策略模式 - 蒙奇・D・路飞

发表于 2022-09-19 | 更新于 2023-04-06 | 设计模式

| 字数总计: 2k | 阅读时长: 6 分钟 | 阅读量: 640 | 评论数: 3

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. 橡胶人

路飞是要成为海贼王的男人！在小时候因为误食了红发香克斯找到的橡胶恶魔果实成了橡胶人。对于果实能力，路飞现在已经开发出了五个档位。对于路飞而言在战斗的时候，必须要根据敌人的情况来实时制定合适的策略，使用不同的档位的不同招式去应对来自对方的攻击。

路飞在战斗的时候需要制定策略，在设计模式中也有一种和策略相关的模式叫做策略模式。策略模式需要我们定义一系列的算法，并且将每种算法都放入到独立的类中，在实际操作的时候使这些算法对象可以相互替换。

在日常生活中很多时候都需要制定策略，在程序中就可用使用策略模式来处理这些场景，比如：

目录

- 出行策略，可以选择不同的交通工具： 自行车、公交、地铁、自驾 等
- 战国时期秦国的外交政策：远交近攻
- 收复台湾的策略：武统、文统、恩威并施 。。。
- 电商平台的打折策略：买二赠一、满 300 减 50、购买 VIP 享 8 折优惠。。。

2. 百变路飞

作为橡胶人路飞，平时白痴但战斗时头脑异常清醒，会根据敌我双方的形式做出正确的判断：

- 对手战力弱，使用 1 档并根据实际情况使用相应的招式
- 对手战力一般，切换 2 档并根据实际情况使用相应的招式
- 对手战力强，切换 3 档并根据实际情况使用相应的招式
- 对手战力无敌，切换 4 档并根据实际情况使用相应的招式
- 对手战斗逆天，切换 5 档并根据实际情况使用相应的招式

假设将所有的策略都写到一个类中就会使得路飞这个类过于复杂，而且不容易维护，如果基于策略模式来处理路飞这个类，可以把关于在什么场景下使用什么策略的判断去除，把处理逻辑分散到多个不同的策略类中，这样就可以将复杂的逻辑简化了。

2.1 路飞的形态

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

根据上面的描述路飞一共有五种形态，不论使用哪种形态都需要制定战斗策略，这一点是不变的，所以我们就可以定义出一个抽象的策略类：

C++

```cpp
// 抽象的策略类
class AbstractStrategy
{
public:
    virtual void fight(bool isfar = false) = 0;
    virtual ~AbstractStrategy() {}
};
```

这个抽象类中的 `fight()` 函数有一个布尔类型的参数，表示在当前这个状态下是要进行近距离攻击还是远程攻击，参数不同，在这种状态下使用的招式也不同。有了这个抽象的基类，就可以定义出一系列的子类了：

C++

```cpp
// 一档
class YiDang : public AbstractStrategy
{
public:
    void fight(bool isfar = false) override
    {
        cout << "*** 现在使用的是一档: ";
        if (isfar)
        {
            cout << "橡胶机关枪" << endl;
        }
        else
```

```
        {
            cout << "橡胶·攻城炮" <<endl;
        }
    }
};

// 二挡
class ErDang : public AbstractStrategy
{
public:
    void fight(bool isfar = false) override
    {
        cout << "*** 切换成二挡: ";
        if (isfar)
        {
            cout << "橡胶Jet火箭" << endl;
        }
        else
        {
            cout << "橡胶Jet·铳乱打" << endl;
        }
    }
};

// 三挡
class SanDang : public AbstractStrategy
{
public:
    void fight(bool isfar = false) override
    {
        cout << "*** 切换成三挡: ";
```

这五个子类分别对应的路飞的 一档、二挡、三挡、四挡、五档，也就是五种不同的策略，在它们内部都重写了从基类继承的纯虚函数 `fight()`，并根据函数的参数做了不同的处理。通过这种方式的拆分就把复杂的问题简单化了，有种兄弟分家的感觉，本来是在同一个屋檐下，分家之后都有了自己的房子，各过各的互不干涉。

