建造者模式 - 卡雷拉公司

发表于 2022-09-02 | 更新于 2023-04-06 | 设计模式

| 字数总计: 2.8k | 阅读时长: 9 分钟 | 阅读量: 995 | 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. 造船，我是专业的

在海贼世界中， 水之都 拥有全世界最好的造船技术，三大古代兵器之一的 冥王 就是由岛上的造船技师们制造出来的。现在岛上最大、最优秀的造船公司就是卡雷拉公司，它的老板还是水之都的市长，财富权力他都有，妥妥的人生赢家。

众所周知，在冰山身边潜伏着很多卧底，他们都是 世界政府直属秘密谍报机关 CP9 成员，目的是要得到古代兵器冥王的设计图，但是很不幸图纸后来被弗兰奇烧掉了。既然他们造船这么厉害，我也来到了卡雷拉公司，学习一下他们是怎么造船的。

文章 134 标签 37 分类 12

大丙课堂

公告

微信公众号 爱编程的大丙 和 大丙课堂 上线了，可点击上方 图标关注～～～

目录

最新文章

1.1 桑尼号

以下是我拿到的冰山和弗兰奇给路飞造的 桑尼号 的部分设计图纸:

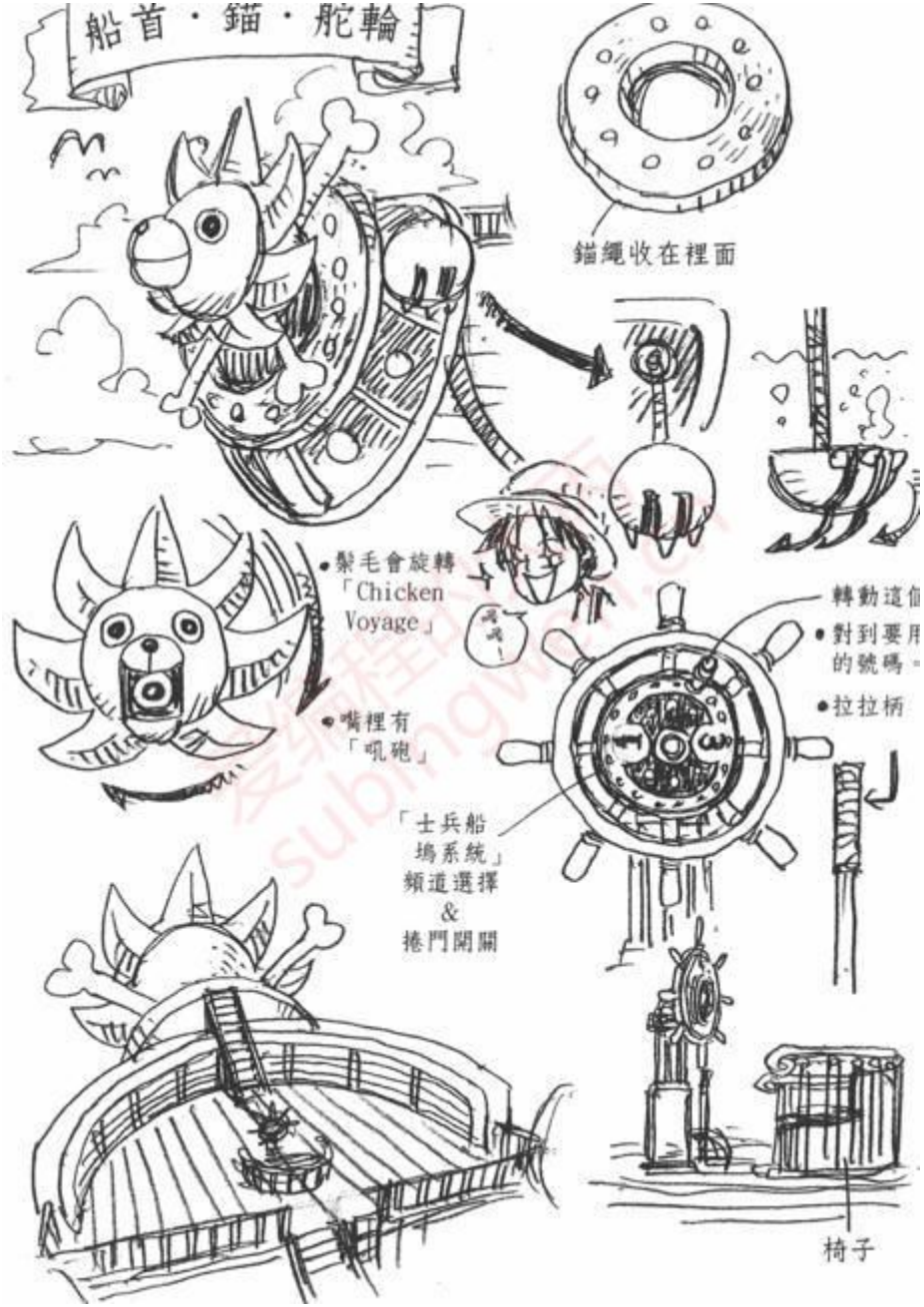
CMake 保姆级教程（下）
2023-03-15

