软件构造实验报告六

实验名称：组合模式与职责链模式编程实现

实验时间：2019.5.15

学号：E21614061

姓名：徐奕

所在院系：计算机科学与技术学院

所在专业：软件工程

【实验目的和要求】

a）熟悉并理解组合模式与职责链模式的原理与方法

b）熟练掌握组合模式与职责链模式的代码与方法

【实验原理】

组合模式

简介

- 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。

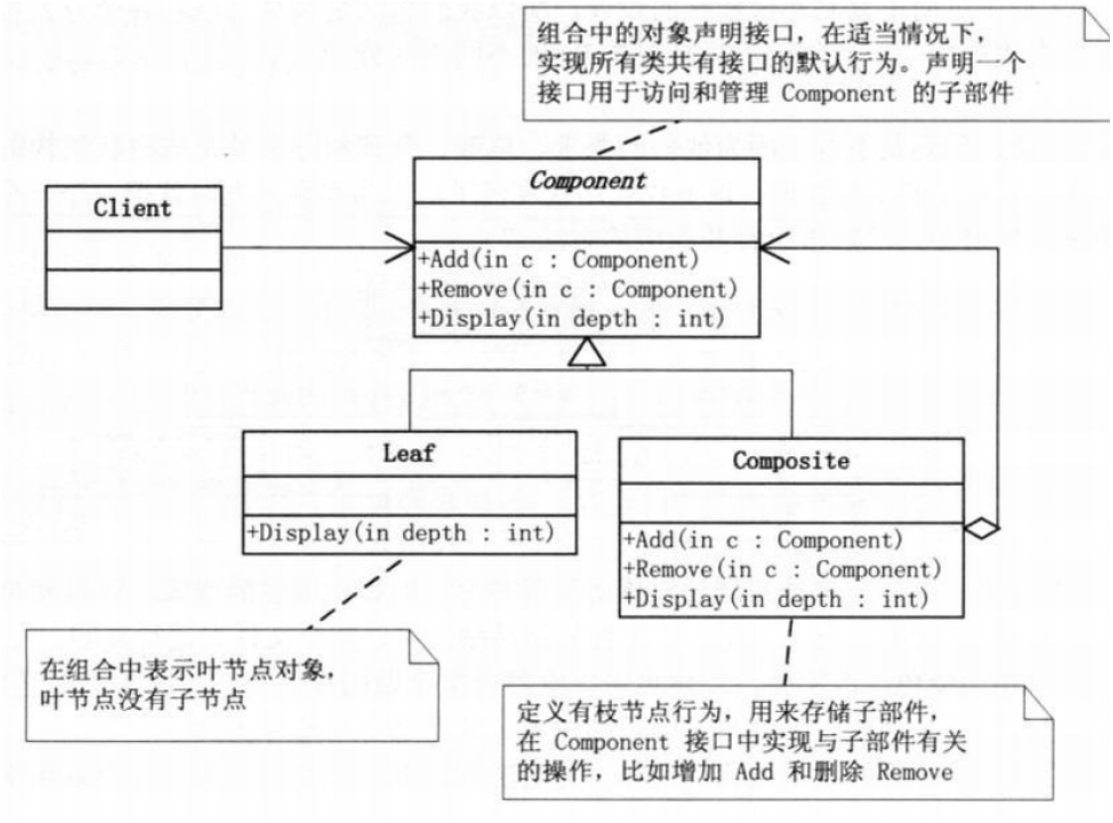
动机

- 总部、分部和办事处是成树状结构，也就是有组织结构的，不可以简单的平行管理。
- 希望总公司的组织结构，比如人力资源部、财务部的管理功能可以复用于分公司。这其实是整体与部分可以被一致对待的问题。

适用性

- 想表示对象的“部分-整体”层次结构。
- 希望用户忽略组合对象与单个对象的不同，用户将统一地使用组合结构中的所有对象。

结构



参与者

- Component
  - 为组合中的对象声明接口。
  - 在适当的情况下，实现所有类共有接口的缺省行为。
  - 声明一个接口用于访问和管理 Component 的子组件。
  - 在递归结构中定义一个接口，用于访问一个父部件，并在合适的情况下实现它。（可选）
- Leaf
  - 在组合中表示叶节点对象，叶节点没有子节点。
  - 在组合中定义叶节点对象的行为。

