

原

**C++基础——用C++实例理解UML类图**

2015年11月01日 11:54:04 [Inside\_Zhang](https://me.csdn.net/lanchunhui) 阅读数：10208

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/lanchunhui/article/details/49557083

[类展示](https://blog.csdn.net/lanchunhui/article/details/49557083#%E7%B1%BB%E5%B1%95%E7%A4%BA)

[基本概念及术语](https://blog.csdn.net/lanchunhui/article/details/49557083#%E5%9F%BA%E6%9C%AC%E6%A6%82%E5%BF%B5%E5%8F%8A%E6%9C%AF%E8%AF%AD)

[可见性visibility](https://blog.csdn.net/lanchunhui/article/details/49557083#%E5%8F%AF%E8%A7%81%E6%80%A7visibility)

[参数的方向parameter direction](https://blog.csdn.net/lanchunhui/article/details/49557083#%E5%8F%82%E6%95%B0%E7%9A%84%E6%96%B9%E5%90%91parameter-direction)

[类成员变量或者函数的类型](https://blog.csdn.net/lanchunhui/article/details/49557083#%E7%B1%BB%E6%88%90%E5%91%98%E5%8F%98%E9%87%8F%E6%88%96%E8%80%85%E5%87%BD%E6%95%B0%E7%9A%84%E7%B1%BB%E5%9E%8B)

[类关系](https://blog.csdn.net/lanchunhui/article/details/49557083#%E7%B1%BB%E5%85%B3%E7%B3%BB)

[关系的多重性multiplicity](https://blog.csdn.net/lanchunhui/article/details/49557083#%E5%85%B3%E7%B3%BB%E7%9A%84%E5%A4%9A%E9%87%8D%E6%80%A7multiplicity)

[类关系](https://blog.csdn.net/lanchunhui/article/details/49557083#%E7%B1%BB%E5%85%B3%E7%B3%BB-1)

[assocationkonws a](https://blog.csdn.net/lanchunhui/article/details/49557083#assocationkonws-a)

[dependency](https://blog.csdn.net/lanchunhui/article/details/49557083#dependency)

[Aggregation](https://blog.csdn.net/lanchunhui/article/details/49557083#aggregation)

[Composition](https://blog.csdn.net/lanchunhui/article/details/49557083#composition)

[继承泛化](https://blog.csdn.net/lanchunhui/article/details/49557083#%E7%BB%A7%E6%89%BF%E6%B3%9B%E5%8C%96)

[类模板](https://blog.csdn.net/lanchunhui/article/details/49557083#%E7%B1%BB%E6%A8%A1%E6%9D%BF)

[References](https://blog.csdn.net/lanchunhui/article/details/49557083#references)

本文包括以下内容：

类间存在哪几种常见关系？

它们之间的区别和联系是什么？

如何在代码中反映类间的关系？

如何理解 **IN/OUT** mode型的参数？

**类展示**

class Circle

{

private:

double radius\_;

Point center\_;

public:

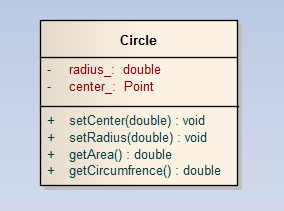
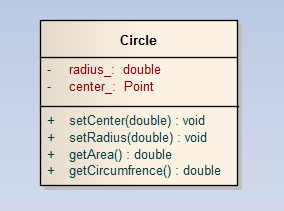
void setRadius(double \_radius);

void setCenter(Point \_center);

double getArea() const;

double getCircumfrence() const;

}



**基本概念及术语**

**可见性（visibility）**

‘+’ public

‘-’ private

‘#’ protected

**参数的方向（parameter direction）**

‘in’：用于输入的参数，get the value

‘out’：用于输出的参数， set the value

‘inout’：既可作为输入又可作为输出， get the value and set the value

这种参数形式常见于，这样一种情况，传递的是一个指针变量（如果传递的是一个既非指针，也非引用的变量，就纯做输入（in）了），既然是指针变量，便可提领指针，修改其指向的内容，既具备提供输入，又具有修改原值 的双重属性。