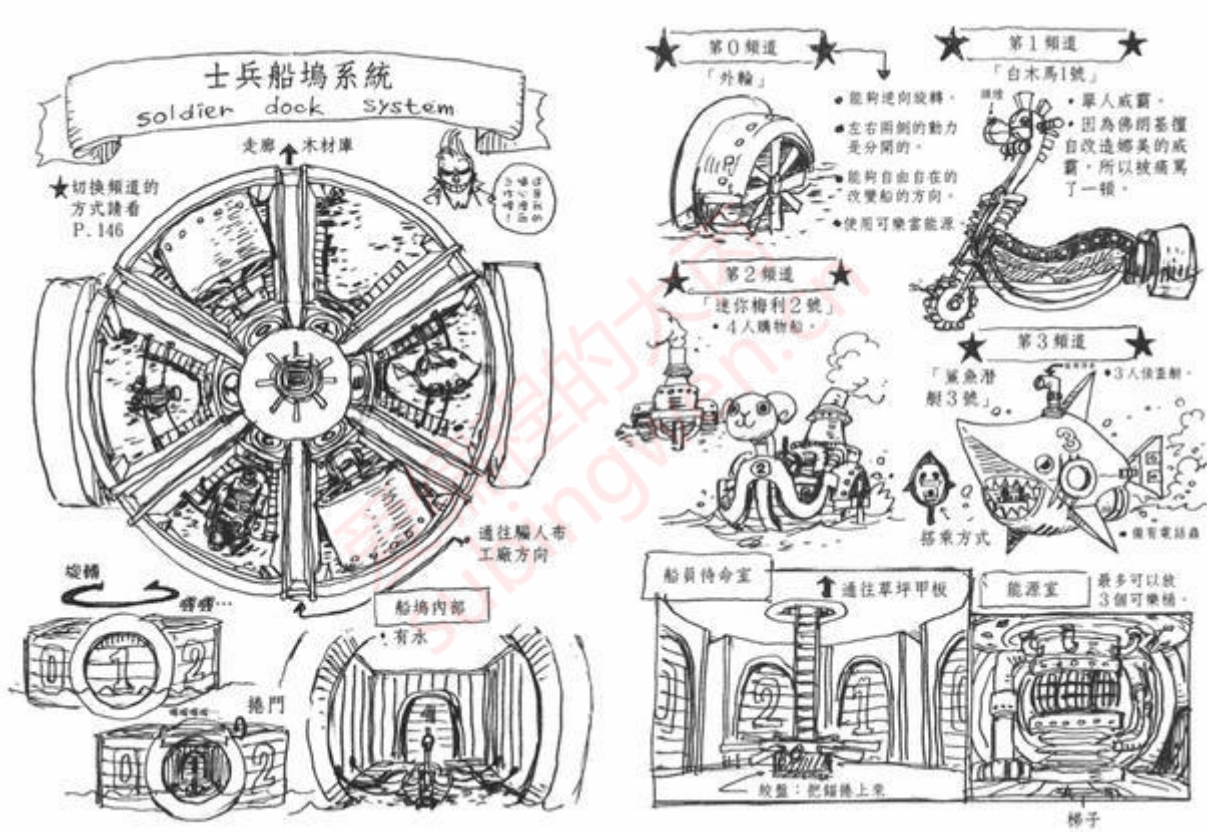
CMake 保姆级教程（上）
2023-03-06

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


船首・錨・舵輪
錨繩收在裡面
●梨毛會旋轉
「Chicken
Voyage」
●嘴裡有
「吼砲」
轉動這個
●對到要用
的號碼
●拉拉柄
「士兵船
塢系統」
頻道選擇
&
捲門開關
椅子

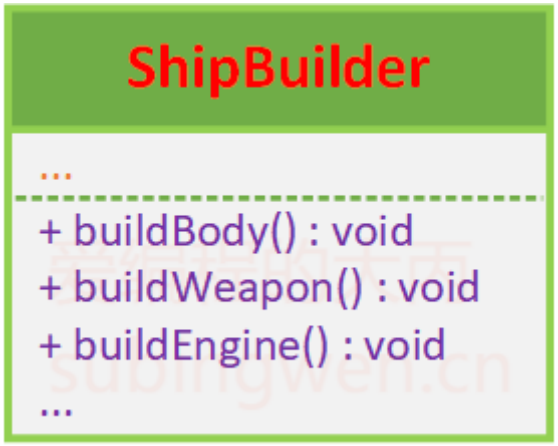


通过图纸可以感受到造一艘船的工序是极其复杂的，看到这儿，曾经作为程序猿的我又想到了写程序，如果只通过一个类直接把这种结构的船构建出来完全是不可能，先不说别的光构造函数的参数就已经不计其数了。

冰山给出的解决方案是 化繁为简，逐个击破。也就是分步骤创建复杂的对象，并且允许使用相同的代码生成不同类型和形式的对象 ，他说这种模式叫做 生成器模式（也叫建造者模式） 。

1.2 生成器

生成器模式建议将造船工序的代码从产品类中抽取出来， 并将其放在一个名为 生成器 的独立类中。



在这个生成器类中有一系列的构建步骤，每次造船的时候，只需 从中选择需要的步骤并调用 就可以得到满足需求的实例对象。

假设我们要通过上面的生成器建造很多不同规格、型号的海贼船，那么就需要创建多个生成器，但是有一点是不变的： 生成器内部的构建步骤不变。

