本文为 Android设计模式源码中 组合模式 分析
Android系统版本： 5.0
分析者：Tiny-Times，分析状态：未完成，校对者：无，校对状态：未开始

1. 模式介绍

模式的定义

组合模式(Composite Pattern)又叫作部分-整体模式，它使我们树型结构的问题中，模糊了简单元素和复杂元素的概念，客户程序可以向处理简单元素一样来处理复杂元素,从而使得客户程序与复杂元素的内部结构解耦。 GoF在《设计模式》一书中这样定义组合模式：将对象组合成树形结构以表示“部分-整体”的层次结构。使得用户对单个对象和组合对象的使用具有一致性。

模式的使用场景

- 表示对象的部分-整体层次结构。
- 从一个整体中能够独立出部分模块或功能的场景。

2. UML类图

角色分析

- Component抽象构件角色 ：定义参加组合对象的共有方法和属性，可以定义一些默认的行为或属性。
- Leaf叶子构件 : 叶子对象，其下再也没有其他的分支。
- Composite树枝构件 ：树枝对象，它的作用是组合树枝节点和叶子节点形成一个树形结构。

3. 该模式的实现实例

抽象构件 Component.java：

```
 public abstract class Component {
    //个体和整体都具有的共享
   public void doSomething(){
    //业务逻辑
   }
 }
```

树枝构件 Composite.java

```
public class Composite extends Component {
   //构件容器
   private ArrayList<Component> componentArrayList = new ArrayList<Component>();
   //增加一个叶子构件或树枝构件
   public void add(Component component){
      this.componentArrayList.add(component);
   }
   //删除一个叶子构件或树枝构件
   public void remove(Component component){
      this.componentArrayList.remove(component);
   }
   //获得分支下的所有叶子构件和树枝构件
   public ArrayList<Component> getChildren(){
     return this.componentArrayList;
   }
 }
```

树叶构件 Leaf.java

```
public class Leaf extends Component {

   //可以覆写父类方法
   public void doSomething(){
   }

}
```

场景类 Client.java

```
public class Client {
    public static void main(String[] args) {
       //创建一个根节点
      Composite root = new Composite();
      root.doSomething();
      //创建一个树枝构件
      Composite branch = new Composite();
      //创建一个叶子节点
      Leaf leaf = new Leaf();
      //建立整体
      root.add(branch);
      branch.add(leaf);
   }
   //通过递归遍历树
   public static void display(Composite root){
      for(Component c:root.getChildren()){
         if(c instanceof Leaf){ //叶子节点
             c.doSomething();
         }else{ //树枝节点
             display((Composite)c);
            }
        }
   }
}
```

组合模式在Android源码中的应用

Adnroid系统中采用组合模式的组合视图类图：

具体实现代码

View.java

```
public class View ....{
 //此处省略无关代码...
}
```

ViewGroup.java

```
public abstract class ViewGroup extends View ...{

     //增加子节点
    public void addView(View child, int index) {

    }
    //删除子节点
    public void removeView(View view) {

    }
     //查找子节点
    public View getChildAt(int index) {
      try {
            return mChildren[index];
          } catch (IndexOutOfBoundsException ex) {
            return null;
          }
   }
}
```

4. 注意事项

使用组合模式组织起来的对象具有出色的层次结构，每当对顶层组合对象执行一个操作的时候，实际上是在对整个结构进行深度优先的节点搜索。但是这些优点都是用操作的代价换取的，比如顶级每执行一次 store.show 实际的操作就是整一颗树形结构的节点均遍历执行一次。

5. 总结

5. 总结

优点

- 不破坏封装，整体类与局部类之间松耦合，彼此相对独立 。
- 具有较好的可扩展性。
- 支持动态组合。在运行时，整体对象可以选择不同类型的局部对象。
- 整体类可以对局部类进行包装，封装局部类的接口，提供新的接口。

缺点

- 整体类不能自动获得和局部类同样的接口。
- 创建整体类的对象时，需要创建所有局部类的对象 。