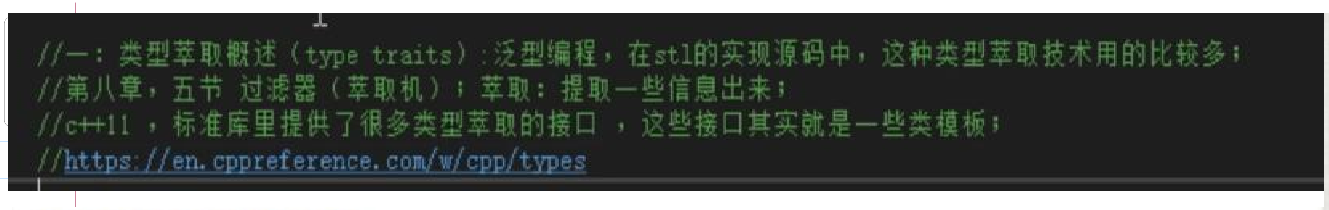
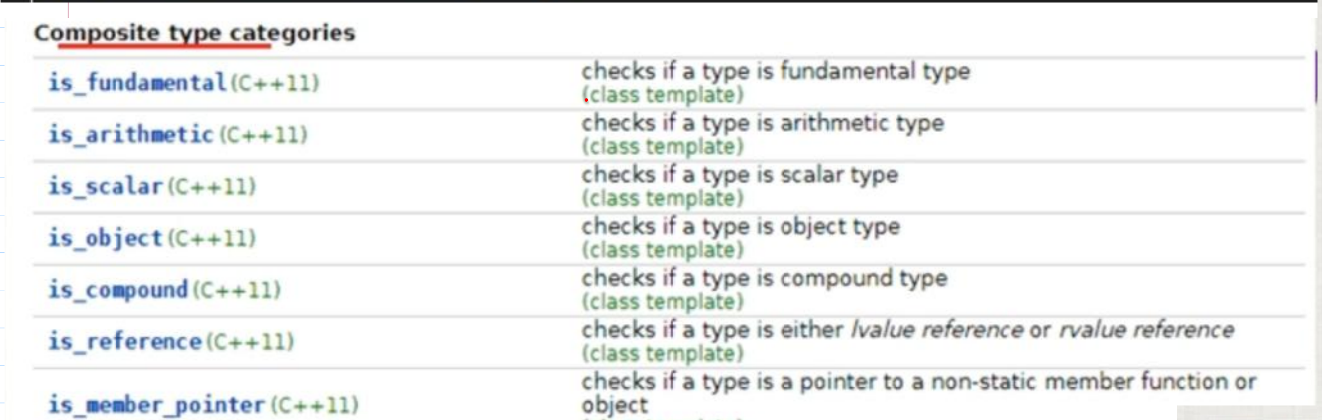
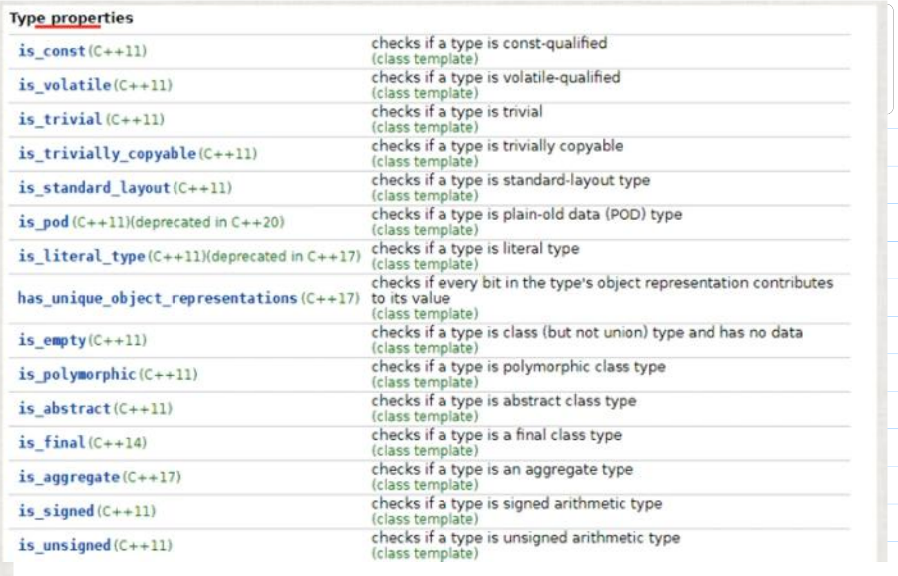


```
//一：类型萃取概述（type traits）：泛型编程，在stl的实现源码中，这种类型萃取技术用的比较多；
//第八章，五节 过滤器（萃取机）；萃取：提取一些信息出来；
//c++11 ，标准库里提供了很多类型萃取的接口 ，这些接口其实就是一些类模板；
//https://en.cppreference.com/w/cpp/types
```



