访问者模式 - 再见，香波地群岛

发表于 2022-09-22 | 更新于 2023-04-06 | 设计模式

| 字数总计: 3.5k | 阅读时长: 12 分钟 | 阅读量: 1566 | 评论数: 2

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. 遭遇香波地

在海贼世界中，香波地群岛位于伟大航路中间的红土大陆前方。岛屿是由多颗大树构成，地面就是树根，会从地面冒出气泡。在这里，路飞的好朋友人鱼凯米被人贩子拐卖并卖给了天龙人，路飞为了救凯米狠狠地揍了天龙人。最后，草帽团被巴索罗缪·大熊的果实能力拍到了世界各地，草帽团就地解散。

对于当时的草帽团来说一共有九个人，面对这个打不过的敌人大熊，九个人都做了最后的反抗和挣扎，在这个过程中这些人的状态和行为或许相同相互不同，归纳总结一下就是 无助、恐惧、愤怒 。

目录

最新文章

如果我们想要通过程序复刻上面动态图片中的这个场景是有很多种处理方式的，最简单的一种就是定义一个人的基类，然后让草帽团的各成员作为这个类的子类，在各个子类中来具体描述他们面对大熊的攻击时的反应和状态。

1. 索隆很愤怒决定要砍了大熊
2. 山治很愤怒决定要踢死大熊
3. 乌索普很恐惧在心里画圈圈诅咒大熊
4. 路飞很愤怒想锤死大熊
5. 乔巴很愤怒想拍死大熊
6. 布鲁克很愤怒用已死的身体阻挡大熊

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

7. 弗兰奇很愤怒思考怎么弄死大熊
8. 娜美很恐惧，请求伙伴的帮助
9. 罗宾很恐惧，请求伙伴的帮助

可以看到这么写有一个弊端，就是比较啰嗦，其实草帽团面对大熊的突如其来的攻击就两个状态：愤怒和恐惧，在这两种状态下的反应就是 战斗和求助 。所以我们可以把上面的这个场景重构一下：

如果草帽团的某些成员在面对大熊攻击时的状态反应是一样的，那么在这些子类中就会出现很多相同的冗余代码。有一种更好的处理思路就是将状态和人分开，其中草帽团的各个成员我们可以看做是对象，草帽团成员的反应和状态我们可以将其看做是算法，这种将算法与其所作用的对象隔离开来的设计模式就叫做访问者模式，其实就是通过被分离出的算法来访问对应的对象。

关于访问者模式，在日常生活中对应的场景也有很多，比如：

- 旅游：去安徽可以爬黄山，去山东可以爬泰山，去陕西可以爬华山
  - 不同的地点，人爬的山是不一样的
- 卖保险：如果是老百姓推销医疗保险，如果是银行推销失窃保险，如果是商铺推销火灾和水灾保险。
  - 不同的受众，保险推销员推销的产品是不一样的
- 小鬼子奇葩的盖章文化：Boss 的章是正的，其他下属职位越低盖章的时候就得越倾斜（真他妈的虚伪），如果不这样就表示对上司有意见。
  - 不同等级的职员，盖章的方式是不一样的。

以上三个例子中前者是对象，后者就是算法，如果用访问者模式处理上边列举的场景就需要使用后者来访问前者。

2. 再见，草帽团

2.1 草帽团成员

在香波地群岛的时候，草帽团的成员一共有 9 人，如果使用访问者模式来处理他们在遭遇大熊之后的状态，那么就需要将状态（算法）和人（对象）分离开来，关于这九个成员我们可以按照性别进行划分： 男人和女人。 不论是哪类成员最终都需要接受对应的被分离出的那个行为状态的访问，我们只需要在这个成员类中提供一个接受访问的函数就可以了，所以这个抽象的成员类定义如下：

C++

```cpp
// 抽象的成员类
class AbstractMember
{
public:
    AbstractMember(string name) : m_name(name){}
    string getName()
    {
        return m_name;
    }
    // 接受状态对象的访问
    virtual void accept(行为/动作类* action) = 0;
    virtual ~AbstractMember() {}
protected:
    string m_name;
};
```

