肖小胖的博客

永久域名 http://flowercat.javaeye.com
使用xStream实现xml与java对象之间的转换 | 内部类的序列化问题
2006-11-09

转：java内部类

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代码

```
1.  public interface Contents {
2.      int value();
3.  }
4.
5.  public interface Destination {
6.      String readLabel();
7.  }
8.
9.  public class Goods {
10.     private class Content implements Contents {
11.         private int i = 11;
12.         public int value() {
13.             return i;
14.         }
15.     }
16.
17.     protected class GDestination implements Destination {
18.         private String label;
19.         private GDestination(String whereTo) {
20.             label = whereTo;
21.         }
22.         public String readLabel() {
23.             return label;
24.         }
25.     }
26.
27.     public Destination dest(String s) {
28.         return new GDestination(s);
29.     }
30.     public Contents cont() {
```

```
        return new Content();
    }
}

class TestGoods {
    public static void main(String[] args) {
        Goods p = new Goods();
        Contents c = p.cont();
        Destination d = p.dest("Beijing");
    }
}
```

Java代码

```
public interface Contents {
    int value();
}

public interface Destination {
    String readLabel();
}

public class Goods {
    private class Content implements Contents {
        private int i = 11;
        public int value() {
            return i;
        }
    }

    protected class GDestination implements Destination {
        private String label;
        private GDestination(String whereTo) {
            label = whereTo;
        }
        public String readLabel() {
            return label;
        }
    }

    public Destination dest(String s) {
        return new GDestination(s);
    }
    public Contents cont() {
        return new Content();
    }
}

class TestGoods {
    public static void main(String[] args) {
        Goods p = new Goods();
        Contents c = p.cont();
```

```
        Destination d = p.dest("Beijing");
    }
}
```

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代码

```
public class Goods {

    private valueRate=2;

    private class Content implements Contents {
        private int i = 11*valueRate;
        public int value() {
            return i;
        }
    }

    protected class GDestination implements Destination {
        private String label;
        private GDestination(String whereTo) {
            label = whereTo;
        }
        public String readLabel() {
            return label;
        }
    }

    public Destination dest(String s) {
        return new GDestination(s);
```

```
    }
    public Contents cont() {
        return new Content();
    }
}

```

Java代码

```
public class Goods {

    private valueRate=2;

    private class Content implements Contents {
        private int i = 11*valueRate;
        public int value() {
            return i;
        }
    }

    protected class GDestination implements Destination {
        private String label;
        private GDestination(String whereTo) {
            label = whereTo;
        }
        public String readLabel() {
            return label;
        }
    }

    public Destination dest(String s) {
        return new GDestination(s);
    }
    public Contents cont() {
        return new Content();
    }
}

```

修改的部分用红色显示了。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代码

```
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() { return label; }
9.          }
10.         return new GDestination(s);
11.     }
12.
13.     public static void main(String[] args) {
14.         Goods1 g= new Goods1();
15.         Destination d = g.dest("Beijing");
16.     }
17. }
```

Java代码

```
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.              }
```

```
            public String readLabel() { return label; }
        }
        return new GDestination(s);
    }

    public static void main(String[] args) {
        Goods1 g= new Goods1();
        Destination d = g.dest("Beijing");
    }
}
```

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代码

```
public class Goods2{
    private void internalTracking(boolean b) {
        if(b) {
            class TrackingSlip {
                private String id;
                TrackingSlip(String s) {
                    id = s;
                }
                String getSlip() { return id; }
            }
            TrackingSlip ts = new TrackingSlip("slip");
            String s = ts.getSlip();
        }
    }

    public void track() { internalTracking(true); }

    public static void main(String[] args) {
        Goods2 g= new Goods2();
        g.track();
    }
}
```

Java代码

```
public class Goods2{
    private void internalTracking(boolean b) {
```

```
        if(b) {
            class TrackingSlip {
                private String id;
                TrackingSlip(String s) {
                    id = s;
                }
                String getSlip() { return id; }
            }
            TrackingSlip ts = new TrackingSlip("slip");
            String s = ts.getSlip();
        }
    }

    public void track() { internalTracking(true); }

    public static void main(String[] args) {
        Goods2 g= new Goods2();
        g.track();
    }
}
```

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代码

```
public class Goods3 {
    public Contents cont(){
        return new Contents(){
            private int i = 11;
            public int value() {
                return i;
            }
        };
    }
}
```

Java代码

```
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代码

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

Java代码

```
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代码

1.

Java代码

1.

使用xStream实现xml与java对象之间的转换 | 内部类的序列化问题

- 13:38
- 浏览 (6483)
- 评论 (0)
- 相关推荐

评论

发表评论

表情图标

B I U Quote Code List Img URL Flash Table

字体颜色: 标准 字体大小: 标准 对齐: 标准

提示：选择您需要装饰的文字, 按上列按钮即可添加上相应的标签

表情图标

B I U Quote Code List Img URL Flash Table

字体颜色: 标准 字体大小: 标准 对齐: 标准

提示：选择您需要装饰的文字, 按上列按钮即可添加上相应的标签

您还没有登录，请登录后发表评论(快捷键 Alt+S / Ctrl+Enter) 提交

flowercat

- 浏览: 68442 次
- 我现在离线
- 详细资料 留言簿

搜索本博客

搜索

最近访客 >>更多访客

gwpking8419

我是小兵

nzinfo

limi1986214

博客分类

- 全部博客 (110)
- 灌水分类 (3)
- Ruby&Rails (8)
- linux (34)
- python (6)
- java (22)
- db (4)
- Google (5)

我的留言簿 >>更多留言

- 路过打酱油
  -- by dongwei_6688

其他分类

- 我的收藏 (61)
- 我的论坛主题贴 (2)
- 我的所有论坛贴 (15)
- 我的精华良好贴 (0)

最近加入圈子

- Python
- Groovy on Grails

存档

- 2010-10 (1)
- 2010-05 (2)
- 2010-04 (5)
- 更多存档...

评论排行榜

- play!存在的Cookie设置的bug
- playframework的Eclipse插件问题
- 使用GA跟踪网站下载文件
- (转)在 CentOS 设置 iptables
- Linux iptables 开放Mysql端口

- RSS
- Add to Google