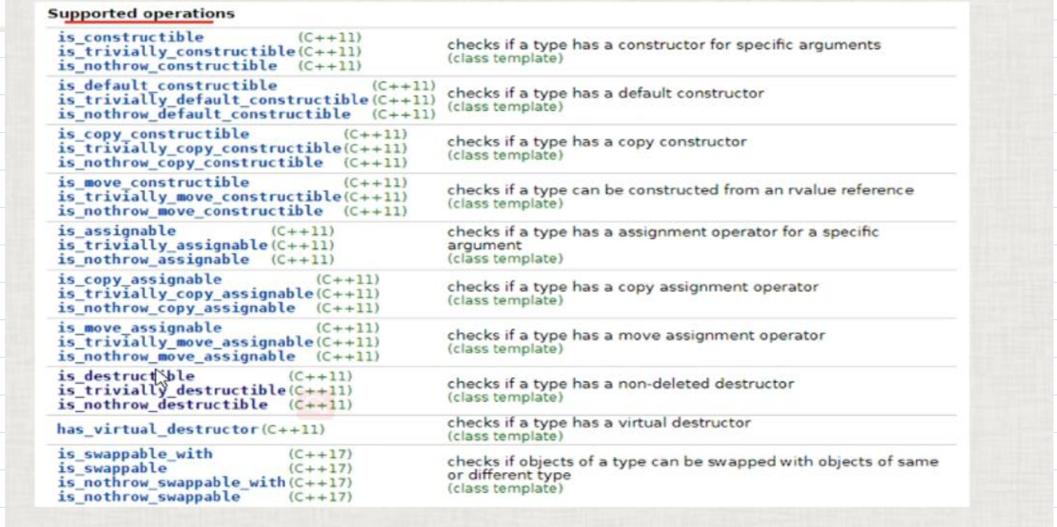
Composite type categories

is_fundamental (C++11)	checks if a type is fundamental type (class template)
is_arithmetic (C++11)	checks if a type is arithmetic type (class template)
is_scalar (C++11)	checks if a type is scalar type (class template)
is_object (C++11)	checks if a type is object type (class template)
is_compound (C++11)	checks if a type is compound type (class template)
is_reference (C++11)	checks if a type is either *lvalue reference* or *rvalue reference* (class template)
is_member_pointer (C++11)	checks if a type is a pointer to a non-static member function or object (class template)