- 这个抽象的基类的构造函数提供了一个参数，用于指定当前成员的名字。
- 调用 `getName()` 函数可以得到当前成员的名字
- 调用 `accept()` 函数表示当前成员对象接受了行为 / 动作类的访问
  - 关于 行为/动作 也可以对应很多种情况，所以此处的类也应该是一个基类
  - 目前 行为/动作 类还没有被定义，所以先用中文注释代替。

有了上面的抽象基类，按照里面的分类就可以把成员子类定义出来了:

C++

```cpp
// 男性成员
class MaleMember : public AbstractMember
```

```
{
public:
    AbstractMember::AbstractMember;
    void accept(行为/动作* action) override
    {
        // do something
    }
};

// 女性成员
class FemaleMember : public AbstractMember
{
public:
    AbstractMember::AbstractMember;
    void accept(行为/动作* action) override
    {
        // do something
    }
};
```

关于草帽团成员类，他们的行为全部被分离出去了，所以在当前成员类中剩下的就是这个成员自身的一些属性信息，比如：性别、姓名、在船上的职务、被悬赏的金额等。

如果在成员类中调用了 `accept()` 方法，就可以通过参数传入的行为 / 动作对象调用它的成员函数，这个动作就是当前草帽团某个成员对象被分离出去的动作或者行为，通过这种方式成员类和行为 / 动作类就可以关联到一起了。

2.2 最后的挣扎

草帽一伙在被大熊拍飞之前都做出了最后的挣扎，他们对应的就是一些状态和行为，假设状态就两种： 愤怒和恐惧。 这两个状态的所有者就是上面定义的两个类 男性成员类和女性成员类。 我们先把这两种行为状态对应的基类定义出来：

C++

```cpp
// Visitor.h
// 类声明
class MaleMember;
class FemaleMember;
// 抽象的动作类
class AbstractAction
{
public:
    // 访问男人
    virtual void maleDoing(MaleMember* male) = 0;
    // 访问女人
    virtual void femalDoing(FemaleMember* female) = 0;
    virtual ~AbstractAction() {}
};
```

这个类提供了两个虚函数：

- `maleDoing()`：男性成员的行为函数，所以需要访问一个男性成员的对象，故参数的类型为 `MaleMember*`
- `femalDoing()`：女性成员的行为函数，所以需要访问一个女性成员的对象，故参数的类型为 `FemaleMember*`

- 这两个 `Doing()函数` 之所以参数是成员类的子类类型是因为男性成员和女性成员对象所拥有的函数可以是不一样的（除了从父类继承，可能在子类内部也会定义一些只属于这个子类的成员函数），这样才更方便进行函数的调用。
- 在这个类的头文件中本别对 `MaleMember 类` 和 `FemaleMember 类` 进行了声明，但并没有包含他们对应的头文件，目的是防止头文件重复包含。

有了行为类的基类，下面把它对应的两个子类定义出来：

头文件 Visitor.h

C++

```cpp
// 愤怒
class Anger : public AbstractAction
{
public:
    void maleDoing(MaleMember* male) override;
    void femalDoing(FemaleMember* female) override;
    void warning();
    void fight();
};

// 恐惧
class Horror : public AbstractAction
{
public:
    void maleDoing(MaleMember* male) override;
    void femalDoing(FemaleMember* female) override;
    void help();
```

```
18      void thinking();
19  };
```

源文件 Visitor.cpp

C++

```cpp
#include <iostream>
#include "Visitor.h"
#include "Member.h"
#include <list>
#include <vector>
using namespace std;

void Anger::maleDoing(MaleMember* male)
{
    cout << "我是草帽海贼团的" << male->getName() << endl;
    fight();
}

void Anger::femalDoing(FemaleMember* female)
{
    cout << "我是草帽海贼团的" << female->getName() << endl;
    warning();
}

void Anger::warning()
{
    cout << "大家块逃，我快顶不住了，不要管我!!!" << endl;
}

