UML 类图

发表于 2022-08-27 | 更新于 2023-04-06 | 设计模式
| 字数总计: 3.1k | 阅读时长: 10 分钟 | 阅读量: 3339 | 评论数: 4

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

面向对象设计主要就是使用 UML 的类图，类图用于描述系统中所包含的类以及它们之间的相互关系，帮助人们简化对系统的理解，它是**系统分析和设计阶段的重要产物，也是系统编码和测试的重要模型依据。**下面基于 C++ 这门语言给大家讲一下 UML 类图的画法。

类的 UML 画法

类（class /struct）封装了数据和行为，是面向对象的重要组成部分，它是具有相同 属性 、 操作 、 关系 的对象集合的总称。在系统中，每个类都具有一定的职责，职责指的是类要完成什么样子的功能，要承担什么样子的义务。一个类可以有多种职责，但是设计得好的类一般只有一种职责。

比如，我现在定义了猎人类：

C++

```
class Hunter
{
public:
    int m_age = 32;
    static int m_times;
    string getName()
    {
        return m_name;
    }

    void setName(string name)
    {
```


目录 3

```
15
16      void goHunting()
17      {
18          aiming();
19          shoot();
20      }
21      static void saySorry()
22      {
23          string count = to_string(m_times);
24          cout << "Say sorry to every animal " + count + " times!" << endl;
25      }
26
27  protected:
28      string m_name = "Jack";
29      void aiming()
30      {
31          cout << "使用" + m_gunName + "瞄准猎物..." << endl;
32      }
33
34  private:
35      string m_gunName = "AK-47";
36      void shoot()
37      {
38          cout << "使用" + m_gunName + "射击猎物..." << endl;
39      }
40  };
41  int Hunter::m_times = 3;
```

