decltype

含义

- decltype 用于对变量或者表达式进行类型推导，返回变量或者表达式的实际类型
- **和 auto 一样，decltype 进行类型推断同样发生在编译期。因此，当我们推导表达式的类型时，decltype 并不会真正地执行表达式，或者说构造临时对象**
- **和 auto 不同的是，当我们使用值类型的 auto 类型推断时，引用和 const 属性都会被忽略（指针类型除外）。但是 decltype 不会，即保留变量或表达式的原有类型，包括引用和 const 属性**

基本使用

decltype(变量)

```
int i = 1024;
const int &j = i;
const int &&k = 128;
```

- `auto m = j;` — 此时 m 的类型为 int
- `decltype(j) n = 2048;` `decltype(k) o = 2048;` — 此时 n 的类型为 const int &，o 的类型为 const int &&

- decltype 对变量的推导会保留变量的 const 属性以及引用属性，auto 就不行
- 正因为如此，我们才能够使用 type_id_with_cvr<decltype(T)>().pretty_name() 来输出函数模板中的类型参数 T 到底被推断成了哪一个类型

decltype(表达式)

- 我们也可以使用 decltype(表达式) 来获取表达式执行结果对应的类型，只不过并没有执行表达式而已
- `decltype(16 + 8) i;`
  - 那么此时 i 的类型即为 int，因为 decltype(16 + 8) 将得到 int
  - 同样的，decltype(16 + 8.0) 将得到 double
- **除此以外，如果表达式的结果能够作为赋值语句左边儿的内容，也就是推导结果是一个左值的话，decltype 将返回引用**

```
int *ptr = &i;
cout << type_id_with_cvr<decltype(*ptr)>().pretty_name() << endl;
```

- *ptr 是一个表达式，表示获取 ptr 指针所指向的对象，我们可以为其赋值: *ptr = 256; 所以 *ptr 的结果为左值
- 输出结果为 int &，即整型引用类型

- 另外，(变量) 也是一种表达式，例如 (i)、(j)、((k)) 等，并且这类表达式的结果为左值，所以使用 decltype((x)) 对其进行类型获取时，将会得到一个引用类型

decltype(函数)

```
int foo(int &value) {
    cout << "foo function" << endl;
    return 1024;
}
```

```
    cout << type_id_with_cvr<decltype(foo)>().pretty_name() << endl;
    cout << type_id_with_cvr<decltype(foo(i))>().pretty_name() << endl;
```

我们既可以对函数名称进行类型判断，也可以对函数的返回类型进行类型判断，后者并不会实际调用函数

- 上述语句分别输出 int (int&) 和 int，前者是函数原型，后者则是函数的返回类型

主要用途

应付可变数据类型

可变数据类型一般来说会出现在函数模板或者是类模板中，而 decltype 的主战地还是在类模板中，并且主要应对于 const 对象和非 const 对象

尾置返回类型

- C++ 中的位置返回类型非常类似于 Python 的 TypeHint，也是使用 -> 来表明函数返回类型

```
auto foo(int x, int y) -> int {
    return x + y;
}
```

尾置返回类型必须配合 auto 使用，此时的 auto 就是个占位符

- 对于一些比较复杂的返回类型，我们可以直接使用 decltype 替换掉

```
auto foo(int x, int y) -> decltype(x + y) {
    return x + y;
}
```

当然了，这里的返回类型就是一个 int，比较简单，但是复杂情况下使用 decltype 将简化代码的复杂度

decltype(auto)

decltype(auto) 这东西挺有意思的，auto 用于自动类型推断，而 decltype 则完整地返回变量或表达式的确切类型，两者一结合，就会变得很智能

① 函数返回类型自动推断

- 还是前面的那个例子，我们需要把 decltype(x + y) 这个表达式抄过来，有点儿麻烦

```
decltype(auto) foo(int x, int y) {
    return x + y;
}
```

表达式都可以省略，直接怼上一个 auto，函数返回啥类型由聪明的编译器自己推断

```
auto foo(int x, int y) {
    return x + y;
}
```

虽然编译不会报错，但是只用 auto 是有问题的。原因在于 auto 会选择性的忽略引用属性和 const 属性

- **因此，auto 所忽略的引用属性、const 属性能够通过 decltype(auto) 再重新捡回来**

② 变量声明

相比于用在函数上，decltype(auto) 在变量声明上用的比较少，因为我们不用 auto 也能够完成这件事儿

```
int i = 1024;
const int &k = i;

decltype(auto) m = k;
```

decltype(k) m; 这样也能可以的，并且语义在个人看来会更加清晰