# 组合模式简介

## 概述

1. 组合模式关注那些包含叶子构件和容器构件的结构以及它们的的组织形式，在叶子构件中不包含成员对象，而容器构件成员对象（这里的成员对象指的就是叶子构件和容器构件），这些对象通过递归组合可构成一个树形结构。
2. 在树形结构中（如操作系统中的目录结构、应用软件中的菜单结构、办公系统中的公司组织结构），都蕴含者组合模式的道理；如在操作系统中的目录结构中，文件夹为容器（Container），不同类型的文件是其成员，称为叶子（Leaf）；对于所有与目录结构相类似的树形结构，当容器对象（例如文件夹）的某一个方法被调用时将遍历整个树形结构，寻找也包含这个方法的成员对象（可以容器对象，可以是叶子对象，例如子文件夹和文件）并调用执行，其中会使用递归的机制对整个结构进行处理。由于容器对象和叶子对象在功能上的区别，在使用这些对象的代码中必须有区别地对待容器对象和叶子对象，而实际上在大多数情况下客户端希望一致地处理它们，因为对于这些对象的区别对待将会使程序非常复杂。组合模式使得用户可以一致性地处理整个树形结构或者树形结构的一部分，描述了如何将容器对象和叶子对象进行递归组合，使得用户在使用时无需对它们进行区分，可以一致地对待容器对象和叶子对象，这就是组合模式的动机。
3. 组合模式（Composite Pattern）的定义如下：组合多个对象形成树形结构以表示具有部分-整体关系的层次结构。组合模式让客户端可以统一对待单个对象和组合对象。
4. 组合模式为叶子构件和容器构件提供了一个公共的抽象构件类，客户端可以针对抽象构件进行编程。
5. 组合模式又称“部分-整体”模式，属于对象结构型模式，它将对象组织到树形结构中，可以用来描述整体与部分的关系。

## 结构

组合模式的结构图如下所示：



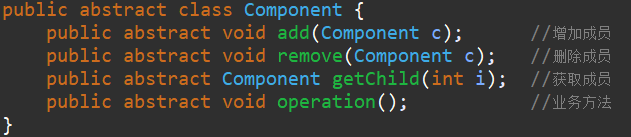
组合模式包含以下3个角色：

1. Component（抽象构件）：它可以是接口或抽象类，为叶子构件和容器构件对象声明接口，让叶子构件或容器构件去继承或实现；在该角色中包含所有子类共有行为的声明和实现；在抽象构件中定义了访问及管理它的子构件（容器构件和叶子构件）的方法，如增加子构件、删除子构件、获取子构件等。
2. Leaf（叶子构件）：抽象构件的子类，它在组合模式中表示叶子结点对象，叶子结点没有子结点，它实现了在抽象构件中定义的行为；对于那些访问及管理子构件的方法，可以通过抛出异常、提示错误等方式进行处理。
3. Composite（容器构件）：抽象构件的子类，它在组合结构中表示容器结点对象，容器结点包含子结点，其子结点可以是叶子结点，也可以是容器结点；它提供一个集合用于存储子结点；实现了在抽象构件中定义的行为，包括那些访问及管理子构件的方法；在其业务方法中可以递归调用其子结点的业务方法。

