**[内部类详解（很详细）](http://www.cnblogs.com/jiangao/archive/2012/02/23/2364119.html)**

（转）内部类有两种情况：

(1) 在类中定义一个类(私有内部类，静态内部类)

(2) 在方法中定义一个类(局部内部类，匿名内部类)

**1、私有内部类 —— 在方法之间定义的内部类，非静态**

      我们首先看看类中内部类的两个特点：

     (1) 在外部类的作用范围内可以任意创建内部类对象，即使内部类是私有的(私有内部类)。即内部类对包围它的外部类可见。

1. //代码1：内部类对外部类可见
2. **class** Outer{
3. //创建私有内部类对象
4. **public** Inner in=**new** Inner();
5. //私有内部类
6. **private** **class** Inner{
7. ...
8. }
9. }

      (2) 在内部类中可以访问其外部类的所有域，即使是私有域。即外部类对内部类可见。

1. //代码2：外部类对内部类可见
2. **class** Outer{
3. //外部类私有数据域
4. **private** **int** data=0;
5. //内部类
6. **class** Inner{
7. **void** print(){
8. //内部类访问外部私有数据域
9. System.out.println(data);
10. }
11. }
12. }

       问题来了：上面两个特点到底如何办到的呢？内部类的"内部"到底发生了什么？

其实，内部类是Java编译器一手操办的。虚拟机并不知道内部类与常规类有什么不同。 编译器是如何瞒住虚拟机的呢？

     对内部类进行编译后发现有两个class文件：**Outer.class**和**Outer$Inner.class**。这说明内部类Inner仍然被编译成一个独立的类(Outer$Inner.class)，而不是Outer类的某一个域。 虚拟机运行的时候，也是把Inner作为一种常规类来处理的。

        但问题来了，即然是两个常规类，为什么他们之间可以互相访问私有域那(最开始提到的两个内部类特点)？这就要问问编译器到底把这两个类编译成什么东西了。

        我们利用reflect反射机制来探查了一下内部类编译后的情况（关于探查类内部机制的代码提供在下面的附件里Reflect.java）。

        (1)、编译代码1生成 Outer$Inner.class 文件后使用 ReflectUtil.reflect("Outer$Inner") 对内部类Inner进行反射。运行结果 发现了三个隐含的成分：

1. //反编译1
2. class Outer$Inner
3. {
4. Outer$Inner(Outer,Outer$Inner);  //包可见构造器
5. private Outer$Inner(Outer);   //私有构造器将设置this$0域
6. final Outer this$0;   //外部类实例域this$0
7. }

      好了，现在我们可以解释上面的第一个内部类特点了： **为什么外部类可以创建内部类的对象？并且内部类能够方便的引用到外部类对象?**

     首先编译器将外、内部类编译后放在同一个包中。在内部类中附加一个包可见构造器。这样， 虚拟机运行Outer类中Inner in=new Inner(); 实际上调用的是包可见构造： new Outer$Inner(this,null)。因此即使是private内部类，也会通过隐含的包可见构造器成功的获得私有内部类的构造权限。

      再者，Outer$Inner类中有一个指向外部类Outer的引用this$0，那么通过这个引用就可以方便的得到外部类对象中可见成员。但是Outer类中的private成员是如何访问到的呢？这就要看看下面Outer.class文件中的秘密了。

      (2)、编译代码2生成 Outer.class文件，然后使用 ReflectUtil.reflect("Outer") 对外部类Outer进行反射 。 运行结果 发现一个隐含成分如下：

1. //反编译2
2. class Outer
3. {
4. static int access$0(Outer);  //静态方法，返回值是外部类私有域 data 的值。
5. }

      现在可以解释第二个特点了：**为什么内部类可以引用外部类的私有域？**

         原因的关键就在编译器在外围类中添加了静态方法access$0。 它将返回值作为参数传递给他的对象域data。这样内部类Inner中的打印语句：

                     System.out.println(data);

         实际上运行的时候调用的是：

                 S ystem.out.println(this$0.access$0(Outer));