举例如下：

对一个接口作如下documentation：

Prototype: ULONG GetActivationState( ULONG \* pActivationState );

Parameters

Type: ULONG\*

Variable: pActivationState

Mode: IN/OUT

客户端代码

ULONG activationState = 1;

ULONG result = GetActivationState(&activationState);

既然提供的是变量的地址，便可轻易地get the value 和 set the value。 所以根据GetActivationState(ULONG\*) 参数的IN/OUT状态，推测其中的代码如下：

ULONG GetActivationState()

{

if (\*activationState == 1) *// 可读属性，in*

{

*// TODO*

*// 对变量值进行修改，例如，改为0*

\*activationState = 0; *// 可写属性，out*

}

return \*activationState;

}

**类成员（变量或者函数）的类型**

静态成员，下划线表示

纯虚函数，斜体

**类关系**

Assocation (knows a)

Dependency (uses a)

Composition (has a)

Aggregation (has a)

Inheritance (is a)

Class template

**关系的多重性（multiplicity）**

‘0..1’， 无实例，或者1个实例

‘1’，只有一个实例

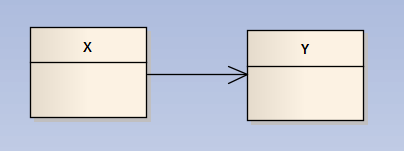
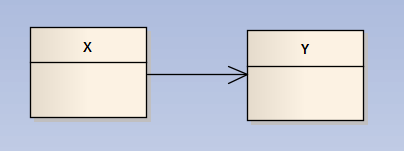
‘0..\*’，

‘1..\*’

**类关系**

**assocation（konws a）**

一个对象知道另一个对象的存在，该对象持有另一个对象的指针或者引用。



class X

{

public:

X(Y\* \_ptrY):ptrY\_(\_ptrY) {}

void setY(Y \*y) { ptrY\_ = y;}

void f() { ptrY\_->foo();}

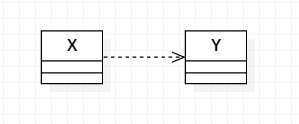
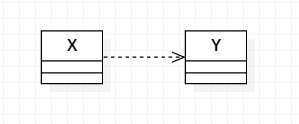
private/public:

Y\* ptrY\_; *// X类持有Y的一个指针，可据此调用Y中的成员方法*

}

**dependency**

当类Y与类X彼此依赖，而类Y是类X成员函数的一个参数，或者X中成员函数的一个局部变量。



class X

{

...

void f1(Y y) { ...; y.foo(); }

void f2(Y& y) { ...; y.foo(); }

void f3(Y\* y) { ...; y->foo(); }

void f4() { ...; Y y; y.foo(); }

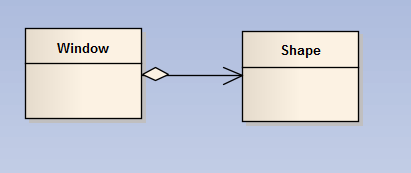
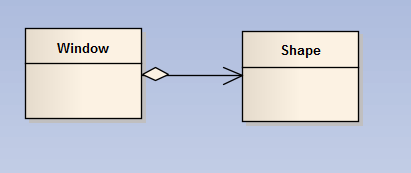
void f5() { Y::staticFoo();}

...

