[**java提高篇（七）-----详解内部类**](http://blog.csdn.net/chenssy/article/details/13024951)

2013-10-25 20:02 3700人阅读 [评论](http://blog.csdn.net/chenssy/article/details/13024951#comments)(2) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/chenssy/article/details/13024951#report)

http://static.blog.csdn.net/images/category_icon.jpg 分类：

【JAVA开发】-----Java提高篇（36） http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

版权声明：本文为博主原创文章，未经博主允许不得转载。

目录[(?)[-]](http://blog.csdn.net/chenssy/article/details/13024951)

1. [第一次见面](http://blog.csdn.net/chenssy/article/details/13024951#t0)
2. [一为什么要使用内部类](http://blog.csdn.net/chenssy/article/details/13024951#t1)
3. [二内部类基础](http://blog.csdn.net/chenssy/article/details/13024951#t2)
4. [三成员内部类](http://blog.csdn.net/chenssy/article/details/13024951#t3)
5. [四局部内部类](http://blog.csdn.net/chenssy/article/details/13024951#t4)
6. [五匿名内部类](http://blog.csdn.net/chenssy/article/details/13024951#t6)
7. [六静态内部类](http://blog.csdn.net/chenssy/article/details/13024951#t7)

**可以将一个类的定义放在另一个类的定义内部，这就是内部类。**

       内部类是一个非常有用的特性但又比较难理解使用的特性(鄙人到现在都没有怎么使用过内部类，对内部类也只是略知一二)。

**第一次见面**

       内部类我们从外面看是非常容易理解的，无非就是在一个类的内部在定义一个类。

**[java]** [view plain](http://blog.csdn.net/chenssy/article/details/13024951) [copy](http://blog.csdn.net/chenssy/article/details/13024951)

[print?](http://blog.csdn.net/chenssy/article/details/13024951)[在CODE上查看代码片](https://code.csdn.net/snippets/129617)

1. **public** **class** OuterClass {
2. **private** String name ;
3. **private** **int** age;
5. **public** String getName() {
6. **return** name;
7. }
9. **public** **void** setName(String name) {
10. **this**.name = name;
11. }
13. **public** **int** getAge() {
14. **return** age;
15. }
17. **public** **void** setAge(**int** age) {
18. **this**.age = age;
19. }
21. **class** InnerClass{
22. **public** InnerClass(){
23. name = "chenssy";
24. age = 23;
25. }
26. }
27. }

       在这里InnerClass就是内部类，对于初学者来说内部类实在是使用的不多，鄙人菜鸟一个同样没有怎么使用过(貌似仅仅只在做swing 注册事件中使用过)，但是随着编程能力的提高，我们会领悟到它的魅力所在，它可以使用能够更加优雅的设计我们的程序结构。在使用内部类之间我们需要明白为什么要使用内部类，内部类能够为我们带来什么样的好处。

**一、为什么要使用内部类**

       为什么要使用内部类？在《Think in [**Java**](http://lib.csdn.net/base/17)》中有这样一句话：使用内部类最吸引人的原因是：每个内部类都能独立地继承一个（接口的）实现，所以无论外围类是否已经继承了某个（接口的）实现，对于内部类都没有影响。

      在我们程序设计中有时候会存在一些使用接口很难解决的问题，这个时候我们可以利用内部类提供的、可以继承多个具体的或者抽象的类的能力来解决这些程序设计问题。可以这样说，接口只是解决了部分问题，而内部类使得多重继承的解决方案变得更加完整。

**[java]** [view plain](http://blog.csdn.net/chenssy/article/details/13024951) [copy](http://blog.csdn.net/chenssy/article/details/13024951)

[print?](http://blog.csdn.net/chenssy/article/details/13024951)[在CODE上查看代码片](https://code.csdn.net/snippets/129617)

1. **public** **interface** Father {
3. }
5. **public** **interface** Mother {
7. }
9. **public** **class** Son **implements** Father, Mother {
11. }
13. **public** **class** Daughter **implements** Father{
15. **class** Mother\_ **implements** Mother{
17. }
18. }

      其实对于这个实例我们确实是看不出来使用内部类存在何种优点，但是如果Father、Mother不是接口，而是抽象类或者具体类呢？这个时候我们就只能使用内部类才能实现多重继承了。

      其实使用内部类最大的优点就在于它能够非常好的解决多重继承的问题，但是如果我们不需要解决多重继承问题，那么我们自然可以使用其他的编码方式，但是使用内部类还能够为我们带来如下特性（摘自《Think in java》）：

