责任链模式 - 巴洛克工作社

发表于 2022-09-13 | 更新于 2023-04-06 | 设计模式

| 字数总计: 2.2k | 阅读时长: 7 分钟 | 阅读量: 476 | 评论数: 1

配套视频课程已更新完毕，大家可通过以下两种方式观看视频讲解：

关注公众号： 爱编程的大丙 ，或者进入 大丙课堂 学习。

苏丙榅

合抱之木，生于毫末；九层之台，起于垒土；千里之行，始于足下。

1. 犯罪公司

在海贼世界中，巴洛克工作社是驻扎于阿拉巴斯坦的秘密犯罪公司，社长是王下七武海之一的沙·克洛克达尔。巴洛克工作社的名义上的目的是推翻阿拉巴斯坦建立理想王国，真正目的是得到古代兵器“冥王”从而获得更强大的力量。先来看一下这个公司的组织结构：

目录

CMake 保姆级教程（下）
2023-03-15

CMake 保姆级教程（上）
2023-03-06

访问者模式 - 再见，香波地群岛
2022-09-22

模板方法模式 - 和平主义者
2022-09-21

状态模式 - 文斯莫克・山治
2022-09-20

虽然巴洛克工作社是个犯罪公司，但是其内部的组织结构和分工是非常明确的。假设大聪明是这个组织中最卑微的 百万长者 ，那么有任何需求都需要向上级汇报等待审批，比如 请假、涨工资、离职 等，对于不同的请求，各个层级的领导也有不同的审批权限：

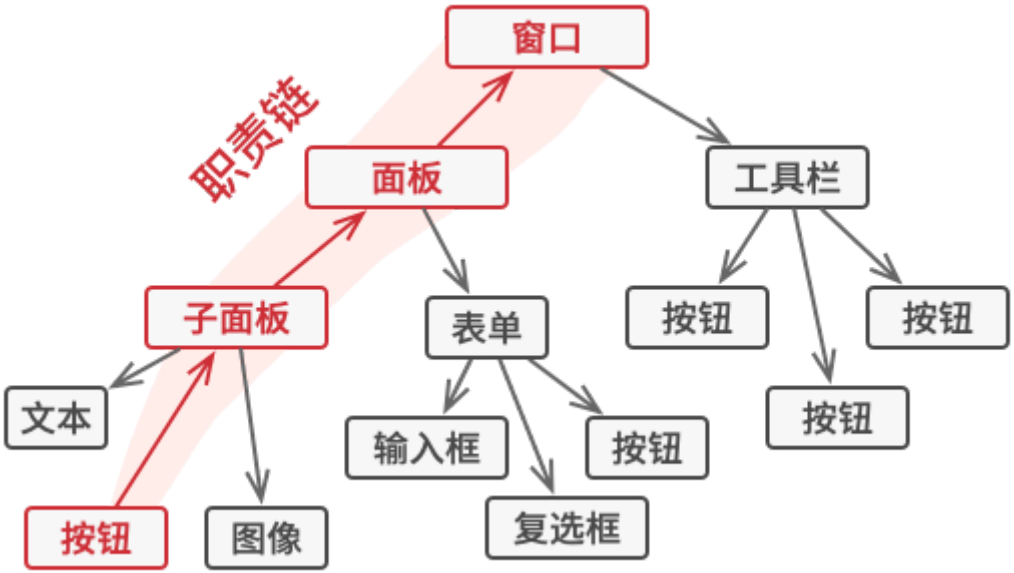
- 请假：直接上级有权进行处理
- 涨工资：只有副社长和社长有权利进行处理
- 离职：只有社长有权利进行处理

也就是说对于某个员工的请求可能需要一级一级向上传递，如果有权处理那就将其处理掉，如果无权处理还需继续向上传递该请求。像上面这种将对象连成一条链，并沿着这条链传递请求，直到链上有一个对象将请求处理掉为止，这种处理数据的模式叫做责任链模式。使用这种模式有一个好处：处理者可以决定不再沿着链传递请求， 这可高效地取消所有后续处理步骤。

责任链会将特定行为转换为被称作 处理者 的独立对象。 在巴洛克工作社这个例子中， 每个审批步骤都可被抽取为仅有单个方法的类， 并执行审批操作，请求及其数据则会被作为参数传递给该方法。

关于责任链这种模式，在日常生活中也比较常见，比如：

- 公司的 OA 审批系统
- 移动、联通语音服务器系统
- GUI 操作界面



2. 浪子回头

大聪明是巴洛克工作社最底层的一个小坏蛋，因为总干坏事儿受到了良心的谴责，所以要请假，被顶头上司批准，随后提出涨工资，被罗宾（副社长）拒绝，最后提出离职被克洛克达尔（社长）拒绝。下面作为程序猿的我基于责任链模式写一段代码来描述一下大聪明的遭遇。

2.1 执剑人

在责任链模式中，每个节点都有相同的处理动作（处理函数），只不过因为每个节点的权限、地位不同，（在处理函数内部）这个请求可能被处理了，也可能没有被处理，所以这些节点应该有一个共同的基类，也就是一个抽象的管理者节点类：