- Composite
  - 定义有子部件的那些部件的行为。
  - 存储子部件。
  - 在 Component 接口中实现与子部件有关的操作。
- Client
  - 通过 Component 接口操纵组合部件的对象。

协作

- 用户使用 Component 类接口与组合结构中的对象进行交互。
- 如果接收者是一个叶节点，则直接处理请求。
- 如果接收者是一个 Composite，它通常将请求发送给它的子部件，在转发请求之前与/或之后可能执行一些辅助操作。

效果

- 定义了包含基本对象和组合对象的类层次结构。
- 简化客户代码。
- 使得更容易添加新类型的组件。

职责链模式

简介

- 为解除请求的发送者和接收者之间的耦合，而使多个对象都有机会处理这个请求。
- 将这些对象连成一条链，并沿着这条链传递该请求，指导有一个对象处理它为止。

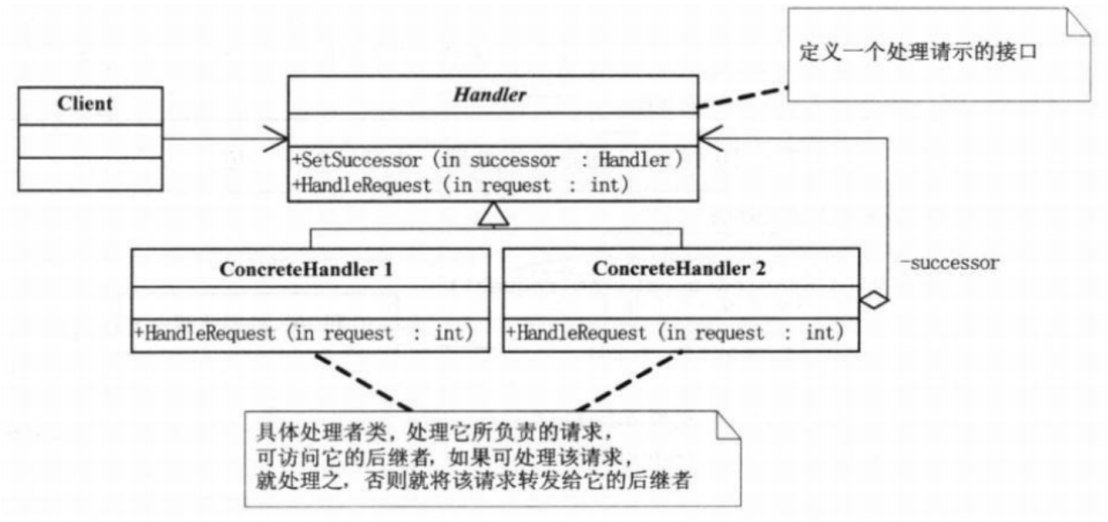
动机

- 考虑公司中的请假申请。
- 公司中管理人员按等级从低到高分别有：经理、总监、总经理。
- 请假需要根据天数长短，向不同等级的管理人员申请，例如少于 2 天直接向经理申请即可，少于一周必须要总监批准，更长时间则需要总经理批准。

适用性

- 有多个对象可以处理一个请求，哪个对象处理该请求运行时刻自动确定。
- 想在不明确指定接收者的情况下，向多个对象中的一个提交一个请求。
- 可处理一个请求的对象集合应被动态指定。

结构



参与者

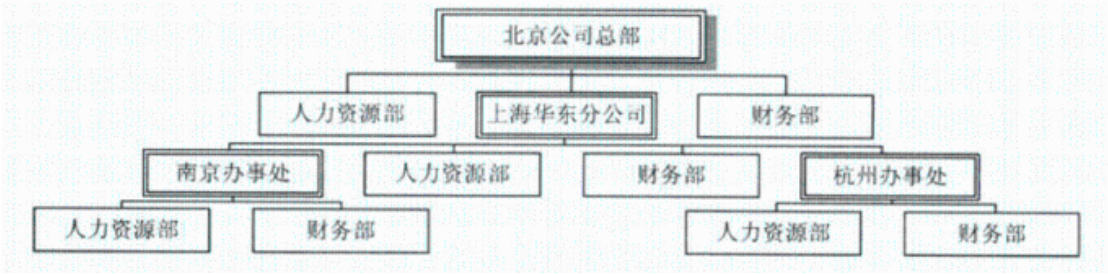
- Handler
  - 定义一个处理请求的接口。
  - 实现后继链。
- ConcreteHandler
  - 处理它所负责的请求。
  - 可访问它的后继者。
  - 如果可处理该请求，就处理之；否则将该请求转发给它的后继者。
- Client
  - 向链上的 ConcreteHandler 对象提交请求。