**1、**内部类可以用多个实例，每个实例都有自己的状态信息，并且与其他外围对象的信息相互独立。

**2、**在单个外围类中，可以让多个内部类以不同的方式实现同一个接口，或者继承同一个类。

**3、**创建内部类对象的时刻并不依赖于外围类对象的创建。

**4、**内部类并没有令人迷惑的“is-a”关系，他就是一个独立的实体。

**5、**内部类提供了更好的封装，除了该外围类，其他类都不能访问。

**二、内部类基础**

       在这个部分主要介绍内部类如何使用外部类的属性和方法，以及使用.this与.new。

       当我们在创建一个内部类的时候，它无形中就与外围类有了一种联系，依赖于这种联系，它可以无限制地访问外围类的元素。

**[java]** [view plain](http://blog.csdn.net/chenssy/article/details/13024951) [copy](http://blog.csdn.net/chenssy/article/details/13024951)

[print?](http://blog.csdn.net/chenssy/article/details/13024951)[在CODE上查看代码片](https://code.csdn.net/snippets/129617)

1. **public** **class** OuterClass {
2. **private** String name ;
3. **private** **int** age;
5. /\*\*省略getter和setter方法\*\*/
7. **public** **class** InnerClass{
8. **public** InnerClass(){
9. name = "chenssy";
10. age = 23;
11. }
13. **public** **void** display(){
14. System.out.println("name：" + getName() +"   ;age：" + getAge());
15. }
16. }
18. **public** **static** **void** main(String[] args) {
19. OuterClass outerClass = **new** OuterClass();
20. OuterClass.InnerClass innerClass = outerClass.**new** InnerClass();
21. innerClass.display();
22. }
23. }
24. --------------
25. Output：
26. name：chenssy   ;age：23

       在这个应用程序中，我们可以看到内部了InnerClass可以对外围类OuterClass的属性进行无缝的访问，尽管它是private修饰的。这是因为当我们在创建某个外围类的内部类对象时，此时内部类对象必定会捕获一个指向那个外围类对象的引用，只要我们在访问外围类的成员时，就会用这个引用来选择外围类的成员。

       其实在这个应用程序中我们还看到了如何来引用内部类：引用内部类我们需要指明这个对象的类型：OuterClasName.InnerClassName。同时如果我们需要创建某个内部类对象，必须要利用外部类的对象通过.new来创建内部类：OuterClass.InnerClass innerClass = outerClass.new InnerClass();。

       同时如果我们需要生成对外部类对象的引用，可以使用OuterClassName.this，这样就能够产生一个正确引用外部类的引用了。当然这点实在编译期就知晓了，没有任何运行时的成本。

**[java]** [view plain](http://blog.csdn.net/chenssy/article/details/13024951) [copy](http://blog.csdn.net/chenssy/article/details/13024951)

[print?](http://blog.csdn.net/chenssy/article/details/13024951)[在CODE上查看代码片](https://code.csdn.net/snippets/129617)

1. **public** **class** OuterClass {
2. **public** **void** display(){
3. System.out.println("OuterClass...");
4. }
6. **public** **class** InnerClass{
7. **public** OuterClass getOuterClass(){
8. **return** OuterClass.**this**;
9. }
10. }
12. **public** **static** **void** main(String[] args) {
13. OuterClass outerClass = **new** OuterClass();
14. OuterClass.InnerClass innerClass = outerClass.**new** InnerClass();
15. innerClass.getOuterClass().display();
16. }
17. }
18. -------------
19. Output:
20. OuterClass...

       到这里了我们需要明确一点，内部类是个编译时的概念，一旦编译成功后，它就与外围类属于两个完全不同的类（当然他们之间还是有联系的）。对于一个名为OuterClass的外围类和一个名为InnerClass的内部类，在编译成功后，会出现这样两个class文件：OuterClass.class和OuterClass$InnerClass.class。

       在Java中内部类主要分为成员内部类、局部内部类、匿名内部类、静态内部类。

**三、成员内部类**

       成员内部类也是最普通的内部类，它是外围类的一个成员，所以他是可以无限制的访问外围类的所有 成员属性和方法，尽管是private的，但是外围类要访问内部类的成员属性和方法则需要通过内部类实例来访问。

       在成员内部类中要注意两点，**第一：**成员内部类中不能存在任何static的变量和方法；**第二：**成员内部类是依附于外围类的，所以只有先创建了外围类才能够创建内部类。

**[java]** [view plain](http://blog.csdn.net/chenssy/article/details/13024951) [copy](http://blog.csdn.net/chenssy/article/details/13024951)

