|6.1 Prototype 模式

【通过复制生成实例】

我们通常会使用以下方式生成Something类的实例。

new Something（）

在Java中，我们可以使用new关键字指定类名来生成类的实例。像这样使用new来生成实例时，是必须指定类名的。但是，在开发过程中，有时候也会有“在不指定类名的前提下生成实例”的需求。例如，在以下情况下，我们就不能根据类来生成实例，而要根据现有的实例来生成新的实例。

（1）对象种类繁多，无法将它们整合到一个类中时

第一种情况是需要处理的对象太多，如果将它们分别作为一个类，必须要编写很多个类文件。

（2）难以根据类生成实例时

第二种情况是生成实例的过程太过复杂，很难根据类来生成实例。例如，我们假设这里有一个实例，即表示用户在图形编辑器中使用鼠标制作出的图形的实例。想在程序中创建这样的实例是非常困难的。通常，在想生成一个和之前用户通过操作所创建出的实例完全一样的实例的时候，我们会事先将用户通过操作所创建出的实例保存起来，然后在需要时通过复制来生成新的实例。

（3）想解耦框架与生成的实例时

第三种情况是想要让生成实例的框架不依赖于具体的类。这时，不能指定类名来生成实例，而要事先“注册”一个“原型”实例，然后通过复制该实例来生成新的实例。

根据实例生成实例与使用复印机复印文档相类似。即使不知道原来的文档中的内容，我们也可以使用复印机复制出完全相同的文档，无论多少份都行。

在本章中，我们将要学习不根据类来生成实例，而是根据实例来生成新实例的Prototype模式。

Prototype有“原型”“模型”的意思。在设计模式中，它是指根据实例原型、实例模型来生成新的实例。

在Java语言中，我们可以使用c1one创建出实例的副本。在本章中，我们将学习clone方法与Cloneab1e接口的使用方法。

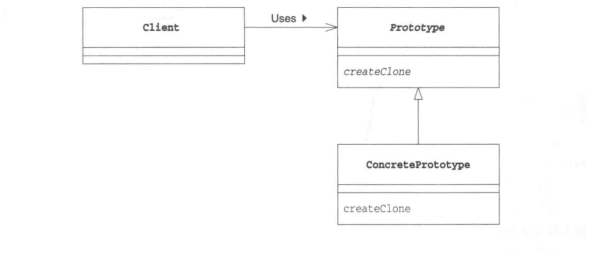
|6.3 Prototype模式中的登场角色

在Prototype模式中有以下登场角色。

·Prototype（原型）Product角色负责定义用于复制现有实例来生成新实例的方法。在示例程序中，由Product接口扮演此角色。

·ConcretePrototype（具体的原型）ConcretePrototype角色负责实现复制现有实例并生成新实例的方法。在示例程序中，由MessageBox类和UnderlinePen类扮演此角色。

◆Client（使用者）Client角色负责使用复制实例的方法生成新的实例。在示例程序中，由Manager类扮演此角色。



6.4拓展思路的要点

|不能根据类来生成实例吗

笔者在最初学习Prototype模式时也曾感觉到迷茫，既然是要创建新的实例，直接用下面这个语句不就好了吗？为什么还需要Prototype模式呢？

new Something（）

在本章开头，我们对这个问题做了简单的回答，现在让我们回顾一下示例程序，并谈谈这个问题。

（1）对象种类繁多，无法将它们整合到一个类中时在示例程序中，一共出现了如下3种样式。

·使用~，为字符串添加下划线

·使用，\*，为字符串添加边框

·使用/，为字符串添加边框

本例比较简单，只生成了3种样式，不过只要想做，不论多少种样式都可以生成。但是请试想一下，如果将每种样式都编写为一个类，类的数量将会非常庞大，源程序的管理也会变得非常困难。

（2）难以根据类生成实例时

本例中感觉不到这一点。大家可以试想下要开发一个用户可以使用鼠标进行操作的、类似于图形编辑器的应用程序，这样可能更加容易理解。假设我们想生成一个和用户通过一系列鼠标操作所创建出来的实例完全一样的实例。这个时候，与根据类来生成实例相比，根据实例来生成实例要简单得多。

（3）想解耦框架与生成的实例时

在示例程序中，我们将复制（clone）实例的部分封装在framework包中了。

在Manager类的create方法中，我们并没有使用类名，取而代之使用了"strong message"和"slash box"等字符串为生成的实例命名。与Java语言自带的生成实例的new Something（）方式相比，这种方式具有更好的通用性，而且将框架从类名的束缚中解脱出来了。