上面这个类对应的类图应该是这样的：

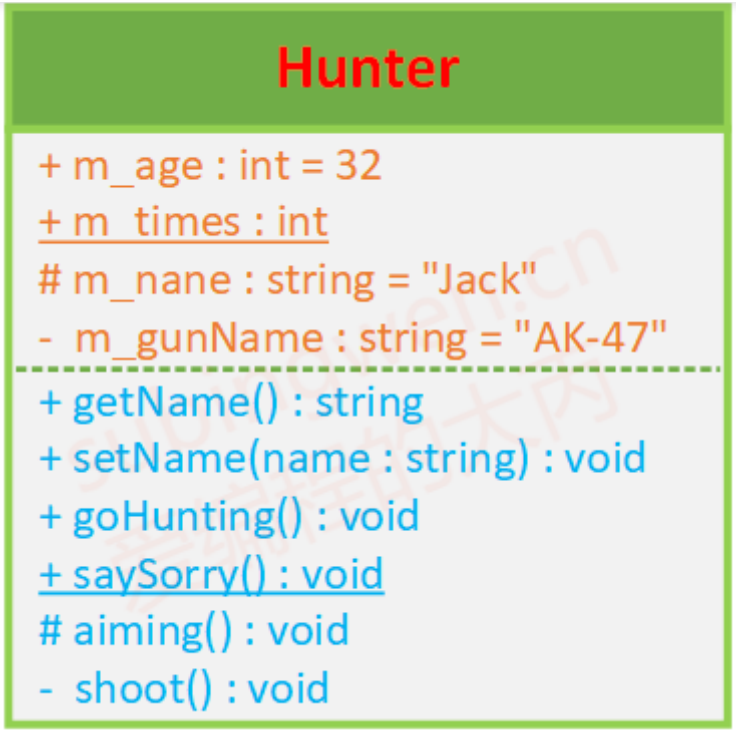
CMake 保姆级教程（下）
2023-03-15

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

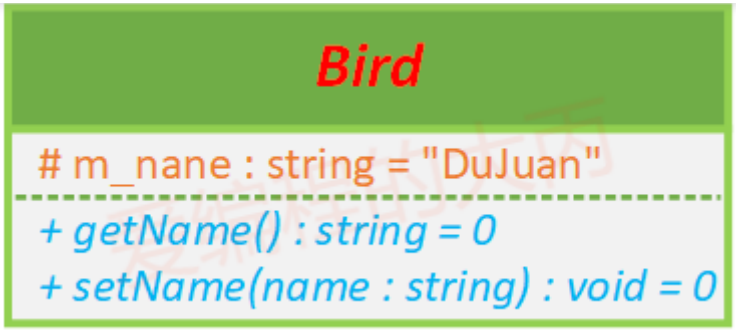
状态模式 - 文斯莫克・山治
2022-09-20



可以看到该图分为上中下三部分：上层是类名，中间层是属性（类的成员变量），下层是方法（类的成员函数）。

- 可见性：`+` 表示 `public`、`#` 表示 `protected`、`-` 表示 `private`、`__(下划线)` 表示 `static`
- 属性的表示方式：【可见性】【属性名称】:【类型】= {缺省值，可选}
- 方法的表示方式：【可见性】【方法名称】(【参数名：参数类型，.....】)：【返回值类型】

如果我们定义的类是一个 抽象类 （类中有纯虚函数），在画 UML 类图的时候， 类名需要使用斜体显



> 在使用 UML 画类图的时候，虚函数的表示方跟随类名，也就是使用斜体，如果是纯虚函数则需要在最后给函数指定 `=0` 。

类与类之间的关系

继承关系

继承也叫作泛化（Generalization），用于描述父子类之间的关系，父类又称为基类或者超类，子类又称作派生类。在 UML 中，泛化关系用 带空心三角形的实线 来表示。

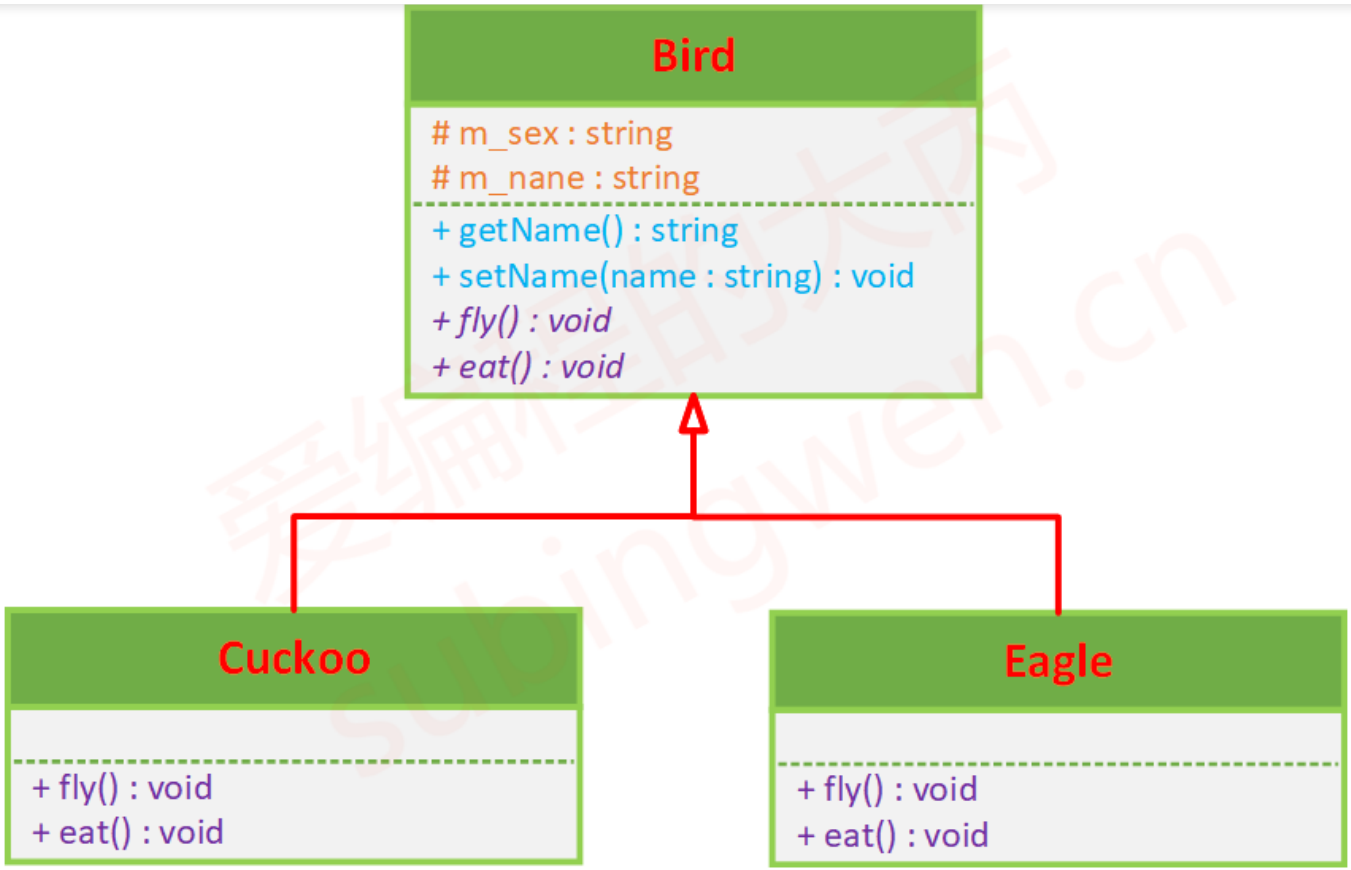
关于继承关系一共有两种： 普通继承关系 和 抽象继承关系 ，但是不论哪一种表示继承关系的线的样式是不变的。