2.2 路飞

上面说到了分家，那个这几个儿子的爹是谁呢？没错，就是路飞，这五个子类都是路飞的果实能力，但是现在他们从路飞这个对象中分离出去了。所以现在路飞和这几个技能的状态类之间就变成了组合关系。关于路飞这个类我们可以这样定义：

C++

```cpp
// 难度级别
enum class Level:char {Easy, Normal, Hard, Experts, Professional};

// 路飞
class Luffy
{
public:
    void fight(Level level, bool isfar = false)
    {
        if (m_strategy)
        {
            delete m_strategy;
            m_strategy = nullptr;
        }
        switch (level)
```

```
        {
        case Level::Easy:
            m_strategy = new YiDang;
            break;
        case Level::Normal:
            m_strategy = new ErDang;
            break;
        case Level::Hard:
            m_strategy = new SanDang;
            break;
        case Level::Experts:
            m_strategy = new SiDang;
            break;
        case Level::Professional:
            m_strategy = new WuDang;
            break;
        default:
            break;
        }
        m_strategy->fight(isfar);
    }
    ~Luffy()
    {
        delete m_strategy;
    }
private:
    AbstractStrategy* m_strategy = nullptr;
};
```

在 `Luffy 类` 中的 `fight()` 方法里边根据参数传递进来的难度级别，路飞在战斗的时候就可以选择开启对应的档位使用相关的招式来击败对手。

2.3 对症下药

路飞在战斗的时候头脑还是非常灵活的，下面来看一下路飞经历过的一些战斗：

C++

```cpp
int main()
{
    Luffy* luffy = new Luffy;
    cout << "--- 在香波地群岛遇到了海军士兵: " << endl;
    luffy->fight(Level::Easy);
    cout << "--- 在魔谷镇遇到了贝拉米: " << endl;
    luffy->fight(Level::Normal);
    cout << "--- 在司法岛遇到了罗布·路奇: " << endl;
    luffy->fight(Level::Hard);
    cout << "--- 在德雷斯罗萨遇到了多弗朗明哥: " << endl;
    luffy->fight(Level::Experts);
    cout << "--- 在鬼岛遇到了凯多: " << endl;
    luffy->fight(Level::Professional);

    delete luffy;
    return 0;
}
```