# 实现

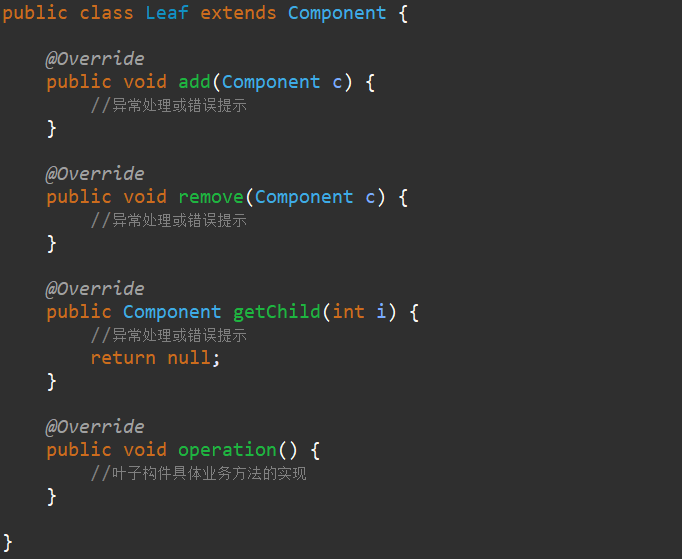
## 实现原理

1. 组合模式的关键在于定义一个抽象构件类，它既可以代表叶子，又可以代表容器，客户端针对该抽象构件类进行编程，无需知道它到底表示的是叶子还是容器，可以对其进行统一处理。同时容器对象类与抽象构件类之间还建立一个聚合关联关系，在容器对象中既可以包含叶子，又可以包含容器，以此实现递归组合，形成树形结构。
2. 如果不使用组合模式，客户端代码将过多地依赖于容器对象复杂的内部实现结构，容器对象内部实现结构的变化将引起客户代码的频繁变化，造成代码维护困难、可扩展性差等问题，组合模式的使用将在一定程度上解决这些问题。
3. 对于组合模式中的抽象构件角色，其典型代码如下：



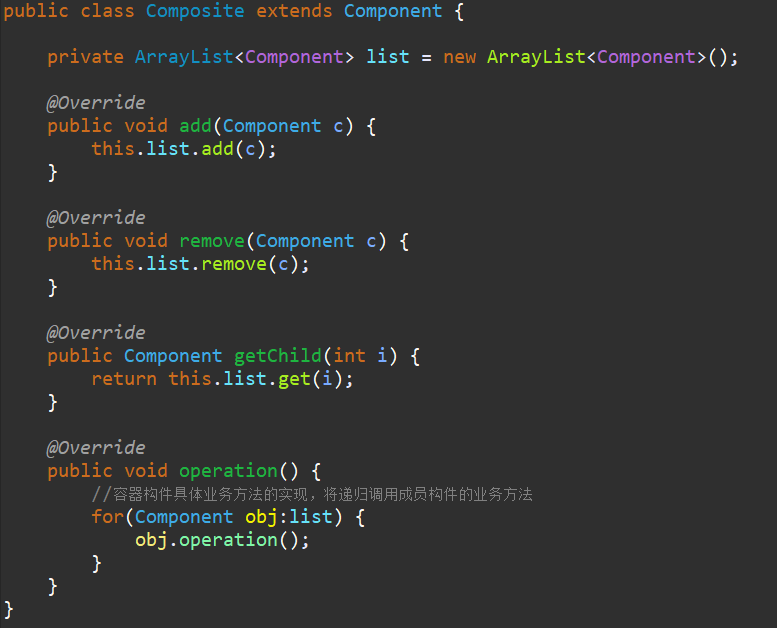
一般将抽象构件类设计为接口或抽象类，将所有子类共有方法的声明和实现定义在抽象构件类中；对于客户端而言，将针对抽象构件编程，而无须关心其具体子类是容器构件还是叶子构件。

1. 对于组合模式中的叶子构件，其典型代码如下：



作为抽象构件类的子类，在叶子构件中需要实现在抽象构件类中声明的所有抽象方法，包括业务方法以及管理和访问子构件的方法，但是叶子构件不能再包含子构件，因此在叶子构件中实现子构件管理和访问的方法时需要提供异常处理或错误提示。显然，这会给叶子构件的实现带来麻烦。

1. 对于组合模式中的容器构件，其典型代码如下：



在容器构件中实现了抽象构件中声明的所有方法，即包括业务方法，也包括用于访问和管理成员子构件的方法，例如上述代码中的add（）、remove（）和getChild（）等方法；

需要注意的是在实现具体业务方法时由于容器构件充当的是容器角色，包含成员构件（容器构件或叶子构件），因此它将递归调用其成员构件的业务方法，例如上述代码中的operation（）方法中的代码；