假如现在我定义了一个父类（ `Bird` ）和两个子类（ `Cuckoo` 、 `Eagle` ）：

```
class Bird
{
public:
    string getName()
    {
        return m_name;
    }

    void setName(string name)
    {
        m_name = name;
    }

    virtual void fly() {}
    virtual void eat() {}
protected:
    string m_sex;
    string m_name;
};

class Cuckoo : public Bird
{
public:
    void fly() override
    {
        cout << "我拍打翅膀飞行..." << endl;
    }

    void eat() override
```

```
    }
};

class Eagle : public Bird
{
public:
    void fly() override
    {
        cout << "我展翅翱翔..." << endl;
    }

    void eat() override
```

使用 UML 表示上述这种关系应当是：



父类 `Bird` 中的 `fly()` 和 `eat()` 是虚函数，它有两个子类 `Cuckoo` 和 `Eagle` 在这两个子类中重写了父类的虚函数，在使用 带空心三角的实现 表示继承关系的时候， 有空心三角的一端指向父类，另一端连接子类 。

关联关系

关联（Association）关系是类与类之间最常见的一种关系，它是一种结构化的关系，表示一个对象与

或不带箭头的）实线连接有关联关系的类 。在 C++ 中这种关联关系在类中是这样体现的， 通常将一个类的对象作为另一个类的成员变量 。

类之间的关联关系有三种，分别是： 单向关联 、 双向关联 、 自关联 。下面逐一给大家进行介绍。

单向关联关系

单向关联指的是关联只有一个方向，比如每个孩子（Child）都拥有一个父亲（Parent），其代码实现为：

C++

```
class Parent
{
};

class Child
{
private:
    Parent m_father;
};
```

通过 UML 来说描述这两个类之间的关系，如下图：

如果是单向关联，使用的连接线是带单向箭头的实线，哪个类作为了当前类的成员变量，那么箭头就指向哪个类。在这个例子中 `Parent 类` 作为了 `Child 类` 的成员变量，因此箭头端应该指向 `Parent 类`，另一端连接 `Child 类`。

