态变量](http://zhidao.baidu.com/search?word=%E9%9D%99%E6%80%81%E5%8F%98%E9%87%8F&fr=qb_search_exp&ie=utf8)，可以访问外部类的[局部变量](http://zhidao.baidu.com/search?word=%E5%B1%80%E9%83%A8%E5%8F%98%E9%87%8F&fr=qb_search_exp&ie=utf8)(即方法内的变量)，但是变量必须是final的。  
public class Outer {  
 private int s = 100;  
 private int out\_i = 1;  
 public void f(final int k){  
 final int s = 200;  
 int i = 1;  
 final int j = 10;  
 class Inner{ //定义在方法内部  
 int s = 300;//可以定义与外部类同名的变量  
 //static int m = 20;//不可以定义静态变量  
 Inner(int k){  
 inner\_f(k);  
 }  
 int inner\_i = 100;  
 void inner\_f(int k){  
 System.out.println(out\_i);//如果内部类没有与外部类同名的变量，在内部类中可以直接访问外部类的[实例变量](http://zhidao.baidu.com/search?word=%E5%AE%9E%E4%BE%8B%E5%8F%98%E9%87%8F&fr=qb_search_exp&ie=utf8)  
 System.out.println(k);//\*\*\*\*\*可以访问外部类的[局部变量](http://zhidao.baidu.com/search?word=%E5%B1%80%E9%83%A8%E5%8F%98%E9%87%8F&fr=qb_search_exp&ie=utf8)(即方法内的变量)，但是变量必须是final的\*\*\*\*\*  
// System.out.println(i);  
 System.out.println(s);//如果内部类中有与外部类同名的变量，直接用变量名访问的是内部类的变量  
 System.out.println(this.s);//用"this.变量名" 访问的也是内部类变量  
 System.out.println(Outer.this.s);//用外部"外部类类名.this.变量名" 访问的是外部类变量  
 }  
 }  
 new Inner(k);  
 }  
  
 public static void main(String[] args) {  
 //访问局部内部类必须先有外部类对象  
 Outer out = new Outer();  
 out.f(3);  
 }  
  
}  
  
注意：  
在类外不可直接生成局部内部类（保证局部内部类对外是不可见的）。要想使用局部内部类时需要生成对象，对象调用方法，在方法中才能调用其局部内部类。通过内部类和接口达到一个强制的弱耦合，用局部内部类来实现接口，并在方法中返回[接口类型](http://zhidao.baidu.com/search?word=%E6%8E%A5%E5%8F%A3%E7%B1%BB%E5%9E%8B&fr=qb_search_exp&ie=utf8)，使局部内部类不可见，屏蔽实现类的可见性。  
  
3、静态内部类： 静态内部类定义在类中，任何方法外，用static定义。  
注意：静态内部类中可以定义静态或者非静态的成员  
public class Outer {  
 private static int i = 1;  
 private int j = 10;  
 public static void outer\_f1(){  
   
 }  
 public void outer\_f2(){  
   
 }  
// 静态内部类可以用public,protected,private修饰  
// 静态内部类中可以定义静态或者非静态的成员  
 static class Inner{  
 static int inner\_i = 100;  
 int inner\_j = 200;  
 static void inner\_f1(){  
 System.out.println("Outer.i"+i);//静态内部类只能访问外部类的[静态成员](http://zhidao.baidu.com/search?word=%E9%9D%99%E6%80%81%E6%88%90%E5%91%98&fr=qb_search_exp&ie=utf8)  
 outer\_f1();//包括静态变量和[静态方法](http://zhidao.baidu.com/search?word=%E9%9D%99%E6%80%81%E6%96%B9%E6%B3%95&fr=qb_search_exp&ie=utf8)  
 }  
 void inner\_f2(){  
// System.out.println("Outer.i"+j);//静态内部类不能访问外部类的非[静态成员](http://zhidao.baidu.com/search?word=%E9%9D%99%E6%80%81%E6%88%90%E5%91%98&fr=qb_search_exp&ie=utf8)  
// outer\_f2();//包括非静态变量和非静态方法  
 }   
   
 }  
   
 public void outer\_f3(){  
// 外部类访问内部类的[静态成员](http://zhidao.baidu.com/search?word=%E9%9D%99%E6%80%81%E6%88%90%E5%91%98&fr=qb_search_exp&ie=utf8)：内部类.静态成员  
 System.out.println(Inner.inner\_i);  
 Inner.inner\_f1();  
// 外部类访问内部类的非静态成员:[实例化](http://zhidao.baidu.com/search?word=%E5%AE%9E%E4%BE%8B%E5%8C%96&fr=qb_search_exp&ie=utf8)内部类即可  
 Inner inner = new Inner();  
 inner.inner\_f2();  
   
 }  
 public static void main(String[] args) {  
 new Outer().outer\_f3();  
 }  
  
}  
  
注意：\*\*\*\*\*\*\*生成（new）一个静态内部类不需要外部类成员：这是静态内部类和成员内部类的区别。静态内部类的对象可以直接生成：  
Outer.Inner in=new Outer.Inner()；  
而不需要通过生成外部类对象来生成。这样实际上使静态内部类成为了一个顶级类。静态内部类不可用private来进行定义。\*\*\*\*\*\*\*  
  