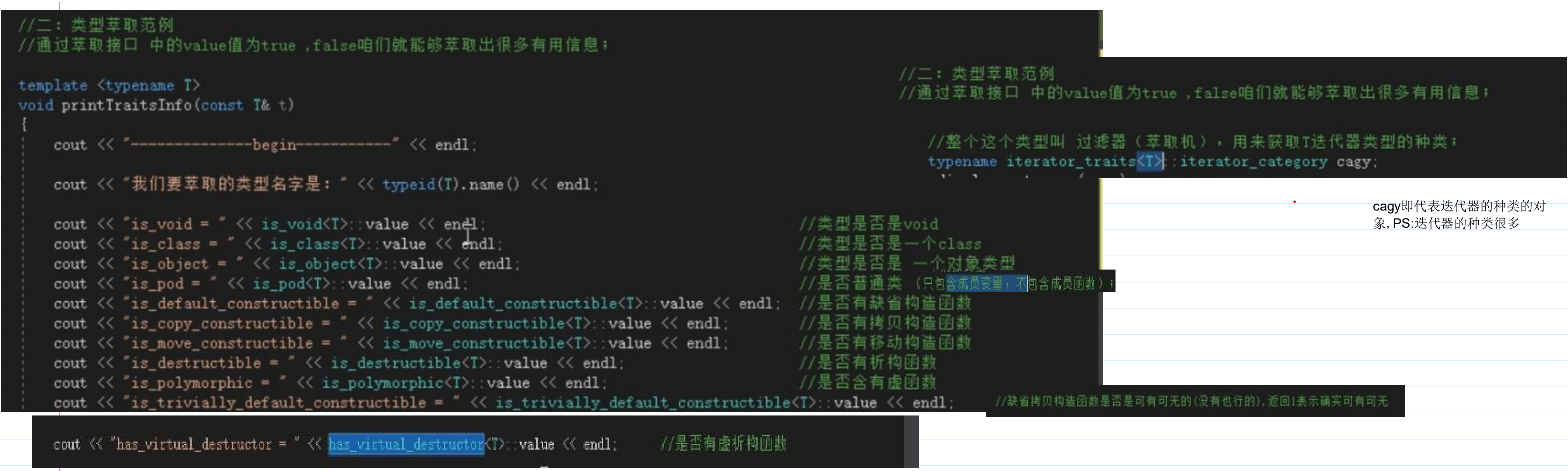
Type properties

is_const (C++11)	checks if a type is const-qualified (class template)
is_volatile (C++11)	checks if a type is volatile-qualified (class template)
is_trivial (C++11)	checks if a type is trivial (class template)
is_trivially_copyable (C++11)	checks if a type is trivially copyable (class template)
is_standard_layout (C++11)	checks if a type is standard-layout type (class template)
is_pod (C++11)(deprecated in C++20)	checks if a type is plain-old data (POD) type (class template)
is_literal_type (C++11)(deprecated in C++17)	checks if a type is literal type (class template)
has_unique_object_representations (C++17)	checks if every bit in the type's object representation contributes to its value (class template)
is_empty (C++11)	checks if a type is class (but not union) type and has no data (class template)
is_polymorphic (C++11)	checks if a type is polymorphic class type (class template)
is_abstract (C++11)	checks if a type is abstract class type (class template)
is_final (C++14)	checks if a type is a final class type (class template)
is_aggregate (C++17)	checks if a type is an aggregate type (class template)
is_signed (C++11)	checks if a type is signed arithmetic type (class template)
is_unsigned (C++11)	checks if a type is unsigned arithmetic type (class template)