双向关联关系

现实生活中每个孩子都有父母，每个父母同样有自己的孩子，如果想要通过类来描述这样的亲情关系，代码如下：

C++

```
class Parent
{
private:
    Child m_son;
};

class Child
{
private:
    Parent m_father;
};
```

通过 UML 来说描述这两个类之间的关系，如下图：

在画 UML 类图的时候，一般使用 没有箭头的实线 来连接有双向关联关系的两个类，这两个类的对象分别作为了对方类的成员变量。

> 有些 UML 绘图软件使用的是带双向箭头的实线来表示双向关联关系。

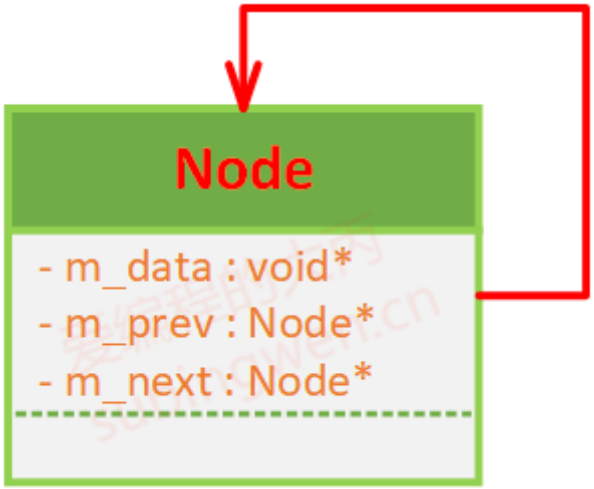
自关联关系

自关联指的就是 当前类中包含一个自身类型的对象成员 ，这在链表中非常常见，单向链表中都会有一个指向自身节点类型的后继指针成员，而双向链表中会包含一个指向自身节点类型的前驱指针和一个指向自身节点类型的后继指针。就以双向链表节点类为例，它的 C++ 写法为：

C++

```
3  private:
4      void* m_data;
5      Node* m_prev;
6      Node* m_next;
7  };
```

对应的 UML 类图应当是：



一般使用 带箭头的实线 来描述自关联关系，我中有我，独角戏。

> 有些 UML 绘图软件表示类与类的关联关系，使用的就是一条实线，没有箭头。

聚合关系

聚合（Aggregation）关系表示**整体与部分**的关系。在聚合关系中，**成员对象是整体的一部分，但是**

聚合关系的例子：

- 汽车（Car）与 引擎（Engine）、轮胎（Wheel）、车灯（Light）
- 森林（Forest）与 植物（Plant）、动物（Animal）、水（Water）、阳光（Sunshine）