例子：  
对于两个类，拥有相同的方法：  
class People  
{  
 run();  
}  
class Machine{  
 run();  
}  
此时有一个robot类：  
class Robot extends People implement Machine.  
此时run()不可直接实现。  
注意：当类与接口（或者是接口与接口）发生方法命名冲突的时候，此时必须使用内部类来实现。用接口不能完全地实现多继承，用接口配合内部类才能实现真正的多继承。  
  
4、匿名内部类   
匿名内部类是一种特殊的局部内部类，它是通过匿名类实现接口。  
IA被定义为接口。  
IA I=new IA(){};  
  
匿名内部类的特点：  
  
1，一个类用于继承其他类或是实现接口，并不需要增加额外的方法，只是对继承方法的事先或是覆盖。  
2，只是为了获得一个对象实例，不需要知道其实际类型。  
3，类名没有意义，也就是不需要使用到。  
  
public class Outer {  
 private static int i = 1;  
 private int j = 10;  
 public static void outer\_f1(){  
   
 }  
 public void outer\_f2(){  
   
 }  
// 静态内部类可以用public,protected,private修饰  
// 静态内部类中可以定义静态或者非静态的成员  
 static class Inner{  
 static int inner\_i = 100;  
 int inner\_j = 200;  
 static void inner\_f1(){  
 System.out.println("Outer.i"+i);//静态内部类只能访问外部类的静态成员  
 outer\_f1();//包括静态变量和静态方法  
 }  
 void inner\_f2(){  
// System.out.println("Outer.i"+j);//静态内部类不能访问外部类的非静态成员  
// outer\_f2();//包括非静态变量和非静态方法  
 }  
 }  
   
 public void outer\_f3(){  
// 外部类访问内部类的静态成员：内部类.静态成员  
 System.out.println(Inner.inner\_i);  
 Inner.inner\_f1();  
// 外部类访问内部类的非静态成员:[实例化](http://zhidao.baidu.com/search?word=%E5%AE%9E%E4%BE%8B%E5%8C%96&fr=qb_search_exp&ie=utf8)内部类即可  
 Inner inner = new Inner();  
 inner.inner\_f2();  
   
 }  
 public static void main(String[] args) {  
 new Outer().outer\_f3();  
 }  
  
}  
  
注：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，没有类名，根据[多态](http://zhidao.baidu.com/search?word=%E5%A4%9A%E6%80%81&fr=qb_search_exp&ie=utf8)，我们使用其父类名。因他是局部内部类，那么局部内部类的所有限制都对其生效。匿名内部类是唯一一种无[构造方法](http://zhidao.baidu.com/search?word=%E6%9E%84%E9%80%A0%E6%96%B9%E6%B3%95&fr=qb_search_exp&ie=utf8)类。大部分匿名内部类是用于接口[回调](http://zhidao.baidu.com/search?word=%E5%9B%9E%E8%B0%83&fr=qb_search_exp&ie=utf8)用的。匿名内部类在编译的时候由系统自动起名Out$1.class。如果一个对象编译时的类型是接口，那么其运行的类型为实现这个接口的类。因匿名内部类无[构造方法](http://zhidao.baidu.com/search?word=%E6%9E%84%E9%80%A0%E6%96%B9%E6%B3%95&fr=qb_search_exp&ie=utf8)，所以其使用范围非常的有限。当需要多个对象时使用局部内部类，因此局部内部类的应用相对比较多。匿名内部类中不能定义[构造方法](http://zhidao.baidu.com/search?word=%E6%9E%84%E9%80%A0%E6%96%B9%E6%B3%95&fr=qb_search_exp&ie=utf8)。如果一个对象编译时的类型是接口，那么其运行的类型为实现这个接口的类。  
  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
内部类总结：  
1.首先，把内部类作为外部类的一个特殊的成员来看待，因此它有类成员的封闭等级：private ,protected,默认(friendly),public  
 它有类成员的修饰符: static,final,abstract  
2.非静态内部类nested inner class,内部类隐含有一个外部类的指针this,因此，它可以访问外部类的一切资源（当然包括private）  
 外部类访问内部类的成员，先要取得内部类的对象,并且取决于内部类成员的封装等级。  
 非静态内部类不能包含任何static成员.  
3.静态内部类：static inner class,不再包含外部类的this指针，并且在外部类装载时初始化.  
 静态内部类能包含static或非static成员.  
 静态内部类只能访问外部类static成员.  
 外部类访问静态内部类的成员，循一般类法规。对于static成员，用类名.成员即可访问，对于非static成员，只能  
 用对象.成员进行访问  
  
4.对于方法中的内部类或块中内部类只能访问块中或方法中的final变量。  
  
类成员有两种static , non-static，同样内部类也有这两种  
non-static 内部类的实例，必须在外部类的方法中创建或通过外部类的实例来创建(OuterClassInstanceName.new innerClassName(ConstructorParameter)),并且可直接访问外部类的信息,外部类对象可通过OuterClassName.this来引用  
static 内部类的实例, 直接创建即可，没有对外部类实例的引用。  
内部类不管static还是non-static都有对外部类的引用  
non-static 内部类不允许有static成员  
  
方法中的内部类只允许访问方法中的final局部变量和方法的final参数列表，所以说方法中的内部类和内部类没什麽区别。但方法中的内部类不能在方法以外访问，方法中不可以有static内部类  
匿名内部类如果继承自接口,必须实现指定接口的方法,且无参数   
匿名内部类如果继承自类,参数必须按父类的[构造函数](http://zhidao.baidu.com/search?word=%E6%9E%84%E9%80%A0%E5%87%BD%E6%95%B0&fr=qb_search_exp&ie=utf8)的参数传递