第16章

工厂方法（Factory Method）模式

在创建类时，通常可以同时定义多个构造函数，然后让它们创建类的实例。然而有时候，客户代码虽然需要某个对象，却并不关心或者不需要关心这个对象究竟是由哪个类创建而来的。

工厂方法模式的意图是定义一个用于创建对象的接口，并控制返回哪个类的实例。

经典范例：迭代器

迭代器（Iterator）模式提供了用于顺序访问容器中的各个元素的方法（参见本书第 28 章）。但是，迭代器实例却经常需要运用工厂方法模式来创建。Java JDK 1.2 引进了一个容器类接口，其中包含了一个 `iterator()`方法，所有容器类均实现了该方法，它使得调用者并不知道 `iterator` 对象是由哪个类实例化的。

使用 `iterator()`方法可以创建一个对象，顺序地返回容器内的所有元素。例如，如下代码创建了一个列表，并打印了列表中包含的内容：

```
package app.factoryMethod;
import java.util.*;

public class ShowIterator {
    public static void main(String[] args) {
```

```
        List list = Arrays.asList(
            new String[] {
                "fountain", "rocket", "sparkler"});

        Iterator iter = list.iterator();
        while (iter.hasNext())
            System.out.println(iter.next());
    }
}
```

挑战 16.1

这段代码中的 Iterator 对象的实际类型是什么？

答案参见第 334 页

工厂方法模式使得客户代码无须关心使用哪个类的实例。

识别工厂方法

你可能认为任何可以创建或返回一个新对象的方法都是“工厂方法”。然而，在面向对象编程中，方法返回新对象是非常普遍的，但是并非每个这样的方法都应用了工厂方法模式。

挑战 16.2

说出 Java 类库中两个可以返回新对象的常用方法。

答案参见第 335 页

事实上，并非每个能创建并返回一个新对象的方法，都是工厂方法模式的实例。工厂方法模式不仅要求有一个能够创建新对象的方法，还要让客户代码无须了解具体实例化的类。工厂方法模式通常包含了若干类，这些类实现了相同的操作，返回了相同的抽象类型，然而这些操作的内部，实际上却实例化了不同的类，并且，这些类都实现了上述抽象类型。当客户代码请求一个新对象时，这个新对象该由哪个类实例化，取决于工厂对象接收创建请求时的行为。

挑战 16.3

类 `javax.swing.BorderFactory` 看似运用了工厂方法模式。请说明工厂方法模式的意图和 `BorderFactory` 类的意图有何不同。

答案参见第 335 页

控制要实例化的类

通常，客户代码使用类的某个构造函数来实例化类。但是有时候，客户代码并不知道使用哪个类去创建它所需要的对象。比如，在迭代器模式中，客户代码需要的迭代器的类取决于客户代码想要遍历的容器的类型。这种应用很常见。

假定 Oozinoz 公司允许顾客通过借贷形式购买焰火产品。在早期的信贷授权系统中，你开发一个名为 `CreditCheckOnline` 的类，用于在线核对系统是否允许为客户在 Oozinoz 公司提供信用额度。

开始开发时，你意识到信贷代理机构有时并不在线。此时，项目分析者认为你需要为呼叫中心的代理建立一个对话框（传递信贷请求），通过设置一些问题使其自动做出信贷决定。因此，你创建了 `CreditCheckOffline` 类，使它能够根据某些规定被使用。最初，你的类设计如图 16.1 所示。其中，`creditLimit()`方法接收客户的身份证号码，返回此客户的信贷限额。

有了图 16.1 中设计的类，不管信贷代理机构是否在线，你都可以为客户提供信贷限额信息。现在的问题是，这些类的使用者需要知道究竟实例化了哪个类。然而，事实上却只有你知道信贷代理机构是否在线。

在这种场景下，需要把创建对象的工作转交给一个接口，来完成实例化的控制。一种解决方案是让这两个类同时实现一个标准接口，并创建一个工厂方法来返回该接口的某个实例。具体做法如下：

