私塾在线《研磨设计模式》
——跟着CC学设计系列精品教程

本节课程概览

- 学习组合模式

一：初识组合模式

包括：定义、结构、参考实现

二：体会组合模式

包括：场景问题、不用模式的解决方案、使用模式的解决方案

三：理解组合模式

包括：认识组合模式、安全性和透明性、父组件引用 、环状引用 、组合模式的优缺点

四：思考组合模式

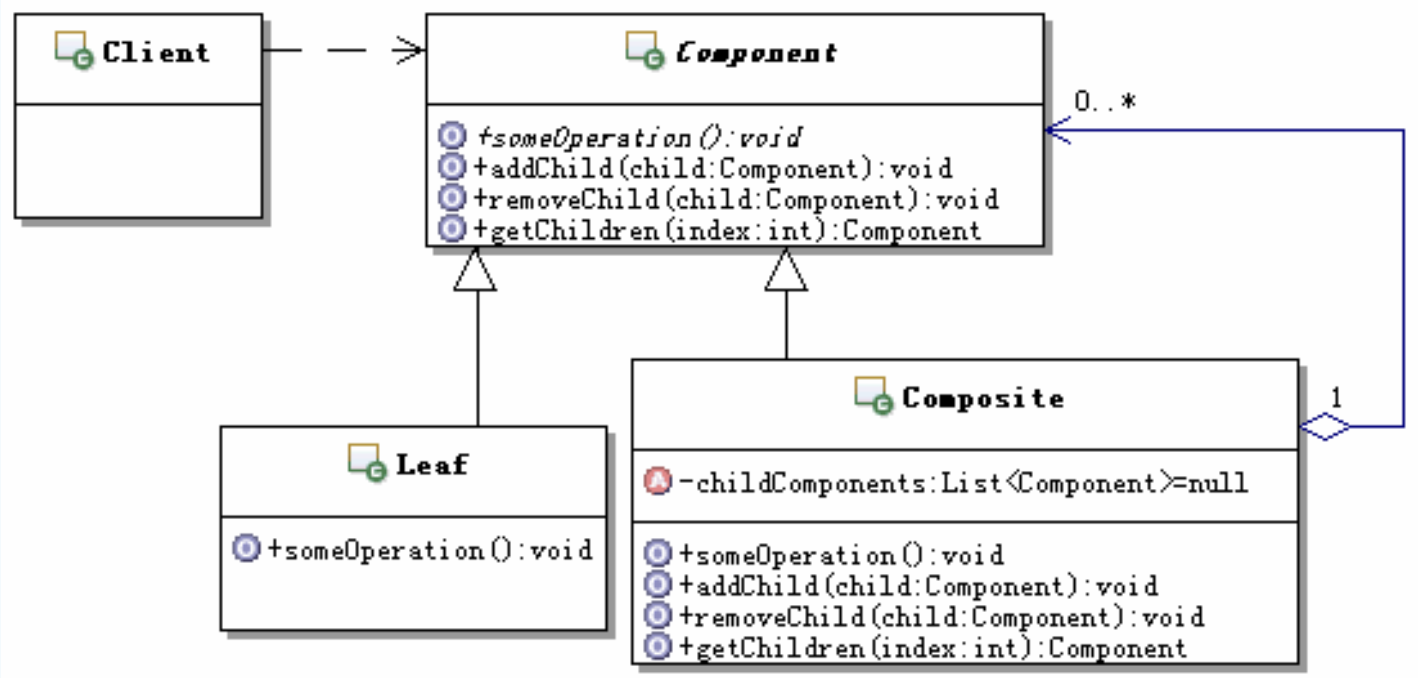
包括：组合模式的本质、何时选用

初识组合模式

- 定义

  将对象组合成树形结构以表示“部分-整体”的层次结构。组合模式使得用户对单个对象和组合对象的使用具有一致性 。

- 结构和说明



初识组合模式

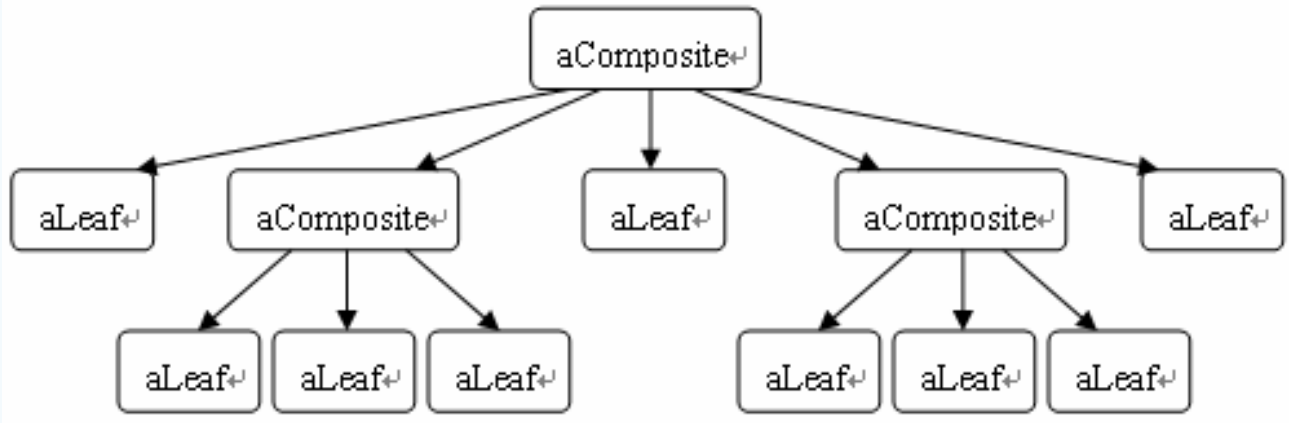
Component：抽象的组件对象，为组合中的对象声明接口，让客户端可以通过这个接口来访问和管理整个对象结构，可以在里面为定义的功能提供缺省的实现。、

Leaf：叶子节点对象，定义和实现叶子对象的行为，不再包含其它的子节点对象。

Composite：组合对象，通常会存储子组件，定义包含子组件的那些组件的行为，并实现在组件接口中定义的与子组件有关的操作。

Client：客户端，通过组件接口来操作组合结构里面的组件对象。

