参考许继写的sgcc的命名空间

一个命名空间是一个作用域，其作用是把全局命名空间划分开来，以避免名字冲突。在不同的命名空间中，可以有同名成员。

namespace sgcc // 名称为sgcc的命名空间

{

}

命名空间中可以出现在全局作用域的任意声明：类、变量（以及他们的初始化），函数（以及他们的定义），模板以及其它命名空间。

命名空间可以是不连续的，也就是说可以在不同的文件中定义同一个命名空间。这样可以像类那样，将命名空间函数和成员的声明放在头文件中，将定义放在单独的源文件中。如下例所示：

例：程序class\_test36

sgcc.h

#ifndef SGCC\_H

#define SGCC\_H

namespace sgcc

{

extern int n; // 命名空间和全局命名空间类似，不能在头文件中定义变量

class A

{

public:

A();

~A();

public:

A& Set();

};

}

#endif // SGCC\_H

sgcc.cpp

#include "sgcc.h"

#include <iostream>

// 定义在命名空间内部，命名空间可以不连续

namespace sgcc

{

int n = 0;

A::A()

{

std::cout << "A()" << std::endl;

}

A::~A()

{

std::cout << "~A()" << std::endl;

}

A& A::Set()

{

std::cout << "Set()" << std::endl;

return \*this;

}

}

// 也可以像下面这样定义命名空间，定义在命名空间外部

//int sgcc::n = 0;

//

//sgcc::A::A()

//{

// std::cout << "A()" << std::endl;

//}

//

//sgcc::A::~A()

//{

// std::cout << "~A()" << std::endl;

//}

//

//sgcc::A& sgcc::A::Set()

//{

// std::cout << "Set()" << std::endl;

// return \*this;

//}

#include <iostream>

#include "sgcc.h"

int main(int argc, char\* argv[])

{

sgcc::A a;

a.Set();

return 0;

}

不能在不相关的命名空间中定义成员。

嵌套命名空间：外围命名空间中声明的名字被嵌套命名空间中同一名字的声明所屏蔽。自定义的命名空间与全局命名空间也构成了嵌套。

例：程序class\_test37

// 嵌套命名空间

#include <iostream>

namespace xxcig

{

int n = 0;

namespace sgcc

{

int n = 1; // 屏蔽xxcig中的n

}

}

int main(int argc, char\* argv[])

{

std::cout << xxcig::sgcc::n << std::endl;

return 0;

}

命名空间成员的使用：

using 声明：

using std::cout;

using指示：

using namespace std;

可以看出：using声明可以控制命名空间某个成员可见，而using指示则无法控制，其会让整个命名空间的所有成员都可见，所以一般情况下慎用using指示。

命名空间别名

例：程序class\_test37

// 嵌套命名空间

#include <iostream>

using std::cout; // using 声明

using std::endl;

namespace xxcig

{

int n = 0;

namespace sgcc

{

int n = 1; // 屏蔽xxcig中的n

}

}

int main(int argc, char\* argv[])

{

cout << xxcig::sgcc::n << endl;

namespace X = xxcig::sgcc; // 命名空间别名

cout << X::n << endl;

return 0;

}

类、命名空间和作用域

接受类类型形参（或类类型指针及引用形参）的函数（包括重载操作符），在用类类型对象作为实参时是可见的，见下例：

例：程序class\_test38

#include <iostream>

using std::cout;

using std::endl;

namespace sgcc

{

class A

{

public:

A()

{

}

};

void f(const A& a) // 与类A在同一个命名空间

{

}

}

int main(int argc, char\* argv[])

{

sgcc::A a;

f(a); // ok，不用显示使用sgcc::f

return 0;

}

重载与命名空间

命名空间对函数重载中函数匹配的影响：

1. using声明或using指示可以将函数加到候选集合。
2. 上文中提到的有一个或多个类类型形参的函数，当查找候选函数而查找定义形参类（以及定义其基类）的每个命名空间，将那些命名空间中任意与被调用（查找）函数名字相同的函数加入候选集合。

针对影响2，参考下例：

例：程序class\_test41

// 重载对命名空间的影响

#include <iostream>

using std::cout;

using std::endl;

namespace sgcc

{

class Base

{

public:

Base()

{

}

};

void Display(const Base& b)

{

cout << "sgcc::Display" << endl;

}

void main()

{

cout << "sgcc::main()" << endl;

}

}

class Derive : public sgcc::Base

{

public:

Derive()

{

}

};

void Display(const Derive& d)

{

cout << "Display(const Derive& d)" << endl;

}

void Display(const sgcc::Base& b)

{

cout << "Display(const sgcc::Base& b)" << endl;

}

int main(int argc, char\* argv[])

{

Derive d;

Display(d); // 调用Display(const Derive& d)

sgcc::Base b;

// Display(b); // error，Display调用不明确，原因是全局作用域中有个与命名空间sgcc完全相同的函数

sgcc::main(); // 居然可以在sgcc中定义一个main函数

return 0;

}

可以看出：由于Display函数形参为一个类类型的引用，所以在全局作用域中是可见，如果在全局作用域中定义了一个与sgcc中完全相同的Display函数，则会出现编译错误。

另外可以看出，可以在一个命名空间定义一个main函数，但这个main与其它普通的函数没有任何区别。

重载与using声明：

using声明声明一个名字，但无法加上形参。using声明的函数，如果在命名空间中时重载的，则重载的所有函数都是可见的。

例：程序class\_test39

// 重载与using声明

#include <iostream>

using std::cout;

using std::endl;

namespace sgcc

{

void Print(int n)

{

cout << "n = " << n << endl;

}

void Print(double f)

{

cout << "f = " << f << endl;

}

}

// using sgcc::Print(double f); // error，using声明无法带上形参

using sgcc::Print;

void Print(char\* str)

{

cout << "str = " << str << endl;

}

//void Print(int n) // error，函数void sgcc::Print(int)已有主体

//{

// cout << "n = " << n << endl;

//}

int main(int argc, char\* argv[])

{

int n = 5;

Print(n); // 调用sgcc::Print(int n)

double f = 3.5;

Print(f); // 调用sgcc::Print(double f)

char\* str = "abcde";

Print(str); // 调用Print(char\* str)

return 0;

}

可以看出：使用了using sgcc::Print后，命名空间sgcc中的Print函数会加入到重载函数的候选集中，如果在全局作用域中，增加一个与sgcc中的Print函数完全相同函数，会出现编译错误。

例：程序class\_test40

重载与using指示：

// 重载与using指示

#include <iostream>

using std::cout;

using std::endl;

namespace sgcc

{

void Print(int n)

{

cout << "sgcc::Print(int n)" << endl;

}

void Print(double f)

{

cout << "sgcc::Print(double f)" << endl;

}

}

namespace xxcig

{

void Print(int n)

{

cout << "xxcig::Print(int n)" << endl;

}

}

using namespace sgcc;

using namespace xxcig;

int main(int argc, char\* argv[])

{

int n = 5;

// Print(n); // error，对重载函数的调用不明确

return 0;

}

可以看出：使用using指示多个命名空间后，这些命名空间的同名函数会构成函数重载的候选集合，如果有完全相同的函数，在调用时，会出现编译错误：调用不明确。