**总结一下编译器对类中内部类做的手脚吧：**

**(1)  在内部类中偷偷摸摸的创建了包可见构造器，从而使外部类获得了创建权限。**

**(2)  在外部类中偷偷摸摸的创建了访问私有变量的静态方法，从而 使 内部类获得了访问权限。**

**这样，类中定义的内部类无论私有，公有，静态都可以被包围它的外部类所访问。**

**2、静态内部类  ——  在方法间定义的内部类，静态**

     内部类也有静态的区别，这就是静态内部类，我们来看看代码：

1. **package** hr.test;
2. //代码3：静态内部类对外部变量的引用
3. **public** **class** Outer{
4. **private** **static** **int** i=0;
5. //创建静态内部类对象
6. **public** Inner in=**new** Inner();
7. //静态
8. **private** **static** **class** Inner{
9. **public** **void** print(){
10. System.out.println(i);   //如果i不是静态变量，这里将无法通过编译。
11. }
12. }
14. }

       静态内部类和私有内部类最大的区别在于，静态内部类中无法引用到其外围类的非静态成员。这是为什么？我们还是来看看静态内部类Outer$Inner中发生了什么吧？

1. //反编译3
2. class Outer$Inner
3. {
4. private Outer$Inner();
5. Outer$Inner(hr.test.Outer$Inner);
6. }

        与上面私有内部类反编译1比较发现，少了一个指向外围类对象的引用final Outer this$0; 也就是说静态内部类无法得到其外围类对象的引用，那么自然也就无法访问外围类的非静态成员了。**因此，静态内部类只能访问其外围类的静态成员，除此之外与非静态内部类没有任何区别。**

**3、局部内部类 —— 在方法中定义的内部类**

      方法内部类也有两个特点

      (1)  方法中的内部类没有访问修饰符， 即方法内部类对包围它的方法之外的任何东西都不可见。

      (2)  方法内部类只能够访问该方法中的局部变量，所以也叫局部内部类。而且这些局部变量一定要是final修饰的常量。

1. **class** Outter{
2. **public** **void** outMethod(){
3. **final** **int** beep=0;
4. **class** Inner{
5. //使用beep
6. }
7. Inner in=**new** Inner();
8. }
9. }

      这又是为什么呢？

      (1) 我们首先对Outter类进行反射发现，Outter中再也没有返回私有域的隐藏方法了。

      (2) 对Inner类的反射发现，Inner类内部多了一个对beep变量的备份隐藏域：final int val$i;

      我们可以这样解释Inner类中的这个备份常量域，首先当JVM运行到需要创建Inner对象之后，Outter类已经全部运行完毕，这是垃圾回收机制很有可能释放掉局部变量beep。那么Inner类到哪去找beep变量呢？

**编译器又出来帮我们解决了这个问题，他在Inner类中创建了一个beep的备份**，也就是说即使Ouuter中的beep被回收了，Inner中还有一个备份存在，自然就不怕找不到了。

      但是问题又来了。如果Outter中的beep不停的在变化那。那岂不是也要让备份的beep变量无时无刻的变化。**为了保持局部变量与局部内部类中备份域保持一致。**编译器不得不规定死这些局部域必须是常量，一旦赋值不能再发生变化了。

      所以为什么局部内部类应用外部方法的域必须是常量域的原因所在了。

**内部类的特点总结**

**(1)  在方法间定义的非静态内部类：**

**● 外围类和内部类可互相访问自己的私有成员。**

**● 内部类中不能定义静态成员变量。**

**(2) 在方法间定义的静态内部类：**

**● 只能访问外部类的静态成员。**

**(3) 在方法中定义的局部内部类：**

**● 该内部类没有任何的访问控制权限**

**● 外围类看不见方法中的局部内部类的，但是局部内部类可以访问外围类的任何成员。**

**● 方法体中可以访问局部内部类，但是访问语句必须在定义局部内部类之后。**

**● 局部内部类只能访问方法体中的常量，即用final修饰的成员。**

**(4) 在方法中定义的匿名内部类：**

**● 没有构造器，取而代之的是将构造器参数传递给超类构造器。**