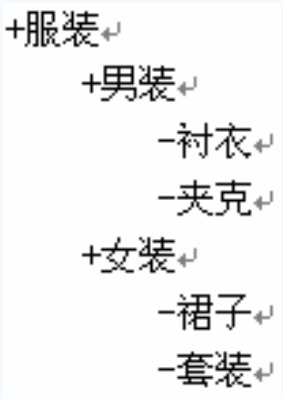
一种典型的Composite对象结构通常是如图所示的树形结构



体会组合模式

- 商品类别树

在实现跟商品有关的应用系统的时候，一个很常见的功能就是商品类别树的管理，比如有如下所示的商品类别树：



仔细观察上面的商品类别树，有以下几个明显的特点：

1：有一个根节点，比如服装，它没有父节点，它可以包含其它的节点

2：有一类节点可以包含其它的节点，称之为树枝节点，比如男装、女装

3：有一类节点没有子节点，称之为叶子节点，比如衬衣、夹克、裙子、套装

现在要求能实现输出如上商品类别树的结构的功能，应该如何实现呢？

体会组合模式

- 不用模式的解决方案

要管理商品类别树，就是要管理树的各个节点，现在树上的节点有三类，根节点、树枝节点和叶子节点，再进一步分析发现，根节点和树枝节点是类似的，都是可以包含其它节点的节点，把它们称为容器节点。

这样一来，商品类别树的节点就被分成了两种，一种是容器节点，另一种是叶子节点。容器节点可以包含其它的容器节点或者叶子节点。把它们分别实现成为对象，也就是容器对象和叶子对象，容器对象可以包含其它的容器对象或者叶子对象，换句话说，容器对象是一种组合对象。