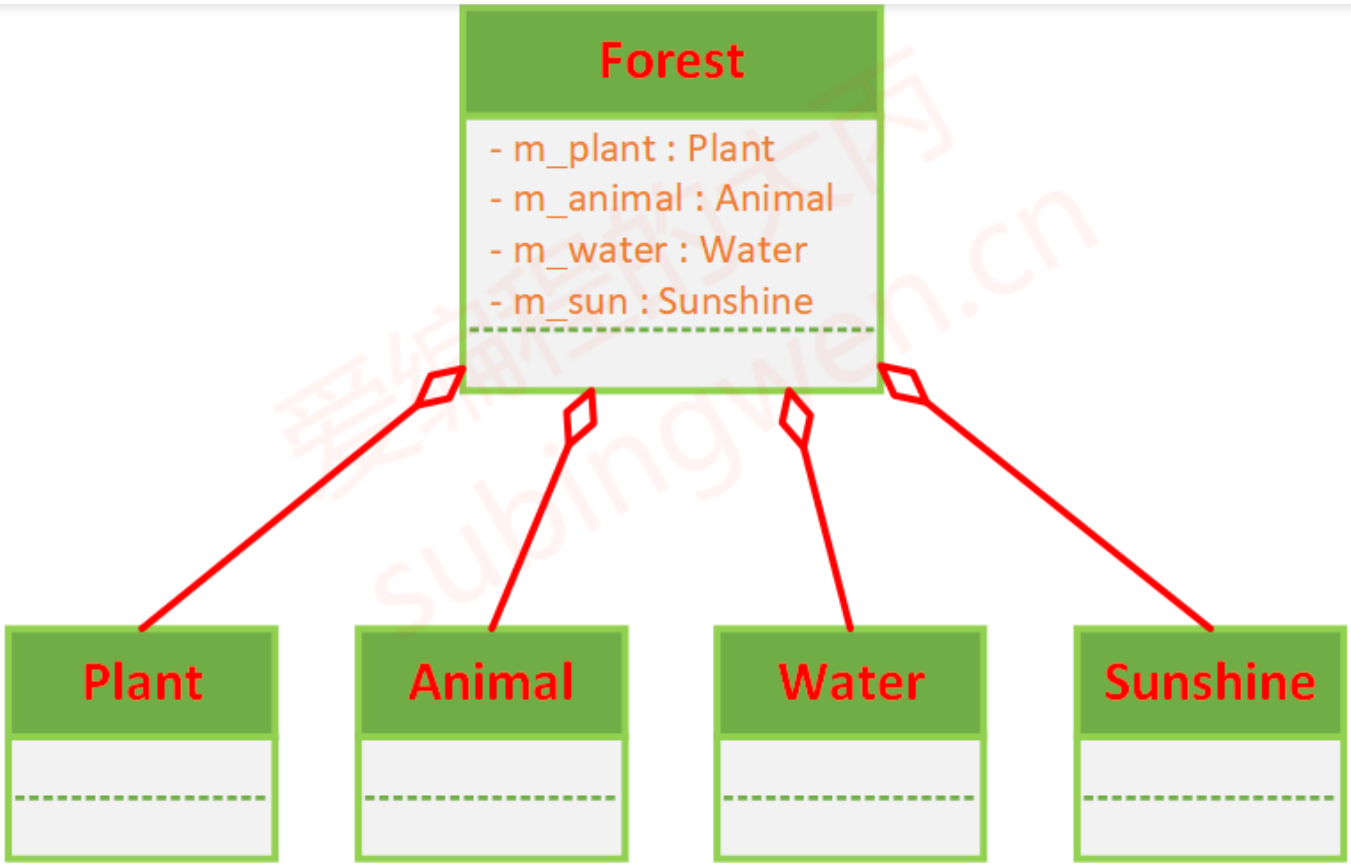
以森林为例，对应的 C++ 类的定义如下：

C++

```cpp
class Plant
{
    // 植物
};

class Animal
{
    // 动物
};

class Water
{
    // 水
};

class Sunshine
{
    // 阳光
};

```

```
23  public:
24      Forest(Plant p, Animal a, Water w, Sunshine s) :
25          m_plant(p),m_animal(a),m_water(w),m_sun(s)
26      {
27      }
28  private:
29      Plant m_plant;
30      Animal m_animal;
31      Water m_water;
32      Sunshine m_sun;
33  };
```

对应的 UML 类图为：



代码实现聚合关系，成员对象通常以构造方法、Setter 方法的方式注入到整体对象之中，因为成员对象可以脱离整体对象独立存在。

表示聚合关系的线，有空心菱形的一端指向整体对象，另一端连接局部对象（有些 UML 绘图软件在这一端还带一个箭头）。

组合关系

组合（Composition）关系也表示的是一种整体和部分的关系，但是在组合关系中整体对象可以控制成员对象的生命周期，一旦整体对象不存在，成员对象也不存在，**整体对象和成员对象之间具有同生共死的关系。**