	简约型	标准型	豪华型
船体	有	有	有
动力	有	有	有
武器	无	有	有
内室	毛坯	毛坯	精装

比如，我想建造两种型号的海贼船： 桑尼号 和 梅利号 ， 并且按照上面的三个规格，每种造一艘，此时就需要两个生成器： 桑尼号生成器 和 梅利号生成器 ， 并且这两个生成器还需要对应一个父类，父类生成器中的建造函数应该设置为虚函数。

1.3 主管

冰山说可以进一步将用于创建产品的一系列生成器步骤调用抽取成为单独的 主管类 。 主管类可定义创建步骤的执行顺序， 而生成器则提供这些步骤的实现。

严格来说， 程序中并不一定需要主管类。 客户端代码可直接以特定顺序调用创建步骤。 不过， 主管类中非常适合放入各种例行构造流程， 以便在程序中反复使用。

此外， 对于客户端代码来说， 主管类完全隐藏了产品构造细节。 客户端只需要将一个生成器与主管类关联， 然后使用主管类来构造产品， 就能从生成器处获得构造结果了。

2. 说干就干

2.1 船模型

现在我们开始着手把路飞的海贼船 桑尼号 和 梅利号 使用生成器模式键造出来。

- 一共需要三个生成器类，一共父类，两个子类
- 父类可以是一个抽象类，提供的建造函数都是虚函数
- 在两个生成器子类中，使用建造函数分别将 桑尼号 和 梅利号 各个零部件造出来。