[print?](http://blog.csdn.net/chenssy/article/details/13024951)[在CODE上查看代码片](https://code.csdn.net/snippets/129617)

1. **public** **class** OuterClass {
2. **private** String str;
4. **public** **void** outerDisplay(){
5. System.out.println("outerClass...");
6. }
8. **public** **class** InnerClass{
9. **public** **void** innerDisplay(){
10. //使用外围内的属性
11. str = "chenssy...";
12. System.out.println(str);
13. //使用外围内的方法
14. outerDisplay();
15. }
16. }
18. /\*推荐使用getxxx()来获取成员内部类，尤其是该内部类的构造函数无参数时 \*/
19. **public** InnerClass getInnerClass(){
20. **return** **new** InnerClass();
21. }
23. **public** **static** **void** main(String[] args) {
24. OuterClass outer = **new** OuterClass();
25. OuterClass.InnerClass inner = outer.getInnerClass();
26. inner.innerDisplay();
27. }
28. }
29. --------------------
30. chenssy...
31. outerClass...

**推荐使用getxxx()来获取成员内部类，尤其是该内部类的构造函数无参数时 。**

**四、局部内部类**

       有这样一种内部类，它是嵌套在方法和作用于内的，对于这个类的使用主要是应用与解决比较复杂的问题，想创建一个类来辅助我们的解决方案，到那时又不希望这个类是公共可用的，所以就产生了局部内部类，局部内部类和成员内部类一样被编译，只是它的作用域发生了改变，它只能在该方法和属性中被使用，出了该方法和属性就会失效。

       对于局部内部类实在是想不出什么好例子，所以就引用《Think in java》中的经典例子了。

       定义在方法里：

**[java]** [view plain](http://blog.csdn.net/chenssy/article/details/13024951) [copy](http://blog.csdn.net/chenssy/article/details/13024951)

[print?](http://blog.csdn.net/chenssy/article/details/13024951)[在CODE上查看代码片](https://code.csdn.net/snippets/129617)

1. **public** **class** Parcel5 {
2. **public** Destionation destionation(String str){
3. **class** PDestionation **implements** Destionation{
4. **private** String label;
5. **private** PDestionation(String whereTo){
6. label = whereTo;
7. }
8. **public** String readLabel(){
9. **return** label;
10. }
11. }
12. **return** **new** PDestionation(str);
13. }
15. **public** **static** **void** main(String[] args) {
16. Parcel5 parcel5 = **new** Parcel5();
17. Destionation d = parcel5.destionation("chenssy");
18. }
19. }

       定义在作用域内:

**[java]** [view plain](http://blog.csdn.net/chenssy/article/details/13024951) [copy](http://blog.csdn.net/chenssy/article/details/13024951)

[print?](http://blog.csdn.net/chenssy/article/details/13024951)[在CODE上查看代码片](https://code.csdn.net/snippets/129617)

1. **public** **class** Parcel6 {
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("chenssy");
14. String string = ts.getSlip();
15. }
16. }
18. **public** **void** track(){
19. internalTracking(**true**);
20. }
22. **public** **static** **void** main(String[] args) {
23. Parcel6 parcel6 = **new** Parcel6();
24. parcel6.track();
25. }
26. }

**五、匿名内部类**

       在做Swing编程中，我们经常使用这种方式来绑定事件

**[html]** [view plain](http://blog.csdn.net/chenssy/article/details/13024951) [copy](http://blog.csdn.net/chenssy/article/details/13024951)

[print?](http://blog.csdn.net/chenssy/article/details/13024951)[在CODE上查看代码片](https://code.csdn.net/snippets/129617)

1. button2.addActionListener(
2. new ActionListener(){
3. public void actionPerformed(ActionEvent e) {
4. System.out.println("你按了按钮二");
5. }
6. });

        我们咋一看可能觉得非常奇怪，因为这个内部类是没有名字的，在看如下这个例子：

**[java]** [view plain](http://blog.csdn.net/chenssy/article/details/13024951) [copy](http://blog.csdn.net/chenssy/article/details/13024951)

[print?](http://blog.csdn.net/chenssy/article/details/13024951)[在CODE上查看代码片](https://code.csdn.net/snippets/129617)