Supported operations

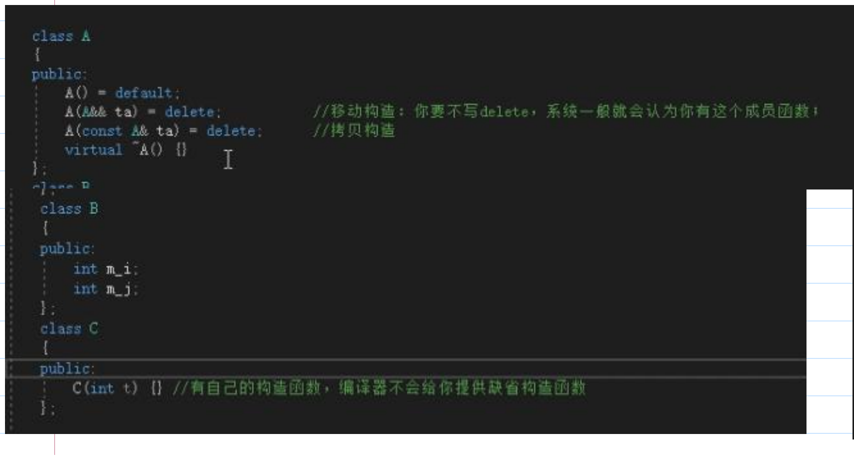
is_constructible (C++11) is_trivially_constructible (C++11) is_nothrow_constructible (C++11)	checks if a type has a constructor for specific arguments (class template)
is_default_constructible (C++11) is_trivially_default_constructible (C++11) is_nothrow_default_constructible (C++11)	checks if a type has a default constructor (class template)
is_copy_constructible (C++11) is_trivially_copy_constructible (C++11) is_nothrow_copy_constructible (C++11)	checks if a type has a copy constructor (class template)
is_move_constructible (C++11) is_trivially_move_constructible (C++11) is_nothrow_move_constructible (C++11)	checks if a type can be constructed from an rvalue reference (class template)
is_assignable (C++11) is_trivially_assignable (C++11) is_nothrow_assignable (C++11)	checks if a type has a assignment operator for a specific argument (class template)
is_copy_assignable (C++11) is_trivially_copy_assignable (C++11) is_nothrow_copy_assignable (C++11)	checks if a type has a copy assignment operator (class template)
is_move_assignable (C++11) is_trivially_move_assignable (C++11) is_nothrow_move_assignable (C++11)	checks if a type has a move assignment operator (class template)
is_destructible (C++11) is_trivially_destructible (C++11) is_nothrow_destructible (C++11)	checks if a type has a non-deleted destructor (class template)
has_virtual_destructor (C++11)	checks if a type has a virtual destructor (class template)
is_swappable_with (C++17) is_swappable (C++17) is_nothrow_swappable_with (C++17) is_nothrow_swappable (C++17)	checks if objects of a type can be swapped with objects of same or different type (class template)



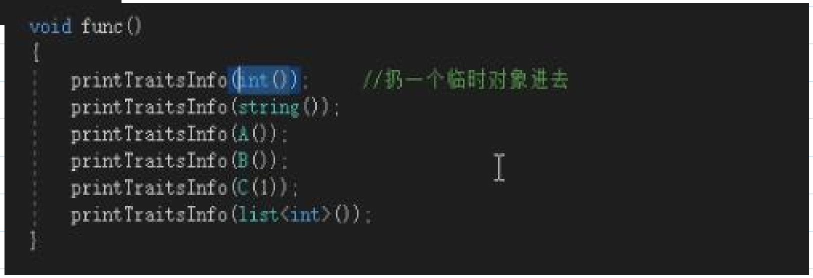
```
//二：类型萃取范例
//通过萃取接口 中的value值为true ,false咱们就能够萃取出很多有用信息；

template <typename T>
void printTraitsInfo(const T& t)
{
    cout << "--------------begin-----------" << endl;

    cout << "我们要萃取的类型名字是：" << typeid(T).name() << endl;

    cout << "is_void = " << is_void<T>::value << endl;                          //类型是否是void
    cout << "is_class = " << is_class<T>::value << endl;                        //类型是否是一个class
    cout << "is_object = " << is_object<T>::value << endl;                      //类型是否是 一个对象类型
    cout << "is_pod = " << is_pod<T>::value << endl;                            //是否普通类 （只包含成员变量，不包含成员函数）；
    cout << "is_default_constructible = " << is_default_constructible<T>::value << endl;  //是否有缺省构造函数
    cout << "is_copy_constructible = " << is_copy_constructible<T>::value << endl;        //是否有拷贝构造函数
    cout << "is_move_constructible = " << is_move_constructible<T>::value << endl;        //是否有移动构造函数
    cout << "is_destructible = " << is_destructible<T>::value << endl;          //是否有析构函数
    cout << "is_polymorphic = " << is_polymorphic<T>::value << endl;            //是否含有虚函数
    cout << "is_trivially_default_constructible = " << is_trivially_default_constructible<T>::value << endl;    //缺省拷贝构造函数是否是可有可无的(没有也行的),返回1表示确实可有可无

    cout << "has_virtual_destructor = " << has_virtual_destructor<T>::value << endl;      //是否有虚析构函数
```

```
//二：类型萃取范例
//通过萃取接口 中的value值为true ,false咱们就能够萃取出很多有用信息；

    //整个这个类型叫 过滤器（萃取机），用来获取T迭代器类型的种类；
    typename iterator_traits<T>::iterator_category cagy;
```