C++

```cpp
enum class RequestType:char {QingJia, ZhangXin, CiZhi};
// 抽象的任务节点类
class AbstractManager
{
public:
    void setNext(AbstractManager* manager)
    {
        m_next = manager;
    }
    virtual void handleRequest(RequestType type) = 0;
    virtual ~AbstractManager() {}
protected:
    AbstractManager* m_next = nullptr;
};
```

在这个抽象类中提供了一个 `setNext()` 方法，这样就可以将各个节点连接起来了，并且提供了一个处理请求的纯虚函数 `handleRequest()`，关于请求的类型被定义到了枚举 `RequestType` 中。

在巴洛克工作社中不同等级的管理者权利也不同，下面需要根据管理者的权限将处理任务的函数实现出来：

C++

```cpp
// 初级管理者
class Manager : public AbstractManager
{
public:
    void handleRequest(RequestType type)
    {
        switch (type)
```

```
        {
        case RequestType::QingJia:
            cout << "请假: 同意请假，好好休息~~~" << endl;
            break;
        case RequestType::ZhangXin:
            cout << "涨薪: 这个我得请示一下咱们CEO..." << " ====> ";
            m_next->handleRequest(type);
            break;
        case RequestType::CiZhi:
            cout << "辞职: 我给你向上级反应一下..." << " ====> ";
            m_next->handleRequest(type);
            break;
        default:
            break;
        }
    }
};

// CEO
class CEO : public AbstractManager
{
public:
    void handleRequest(RequestType type)
    {
        switch (type)
        {
        case RequestType::QingJia:
            cout << "请假: 同意请假, 下不为例..." << endl;
            break;
        case RequestType::ZhangXin:
            cout << "涨薪: 你工资不少了, 给你个购物券吧..." << endl;
```

```
39              break;
40          case RequestType::CiZhi:
41              cout << "辞职: 这个我得问问咱们老板..." << " ====> ";
```

在上面三个子类中根据各自权限分别重写了 `handleRequest()` 函数。

2.2 大聪明

大聪明作为巴洛克工作社中一个身份卑微的小弟，也对应一个类，通过这个类中提供的函数就可以向领导提出请求了:

C++

```cpp
// 卑微的大聪明
class DaCongMing
{
public:
    void request(RequestType type, AbstractManager* manager)
    {
        manager->handleRequest(type);
    }
};
```

2.3 宿命

下面是不甘心的大聪明分别向三个不同的领导提出请求的经过:

C++

```cpp
int main()
{
    Manager* manager = new Manager;
    CEO* ceo = new CEO;
    Boss* boss = new Boss;
    // 设置关联关系
    manager->setNext(ceo);
    ceo->setNext(boss);

    // 卑微的大聪明的请求
    DaCongMing* boy = new DaCongMing;
    cout << "========== 大聪明向顶头上司提要求 ==========" << endl;
    boy->request(RequestType::QingJia, manager);
    boy->request(RequestType::ZhangXin, manager);
    boy->request(RequestType::CiZhi, manager);
    cout << "========== 大聪明越级找CEO提要求 ==========" << endl;
    boy->request(RequestType::QingJia, ceo);
    boy->request(RequestType::ZhangXin, ceo);
    boy->request(RequestType::CiZhi, ceo);
    cout << "========== 大聪明直接找BOSS提要求 ==========" << endl;
    boy->request(RequestType::QingJia, boss);
    boy->request(RequestType::ZhangXin, boss);
    boy->request(RequestType::CiZhi, boss);

    delete boy;
    delete manager;
    delete ceo;
    delete boss;

    return 0;
}
```

大聪明到的答复如下:

C++

```
========== 大聪明向顶头上司提要求 ==========
请假: 同意请假，好好休息~~~
涨薪: 这个我得请示一下咱们CEO... ====> 涨薪: 你工资不少了，给你个购物券吧...
辞职: 我给你向上级反应一下... ====> 辞职: 这个我得问问咱们老板... ====> 辞职: 巴洛克
========== 大聪明越级找CEO提要求 ==========
请假: 同意请假，下不为例...
涨薪: 你工资不少了，给你个购物券吧...
辞职: 这个我得问问咱们老板... ====> 辞职: 巴洛克工作社就是你的家，这次把你留下，下次别
========== 大聪明直接找BOSS提要求 ==========
请假: 只有工作才能实现人生价值，回去好好坚守岗位！！！
涨薪: 钱财乃身外之物，要视其如粪土!!!
辞职: 巴洛克工作社就是你的家，这次把你留下，下次别再提了!!!
```

