**Chapter 14 代码重用**

1. Has-a
2. Containment/composition/layering
3. 能用的东西更多，比较方便
4. 可以继承一个类的多个对象
5. 访问时用object name . function name
6. Private inheritance
7. 很多限制
8. 只能继承一个类（继承的是类定义）
9. 访问时用class name :: function name
10. 必须明确declare基类的友元函数，因为基类的友元函数的指针不能default地设成派生类的指针
11. 模板类valarray

Valarray<typename> objectname （eg: valarray<int> q\_values）

1. 保护继承
2. 如果用private inheritance，从派生类再派生后的类就不能用最开始的基类里的接口了，因为原始基类里的接口是private的，但是如果是protected inheritance，第二次派生后的类仍然可以使用原始基类里的接口/数据，因为它们一直都是protected状态
3. 用Using可以重新定义存取的方式
4. 比如valarray里的sum函数，如果派生类里要继续使用可以在派生类里定义一个sum函数，然后实现方式为在sum函数里直接return std :: valarray<double> :: sum(); //use privately-inherited method
5. Using std :: valarray<double> :: min;

Using std :: valarray<double> :: min;

1. 多重继承

有一个以上的基础类别，就叫multiple inheritance

避免出现环状，比如a :: b ::c , a :: d :: c , 这时候就要用到虚基类，将c设为虚基类

1. 虚基类
2. 为了多重继承后只有一个对象，才会设虚基类（+virtual≈唯一化）
3. Declaration: class Waiter :: virtual public Worker / class Waiter :: public virtual Worker
4. 创建类模板template class, STL（standard template library）
5. Eg. Valarray string vector Stack
6. Ps. 使用strlen函数之后一般要+1的原因是字符串以‘\0’结尾，但strlen不包含‘\0’
7. 模板定义：template <typename Type>
8. Explain more in Chap 16
9. Ways of specialization具体化的方法:
10. Implicit specialization
11. Explicit specialization
12. Partial specialization
13. 模板的具体化
14. 模板的友元函数
15. Non-template friends

没有模板类的类型，就是没有尖括号<>

1. Bound template friends

当类别具体化时，friend的类型取决于这个class的type，比如<TT>

1. Unbounded template friends

无论模板类具体化后是什么type，Friend的类型都不改变，比如<typename T>

|  |  |  |  |
| --- | --- | --- | --- |
| All kinds of Inheritance | | | |
| 性质 | Public | Protected | Private |
| Public members change into | Public in derived class | Protected in derived class | Private in derived class |
| Protected change into | Protected in derived class | Protected in derived class | Private in derived class |
| Private members change into | Can only be done by the functions in base class | Can only be done by the functions in base class | Can only be done by the functions in base class |
| 隐式向上转型 | yes | Yes（but in derived class only） | no |

《Inside the C++ Object》