- 创建一个名为 `CreditCheck` 的接口，使其包含 `creditLimit()`方法。

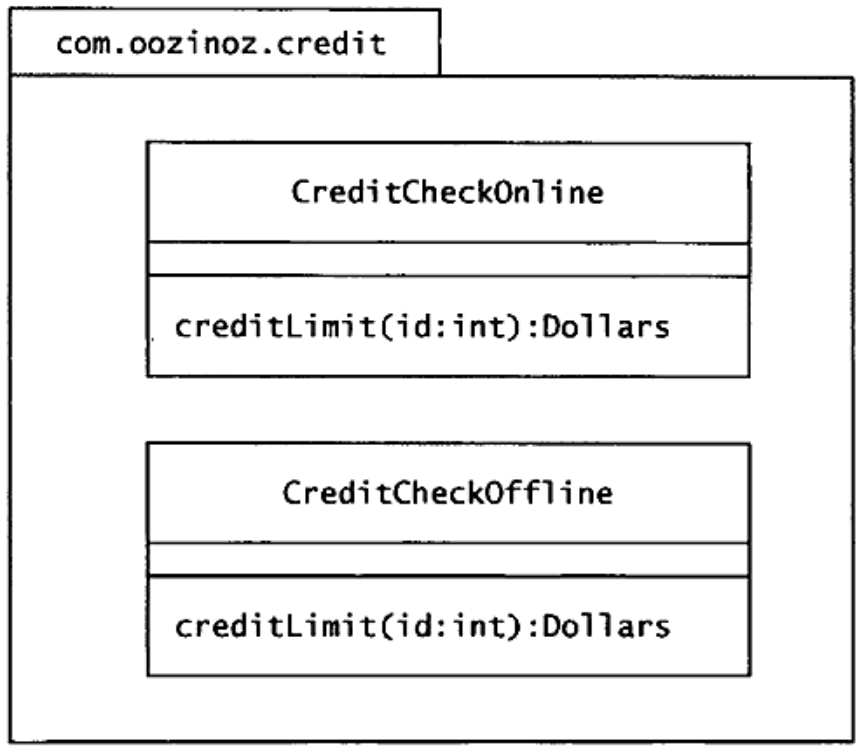

图 16.1 你可以判断信贷代理机构是否在线，从而决定实例化图中的哪一个类，以此核对客户的信贷信息

- 使所有信贷核对类声明实现 CreditCheck 接口。
- 创建一个名为 CreditCheckFactory 的类，此类包含一个名为 createCreditCheck() 的方法，并返回一个类型为 CreditCheck 的对象。

在实现 createCreditCheck()方法时，通过掌握的信贷代理机构是否在线的信息，你可以决定究竟实例化哪一个类。

挑战 16.4

为这个新设计设计一个类图，使其不仅可以创建信贷核对对象，还可以确定由哪个类去实例化对象。

答案参见第 335 页

通过应用工厂方法模式，使用此服务的用户可以通过调用 createCreditCheck()方法获得一个信贷核对的对象，这样不管信贷代理机构是否在线都可以满足顾客的服务请求。

挑战 16.5

假定 CreditCheckFactory 类有一个名叫 isAgencyUp()的方法，它确定信贷代理机构是否在线，请写出 createCreditCheck()的代码。

答案参见第 336 页

并行层次结构中的工厂方法模式

在使用并行层次结构对问题域进行建模时，常常会使用工厂方法模式。一个并行层次结构是一对类层次结构。其中，一个类在一个层次，与其相关的类在另一层次。当将现有类层次结构的部分行为移出该层次后，并行层次结构就会显露出来。

第 5 章的合成模式介绍了焰火弹的制造，Oozinoz 公司就是使用图 16.2 所示类图的机器生产这些焰火弹的。

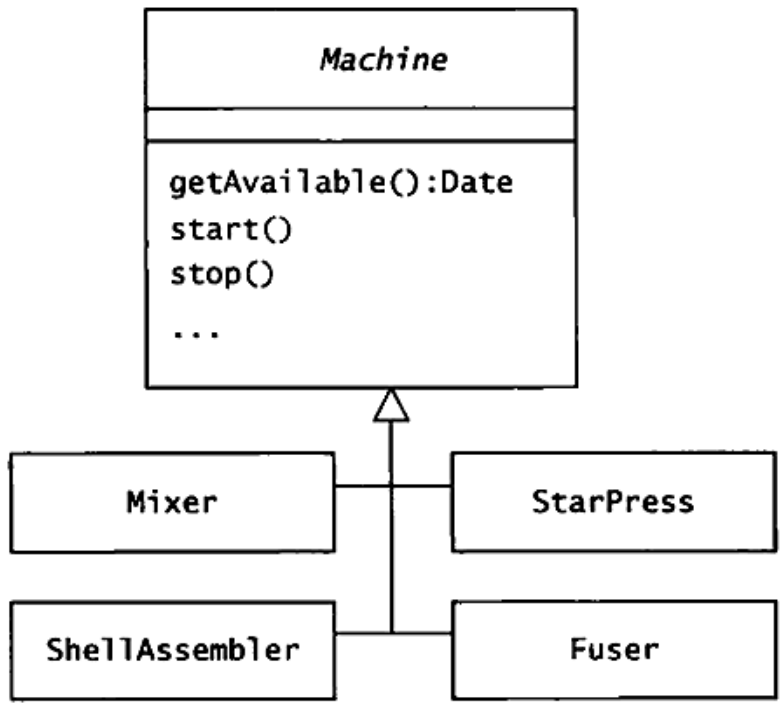

图 16.2　Machine 层次结构包含了控制物理机器和规划的逻辑

要生产一个焰火弹，需要混合相关的化学原料，然后将这些化学原料放入一个星形的压力

机中，这样混合过的原料就被挤压成了一个个星形的混合物。接着，使用一个外壳装备器，把这些星形的混合物装到一个具有黑色粉末内核的壳内，并且放置在推进药的上方。最后，再使用一根导火线插入到焰火弹中，一旦导火线被点燃，推进药和内核都会被引燃。