大聪明直接哭晕在厕所！在真实的业务场景中，不同级别的管理者在处理请求的时候会进行更加细致的判定，在当前的场景中就忽略了。

关键点：在处理请求之前必须先要把各个管理者对象按照等级关系串联起来：

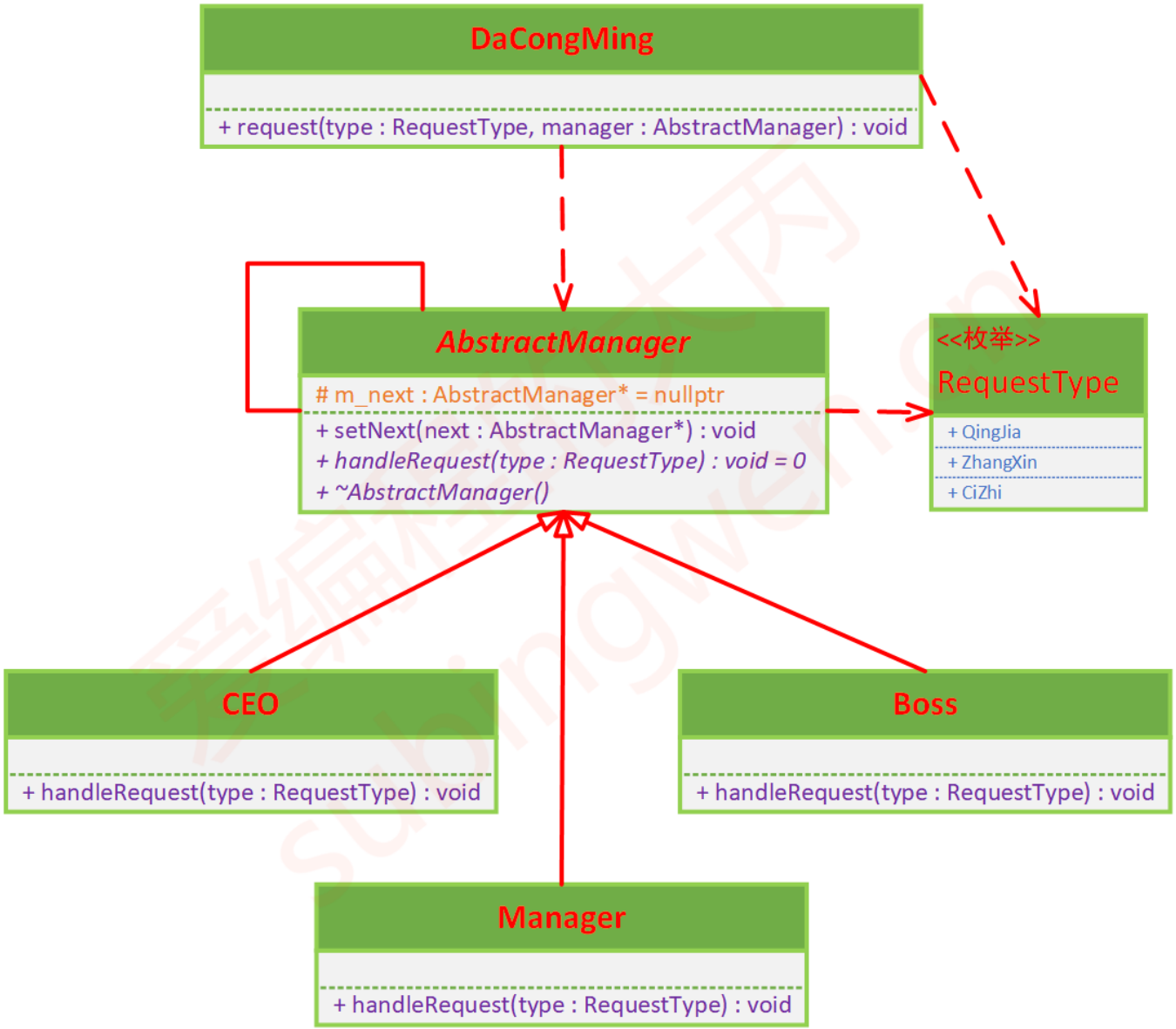
C++

```
Manager* manager = new Manager;
CEO* ceo = new CEO;
Boss* boss = new Boss;
// 设置关联关系
manager→setNext(ceo);
ceo→setNext(boss);
```

> 责任链模式就是将这些处理者连成一条链。 链上的每个处理者都有一个成员变量来保存下一个处理者。 除了处理请求外， 处理者还负责沿着链传递请求， 请求会在链上移动， 直至所有处理者都有机会对其进行处理。

3. 结构图

最后将上面的例子对应的 UML 类图画一下（学会了责任链模式之后，需要先画 UML 类图，再写程序。）



在上图中，通过自关联的方式，并且基于多态把当前管理者对象的下一级对象（`AbstractManager` 的子类对象）保存到 `m_next` 中了，这样就可以得到一个单向链，它其实还可以有其他的变种：

- 双向链：将当前节点的后继对象和前驱对象都记录下来
- 树状链：当前节点可以有多个子节点，也就是多个后继节点，可以将它们保存到一个 STL 容器中。

总之，举一反三，根据实际情况随机应变就对了。

设计模式

打赏

命令模式 - 海上餐厅巴拉蒂

享元模式 - 铁拳卡普

相关推荐

评论

昵称 邮箱 网址(http://)

来都来了, 说点什么吧...

提交

1 评论

Anonymous Chrome 110.0.0.0 Linux

2023-02-18

回复

好啊

Powered By Valine

v1.5.1