组合模式 - 草帽大船团

发表于 2022-09-03 | 更新于 2023-04-06 | 设计模式

| 字数总计: 2.9k | 阅读时长: 10 分钟 | 阅读量: 426 | 评论数: 0

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. 好大一棵树

路飞在德雷斯罗萨打败多弗朗明哥之后，一些被路飞解救的海贼团自愿加入路飞麾下，自此组成了草帽大船团，旗下有 7 为船长，分别是：

1. 俊美海贼团 75 人
2. 巴托俱乐部 56 人
3. 八宝水军 1000 人
4. 艾迪欧海贼团 4 人
5. 咚塔塔海贼团 200 人
6. 巨兵海贼团 5 人
7. 约塔玛利亚大船团 4300 人

小弟数量总计 5640 人。

目录

对于 草帽大船团 的结构组成，很像一棵树：路飞是这棵树的根节点，旗下的七个船长是路飞的子节点。在这七个船长的旗下可能还有若干个船长。。。

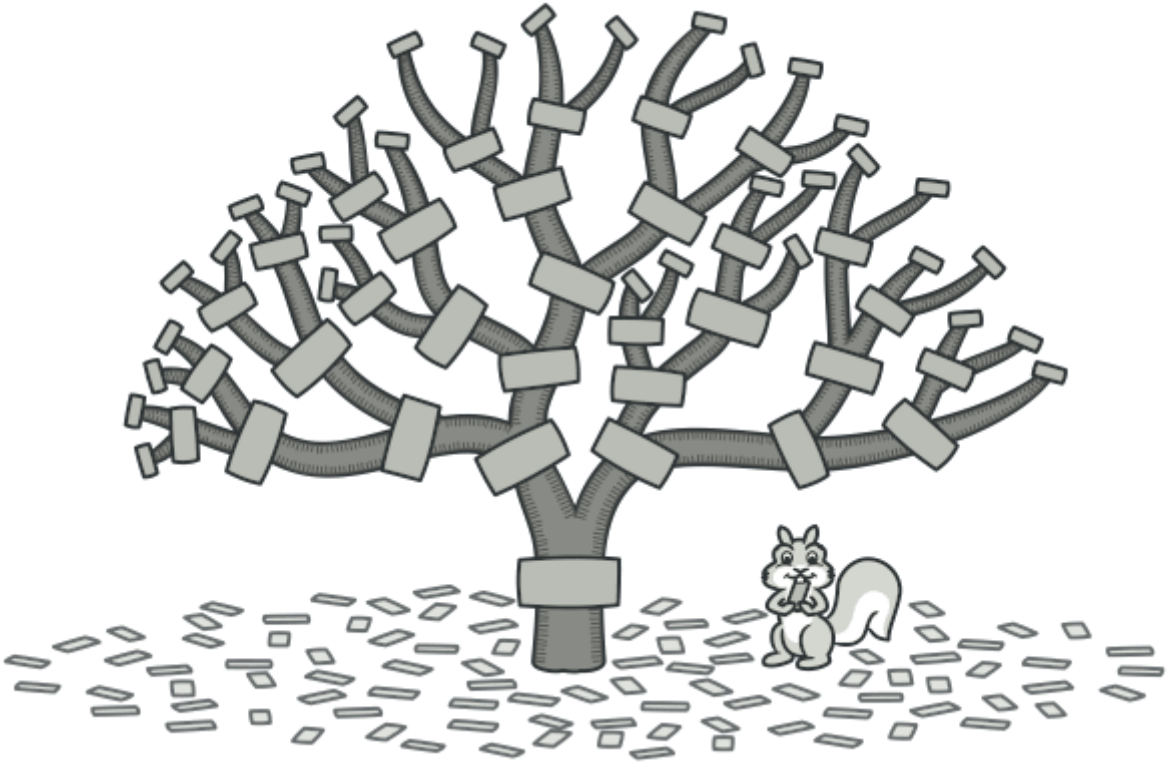
CMake 保姆级教程（下）
2023-03-15

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



像草帽大船团这样，能将多个对象组成一个树状结构，用以描述部分 — 整体的层次关系，使得用户对单个对象和组合对象的使用具有一致性，这样的结构性设计模式叫做组合模式。