体会组合模式

- 有何问题

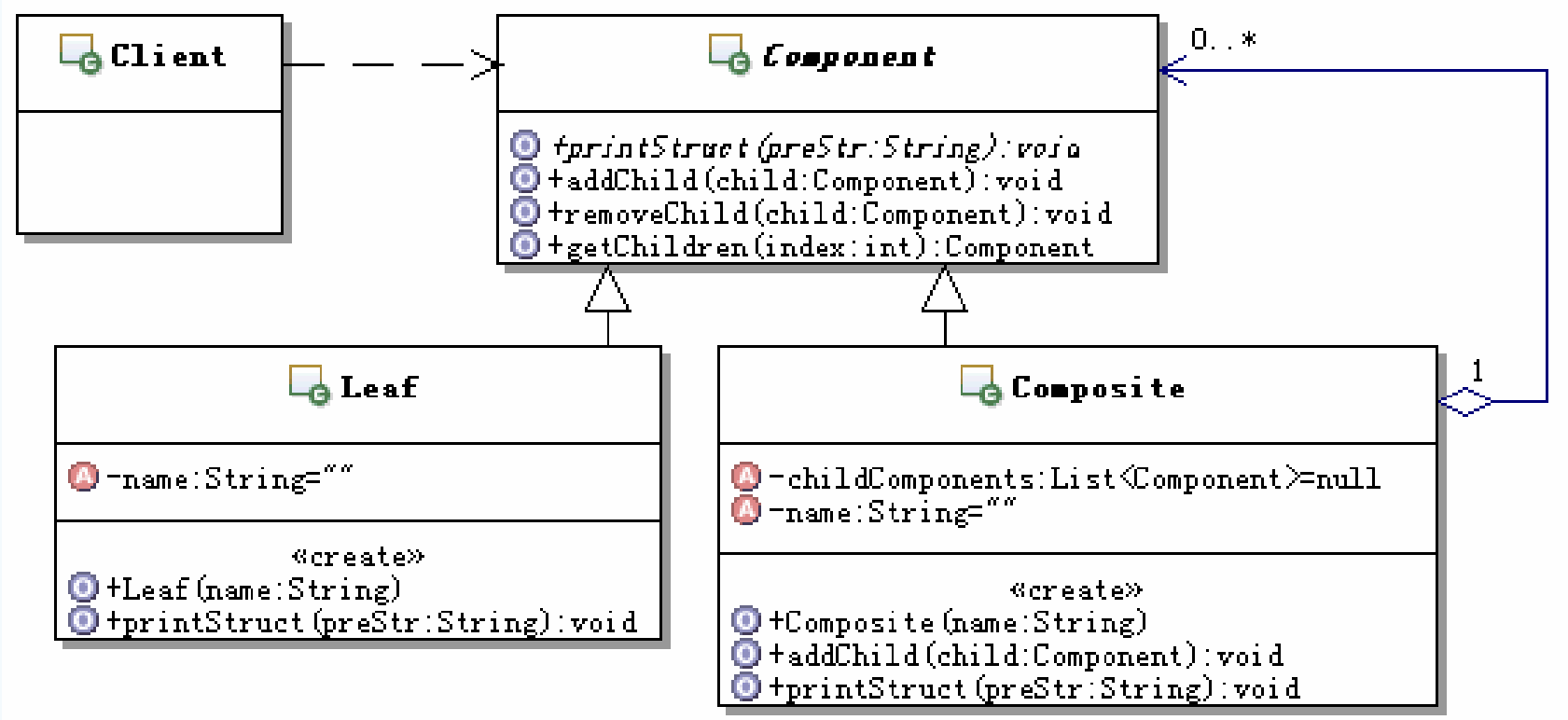
上面的实现，虽然能实现要求的功能，但是有一个很明显的问题：那就是必须区分组合对象和叶子对象，并进行有区别的对待，比如在Composite和Client里面，都需要去区别对待这两种对象。

区别对待组合对象和叶子对象，不仅让程序变得复杂，还对功能的扩展也带来不便。实际上，大多数情况下用户并不想要去区别它们，而是认为它们是一样的，这样他们操作起来最简单。

换句话说，对于这种具有整体与部分关系，并能组合成树形结构的对象结构，如何才能够以一个统一的方式来进行操作呢？

体会组合模式

- 使用模式的解决方案的类图



理解组合模式

- 认识组合模式

1：组合模式的目的

组合模式的目的是：让客户端不再区分操作的是组合对象还是叶子对象，而是以一个统一的方式来操作。

实现这个目标的关键之处，是设计一个抽象的组件类，让它可以代表组合对象和叶子对象。这样一来，客户端就不用区分到底是组合对象还是叶子对象了，只需要全部当成组件对象进行统一的操作就可以了。