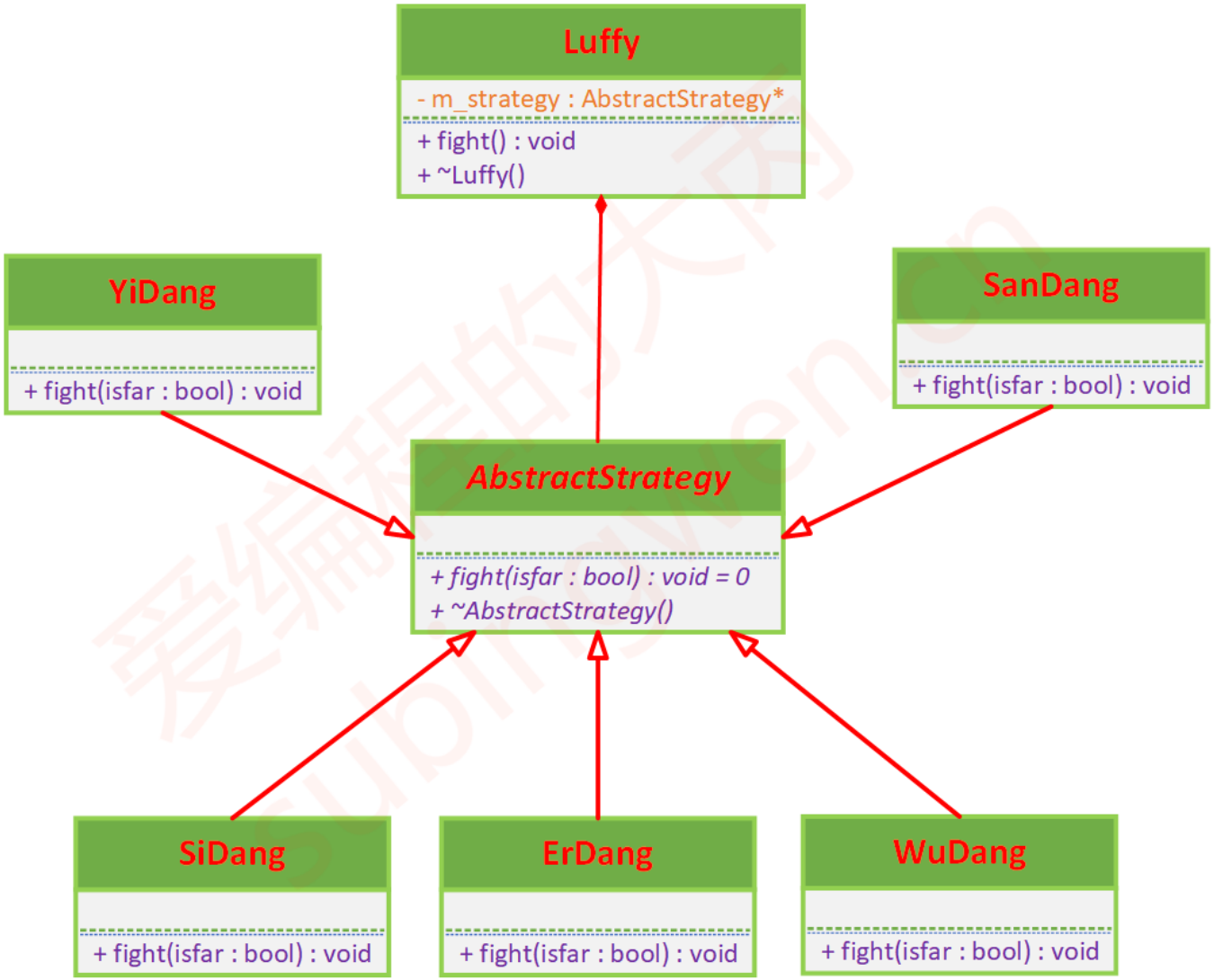
当时战斗的场景是这样的:

C++

```
--- 在香波地群岛遇到了海军士兵：
*** 现在使用的是一档：橡胶·攻城炮
--- 在魔谷镇遇到了贝拉米：
*** 切换成二挡：橡胶Jet·铳乱打
--- 在司法岛遇到了罗布·路奇：
*** 切换成三挡：橡胶巨人战斧
--- 在德雷斯罗萨遇到了多弗朗明哥：
*** 切换成四挡：橡胶犀牛榴弹炮
--- 在鬼岛遇到了凯多：
*** 切换成五挡：变成尼卡形态可以把物体变成橡胶，并任意改变物体的形态对其进行攻击!!!
```

3. 结构图

最后画一下策略模式对应的 UML 类图（学会策略模式之后，要先画 UML 类图再写程序。）



策略模式中的若干个策略对象相互之间是完全独立的， 它们不知道其他对象的存在。当我们想使用对象中各种不同的算法变体，并希望能够在运行的时候切换这些算法时，可以选择使用策略模式来处理这个问题。

设计模式

打赏

上一篇
状态模式 - 文章作者

下一篇
观察者模式 - 摩根斯

相关推荐

评论

昵称 邮箱 网址(http://)

来都来了, 说点什么吧...

提交

3 评论

令樽 Safari 16.5 iOS 16.5

10 小时前 回复

大丙老师不但技术牛，而且课讲得也好👍我相信国内任何一个互联网或电子信息大厂都有大丙老师的学生，至少都从老师的课程中受益。毕业了，现在能自己赚钱了，等这段忙完了一定停止白嫖，买一门课好好学习下

Anonymous Chrome 90.0.4430.61 Android 12

4 天前 回复

牛

Anonymous Chrome 113.0.0.0 Windows 11

2023-05-11 回复

太牛 x 了，看了好多版设计模式这个是最通俗易懂的

Powered By Valine

v1.5.1