如果我们仔细分析，发现还需要解决另外一个问题，通过生成器得到了海贼船的各个零部件，这些零部件必须有一个载体，那就是 海贼船对象 。因此，还需要提供一个或多个海贼船类。

因为 桑尼号 和 梅利号 这两艘的差别非常之巨大，所以我们定义两个海贼船类，代码如下：

C++

```
// 桑尼号
class SunnyShip
{
public:
    // 添加零件
    void addParts(string name)
    {
        m_parts.push_back(name);
    }
    void showParts()
```

```
    {
        for (const auto& item : m_parts)
        {
            cout << item << "   ";
        }
        cout << endl;
    }
private:
    vector<string> m_parts;
};

// 梅利号
class MerryShip
{
public:
    // 组装
    void assemble(string name, string parts)
    {
        m_patrs.insert(make_pair(name, parts));
    }
    void showParts()
    {
        for (const auto& item : m_patrs)
        {
            cout << item.first << ": " << item.second << "  ";
        }
        cout << endl;
    }
private:
    map<string, string> m_patrs;
};
```

在上面的两个类中，通过一个字符串来代表某个零部件，为了使这两个类有区别 `SunnyShip 类` 中使用 `vector 容器` 存储数据， `MerryShip 类` 中使用 `map 容器` 存储数据。

2.2 船生成器

虽然有海贼船类，但是这两个海贼船类并不造船，每艘船的零部件都是由他们对应的生成器类构建完成的，下面是生成器类的代码：

抽象生成器

C++

```
// 生成器类
class ShipBuilder
{
public:
    virtual void reset() = 0;
    virtual void buildBody() = 0;
    virtual void buildWeapon() = 0;
    virtual void buildEngine() = 0;
    virtual void buildInterior() = 0;
    virtual ~ShipBuilder() {}
};
```

在这个抽象类中定义了建造海贼船所有零部件的方法，在这个类的子类中需要重写这些虚函数，分别完成 桑尼号 和 梅利号 零件的建造。

桑尼号生成器

C++

```cpp
// 桑尼号生成器
class SunnyBuilder : public ShipBuilder
{
public:
    SunnyBuilder()
    {
        reset();
    }
    ~SunnyBuilder()
    {
        if (m_sunny =/= nullptr)
        {
            delete m_sunny;
        }
    }
    // 提供重置函数，目的是能够使用生成器对象生成多个产品
    void reset() override
    {
        m_sunny = new SunnyShip;
    }
    void buildBody() override
    {
        m_sunny→addParts("神树亚当的树干");
    }
    void buildWeapon() override
    {
        m_sunny→addParts("狮吼炮");
    }
    void buildEngine() override
    {
```

```
        m_sunny->addParts("可乐驱动");
    }
    void buildInterior() override
    {
        m_sunny->addParts("豪华内室精装");
    }
    SunnyShip* getSunny()
    {
        SunnyShip* ship = m_sunny;
        m_sunny = nullptr;
        return ship;
    }
private:
    SunnyShip* m_sunny = nullptr;
};
```

在这个生成器类中只要调用 `build 方法`，对应的零件就会被加载到 `SunnyShip 类的对象 m_sunny 中`，当船被造好之后就可以通过 `SunnyShip* getSunny()` 方法得到 `桑尼号的实例对象`，当这个对象地址被外部指针接管之后，当前生成器类就不会再维护其内存的释放了。如果想通过生成器对象建造第二艘桑尼号就可以调用这个类的 `reset()` 方法，这样就得到了一个新的桑尼号对象，之后再调用相应的建造函数，这个对象就被初始化了。