在 UML 中组合关系用带实心菱形的直线表示，下面举两个组合关系的例子：

- 头（Head）和 嘴巴（Mouth）、鼻子（Nose）、耳朵（Ear）、眼睛（Eye）
- 树（Tree）和 树根（Root）、树干（Trunk）、树枝（Branch）、树叶（Leaf）

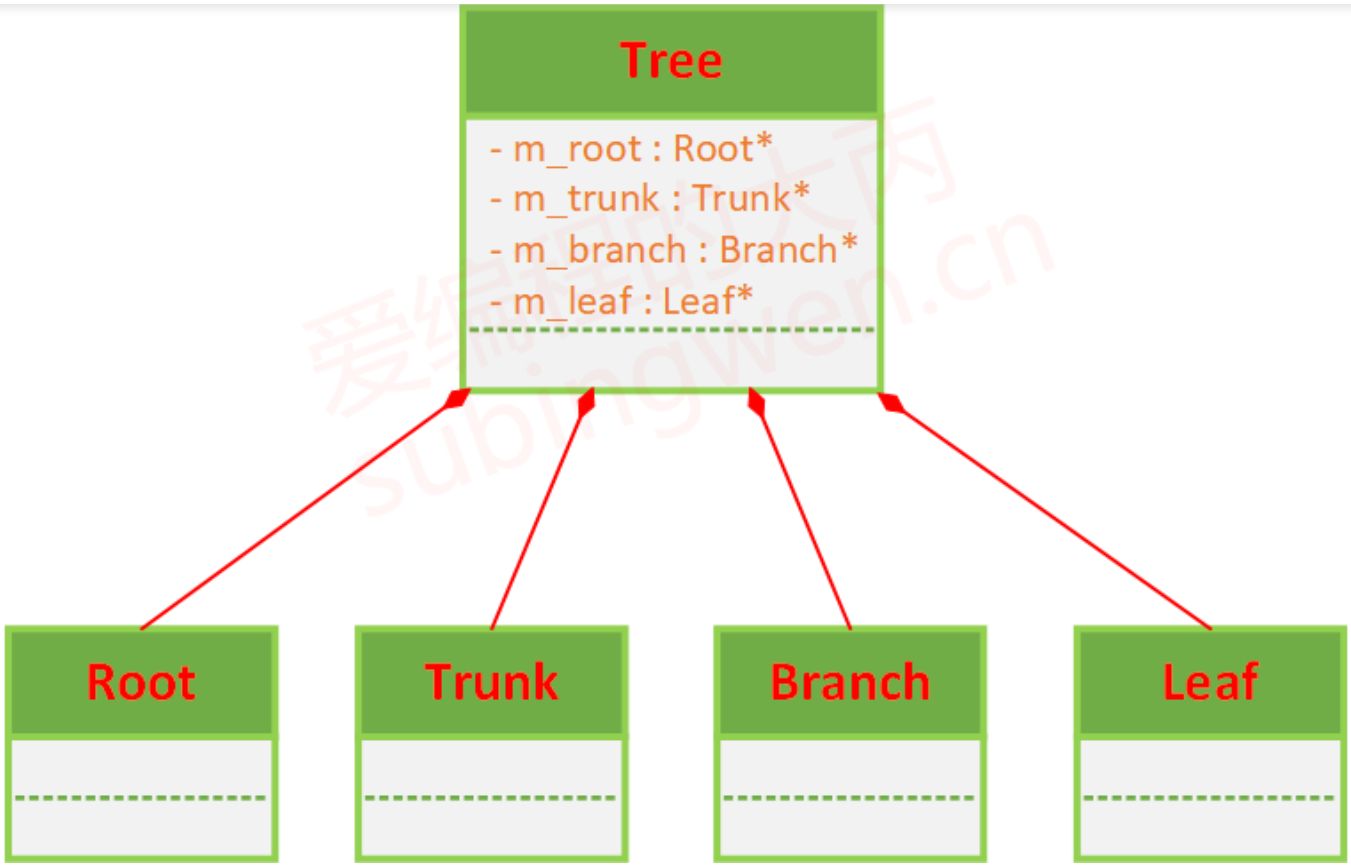
以树为例，对应的 C++ 类的定义如下：

C++

```
class Root
{
};

class Trunk
{
};

class Branch
{
};

class Leaf
{
};
```

```
{
public:
    Tree()
    {
        m_root = new Root;
        m_trunk = new Trunk;
        m_branch = new Branch;
        m_leaf = new Leaf;
    }
    ~Tree()
    {
        delete m_root;
        delete m_trunk;
        delete m_branch;
        delete m_leaf;
    }
private:
    Root* m_root;
    Trunk* m_trunk;