效果

- 降低耦合度。
- 增强了给对象指派职责的灵活性。
- 不保证被接受。

【实验内容】

根据下图，利用组合模式构建该公司的结构。

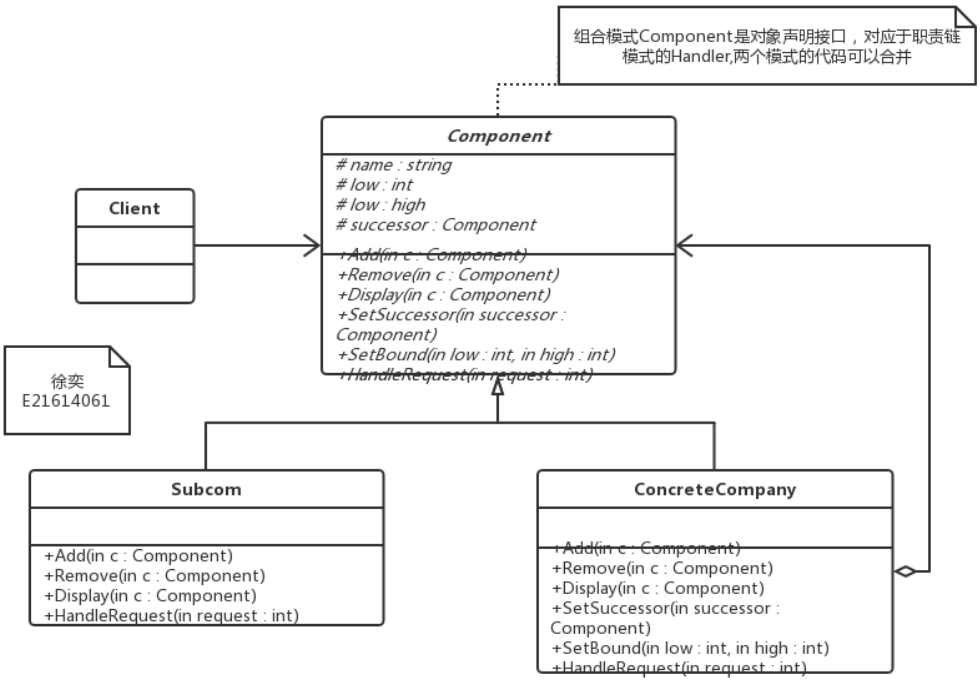


即总公司下设上海华东分公司，华东分公司下设南京办事处和杭州办事处。其中，各级分公司均设有人力资源部和财务部。

利用组合模式构建好该结构后，利用职责链模式处理各地员工加薪请求。

例如，一南京员工提出加薪请求，如果加薪不超过 1000 元，南京办事处即可批准；如果超过 1000 元，则上报华东分公司，如果 2000 元以内，则批准；超过则上报北京总公司，如果不超过 3000 元，则总公司批准，否则拒绝。

