【容器与内容的一致性】

在计算机的文件系统中，有“文件夹”的概念（在有些操作系统中，也称为“目录”）。文件夹里面既可以放入文件，也可以放入其他文件夹（子文件夹）。在子文件夹中，一样地既可以放入文件，也可以放入子文件夹。可以说，文件夹是形成了一种容器结构、递归结构。

我们接着再想一想。虽然文件夹与文件是不同类型的对象，但是它们都“可以被放入到文件夹中”。文件夹和文件有时也被统称为“目录条目”（directory entry）。在目录条目中，文件夹和文件被当作是同一种对象看待（即一致性）。

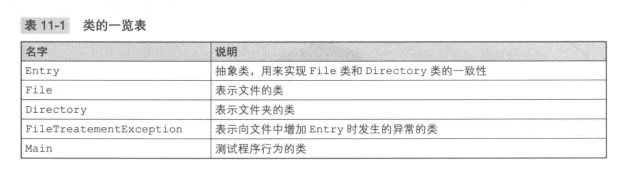
例如，想查找某个文件夹中有什么东西时，找到的可能是文件夹，也可能是文件。简单地说，找到的都是目录条目。

有时，与将文件夹和文件都作为目录条目看待一样，将容器和内容作为同一种东西看待，可以帮助我们方便地处理问题。在容器中既可以放入内容，也可以放入小容器，然后在那个小容器中，又可以继续放入更小的容器。这样，就形成了容器结构、递归结构。

在本章中，我们要学习的Composite模式就是用于创造出这样的结构的模式。能够使容器与内容具有一致性，创造出递归结构的模式就是Composite模式。Composite在英文中是“混合物”“复合物”的意思。

|11.2示例程序

下面我们来看一段Composite模式的示例程序。这段示例程序的功能是列出文件和文件夹的一览。在示例程序中，表示文件的是File类，表示文件夹的是Directory类，为了能将它们统一起来，我们为它们设计了父类Entry类。Entry类是表示“目录条目”的类，这样就实现了File 类和Directory类的一致性。



|11.3 composite 模式中的登场角色

在Composite模式中有以下登场角色。

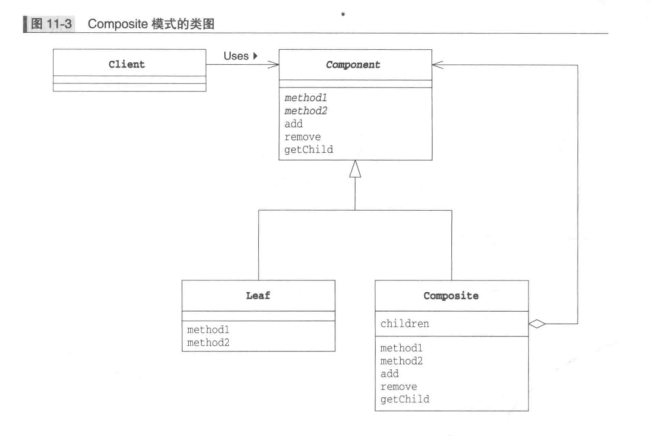
◆Leaf（树叶）表示“内容”的角色。在该角色中不能放入其他对象。在示例程序中，由File类扮演此角色。

·Composite（复合物）表示容器的角色。可以在其中放入Leaf角色和Composite角色。在示例程序中，由Directory类扮演此角色。

·Component使Leaf角色和Composite角色具有一致性的角色。Composite角色是Leaf角色和Composite角色的父类。在示例程序中，由Entry类扮演此角色。

·Client使用Composite模式的角色。在示例程序中，由Main类扮演此角色。

Composite模式的类图如图11-3所示。在该图中，可以将Composite角色与它内部的Component角色（即Leaf角色或Composite角色）看成是父亲与孩子们的关系。getchild方法的作用是从Component角色获取这些“孩子们”。



|11.4拓展思路的要点

|多个和单个的一致性

使用Composite模式可以使容器与内容具有一致性，也可以称其为多个和单个的一致性，即将多个对象结合在一起，当作一个对象进行处理。

例如，让我们试想一下测试程序行为时的场景。现在假设Testl是用来测试输入数据来自键盘输入时的程序的行为，Test2是用来测试输入数据来自文件时的程序的行为，Test3是用来测试输入数据来自网络时的程序的行为。如果我们想将这3种测试统一为“输入测试”，那么Composite模式就有用武之地了。我们可以将这几个测试结合在一起作为“输入测试”，或是将其他几个测试结合在一起作为“输出测试”，甚至可以最后将“输入测试”和“输出测试”结合在一起作为“输入输出测试”。

例如，在以下网址介绍的测试场景中，测试程序中使用了Composite模式。

·Kent Beck Testing Framework入门

http://objectclub，jp/community/memorial/homepage3.nifty.com/masar1/article/

testing-framework.html

·Simple Smaltalk Testing:With Patterms（by Kent Beck）

http://swing.fit.cvut.cz/projects/stx/doc/online/english/tools/misc/testfram.html

Add方法应该放在哪里

在示例程序中，Entry类中定义了add方法，所做的处理是抛出异常，这是因为能使用add方法的只能是Directory类。下面我们学习一下各种add方法的定义位置和实现方法。

◆方法1：定义在Entry类中，报错

将add方法定义在Entry类中，让其报错，这是示例程序中的做法。能使用add方法的只有Directory类，它会重写add方法，根据需求实现其处理。

File类会继承Entry类的add方法，虽然也可以调用它的add方法，不过会抛出异常。

·方法2：定义在Entry类中，但什么都不做

也可以将add方法定义在Entry类中，但什么处理都不做。

·方法3：声明在Entry类中，但不实现

也可以在Entry类中声明add抽象方法。如果子类需要add方法就根据需求实现该方法，如果不需要add方法，则可以简单地报错。该方法的优点是所有子类必须都实现add方法，不需要add方法时的处理也可以交给子类自己去做决定。不过，使用这种实现方法时，在File一方中也必须定义本来完全不需要的add（有时还包括remove和getchild）方法。

◆方法4：只定义在Directory类中

因为只有Directory类可以使用add方法，所以可以不在Entry类中定义add方法，而是只将其定义在Directory类中。不过，使用这种方法时，如果要向Entry类型的变量（实际保存的是Directory类的实例）中add时，需要先将它们一个一个地类型转换（cast）为Directory类型。

到处都存在递归结构

在示例程序中，我们以文件夹的结构为例进行了学习，但实际上在程序世界中，到处都存在递归结构和Composite模式。例如，在视窗系统中，一个窗口可以含有一个子窗口，这就是Composite模式的典型应用。此外，在文章的列表中，各列表之间可以相互嵌套，这也是一种递归结构。将多条计算机命令合并为一条宏命令时，如果使用递归结构实现宏命令，那么还可以编写出宏命令的宏命令。另外，通常来说，树结构的数据结构都适用Composite模式。

|11.5相关的设计模式

◆Command模式（第22章）使用Command模式编写宏命令时使用了Composite模式。

·Visitor模式（第13章）可以使用Visitor模式访问Composite模式中的递归结构。

◆Decorator模式（第12章）Composite模式通过Component角色使容器（Composite角色）和内容（Leaf角色）具有一致性。

Decorator模式使装饰框和内容具有一致性。