在组合模式结构中，由于容器构件中仍然可以包含容器构件，所以定义了一个集合用于保存成员构件，在对容器构件进行处理时需要使用递归算法，即在容器构件的operator（）方法中递归调用其成员构件的operator（）方法。

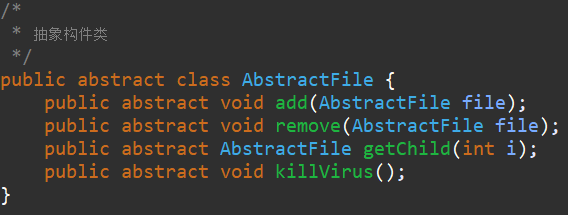
## 实例

实例说明：某软件公司要开发一个杀毒软件（Antivirus）软件，该软件既可以对某个文件夹（Folder）杀毒，也可以对某个指定的文件（File）杀毒。该杀毒软件还可以根据各类文件的特点为不同类型的文件提供不同的杀毒方式，例如图像文件（ImageFile）和文本文件（TextFile）的杀毒方式就有所差异。先使用组合模式来设计该杀毒软件的整体框架。

AbstractFile充当抽象构件类，Folder充当容器构件类，ImageFile、TextFile、VideoFile充当叶子构件类，结构图如下：



1. AbstractFile：抽象文件类，充当抽象构件类。



1. ImageFile：图像文件类，充当叶子构件类



1. TextFile：文本文件类，充当叶子构件类



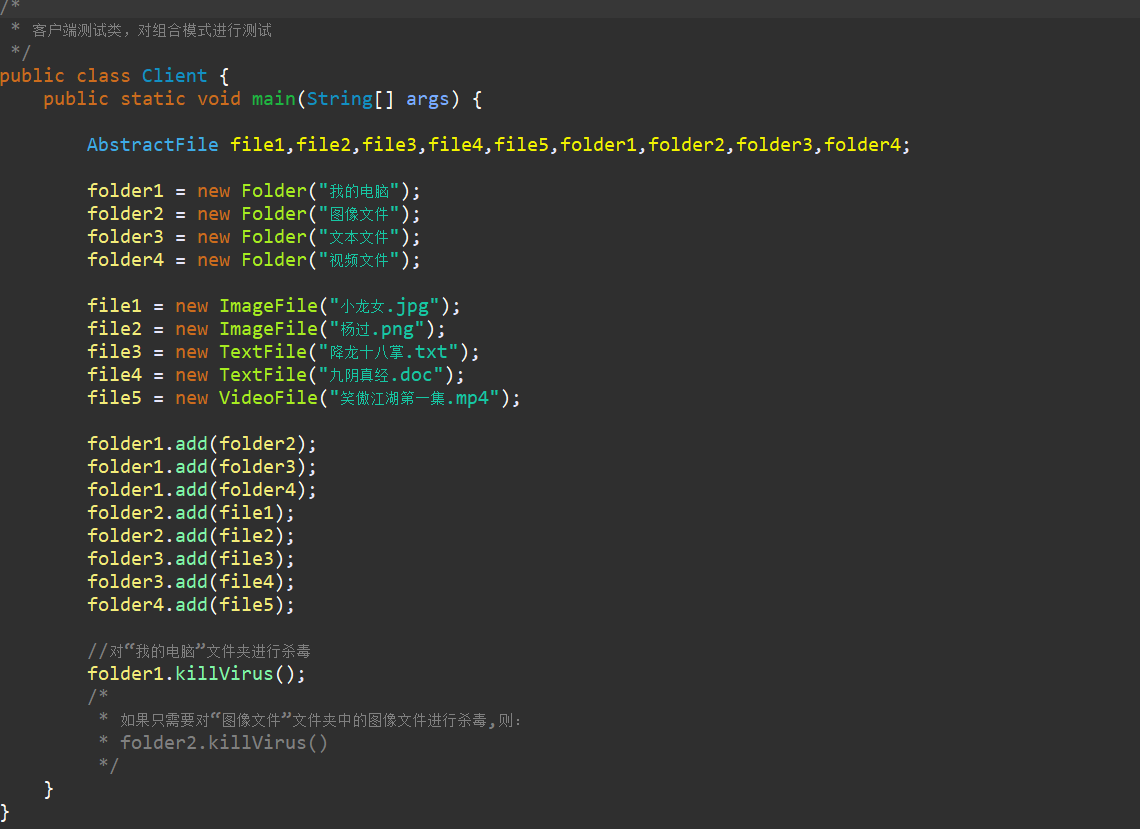
1. Video：视频文件类，充当叶子构件类



1. Folder：文件夹类，充当容器构件类



1. Client：客户端测试类



# 透明组合模式与安全组合模式

组合模式根据抽象构件类的定义形式又分为透明组合模式和安全组合模式。

## 透明组合模式

在透明组合模式中，抽象构件Component中声明了所有用于管理成员对象的方法，如add（）、remove（）以及getChild（）等方法。其结构如下图所示：



这样做的好处是确保所有的构件类都有相同的接口。在客户端看来，叶子对象与容器对象所提供的方法是一致的，客户端可以一致地对待所有的对象。

透明组合模式的缺点是不够安全，因为叶子对象和容器对象在本质上是有区别的，叶子对象不可能有下一层次的对象，即不可能包含成员对象（叶子对象或容器对象），因此为其提供add（）、remove（）以及getChild（）等方法是没有意义的，这在编译阶段不会出错，但在运行阶段如果调用这些方法可能会出错（如果没有提供相应的错误处理代码）。