1. **public** **class** OuterClass {
2. **public** InnerClass getInnerClass(**final** **int** num,String str2){
3. **return** **new** InnerClass(){
4. **int** number = num + 3;
5. **public** **int** getNumber(){
6. **return** number;
7. }
8. };        /\* 注意：分号不能省 \*/
9. }
11. **public** **static** **void** main(String[] args) {
12. OuterClass out = **new** OuterClass();
13. InnerClass inner = out.getInnerClass(2, "chenssy");
14. System.out.println(inner.getNumber());
15. }
16. }
18. **interface** InnerClass {
19. **int** getNumber();
20. }
22. ----------------
23. Output:
24. 5

       这里我们就需要看清几个地方

**1、** 匿名内部类是没有访问修饰符的。

**2、** new 匿名内部类，这个类首先是要存在的。如果我们将那个InnerClass接口注释掉，就会出现编译出错。

**3、** 注意getInnerClass()方法的形参，第一个形参是用final修饰的，而第二个却没有。同时我们也发现第二个形参在匿名内部类中没有使用过，所以当所在方法的形参需要被匿名内部类使用，那么这个形参就必须为final。

**4、** 匿名内部类是没有构造方法的。因为它连名字都没有何来构造方法。

**PS：**由于篇幅有限，对匿名内部类就介绍到这里，有关更多关于匿名内部类的知识，我就会在下篇博客（java提高篇-----详解匿名内部类）做详细的介绍，包括为何形参要定义成final，怎么对匿名内部类进行初始化等等，敬请期待……

**六、静态内部类**

       在java提高篇-----关键字static中提到Static可以修饰成员变量、方法、代码块，其他它还可以修饰内部类，使用static修饰的内部类我们称之为静态内部类，不过我们更喜欢称之为嵌套内部类。静态内部类与非静态内部类之间存在一个最大的区别，我们知道非静态内部类在编译完成之后会隐含地保存着一个引用，该引用是指向创建它的外围内，但是静态内部类却没有。没有这个引用就意味着：

**1、** 它的创建是不需要依赖于外围类的。

**2、** 它不能使用任何外围类的非static成员变量和方法。

**[java]** [view plain](http://blog.csdn.net/chenssy/article/details/13024951) [copy](http://blog.csdn.net/chenssy/article/details/13024951)

[print?](http://blog.csdn.net/chenssy/article/details/13024951)[在CODE上查看代码片](https://code.csdn.net/snippets/129617)

1. **public** **class** OuterClass {
2. **private** String sex;
3. **public** **static** String name = "chenssy";
5. /\*\*
6. \*静态内部类
7. \*/
8. **static** **class** InnerClass1{
9. /\* 在静态内部类中可以存在静态成员 \*/
10. **public** **static** String \_name1 = "chenssy\_static";
12. **public** **void** display(){
13. /\*
14. \* 静态内部类只能访问外围类的静态成员变量和方法
15. \* 不能访问外围类的非静态成员变量和方法
16. \*/
17. System.out.println("OutClass name :" + name);
18. }
19. }
21. /\*\*
22. \* 非静态内部类
23. \*/
24. **class** InnerClass2{
25. /\* 非静态内部类中不能存在静态成员 \*/
26. **public** String \_name2 = "chenssy\_inner";
27. /\* 非静态内部类中可以调用外围类的任何成员,不管是静态的还是非静态的 \*/
28. **public** **void** display(){
29. System.out.println("OuterClass name：" + name);
30. }
31. }
33. /\*\*
34. \* @desc 外围类方法
35. \* @author chenssy
36. \* @data 2013-10-25
37. \* @return void
38. \*/
39. **public** **void** display(){
40. /\* 外围类访问静态内部类：内部类. \*/
41. System.out.println(InnerClass1.\_name1);
42. /\* 静态内部类 可以直接创建实例不需要依赖于外围类 \*/
43. **new** InnerClass1().display();
45. /\* 非静态内部的创建需要依赖于外围类 \*/
46. OuterClass.InnerClass2 inner2 = **new** OuterClass().**new** InnerClass2();
47. /\* 方位非静态内部类的成员需要使用非静态内部类的实例 \*/
48. System.out.println(inner2.\_name2);
49. inner2.display();
50. }
52. **public** **static** **void** main(String[] args) {
53. OuterClass outer = **new** OuterClass();
54. outer.display();
55. }
56. }
57. ----------------
58. Output:
59. chenssy\_static
60. OutClass name :chenssy
61. chenssy\_inner
62. OuterClass name：chenssy

       上面这个例子充分展现了静态内部类和非静态内部类的区别。

       到这里内部类的介绍就基本结束了！对于内部类其实本人认识也只是皮毛，逼近菜鸟一枚，认知有限！我会利用这几天时间好好研究内部类!

**巩固基础，提高技术，不惧困难，攀登高峰！！！！！！**