现实生活中能够和组合模式对应的场景也有很多，下面举例说明：

1. Linux 的树状目录结构
2. 国家的行政区划分（省级、地级、县级、乡级）
3. 解放军编制（军、师、旅、团、营、连、排、班）
4. 公司的组织结构（树状）

2. 大决战

在海贼中，大家都预测路飞的对手应该是同为四皇的黑胡子，黑胡子手下也有很多海贼船，双方一旦开战，必定死伤无数，最后的赢家就可以得到罗杰所留下的大秘宝 `ONE PIECE`，并成为新的海贼王。

为了让路飞成为海贼王，我决定使用组合模式为路飞写一个管理 草帽大船团的程序，其对应的主要操作是这样的： 扩充船员 、 战斗减员 、 显示各船队信息 、 加入战斗 等。

2.1 团队管理

对于组合模式来说，操作这个集合中的任意一个节点的方式都是相同的，所以必须要先定义出单个节点的抽象，在这个抽象类中定义出节点的行为。

C++

```cpp
// 抽象节点
class AbstractTeam
{
public:
    AbstractTeam(string name) :m_name(name) {}
    // 设置父节点
    void setParent(AbstractTeam* node)
    {
        m_parent = node;
    }
    AbstractTeam* getParent()
    {
```

```
        return m_parent;
    }
    string getName()
    {
        return m_name;
    }
    virtual bool hasChild()
    {
        return false;
    }
    virtual void add(AbstractTeam* node) {}
    virtual void remove(AbstractTeam* node) {}
    virtual void fight() = 0;
    virtual void display() = 0;
    virtual ~AbstractTeam() {}
protected:
    string m_name;
    AbstractTeam* m_parent = nullptr;
};
```

草帽大船团中有若干个番队，这个抽象类对应的就是以船为单位的一个团队（一艘船就是一个节点），它内部定义了如下方法:

1. 设置和获得当前船队的名字
   - 设置名字：构造函数
   - 获得名字： `getName()`
2. 设置和得到当前船队节点的父节点
   - 设置父节点： `setParent(AbstractTeam* node)`

- 得到父节点： `getParent()`

3. 给当前番队添加一个子船队节点： `add(AbstractTeam* node)`
4. 跟当前番队删除一个子船队节点： `remove(AbstractTeam* node)`
5. 当前番队和敌人战斗： `fight()`
6. 显示当前番队的信息： `display()`

2.2 叶子节点

草帽大船团是一种组合模式，也就是一种树状结构，在最末端的节点就没有子节点了，这种节点可以将其称之为叶子节点。 叶子节点也是一个船队，所以它肯定是需要继承抽象节点类的。

C++

```cpp
// 叶子节点的小队
class LeafTeam : public AbstractTeam
{
public:
    using AbstractTeam::AbstractTeam;
    void fight() override
    {
        cout << m_parent->getName() + m_name + "与黑胡子的船员进行近距离肉搏战..
    }
    void display() override
    {
        cout << "我是" << m_parent->getName() << "下属的" << m_name << endl;
    }
    ~LeafTeam()
    {
```

```
            cout << "我是" << m_parent->getName() << "下属的" << m_name
                << ", 战斗已经结束, 拜拜..." << endl;
        }
};
```

叶子节点对应的番队由于没有子节点，所以在其对应的类中就不需要重写父类的 `add(AbstractTeam* node)` 和 `remove(AbstractTeam* node)` 方法了，这也是基类中为什么不把这两个虚函数指定为纯虚函数的原因。