```

```
25  void Anger::fight()
26  {
27      cout << "只要还活着就得跟这家伙血战到底!!!" << endl;
28  }
29
30  void Horror::maleDoing(MaleMember* male)
31  {
32      cout << "我是草帽海贼团的" << male→getName() << endl;
33      thinking();
34  }
35
36  void Horror::femalDoing(FemaleMember* female)
37  {
38      cout << "我是草帽海贼团的" << female→getName() << endl;
39      help();
40  }
41
42  void Horror::help()
```

在这个两个字行为类中分别通过 `maleDoing()` 和 `femalDoing()` 函数完成了对成员类的访问，通过参数传递进来的成员对象得到了成员属性，然后在行为类中赋予这个成员对象一系列的行为，这样成员对象和成员对象的行为就又被整合到一起了。

行为类被定义出来之后，我们就可以把前面的成员类补充完整了：

C++

```
1  // Member.h
2  #pragma once
3  #include <iostream>
```

```cpp
#include "Visitor.h"
using namespace std;
// 抽象的成员类
class AbstractMember
{
public:
    AbstractMember(string name) :m_name(name){}
    string getName()
    {
        return m_name;
    }
    // 接受状态对象的访问
    virtual void accept(AbstractAction* action) = 0;
    virtual ~AbstractMember() {}
protected:
    string m_name;
};

// 男性成员
class MaleMember : public AbstractMember
{
public:
    AbstractMember::AbstractMember;
    void accept(AbstractAction* action) override
    {
        action->maleDoing(this);
    }
};

// 女性成员
class FemaleMember : public AbstractMember
```

```
{
public:
    AbstractMember::AbstractMember;
    void accept(AbstractAction* action) override
    {
        action->femalDoing(this);
    }
};
```

在上面的代码中用到了一种 双分派技术：

1. 在调用成员类的 `accept()` 函数 的时候，将具体地行为状态通过参数传递给了男性成员或者女性成员。
2. 在 `accept()` 函数 中通过行为状态对象调用行为函数的时候，将当前成员对象传递给了状态对象。

accept () 函数是一个双分派操作，它得到执行的操作不仅取决于传入的状态类的具体状态，还取决于它访问的人的类别。

2.3 集合

草帽一伙要被拍飞了，我们可以把这九个人先聚到一起，所以可以定义一个草帽团类：

C++

```
// Visitor.cpp
// 草帽团
class CaoMaoTeam
{
```

```
public:
    CaoMaoTeam()
    {
        m_actions.push_back(new Anger);
        m_actions.push_back(new Horror);
    }
    void add(AbstractMember* member)
    {
        m_members.push_back(member);
    }
    void remove(AbstractMember* member)
    {
        m_members.remove(member);
    }
    void display()
    {
        for (const auto& item : m_members)
        {
            int index = rand() % 2;
            item->accept(m_actions[index]);
        }
    }
    ~CaoMaoTeam()
    {
        for (const auto& item : m_members)
        {
            delete item;
        }
        for (const auto& item : m_actions)
        {
            delete item;
```

```
        }
    }
private:
    list<AbstractMember*> m_members;
    vector<AbstractAction*> m_actions;
};
```

在这个类中使用了两个容器:

- `list 容器`： 用于存储草帽团成员
- `vector 容器`： 用于存储草帽团成员的两种行为状态。

通过这个类的 `display()函数` 我们就可以了解到草帽团成员在被大熊拍飞之前的行为状态了。

2.3 解散，再见

草帽团成员已集合完毕，大熊可以下手了：

C++

```
int main()
{
    srand(time(NULL));
    vector<string> names{
        "路飞", "索隆","山治", "乔巴", "弗兰奇", "乌索普", "布鲁克"
    };
    CaoMaoTeam* caomao = new CaoMaoTeam;
    for (const auto& item : names)
    {
        caomao->add(new MaleMember(item));
```

```
    }
    caomao->add(new FemaleMember("娜美"));
    caomao->add(new FemaleMember("罗宾"));
    caomao->display();
    delete caomao;
    return 0;
}
```