## 安全组合模式

在安全组合模式中，抽象构件Component中没有声明任何用于管理成员对象的方法，而是在容器构件Composite中声明并实现这些方法。这种做法是安全的，因为根本不向叶子对象提供这些管理成员对象的方法，对于叶子对象，客户端不可能调用到这些方法。其结构图如下所示：



安全组合模式的缺点是不够透明，因为叶子构件和容器构件具有不同的方法，且容器构件中那些用于管理成员对象的方法没有在抽象构件类中定义，因此客户端不能完成针对抽象构件类编程，必须有区别地对待叶子构件和容器构件。

# 优缺点及适用环境

组合模式使用面向对象的思想来实现树形结构的构建与处理，描述了如何将容器对象和叶子对象进行递归组合。Java SE中的AWT和Swing包的设计就是基于组合模式。

## 优点

组合模式的优点主要如下：

1. 可以清楚地定义分层次的复杂对象，表示对象的全部或部分层次，它让客户端忽略了层次的差异，方便对整个层次结构进行控制。
2. 客户端可以一致地使用一个组合结构或其中单个对象，不必关心处理的是单个对象还是组合结构，简化了客户端的代码。
3. 在组合模式中增加新的容器构件和叶子构件都很方便，无须对现有类库进行任何修改，符合开闭原则。
4. 为树形结构的面向对象实现提供了一种灵活的解决方案，通过叶子对象和容器对象的递归组合可以形成复杂的树形结构，但对树形结构的控制却非常简单。

## 缺点

组合模式的缺点主要是：在增加新构件时很难对容器中构件类型进行限制。有时候希望一个容器中只能有某个特定类型的对象，例如在文件夹中只能包含文本文件，在使用组合模式时不能依赖类型系统来施加这些约束，因为它们都来自于相同的抽象层，在这种情况下必须通过在运行时进行类型检查来实现，这个实现过程较为复杂。

## 适用环境

在以下情况下可以考虑使用组合模式：

1. 在具有整体和部分的层次结构中希望通过一种方式忽略整体与部分的差异，客户端可以一致地对待它们。
2. 在一个使用面向对象语言开发的系统中需要处理一个树形结构。
3. 在一个系统中能够分离出叶子对象和容器对象，而且它们的类型不固定，需要增加一些新的类型。