1.基本使用

decltype的语法是:

decltype (expression)

这里的括号是必不可少的,decltype的作用是“查询表达式的类型”，因此，上面语句的效果是，返回 expression 表达式的类型。注意，decltype 仅仅“查询”表达式的类型，并不会对表达式进行“求值”。

1.1 推导出表达式类型

int i = 4; decltype(i) a; //推导结果为int。a的类型为int。

1.2 与using/typedef合用，用于定义类型。

```
using size_t = decltype(sizeof(0));//sizeof(a)的返回值为size_t类型
using ptrdiff_t = decltype((int*)0 - (int*)0);
using nullptr_t = decltype(nullptr);
vector<int >vec;
typedef decltype(vec.begin()) vectype;
for (vectype i = vec.begin; i != vec.end(); i++)
{
//...
}
```

这样和auto一样，也提高了代码的可读性。

1.3重用匿名类型

在C++中，我们有时候会遇上一些 **匿名类型** ， 如: 而借助decltype，我们可以重新使用这个匿名的结构体:

```
struct
{
    int d ;
    doubel b;
}anon_s;
```

decltype(anon_s) as ;//定义了一个上面匿名的结构体

1.4 泛型编程中结合auto，用于追踪函数的返回值类型

这也是decltype最大的用途了。

```
template <typename T>
auto multiply(T x, T y)->decltype(x*y)
```

```
{
    return x*y;
}
```

2.判别规则

对于decltype(e)而言，其判别结果受以下条件的影响：

> 如果e是一个没有带括号的标记符表达式或者类成员访问表达式，那么的decltype（e）就是e所命名的实体的类型。此外，如果e是一个被重载的函数，则会导致编译错误。 否则，假设e的类型是T，如果e是一个将亡值，那么decltype（e）为T&& 否则，假设e的类型是T，如果e是一个左值，那么decltype（e）为T&。 否则，假设e的类型是T，则decltype（e）为T。

标记符指的是除去关键字、字面量等编译器需要使用的标记之外的程序员自己定义的标记，而单个标记符对应的表达式即为标记符表达式。例如：

```
int arr[4]
```

则arr为一个标记符表达式，而arr[3]+0不是。

举例如下:

```
int i = 4;
int arr[5] = { 0 };
int *ptr = arr;
struct S{ double d; }s ;
void Overloaded(int);
void Overloaded(char);//重载的函数
int && RvalRef();
const bool Func(int);

//规则一：推导为其类型
decltype (arr) var1; //int 标记符表达式

decltype (ptr) var2;//int *  标记符表达式

decltype(s.d) var3;//doubel 成员访问表达式

//decltype(Overloaded) var4;//重载函数。编译错误。

//规则二：将亡值。推导为类型的右值引用。

decltype (RvalRef()) var5 = 1;

//规则三：左值，推导为类型的引用。

decltype ((i))var6 = i;     //int&
```

```
decltype (true ? i : i) var7 = i; //int&  条件表达式返回左值。

decltype (++i) var8 = i; //int&  ++i返回i的左值。

decltype(arr[5]) var9 = i;//int&. []操作返回左值

decltype(*ptr)var10 = i;//int& *操作返回左值

decltype("hello")var11 = "hello"; //const char(&)[9]  字符串字面常量为左值，且为const左值。


//规则四：以上都不是，则推导为本类型

decltype(1) var12;//const int

decltype(Func(1)) var13=true;//const bool

decltype(i++) var14 = i;//int i++返回右值
```