梅利号生成器

C++

```
// 梅利号生成器
class MerryBuilder : public ShipBuilder
{
public:
```

```
    MerryBuilder()
    {
        reset();
    }
    ~MerryBuilder()
    {
        if (m_merry =/= nullptr)
        {
            delete m_merry;
        }
    }
    void reset() override
    {
        m_merry = new MerryShip;
    }
    void buildBody() override
    {
        m_merry->assemble("船体", "优质木材");
    }
    void buildWeapon() override
    {
        m_merry->assemble("武器", "四门大炮");
    }
    void buildEngine() override
    {
        m_merry->assemble("动力", "蒸汽机");
    }
    void buildInterior() override
    {
        m_merry->assemble("内室", "精装");
    }
```

```
36      MerryShip* getMerry()
37      {
38          MerryShip* ship = m_merry;
39          m_merry = nullptr;
40          return ship;
41      }
42  private:
43      MerryShip* m_merry = nullptr;
44  };
```

梅利号的生成器和桑尼号的生成器内部做的事情是一样的，在此就不过多赘述了。

2.3 包工头

如果想要隐藏造船细节，就可以添加一个主管类，这个主管类就相当于一个包工头，脏活累活他都干了，我们看到的就是一个结果。

根据需求，桑尼号和梅利号分别有三个规格， 简约型 、 标准型 、 豪华型 ，根据不同的规格，有选择的调用生成器中不同的建造函数，就可以得到最终的成品了。

C++

```
1  // 主管类
2  class Director
3  {
4  public:
5      void setBuilder(ShipBuilder* builder)
6      {
7          m_builder = builder;
8      }
```

```
    // 简约型
    void builderSimpleShip()
    {
        m_builder->buildBody();
        m_builder->buildEngine();
    }
    // 标准型
    void builderStandardShip()
    {
        builderSimpleShip();
        m_builder->buildWeapon();
    }
    // 豪华型
    void builderRegalShip()
    {
        builderStandardShip();
        m_builder->buildInterior();
    }
private:
    ShipBuilder* m_builder = nullptr;
};
```

在使用主管类的时候，需要通过 `setBuilder(ShipBuilder* builder)` 给它的对象传递一个 生成器对象 ，形参是父类指针，实参应该是子类对象，这样做的目的是为了实现多态，并且在这个地方这个函数是一个 传入传出参数 。

3. 验收

最后测试一个桑尼号和梅利号分别对应的三种规格的船能否被建造出来：

C++

```cpp
// 建造桑尼号
void builderSunny()
{
    Director* director = new Director;
    SunnyBuilder* builder = new SunnyBuilder;
    // 简约型
    director->setBuilder(builder);
    director->builderSimpleShip();
    SunnyShip* sunny = builder->getSunny();
    sunny->showParts();
    delete sunny;

    // 标准型
    builder->reset();
    director->setBuilder(builder);
    director->builderStandardShip();
    sunny = builder->getSunny();
    sunny->showParts();
    delete sunny;

    // 豪华型
    builder->reset();
    director->setBuilder(builder);
    director->builderRegalShip();
    sunny = builder->getSunny();
    sunny->showParts();
    delete sunny;
    delete builder;
```

```
    delete director;
}

// 建造梅利号
void builderMerry()
{
    Director* director = new Director;
    MerryBuilder* builder = new MerryBuilder;
    // 简约型
    director->setBuilder(builder);
    director->builderSimpleShip();
    MerryShip* merry = builder->getMerry();
    merry->showParts();
    delete merry;

```

程序输出:

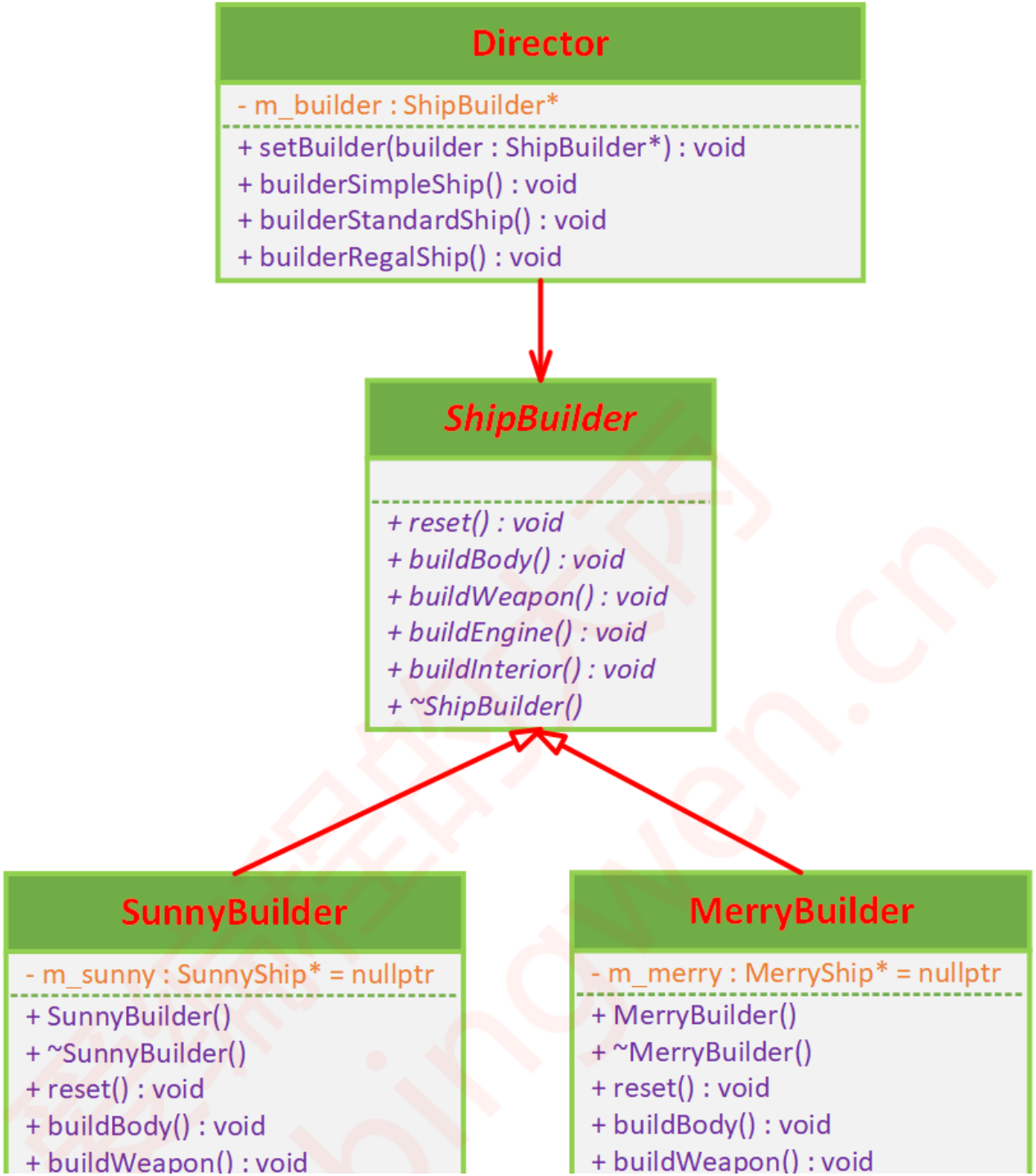
C++

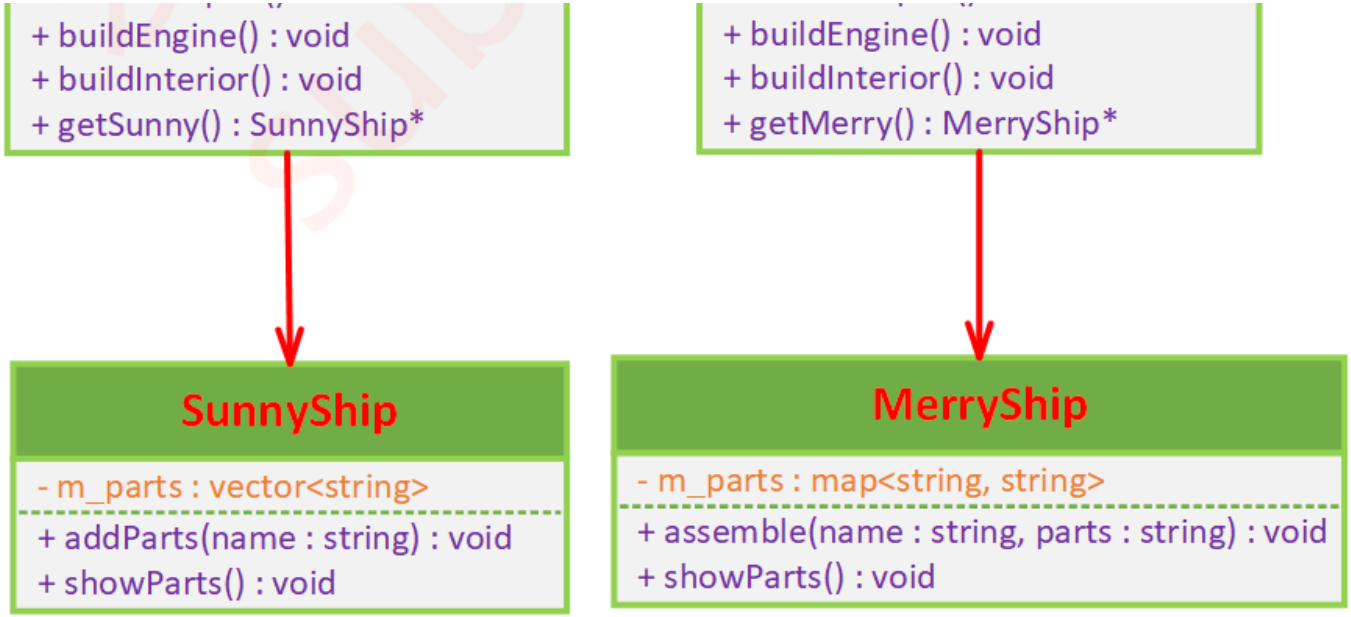
```
神树亚当的树干    可乐驱动
神树亚当的树干    可乐驱动    狮吼炮
神树亚当的树干    可乐驱动    狮吼炮    豪华内室精装
====================================
船体: 优质木材   动力: 蒸汽机
船体: 优质木材   动力: 蒸汽机   武器: 四门大炮
船体: 优质木材   动力: 蒸汽机   内室: 精装   武器: 四门大炮
```

可以看到，输出结果是没问题的，使用生成器模式造船成功!

4. 收工

最后根据上面的代码把 UML 类图画一下（在学习设计模式的时候只能最后出图，在做项目的时候应该是先画 UML 类图，再写程序）。


Director
- m_builder : ShipBuilder*
+ setBuilder(builder : ShipBuilder*) : void
+ builderSimpleShip() : void
+ builderStandardShip() : void
+ builderRegalShip() : void
ShipBuilder
+ reset() : void
+ buildBody() : void
+ buildWeapon() : void
+ buildEngine() : void
+ buildInterior() : void
+ ~ShipBuilder()
SunnyBuilder
- m_sunny : SunnyShip* = nullptr
+ SunnyBuilder()
+ ~SunnyBuilder()
+ reset() : void
+ buildBody() : void
+ buildWeapon() : void
MerryBuilder
- m_merry : MerryShip* = nullptr
+ MerryBuilder()
+ ~MerryBuilder()
+ reset() : void
+ buildBody() : void
+ buildWeapon() : void



通过编写的代码可得知 `Director 类` 和 `ShipBuilder 类` 之间有两种关系 依赖 和 关联，但在描述这二者的关系的时候只能画一条线，一般会选择最紧密的那个关系，在此处就是 关联关系 。

在这个图中，没有把使用这用这些类的客户端画出来，这个客户端对应的是上面程序中的 `main()` 函数中调用的测试代码，在真实场景中对应的应该是一个客户端操作界面，由用户做出选择，从而在程序中根据选择建造不同型号，不同规格的海贼船。

设计模式

打赏


上一篇
适配器模式


设计模式
原型模式
下一篇

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

1 评论

Anonymous Edge 110.0.1587.50 Windows 11

2023-02-22

回复

Powered By Valine

v1.5.1