2：对象树

通常，组合模式会组合出树形结构来，组成这个树形结构所使用的多个组件对象，就自然的形成了对象树。

这也意味着凡是可以使用对象树来描述或操作的功能，都可以考虑使用组合模式，比如读取XML文件，或是对语句进行语法解析等。

理解组合模式

3：组合模式中的递归

组合模式中的递归，指的是对象递归组合，不是常说的递归算法。

而这里的组合模式中的递归，是对象本身的递归，是对象的组合方式，是从设计上来讲的，在设计上称作**递归关联**，是对象关联关系的一种

4：Component中是否应该实现一个Component列表

大多数情况下，一个Composite对象会持有子节点的集合。有些朋友可能就会想，那么能不能把这个子节点集合定义到Component中去呢？因为在Component中还声明了一些操作子节点的方法，这样一来，大部分的工作就可以在Component中完成了。

事实上，这种方法是不太好的，因为在父类来存放子类的实例对象，对于Composite节点来说没有什么，它本来就需要存放子节点，但是对于叶子节点来说，就会导致空间的浪费，因为叶节点本身不需要子节点。

因此只有当组合结构中叶子对象数目较少的时候，才值得使用这种方法。

理解组合模式

5：最大化Component定义

前面讲到了组合模式的目的是：让客户端不再区分操作的是组合对象还是叶子对象，而是以一种统一的方式来操作。

由于要统一两种对象的操作，所以Component里面的方法也主要是两种对象对外方法的和，换句话说，有点大杂烩的意思，组件里面既有叶子对象需要的方法，也有组合对象需要的方法。

这种实现是与类的设计原则相冲突的，类的设计有这样的原则：一个父类应该只定义那些对它的子类有意义的操作。看看上面的实现就知道，Component中的有些方法对于叶子对象是没有意义的。那么怎么解决这一冲突呢？

常见的做法是在Component里面为对某些子对象没有意义的方法，提供默认的实现，或是默认抛出不支持该功能的例外。如果子对象需要这个功能，那就覆盖实现它，如果不需要，那就不用管了，使用父类的默认实现就可以了。

从另一个层面来说，如果把叶子对象看成是一个特殊的Composite对象，也就是没有子节点的组合对象而已。这样看来，对于Component而言，子对象就全部看作是组合对象，因此定义的所有方法都是有意义的了。

理解组合模式

6：子部件排序

在某些应用中，使用组合模式的时候，需要按照一定的顺序来使用子组件对象，比如进行语法分析的时候，使用组合模式构建的抽象语法树，在解析执行的时候，是需要按照顺序来执行的。

对于这样的功能，需要在设计的时候，就要把组件对象的索引考虑进去，并仔细的设计对子节点的访问和管理接口，通常的方式是需要按照顺序来存储，这样在获取的时候就可以按照顺序得到了。可以考虑结合Iterator模式来实现按照顺序的访问组件对象。

理解组合模式

- 安全性和透明性

有一个很重要的问题：到底在组合模式的类层次结构中，在哪一些类里面定义这些管理子组件的操作，到底应该在Component中声明这些操作，还是在Composite中声明这些操作？

这就需要仔细思考，在不同的实现中，进行安全性和透明性的权衡选择。

1：这里所说的安全性是指：从客户使用组合模式上看是否更安全。如果是安全的，那么不会有发生误操作的可能，能访问的方法都是被支持的功能。

2：这里所说的透明性是指：从客户使用组合模式上，是否需要区分到底是组合对象还是叶子对象。如果是透明的，那就是不再区分，对于客户而言，都是组件对象，具体的类型对于客户而言是透明的，是客户无需要关心的。

理解组合模式

- 透明性的实现

如果把管理子组件的操作定义在Component中，那么客户端只需要面对Component，而无需关心具体的组件类型，这种实现方式就是透明性的实现。事实上，前面示例的实现方式都是这种实现方式。

但是透明性的实现是以安全性为代价的，因为在Component中定义的一些方法，对于叶子对象来说是没有意义的，比如：增加、删除子组件对象。而客户不知道这些区别，对客户是透明的，因此客户可能会对叶子对象调用这种增加或删除子组件的方法，这样的操作是不安全的。

组合模式的透明性实现，通常的方式是：在Component中声明管理子组件的操作，并在Component中为这些方法提供缺省的实现，如果是有子对象不支持的功能，缺省的实现可以是抛出一个例外，来表示不支持这个功能。