    Branch* m_branch;
    Leaf* m_leaf;
};
```

其 UML 的表示方法为：



代码实现组合关系，通常在**整体类的构造方法中直接实例化成员类**，因为组合关系的整体和部分是共生关系，整体的实例对象被析构的时候它的子对象也会一并被析构。如果通过外部注入，即使整体不存在了，部分还是存在的，这样的话就变成聚合关系了。

依赖关系

依赖（Dependency）关系是一种使用关系，特定事物的改变有可能会影响到使用该事物的其他事

在 UML 中，依赖关系用带箭头的虚线表示，由依赖的一方指向被依赖的一方，下面举两个聚合关系的例子：

- 驾驶员（Driver）开车，需要将车（Car）对象作为参数传递给 Driver 类的 drive（）方法。

C++

```
class Car
{
public:
    void move() {}
};

class Driver
{
public:
    void drive(Car car)
    {
        car.move();
    }
};
```

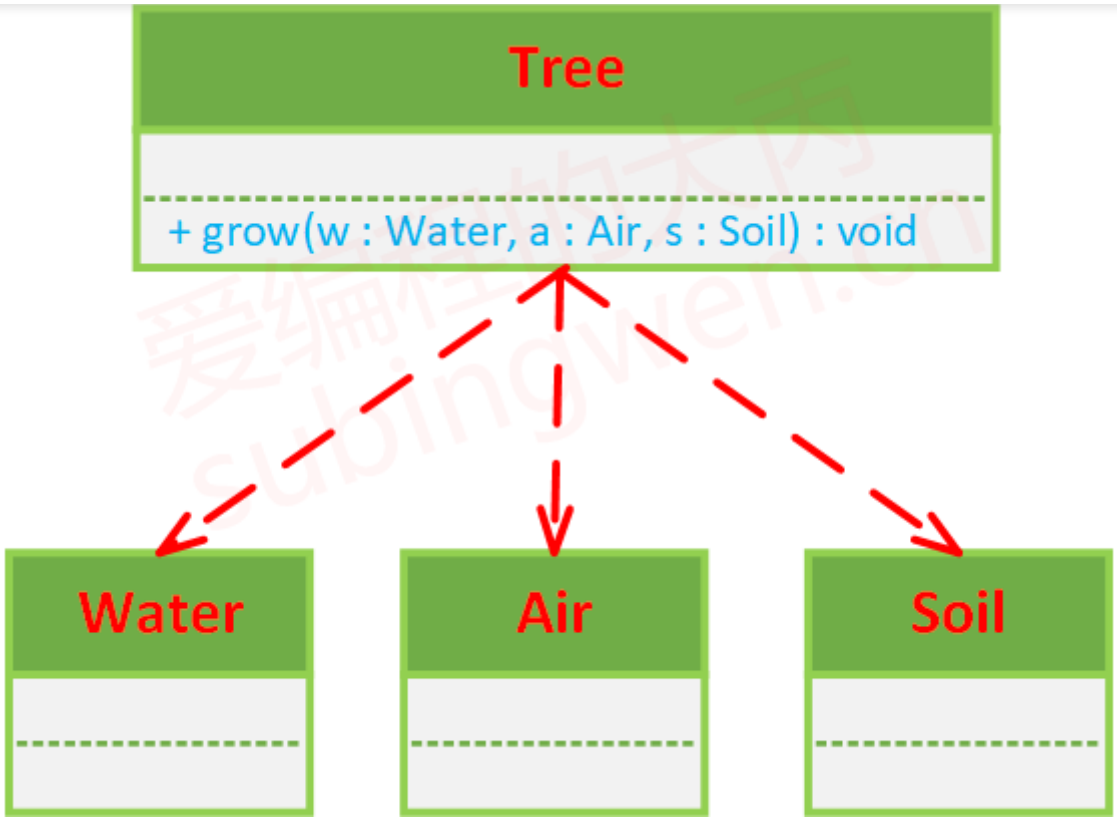
- 树木（Tree）的生长，需要将空气（Air）、水（Water）、土壤（Soil）对象作为参数传递给 Tree 类的 grow（）方法。