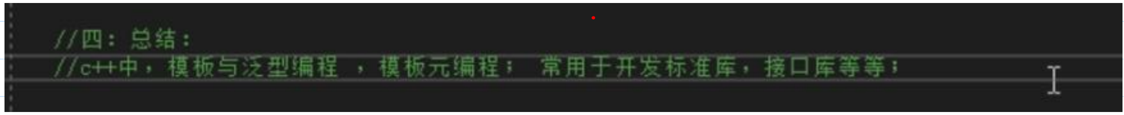
cagy即代表迭代器的种类的对象, PS:迭代器的种类很多



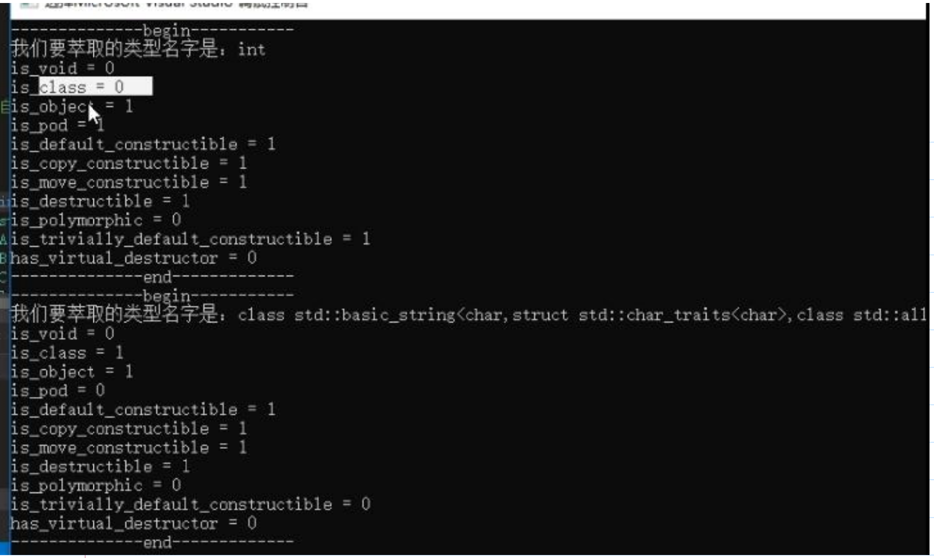
```
class A
{
public:
    A() = default;
    A(A&& ta) = delete;          //移动构造：你要不写delete，系统一般就会认为你有这个成员函数；
    A(const A& ta) = delete;     //拷贝构造
    virtual ~A() {}
};
class B
{
public:
    int m_i;
    int m_j;
};
class C
{
public:
    C(int t) {} //有自己的构造函数，编译器不会给你提供缺省构造函数
};
```



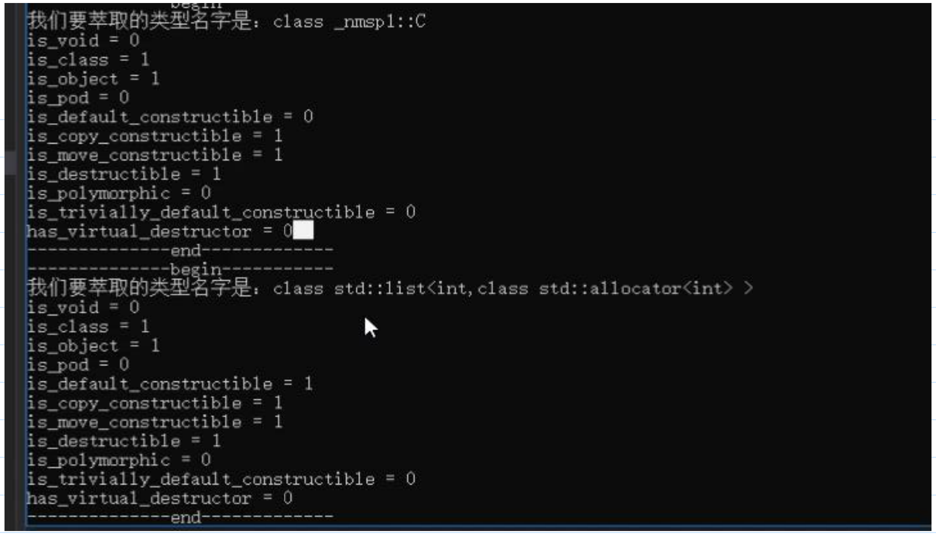
```
void func()
{
    printTraitsInfo(int());     //扔一个临时对象进去
    printTraitsInfo(string());
    printTraitsInfo(A());
    printTraitsInfo(B());
    printTraitsInfo(C(1));
    printTraitsInfo(list<int>());
}
```