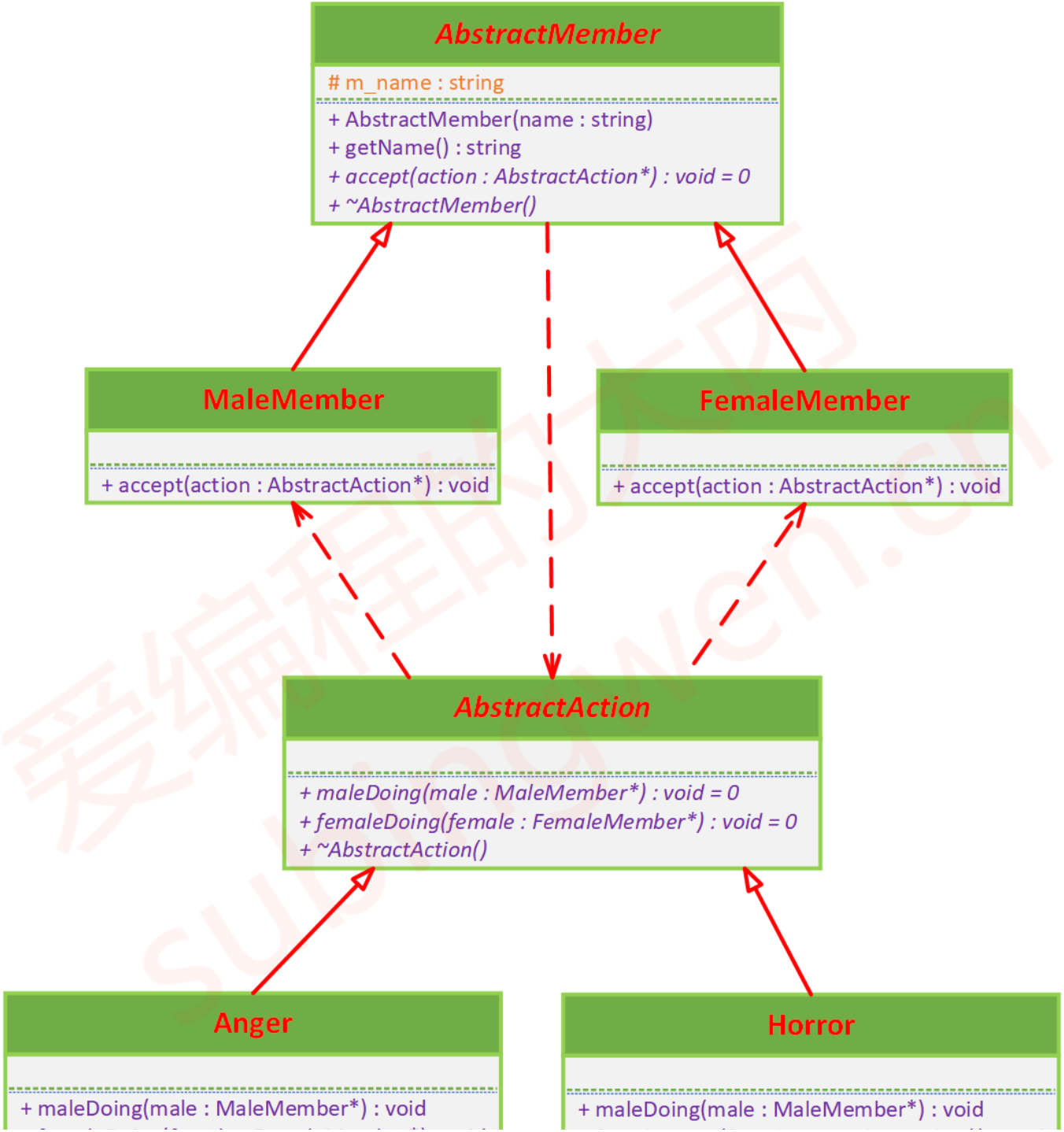
最后看一下，草帽团各成员的反应：

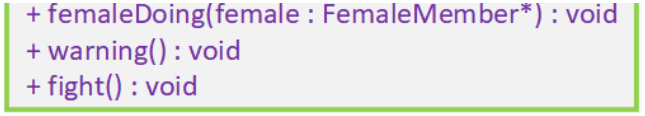
C++

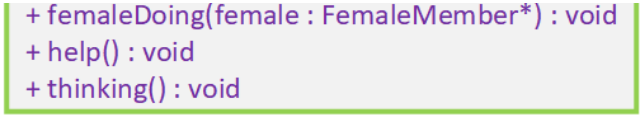
```
我是草帽海贼团的路飞
只要还活着就得跟这家伙血战到底!!!
我是草帽海贼团的索隆
得辅助同伴们一块攻击这个家伙，不然根本打不过呀!!!
我是草帽海贼团的山治
只要还活着就得跟这家伙血战到底!!!
我是草帽海贼团的乔巴
只要还活着就得跟这家伙血战到底!!!
我是草帽海贼团的弗兰奇
得辅助同伴们一块攻击这个家伙，不然根本打不过呀!!!
我是草帽海贼团的乌索普
只要还活着就得跟这家伙血战到底!!!
我是草帽海贼团的布鲁克
只要还活着就得跟这家伙血战到底!!!
我是草帽海贼团的娜美
这个大熊太厉害，太可怕了，快救救我。。。
我是草帽海贼团的罗宾
这个大熊太厉害，太可怕了，快救救我。。。
```

3. 结构图

最后将上面的例子对应的 UML 类图画一下（学会了访问者模式之后，需要先画 UML 类图，再写程序。）


AbstractMember
m_name : string
+ AbstractMember(name : string)
+ getName() : string
+ accept(action : AbstractAction*) : void = 0
+ ~AbstractMember()
MaleMember
+ accept(action : AbstractAction*) : void
FemaleMember
+ accept(action : AbstractAction*) : void
AbstractAction
+ maleDoing(male : MaleMember*) : void = 0