假定你想用 getAvailable()方法来预测机器完成加工的时间，从而决定是否可以执行接下来的工作。该方法需要某些私有方法的支持，总体而言，就是为每个机器类添加一定数量的逻辑。但是这些规划逻辑不应该添加到 Machine 的类层次结构中，更好的方式是独立出 MachinePlanner 层次结构。对于大多数机器类型而言，需要一个独立的计划类；但是，其他的工作总是需要用到混合器和导火线。此时，就可以使用 BasicPlanner 类。

挑战 16.6

完成图 16.3 所示的 Machine/MachinePlanner 并行层次结构图。

答案参见第 336 页

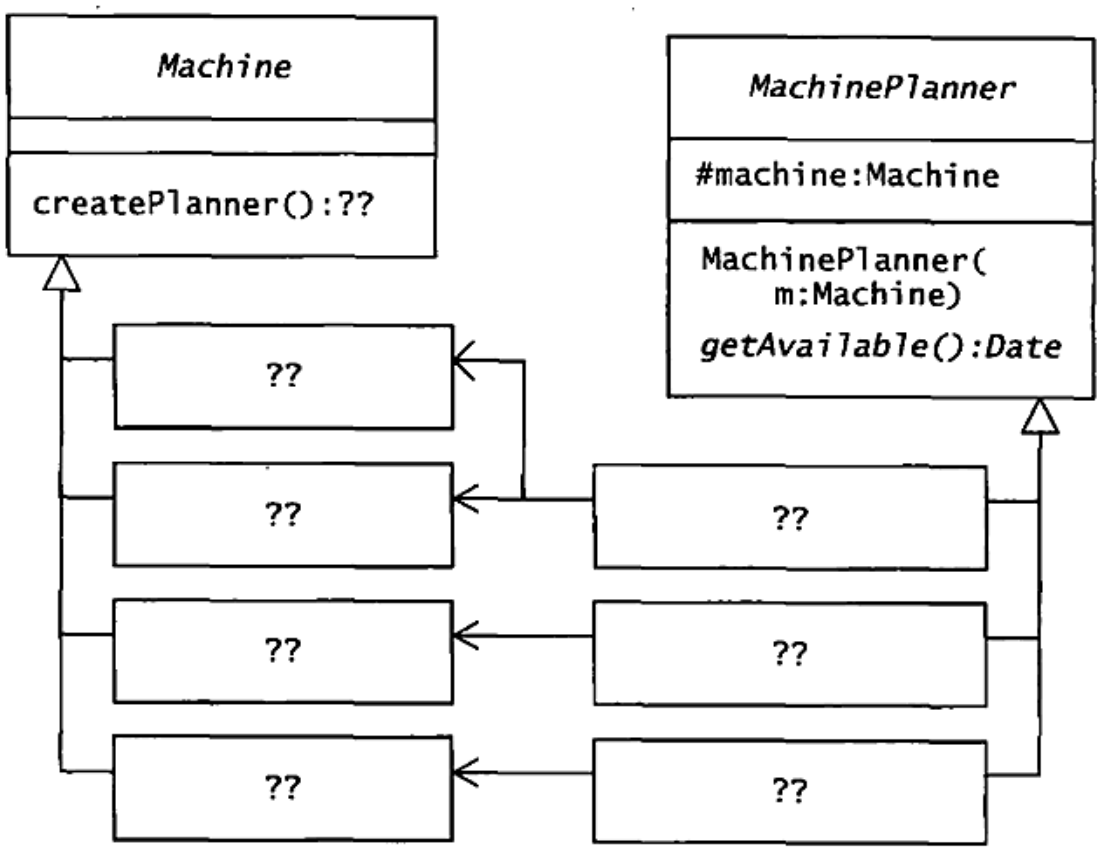

图 16.3　通过将规划逻辑移植到并行层次结构中，可以缩减 Machine 类的层次结构

挑战 16.7

为 Machine 类写一个 createPlanner()方法，使其返回一个 BasicPlanner 对象，并为 StarPress 类编写 createPlanner()方法。

答案参见第 337 页

小结

工厂方法模式的意图是让服务的提供者确定实例化哪个类，而不是客户代码。这种模式在 Java 类库中极为常见，比如 Collection 接口的 iterator()方法。

当你不想让客户代码决定实例化哪个类时，常常可以运用工厂方法模式。此外，工厂方法模式还可用于当客户代码不知道它需要创建哪个对象类的时候，例如客户代码无法获知信贷机构是否在线的情况。另外，在并行类层次结构中使用该模式可以避免类的规模过于庞大。工厂方法模式可以根据一个类层次中的子类，确定另一个相关层次中哪一个类被实例化，从而建立对应的并行层次结构。