局部与全局using，具体操作与使用见下面案例：

```
#include <iostream>
#define isNs1 1
//#define isGlobal 2
using namespace std;
void func()
{
    cout<<"::func"<<endl;
}

namespace ns1 {
    void func()
    {
        cout<<"ns1::func"<<endl;
    }
}

namespace ns2 {
#ifdef isNs1
    using ns1::func;    /// ns1中的函数
#elif isGlobal
    using ::func; /// 全局中的函数
#else
    void func()
    {
        cout<<"other::func"<<endl;
    }
#endif
}

int main()
{
    /**
     * 这就是为什么在c++中使用了cmath而不是math.h头文件
     */
    ns2::func(); // 会根据当前环境定义宏的不同来调用不同命名空间下的func()函数
    return 0;
}
```

改变访问性

(非常实用)

```
class Base{
public:
 std::size_t size() const { return n;  }
protected:
 std::size_t n;
};
class Derived : private Base {
```

```
public:
 using Base::size;
protected:
 using Base::n;
};
```

类Derived私有继承了Base，对于它来说成员变量n和成员函数size都是私有的，如果使用了using语句，可以改变他们的可访问性，如上述例子中，size可以按public的权限访问，n可以按protected的权限访问。

函数重载

> 在继承过程中，派生类可以覆盖重载函数的0个或多个实例，一旦定义了一个重载版本，那么其他的重载版本都会变为不可见。

> 如果对于基类的重载函数，我们需要在派生类中修改一个，又要让其他的保持可见，必须要重载所有版本，这样十分的繁琐。

```
#include <iostream>
using namespace std;

class Base{
    public:
        void f(){ cout<<"f()"<<endl;
        }
        void f(int n){
            cout<<"Base::f(int)"<<endl;
        }
};

class Derived : private Base {
    public:
        using Base::f;
        void f(int n){
            cout<<"Derived::f(int)"<<endl;
        }
};

int main()
{
    Base b;
    Derived d;
    d.f();
    d.f(1);
    return 0;
}
```

> 如上代码中，在派生类中使用using声明语句指定一个名字而不指定形参列表，所以一条基类成员函数的using声明语句就可以把该函数的所有重载实例添加到派生类的作用域中。此时，派生类只需要定义其特有的函数就行了，而无需为继承而来的其他函数重新定义。

取代typedef

> C中常用typedef A B这样的语法，将B定义为A类型，也就是给A类型一个别名B

> 对应typedef A B，使用using B=A可以进行同样的操作。

```
typedef vector<int> V1;
using V2 = vector<int>;
```

//摘自C++那些事。