第14章

构造型模式介绍

若要创建一个 Java 类，通常会为它提供多个构造函数[译注1]。构造函数是有用的，尽管只有客户类知道该使用哪个构造函数以及传递什么参数来创建类。很多设计模式都可以解决常规构造函数无法处理的情况。在检查 Java 语言在构造函数设计上存在的不足之前，先来回顾一下常规的构造函数。

构造函数的挑战

构造函数是一些特殊的方法，在诸如访问修饰符、重载以及参数列表语法等诸多方面，构造函数和常规方法没有什么不同。然而，在语义与语法规则方面，构造函数的行为与使用受到更多的约束。

挑战 14.1

列出使用 Java 构造函数的 4 种约束。

答案参见第 330 页

译注1：若要延续 C++的传统，Constructor 应该被翻译为构造器，事实上，微软的 C#官方文档中，也采用了这一术语。不过在 Java 语言中，似乎翻译为“构造函数”的更为普遍。因此，本书选择用“构造函数”来表示 Constructor。

在一些场景下，Java 提供默认的构造函数与行为。首先，如果某个类没有声明构造函数，Java 会提供一个默认的构造函数。默认构造函数是一个公共的构造函数，没有任何参数，方法体也没有执行语句。

第二个默认行为是，如果构造函数的声明没有显式调用 this()或 super()方法，Java 会自动调用不带参数的 super()方法。这可能会导致出现一些意外的结果，就像如下的代码编译错误一样：

```
package app.construction;
public class Fuse {
      private String name;
   // public Fuse(String name) { this.name = name; }
}
```

与

```
package app.construction;
public class QuickFuse extends Fuse { }
```

只有当你删除了“//”注释标记后，这段代码才能正常编译。

挑战 14.2

请解释为何注释掉 Fuse 类的构造函数后，会发生编译错误？

答案参见第 330 页

初始化对象的常见方法是调用 new 操作符。当然，也可以使用反射。反射提供了能将类型与类型成员像对象一样操作的能力。即使不常使用反射，也可以提供依靠反射实现的程序逻辑，如下所示：

```
package app.construction;
import java.awt.Point;
import java.lang.reflect.Constructor;
public class ShowReflection {
    public static void main(String args[]) {
        Constructor[] cc = Point.class.getConstructors();
        Constructor cons = null;
        for (inti = 0; i<cc.length; i++)
```

```
            if (cc[i].getParameterTypes().length == 2)
                cons = cc[i];
        try {
            Object obj = cons.newInstance(
                new Object[] { new Integer(3), new Integer(4)});
            System.out.println(obj);
        } catch (Exception e) {
            System.out.println("Exception: " + e.getMessage());
        }
    }
}
```

挑战 14.3

ShowReflection 程序输出什么？

答案参见第 331 页

当使用其他手段很难或无法达到想要的结果时，反射能达成你的目标。若要了解使用反射的更多内容，请参见 *Java™ in a Nutshell*（由 Flanagan 在 2005 年编写）。

小结

一般情况下，你需要为自己开发的类提供构造函数使其能够被初始化。这些构造函数可能相互调用协作，并且类中的每个构造函数最终都会去调用超类的构造函数。调用构造函数的常规做法是使用 new 操作符，但也可以使用反射来初始化和使用对象。

超出常规的构造函数

在设计一个新类时，Java 构造函数的这些特性可以给你提供许多选择。然而，只有在类的用户知道该如何初始化类，以及传递所需参数时，构造函数才是有效的。例如，选择怎样的用

户界面，取决于程序运行在哪种显示器上，是手持设备，还是更大的显示器。或许，开发者知道使用哪一个类进行初始化，但他并不知道初始化所需的全部参数或格式。例如，开发者可能需要根据对象的静态方式或文本方式去构建对象。此时，常规的 Java 构造函数很难解决问题，你需要引入新的设计模式。

表 14.1 描述了各种模式的意图。

表 14.1　各种模式的意图

如果你的意图是	使用的模式
• 在请求创建对象前，逐步收集创建对象需要的信息	构建者模式
• 决定推迟实例化类对象	工厂方法模式
• 创建一族具有某些共同特征的对象	抽象工厂模式
• 根据现有对象创建一个新的对象	原型模式
• 通过包含了对象内部状态的静态版本重新构建一个对象	备忘录模式

每种设计模式的意图都是为了解决某种场景下的问题。构造型模式的设计就是为了让客户类不通过类构造函数来创建对象。例如，当你需要逐步获取对象的初始值时，则可能需要使用构建者模式。