}

**Aggregation**

聚类发生的场景是：一个类是另一个类的集合或者容器，但被包含的类与容器本身并不具备相同的生命期，也就是容器销毁时，其包含的内容未必。关联（association）与聚合（aggregation）的不同之处在于，关联关系不存在一个明确的容器包含另外一个类。比如汽车和轮胎的关系，就是一种聚合关系，汽车包含着轮胎的集合，汽车的销毁，轮胎仍然可以存在。



class Window

{

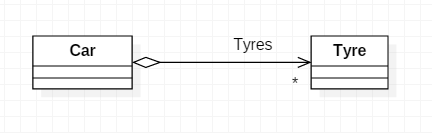
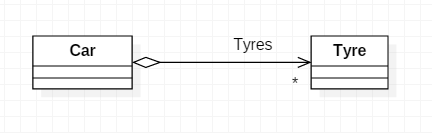
public:

//...

private:

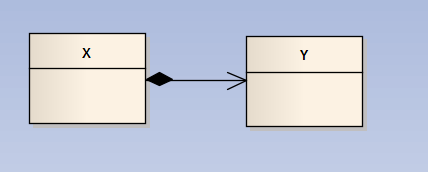
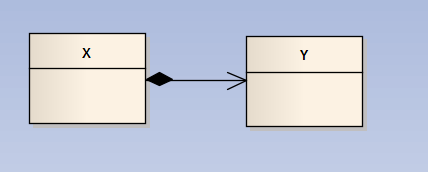
vector<Shape\*> ptrShapes;

}



**Composition**

组合（composition）是聚合（Aggregation）的增强版。组合和聚合的根本不同在于类与其所持有元素的是否具有相同的生命期，要求必须相同是一种组合关系。



class Circle

{

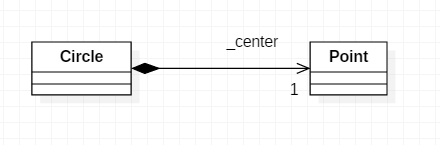
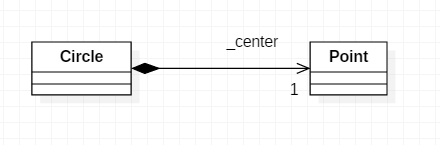
private:

...

Point center\_;

...

}



class X

{

...

Y y; // composition, 相同的生命期，

// 如果是引用或者指针类型，则是关联类型，是一种弱联系

Y b[10]; // 0...10, composition

}

class X

{

X() { a = new Y[10]; }

~X() { delete[] a;}

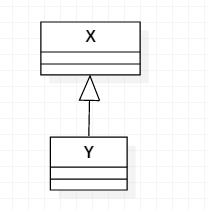
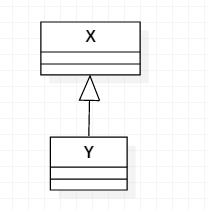
...

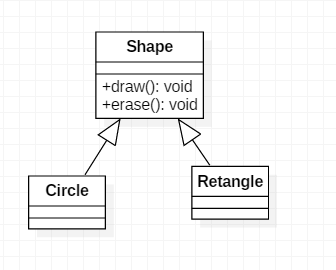
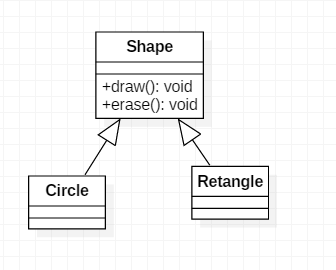
Y \*a; // 0...10, composition，X对象析构时，必须对a进行释放

}

**继承（泛化）**

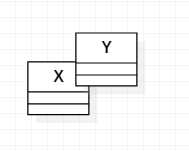
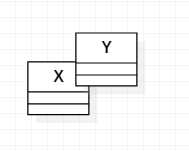
类间的继承关系表达的是一种**derived class B is a base class A**。





**类模板**

类模板意味着泛型类。



template<class T>

class X

{...};

X<Y> a;

**References**

[1] [UML Class Diagram Explained With C++ samples](https://cppcodetips.wordpress.com/2013/12/23/uml-class-diagram-explained-with-c-samples/)

[2] [IN/OUT Parameters and how to work with them in C++](http://stackoverflow.com/questions/6900035/in-out-parameters-and-how-to-work-with-them-in-c)