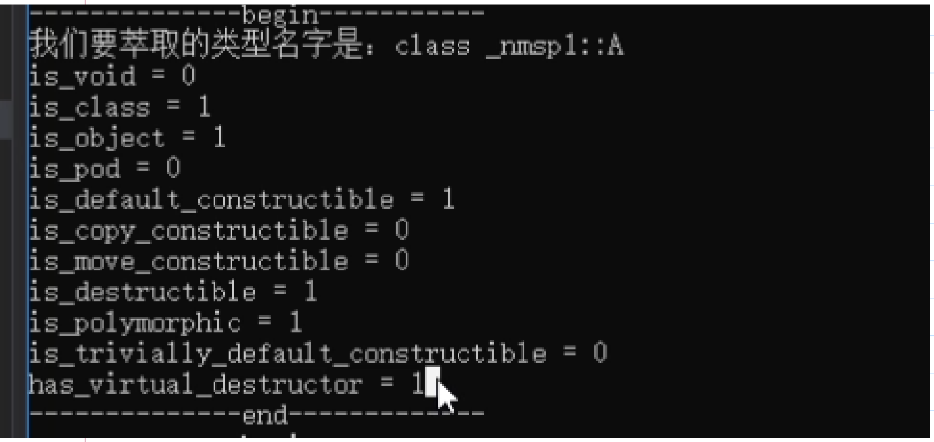
```
    //四：总结：
    //c++中，模板与泛型编程 ，模板元编程； 常用于开发标准库，接口库等等；
```



```
--------------begin-----------
我们要萃取的类型名字是：int
is_void = 0
is_class = 0
is_object = 1
is_pod = 1
is_default_constructible = 1
is_copy_constructible = 1
is_move_constructible = 1
is_destructible = 1
is_polymorphic = 0
is_trivially_default_constructible = 1
has_virtual_destructor = 0
--------------end-------------
--------------begin-----------
我们要萃取的类型名字是：class std::basic_string<char,struct std::char_traits<char>,class std::all
is_void = 0
is_class = 1
is_object = 1
is_pod = 0
is_default_constructible = 1
is_copy_constructible = 1
is_move_constructible = 1
is_destructible = 1
is_polymorphic = 0
is_trivially_default_constructible = 0
has_virtual_destructor = 0
--------------end-------------
```



```
我们要萃取的类型名字是：class _nmsp1::C
is_void = 0
is_class = 1
is_object = 1
is_pod = 0
is_default_constructible = 0
is_copy_constructible = 1
is_move_constructible = 1
is_destructible = 1
is_polymorphic = 0
is_trivially_default_constructible = 0
has_virtual_destructor = 0
--------------end-------------
--------------begin-----------
我们要萃取的类型名字是：class std::list<int,class std::allocator<int> >
is_void = 0
is_class = 1
is_object = 1
is_pod = 0
is_default_constructible = 1
is_copy_constructible = 1
is_move_constructible = 1
is_destructible = 1
is_polymorphic = 0
is_trivially_default_constructible = 0
has_virtual_destructor = 0
--------------end-------------
```



```
--------------begin-----------
我们要萃取的类型名字是：class _nmsp1::A
is_void = 0
is_class = 1
is_object = 1
is_pod = 0
is_default_constructible = 1
is_copy_constructible = 0
is_move_constructible = 0
is_destructible = 1
is_polymorphic = 1
is_trivially_default_constructible = 0
has_virtual_destructor = 1
--------------end-------------
```