2.3 管理者节点

所谓的管理者节点其实就是非叶子节点。这种节点还拥有子节点，它的实现肯定是需要继承抽象节点类的。

C++

```
// 管理者节点
class ManagerTeam : public AbstractTeam
{
public:
    using AbstractTeam::AbstractTeam;
    void fight() override
    {
        cout << m_name + "和黑胡子的恶魔果实能力者战斗!!!" << endl;
    }
    void add(AbstractTeam* node) override
    {
        node->setParent(this);
        m_children.push_back(node);
```

```
    }
    void remove(AbstractTeam* node) override
    {
        node->setParent(nullptr);
        m_children.remove(node);
    }
    bool hasChild()
    {
        return true;
    }
    list<AbstractTeam*> getChildren()
    {
        return m_children;
    }
    void display()
    {
        string info = string();
        for (const auto item : m_children)
        {
            if (item == m_children.back())
            {
                info += item->getName();
            }
            else
            {
                // 优先级: + > +=
                info += item->getName() + ", ";
            }
        }
        cout << m_name + "的船队是【" << info << "】" << endl;
```

在管理者节点类的内部有一个容器 `list`，容器内存储的就是它的子节点对象：

- 通过 `add(AbstractTeam* node)` 把当前番队的子节点存储到 `list` 中
- 通过 `remove(AbstractTeam* node)` 把某一个子节点从当前番队的 `list` 中删除
- 通过 `display()` 来遍历这个 `list` 容器中的节点

2.4 战斗

最后把测试程序写一下：

C++

```cpp
// 内存释放
void gameover(AbstractTeam* root)
{
    if (root == nullptr)
    {
        return;
    }
    if (root && root->hasChild())
    {
        ManagerTeam* team = dynamic_cast<ManagerTeam*>(root);
        list<AbstractTeam*> children = team->getChildren();
        for (const auto item : children)
        {
            gameover(item);
        }
    }
    delete root;
```

```cpp
}

// 和黑胡子战斗
void fighting()
{
    vector<string> nameList = {
        "俊美海贼团", "巴托俱乐部", "八宝水军", "艾迪欧海贼团",
        "咚塔塔海贼团", "巨兵海贼团", "约塔玛利亚大船团"
    };
    // 根节点
    ManagerTeam* root = new ManagerTeam("草帽海贼团");
    for (int i = 0; i < nameList.size(); ++i)
    {
        ManagerTeam* child = new ManagerTeam(nameList.at(i));
        root->add(child);
        if (i == nameList.size() - 1)
        {
            // 给最后一个番队添加子船队
            for (int j = 0; j < 9; ++j)
            {
                LeafTeam* leaf = new LeafTeam("第" + to_string(j + 1) + "番
                child->add(leaf);
                leaf->fight();
                leaf->display();
            }
```

输出的结果为:

