名字空间 (1)

- 可以在不同的模块中定义同名的名字空间。编译器会将不同模块中定义的同名的名字空间整合为一个空间。
- 每个模块中名字空间内的符号，作用范围是当前模块。
- 引用其他模块中同名的名字空间内的符号时，需要在本模块的名字空间中用extern声明。
- 引用名字空间的符号时，只能引用本模块的名字空间中声明/定义的符号。

名字空间 (2)


```
//1.CPP

#include <iostream>
using namespace std;

namespace A {
    int  x = -1;
    int  f( ) { return 1; }
}

using namespace A;
//访问不到2.cpp中A的符号

int main( )
{
    cout << h( );          //error
    cout << y + A::y;  //都访问不到
}
```

```
//2.CPP

#include <iostream>

namespace A {
    int  x;  //error
    int  y = 2;
    int  g( ) { return -2; }
    //extern int f();
}

using namespace A;
//访问不到1.cpp中A的符号

using A::f;
//error, 只能引用本模块A中的符号

int h( )  { return f( ); }
```

名字空间(3)

- 名字空间(包括匿名名字空间)可以分多次定义。

```
//1.CPP
namespace A {
        int x = -1;
        int f( ) { return 1; }
}
using namespace A;
namespace A {
    int y = -1;  //error, 重复定义
    int g( ) { return 2; }  //error, 重复定义
    int h( ) { return 0; }
}
int main( ) {
        f( );   h( );
        g( );  //error, 访问不到2.CPP的g()
        cout << y;  //error, 访问不到
}
```

```
//2.CPP
namespace A {
        int y = 2;
        extern int f( );
}
using namespace A;
namespace A {
        int g( ) { return -2; }
}
int f( ) { return 1; }
void m( ) {
        h( );    //error,访问不到
        f( );    //error,全局f()? A::f()?
        ::f( );  //全局f()优先
        A::f( );
}
```

名字空间(4)

(1) using namespace 名字空间名称;

本模块可以访问这个名字空间中的所有符号(包括在这条语句后面对该名字空间定义的符号)。可以在当前模块中再定义和名字空间中同名的全局标识符。

(2) using 名字空间名称::成员名称;

- 只能在本模块中访问这个成员。编译器将该成员的定义加入当前模块，不能在当前模块中再定义和该成员同名的标识符。
- 引用的成员必须在用using引用前已经声明(即使还没有定义也可以)。

名字空间 (5)


```
//1.CPP
namespace A {
        int y = 2;
        int g( ) { return -2; }
        extern int f( );
}
using namespace A;
int x = 11;     int y = 22;
int f( ) { return 1; }
int g( ) { return 2; }
void m( ) {
    h( );     //error, 访问不到
    f( );     //error, 全局f()? A::f()?
    g( );     //error, 全局g()? A::g()?
    ::f();   //全局f()优先
    ::g();  //全局g()优先
    A::f( );    ::y++;    A::y++;
}
```

```
//2.CPP
namespace A {
        int x = -1;
        //int y = -2; //error,重复定义
        int f( ) { return 1; }
        int h( ) { return 0; }
}

using A::h;
using A::x;
float x = 0;   //error, 重定义

namespace A {
    int h(int x) { return x; }
}

int main( ) {
    h(1);   //error, 没有定义
}
```

将 using A::h 改为 using namespace A ?

名字空间 (6)

匿名名字空间被自动引用，编译器不将其成员的定义加入当前模块。可以在当前模块中重新定义同名的标识符。不同模块定义的匿名名字空间不合并。

```
//1.CPP
namespace {
   //不和2.CPP的匿名空间合并
   void  f( ) { }
}
namespace A {  //将和2.CPP合并
   void g( ) { }
}
void m( ) {
     f( );   // f( ) of 1.CPP
     A::g( );
}
```

```
//2.CPP
namespace {  //不与1.CPP合并
    int  x = 0;
    void  f( ) { }
}
namespace A {   //与1.cpp合并
    void  g( ) { }   //error, 重定义
    void  h( ) { }
}
int  x = 1;
void main() {
   cout << x;     //error, which x ?
   cout << ::x;  //匿名x永远不能访问
   f( );   // f( ) of 2.CPP
}
```

名字空间(7)

The end.