C++

```
class Water
{
```

```
5  class Air
6  {
7  };
8
9  class Soil
10 {
11 };
12
13 class Tree
14 {
15 public:
16     void grow(Water w, Air a, Soil s)
17     {
18         cout << "借助 w 中的水分, s 中的养分和 a 中的二氧化碳, 我就可以茁壮成长
19     }
20 };
```

关于树木这个类，它对应的 UML 类图为：



依赖关系通常通过三种方式来实现：

1. **将一个类的对象作为另一个类中方法的参数**
2. **在一个类的方法中将另一个类的对象作为其对象的局部变量**
3. **在一个类的方法中调用另一个类的静态方法**

> 类之间的关系强弱顺序是这样的：继承（泛化） > 组合 > 聚合 > 关联 > 依赖

关联关系、聚合关系、组合关系之间的区别

从上文可以看出，关联关系、聚合关系和组合关系三者之间比较相似，最后就来总结一下这三者之间的区别：

1. 关联和聚合的区别主要在于语义上：**关联的两个对象之间一般是平等的，聚合则一般是不平等的。**
2. 聚合和组合的区别则在语义和实现上都有差别：
   - 组合的两个对象之间生命周期有很大的关联，被组合的对象在组合对象创建的同时或者创建之后创建在组合对象销毁之前销毁，聚合则无需考虑这些事情。
   - 一般来说被组合对象不能脱离组合对象独立存在，而且也只能属于一个组合对象，聚合则不一样，被聚合的对象可以属于多个聚合对象。

最后，再举例子来描述一下这三种关系：

1. 朋友之间属于关联关系，因为这种关系是平等的，关联关系只是用于表示两个对象之间的一种简单的联系而已。
2. 图书馆看书的时候，人和书属于聚合关系。书是可以独立存在的，而且书不仅可以属于自己，也可以属于别人。
3. 人和自己的心脏属于组合关系，因为心脏不能脱离人体而独自存在。

不过，实际应用中，这三种关系的界限划分其实没有那么清楚，有些时候我们会感觉组合和聚合没

设计模式

打赏

上一篇 设计模式之设计模式三原则

下一篇 数据库连接池 基于 C++11 的数据库连接池

相关推荐

评论

昵称 邮箱 网址(http://)

来都来了，说点什么吧...

提交

4 评论

Anonymous Chrome 99.0.4844.88 Android 10

2023-03-28 回复

大佬，双向关联那里感觉应该是两个指针吧，如果是两个对象不就无限递归了

Anonymous Chrome 111.0.0.0 Windows 11

2023-03-27 回复

大丙老师太棒了，只恨发现太晚了系列

Anonymous Chrome 109.0.0.0 Windows 11

2023-02-17 回复

写的真好，先占个楼，以后会火的，就是不知什么时候出视频

Anonymous Edge 110.0.1587.41 Windows 11

2023-02-15 回复

双向关联关系里面，m_son 和 m_father 应该至少有一个是指针形式吧，否则应该会导致构造的时候发生栈溢出。