C++

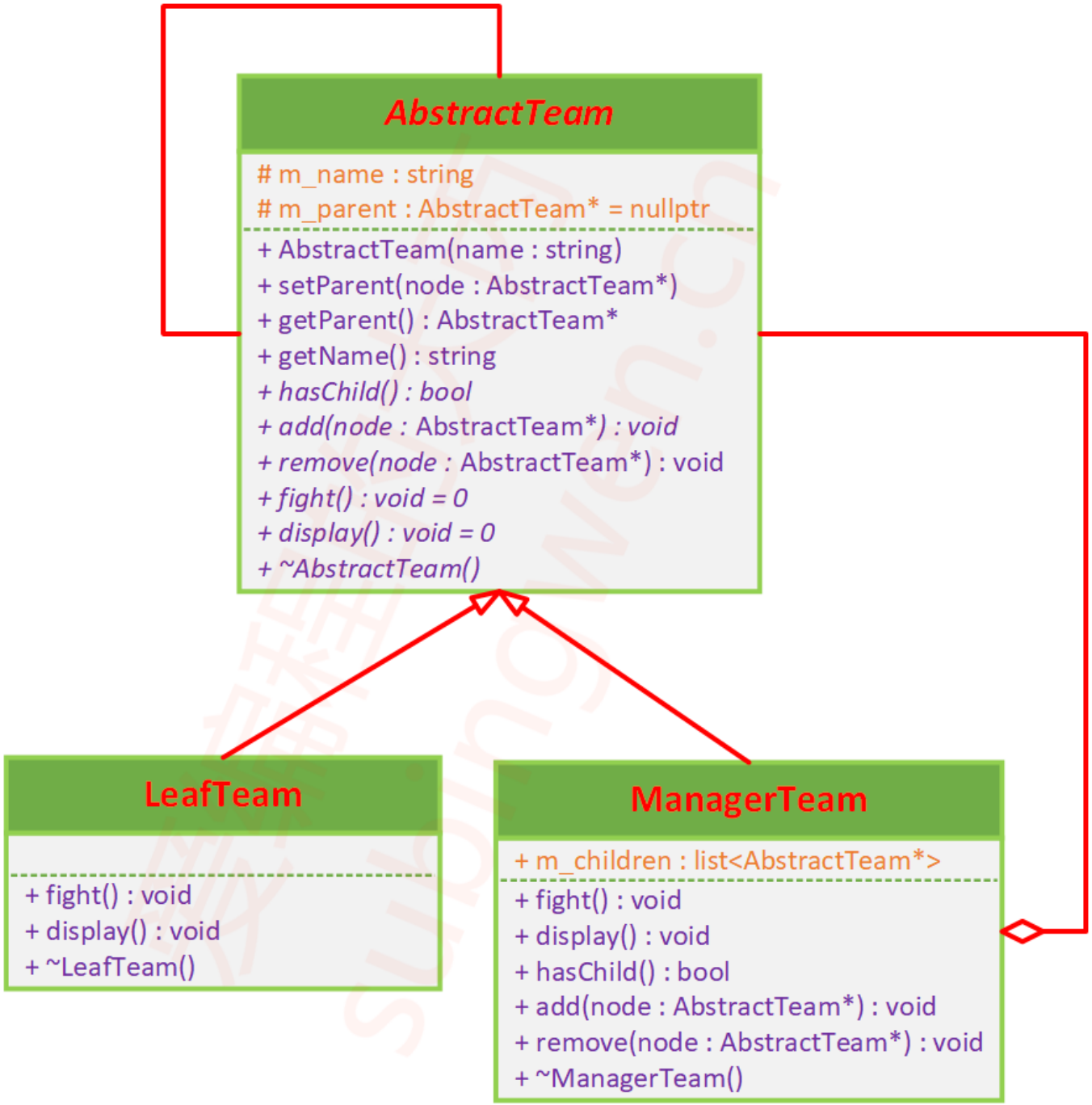
```
约塔玛利亚大船团第1番队与黑胡子的船员进行近距离肉搏战...
我是约塔玛利亚大船团下属的第1番队
约塔玛利亚大船团第2番队与黑胡子的船员进行近距离肉搏战...
我是约塔玛利亚大船团下属的第2番队
约塔玛利亚大船团第3番队与黑胡子的船员进行近距离肉搏战...
我是约塔玛利亚大船团下属的第3番队
约塔玛利亚大船团第4番队与黑胡子的船员进行近距离肉搏战...
我是约塔玛利亚大船团下属的第4番队
约塔玛利亚大船团第5番队与黑胡子的船员进行近距离肉搏战...
我是约塔玛利亚大船团下属的第5番队
约塔玛利亚大船团第6番队与黑胡子的船员进行近距离肉搏战...
我是约塔玛利亚大船团下属的第6番队
约塔玛利亚大船团第7番队与黑胡子的船员进行近距离肉搏战...
我是约塔玛利亚大船团下属的第7番队
约塔玛利亚大船团第8番队与黑胡子的船员进行近距离肉搏战...
我是约塔玛利亚大船团下属的第8番队
约塔玛利亚大船团第9番队与黑胡子的船员进行近距离肉搏战...
我是约塔玛利亚大船团下属的第9番队
约塔玛利亚大船团和黑胡子的恶魔果实能力者战斗!!!
约塔玛利亚大船团的船队是【第1番队，第2番队，第3番队，第4番队，第5番队，第6番队，第7番
草帽海贼团和黑胡子的恶魔果实能力者战斗!!!
草帽海贼团的船队是【俊美海贼团，巴托俱乐部，八宝水军，艾迪欧海贼团，咚塔塔海贼团，巨兵海
==================================
我是【俊美海贼团】战斗结束，拜拜...
我是【巴托俱乐部】战斗结束，拜拜...
我是【八宝水军】战斗结束，拜拜...
我是【艾迪欧海贼团】战斗结束，拜拜...
我是【咚塔塔海贼团】战斗结束，拜拜...
我是【巨兵海贼团】战斗结束，拜拜...
我是约塔玛利亚大船团下属的第1番队，战斗已经结束，拜拜...
我是约塔玛利亚大船团下属的第2番队，战斗已经结束，拜拜...
```

```
我是约塔玛利亚大船团下属的第3番队，战斗已经结束，拜拜...
我是约塔玛利亚大船团下属的第4番队，战斗已经结束，拜拜...
我是约塔玛利亚大船团下属的第5番队，战斗已经结束，拜拜...
我是约塔玛利亚大船团下属的第6番队，战斗已经结束，拜拜...
我是约塔玛利亚大船团下属的第7番队，战斗已经结束，拜拜...
我是约塔玛利亚大船团下属的第8番队，战斗已经结束，拜拜...
我是约塔玛利亚大船团下属的第9番队，战斗已经结束，拜拜...
我是【约塔玛利亚大船团】战斗结束，拜拜...
我是【草帽海贼团】战斗结束，拜拜...
```