【实验 UML 图】



【实验代码与函数】

组合模式 Component 是对象声明接口，对应于职责链模式的 Handler,两个模式的代码可以合并。

另外通过增加 low, bound 两个变量来控制 if 语句数量，减少类的数目。因此职责链中的多个 ConcreteHandler 对应于一个具体的 ConcreteCompany。

```
#include<iostream>
#include<list>

using namespace std;

class Component{
public:
    Component(string name){
        this->name = name;
    }
    virtual void Add(Component* c) = 0;
    virtual void Remove(Component* c) = 0;
    virtual void Display(int depth) = 0;
    void SetSuccessor(Component* s){
        this->successor = s;
    }
    void SetBound(int low, int high){
        this->low = low;
        this->high = high;
    }
    virtual void HandleReauest(int request) = 0;
protected:
    string name;
    int low, high;
    Component* successor;
};

class Subcom : public Component{
public:
    Subcom(string name) : Component(name){}
    void Add(Component* c){
        cout<<"Cannot add to a leaf";
```

```
    }
    void Remove(Component* c){
        cout<<"Cannot remove from a leaf";
    }
    void Display(int depth){
        for(int i = 0; i <= depth; i++)
            cout<<"--";
        cout<<name<<endl;
    }
    void HandleReauest(int request){

    }
};

class ConcreteCompany: public Component{
private:
    list<Component*> children;
public:
    ConcreteCompany(string name): Component(name){}
    void Add(Component *c){
        children.push_back(c);
    }
    void Remove(Component *c){
        children.remove(c);
    }
    void Display(int depth){
        for(int i = 0; i <= depth; i++)
            cout<<"--";
        cout<<name<<endl;
        for(Component* child : children){
            child->Display(depth + 1);
        }
    }
    void HandleReauest(int request){
        if(request >= low && request <= high){
            cout<<this->name<<"处理请求: "<<request<<endl;
        }else if(successor != NULL){
            cout<<this->name<<"处理失败"<<endl;
            successor->HandleReauest(request);
        }else{
            cout<<this->name<<"加薪拒绝"<<request<<endl;
        }
    }
};
```

```
/*
即总公司下设上海华东分公司，华东分公司下设南京办事处和杭州办事处。其中，各级分公司均设有人力资源部和财务部。

用组合模式构建好该结构后，利用职责链模式处理各地员工加薪请求。

例如，一南京员工提出加薪请求，如果加薪不超过1000元，南京办事处即可批准；如果超过1000元，则上报华东分公司，如果2000元以内，则批准；超过则上报北京总公司，如果不超过3000元，则总公司批准，否则拒绝。、

实验报告内容:

 */


int main(){
    Component *root = new ConcreteCompany("总公司");
    root->Add(new Subcom("人力资源部"));
    root->Add(new Subcom("财务部"));
    root->SetBound(2000, 3000);

    Component *huadong = new ConcreteCompany("上海华东分公司");
    huadong->Add(new Subcom("人力资源部"));
    huadong->Add(new Subcom("财务部"));
    huadong->SetBound(1000, 2000);


    Component *nanjing = new ConcreteCompany("南京办事处");
    nanjing->Add(new Subcom("人力资源部"));
    nanjing->Add(new Subcom("财务部"));
    nanjing->SetBound(0, 1000);

    Component *hanzhou = new ConcreteCompany("杭州办事处");
    hanzhou->Add(new Subcom("人力资源部"));
    hanzhou->Add(new Subcom("财务部"));
    hanzhou->SetBound(0, 1000);

    nanjing->SetSuccessor(huadong);
    hanzhou->SetSuccessor(huadong);
    huadong->SetSuccessor(root);
    root->SetSuccessor(NULL);
    huadong->Add(nanjing);
    huadong->Add(hanzhou);
```

```
    root->Add(huadong);
    root->Display(1);
    cout<<"**********************"<<endl;
    nanjing->HandleReauest(3001);
    return 0;
}
```

【实验结果】

```
----总公司
------人力资源部
------财务部
------上海华东分公司
--------人力资源部
--------财务部
--------南京办事处
----------人力资源部
----------财务部
--------杭州办事处
----------人力资源部
----------财务部
**********************
南京办事处处理请求: 100
**********************
杭州办事处处理失败
上海华东分公司处理请求: 1500
**********************
杭州办事处处理失败
上海华东分公司处理失败
总公司处理请求: 2100
**********************
杭州办事处处理失败
上海华东分公司处理失败
总公司加薪拒绝3100
```

【实验总结】

①本次需要将职责链模式与组合模式合并，其中组合模式 Component 是对象声明接口，对应于职责链模式的 Handler。

②组合模式中的节点需要放到当前父节点的 list 中，对于容器 list，需要掌握对应的 STL 语句。

③对于职责链模式，为了减少类的数量和耦合需要设置 low 和 high 两个上下界。类虽然少了，但是对象没有减少，这些对象形成一条链，并沿着这条链传递该请求，直到有一个对象处理了它为止。通过当前对象的 name 来指示客户端选择。