命名空间

命名空间是C++中提供的一种避免*名字*冲突的一种方法,其中基本语法如下:

`namespace 命名空间名 {声明序列}`

```
int x;
int x;
```

该代码会报出"Redefinition of 'x'"，*重定义*x，但是有了命名空间以后可以区别出x

```
namespace A {
    int x = 1;
}

namespace B {
    int x = 2;
}

int x = 3;
```

其中通过：：来限定所查找的名字的范围

```
    std::cout << A::x << "\n";// 输出 1
    std::cout << B::x << "\n";// 输出 2
    std::cout << ::x;// 输出 3，如果：：前没有名字的话代表在全局查找。
```

向命名空间添加成员形式如下

```
#include <iostream>

namespace A {
    int x = 1;
}

namespace A {
    int y = 2, z = 3;
}

namespace A {
    void foo() {
        std::cout << x << " " << y << " " << z << std::endl;
    }
}

int main() {
    A::foo();// 输出 1 2 3
}
```

通过：：标识名字所在类（结构体）

假设定义了如下的结构体

```
struct A {
    using INT = int;//等价typedef int INT,给int取了一个别名
};
```

但是我在后续代码中定义`INT a = 1;`时编译器会给出Unknown type name 'INT'; 提示未能找到INT，因为INT所在的域是A，在没有：：限定时编译器会在全局去查找，但是在全局中并没有INT的声明，所以会编译失败。但是如果改成`A::INT a=1;`此时代码会通过。这也是为什么当你写`std::vector`的迭代器时需要写成`std::vector<typename>::iterator`这是因为`iterator`其实是在`std::vector<typename>`中的一个别名。

现在解析如下代码

```
#include <iostream>

namespace A {

    struct B {
        constexpr static int x = 1;// 在B中定义一个静态变量 x

        static void foo() { //在B中定义一个静态函数 foo
            std::cout << "B::foo()\n";
        }
    };
}

namespace B {
    static ::A::B a;// 在命名空间B中定义一个静态变量 a，其类型声明在::A::B,也就是现在全局中找到A然后再从A命名空间找到B类型
}

int main() {
    ::B::a.foo();//在全局中找到命名空间B，然后再从B中找到静态变量a，调用a的成员函数foo
    std::cout << A::B::x << " \n";//输出全局空间下的A命名空间下的B结构体的x静态成员，全局限定符可不写
}
```