+ femaleDoing(female : FemaleMember*) : void = 0
+ ~AbstractAction()
Anger
+ maleDoing(male : MaleMember*) : void
Horror
+ maleDoing(male : MaleMember*) : void





访问者模式适用于数据结构比较稳定的系统，对于上面的例子而言就是指草帽团成员：只有男性和女性（不会再出现其它性别）。在剥离出的行为状态类中针对男性和女性提供了相对应的 doing 方法。这种模式的优势就是可以方便的给对象添加新的状态和处理动作，也就是添加新的 AbstractAction 子类（算法类），在需要的时候让这个子类去访问某个成员对象，访问者模式的最大优势就是使算法的增加变得更加容易维护。

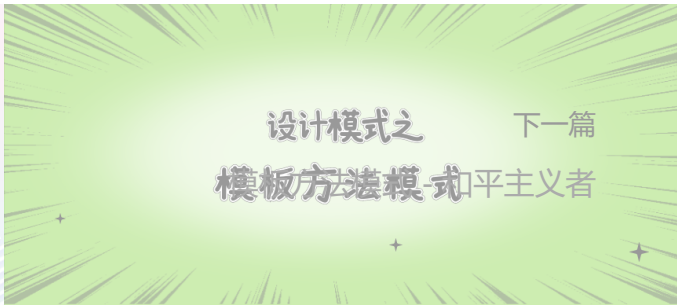
如果不按照性别进行划分，草帽团一共 9 个成员就需要在行为状态类中给每个成员提供一个 doing 方法，当草帽团又添加了新的成员，状态类中也需要给新成员再添加一个对应的 doing 方法，这就破坏了设计模式的开放 – 封闭原则。

> 访问者不是常用的设计模式， 因为它不仅复杂， 应用范围也比较狭窄。

设计模式

打赏


上一篇
CMake
CMake 保姆级教程（上）
保姆级教程-上


设计模式之
下一篇
模板方法模式

相关推荐


UML类图
2022-08-27
UML 类图
设计模式


设计模式之
抽象工厂模式


设计模式之
桥接模式

评论

昵称 邮箱 网址(http://)

来都来了, 说点什么吧...

提交

2 评论

Anonymous Chrome 113.0.0.0 Windows 11

2023-05-22 回复

这个模式好复杂

Anonymous Chrome 109.0.0.0 Windows 11

2023-02-23 回复

牛的

v1.5.1