类名是束缚吗

话说回来，在源程序中使用类名到底会有什么问题呢？在代码中出现要使用的类的名字不是理所当然的吗？

这里，让我们再回忆一下面向对象编程的目标之一，即“作为组件复用”。

在代码中出现要使用的类的名字并非总是坏事。不过，一旦在代码中出现要使用的类的名字，就无法与该类分离开来，也就无法实现复用。

当然，可以通过替换源代码或是改变类名来解决这个问题。但是，此处说的“作为组件复用” 中不包含替换源代码。以Java来说，重要的是当手边只有class文件（.class）时，该类能否被复用。即使没有Java文件（java）也能复用该类才是关键。

当多个类必须紧密结合时，代码中出现这些类的名字是没有问题的。但是如果那些需要被独立出来作为组件复用的类的名字出现在代码中，那就有问题了。

|6.5相关的设计模式

·Flyweight模式（第20章）使用Prototype模式可以生成一个与当前实例的状态完全相同的实例。

而使用Flyweight模式可以在不同的地方使用同一个实例。

◆Memento模式（第18章）使用Prototype模式可以生成一个与当前实例的状态完全相同的实例。而使用Memento模式可以保存当前实例的状态，以实现快照和撤销功能。

·Composite模式（第11章）以及Decorator模式（第12章）经常使用Composite模式和Decorator模式时，需要能够动态地创建复杂结构的实例。这时可以使用Prototype模式，以帮助我们方便地生成实例。

◆Command模式（第22章）想要复制Command模式中出现的命令时，可以使用Prototype模式。

|6.6延伸阅读：clone方法和java.lang.Clonable接口

|Java语言的clone Java语言为我们准备了用于复制实例的clone方法。请注意，要想调用c1one方法，被复制对象的类必须实现ava.lang.Clonable接口，不论是被复制对象的类实现java.lang.

Cloneab1e接口还是其某个父类实现Cloneable接口，亦或是被复制对象的类实现了Cloneab1e接口的子接口都可以。在示例程序中，MessageBox类和UnderlinePen类实现了Product 接口，而Product接口则是Cloneable接口的子接口。

实现了Cloneab1e接口的类的实例可以调用clone方法进行复制，clone方法的返回值是复制出的新的实例（clone方法内部所进行的处理是分配与要复制的实例同样大小的内存空间，接着将要复制的实例中的字段的值复制到所分配的内存空间中去）。

如果没有实现c1oneab1e接口的类的实例调用了clone方法，则会在运行时抛出CloneNotSupportedException（不支持clone方法）异常。

笔者对上文进行了总结，结果如下。

·实现了Cloneable接口的类的实例

→复制

·没有实现Cloneable接口的类的实例

→发生CloneNotSupportedException异常此外，java.lang包是被默认引入的，因此无需显式地引入java.1ang 即可调用clone方法。

Iclone方法是在哪里定义的

clone方法定义在java.lang.object中。因为object类是所有Java类的父类，因此所有的Java类都继承了clone方法。

需要实现Cloneable的哪些方法

提到C1oneab1e接口，很容易让人误以为Cloneab1e接口中声明了clone方法。其实这是错误的。在c1oneab1e接口中并没有声明任何方法。它只是被用来标记“可以使用clone方法进行复制”的。这样的接口被称为标记接口（marker interface）。

|clone方法进行的是浅复制

clone方法所进行的复制只是将被复制实例的字段值直接复制到新的实例中。换言之，它并没有考虑字段中所保存的实例的内容。例如，当字段中保存的是数组时，如果使用c1one方法进行复制，则只会复制该数组的引用，并不会一一复制数组中的元素。

像上面这样的字段对字段的复制（field-to-field-copy）被称为浅复制（shallow copy）。clone方法所进行的复制就是浅复制。

当使用clone方法进行浅复制无法满足需求时，类的设计者可以实现重写clone方法，实现自己需要的复制功能（重写clone方法时，别忘了使用super.clone（）来调用父类的clone方法）。

需要注意的是，clone方法只会进行复制，并不会调用被复制实例的构造函数。此外，对于在生成实例时需要进行特殊的初始化处理的类，需要自己去实现clone方法，在其内部进行这些初始化处理。

详细信息请参见Java的API参考资料中java.lang.object类的clone方法和Cloneable接口这两个相关条目。