由于草帽大船团对应的设计模式是组合模式，它对应的是一个树模型，并且每个节点的操作方式都形同，所以在释放节点的时候就可以使用递归了， `gameover()`函数就是一个递归函数。

3. 结构图

学完了组合模式，根据上面的例子把对应的 UML 类图画一下（学会之后就得先画类图，再写程序了）


AbstractTeam
m_name : string
m_parent : AbstractTeam* = nullptr
+ AbstractTeam(name : string)
+ setParent(node : AbstractTeam*)
+ getParent() : AbstractTeam*
+ getName() : string
+ hasChild() : bool
+ add(node : AbstractTeam*) : void
+ remove(node : AbstractTeam*) : void
+ fight() : void = 0
+ display() : void = 0
+ ~AbstractTeam()
LeafTeam
+ fight() : void
+ display() : void
+ ~LeafTeam()
ManagerTeam
+ m_children : list<AbstractTeam*>
+ fight() : void
+ display() : void
+ hasChild() : bool
+ add(node : AbstractTeam*) : void
+ remove(node : AbstractTeam*) : void
+ ~ManagerTeam()

为了能够更加清楚地描述出设计模式中的组合关系（不是 UML 中的组合关系），在 `AbstractTeam` 和 `ManagerTeam` 之间画了两条线：

- 继承关系：对节点的操作使用的是抽象类中提供的接口，以保证操作的一致性
- 聚合关系： `ManagerTeam` 类型的节点还可以有子节点，父节点和子节点的之间的关系需要具体问题具体分析
  - 子节点跟随父节点一起销毁，二者就是组合关系（UML 中的组合关系）
  - 子节点不跟随父节点一起销毁，二者就是聚合关系
  - 上面的程序中，在父节点的析构函数中没有销毁它管理的子节点，所以在上图中标记的是聚合关系

设计模式

相关推荐

评论

昵称 邮箱 网址(http://)

来都来了, 说点什么吧...

提交

来发评论吧 ~

Powered By Valine

v1.5.1