理解组合模式

- 安全性的实现

  如果把管理子组件的操作定义在Composite中，那么客户在使用叶子对象的时候，就不会发生使用添加子组件或是删除子组件的操作了，因为压根就没有这样的功能，这种实现方式是安全的。

  但是这样一来，客户端在使用的时候，就必须区分到底使用的是Composite对象，还是叶子对象，不同对象的功能是不一样的。也就是说，这种实现方式，对客户而言就不是透明的了。

理解组合模式

- 两种实现方式的选择

对于组合模式而言，在安全性和透明性上，会更看重透明性，毕竟组合模式的功能就是要让用户对叶子对象和组合对象的使用具有一致性。

而且对于安全性的实现，需要区分是组合对象还是叶子对象，但是有的时候，你需要将对象进行类型转换，却发现类型信息丢失了，只好强行转换，这种类型转换必然是不够安全的。

对于这种情况的处理方法是在Component里面定义一个getComposite的方法，用来判断是组合对象还是叶子对象，如果是组合对象，就返回组合对象，如果是叶子对象，就返回null，这样就可以先判断，然后再强制转换。

因此在使用组合模式的时候，建议多用透明性的实现方式，而少用安全性的实现方式。

理解组合模式

- 父组件引用

  所谓父组件引用就是指：子组件对象到父组件对象的引用

  这个在实际开发中也是非常有用的，比如：

1：现在要删除某个商品类别。如果这个类别没有子类别的话，直接删除就好了，没有太大的问题，但是如果它还有子类别，这就涉及到它的子类别如何处理了，一种情况是连带全部删除，一种是上移一层，把被删除的商品类别对象的父商品类别，设置成为被删除的商品类别的子类别的父商品类别。

2：现在要进行商品类别的细化和调整，把原本属于A类别的一些商品类别，调整到B类别里面去，某个商品类别的调整会伴随着它所有的子类别一起调整。这样的调整可能会：把原本是兄弟关系的商品类别变成父子关系，也可能会把原本是父子关系的商品类别调整成了兄弟关系，如此等等会有很多种可能

理解组合模式

要实现上述的功能，一个较为简单的方案就是在保持从父组件到子组件引用的基础上，再增加保持从子组件到父组件的引用，这样在删除一个组件对象或是调整一个组件对象的时候，可以通过调整父组件的引用来实现，这可以大大简化实现。

通常会在Component中定义对父组件的引用，组合对象和叶子对象都可以继承这个引用。那么什么时候来维护这个引用呢？

较为容易的办法就是：在组合对象添加子组件对象的时候，为子组件对象设置父组件的引用；在组合对象删除一个子组件对象的时候，再重新设置相关子组件的父组件引用。把这些实现到Composite中，这样所有的子类都可以继承到这些方法，从而更容易的维护子组件到父组件的引用。

理解组合模式

- 环状引用

所谓环状引用指的是：在对象结构中，某个对象包含的子对象，或是子对象的子对象，或是子对象的子对象的子对象……,如此经过N层后，出现所包含的子对象中有这个对象本身，从而构成了环状引用。比如：A包含B，B包含C，而C又包含了A，转了一圈，转回来了，就构成了一个环状引用。

那么该如何检测是否有环状引用的情况发生呢？

一个很简单的思路就是记录下每个组件从根节点开始的路径，因为要出现环状引用，在一条路径上，某个对象就必然会出现两次。因此只要每个对象在整个路径上只是出现了一次，那么就不会出现环状引用。

理解组合模式

- 组合模式的优缺点
  1：定义了包含基本对象和组合对象的类层次结构
  2：统一了组合对象和叶子对象
  3：简化了客户端调用
  4：更容易扩展
  5：很难限制组合中的组件类型

思考组合模式

- 组合模式的本质

  组合模式的本质是：**统一叶子对象和组合对象**

- 何时选用组合模式

1：如果你想表示对象的部分-整体层次结构，可以选用组合模式，把整体和部分的操作统一起来，使得层次结构实现更简单，从外部来使用这个层次结构也简单。

2：如果你希望统一的使用组合结构中的所有对象，可以选用组合模式，这正是组合模式提供的主要功能