第4章 decltype说明符

《现代C++语言核心特性解析》 谢丙堃

回顾typeof和typeid

- GCC的扩展提供typeof

```
int a = 0;
typeof(a) b = 5;
```

- C++标准提供typeid

```
int x1 = 0;
double x2 = 5.5;
std::cout << typeid(x1).name() << std::endl;
std::cout << typeid(x1 + x2).name() << std::endl;
std::cout << typeid(int).name() << std::endl;
```

使用decltype说明符

- 使用方法与typeof类似

```
int a = 0;
typeof(a) b = 5;

int x1 = 0;
decltype(x1) x2 = 0;
```

推导规则

- decltype(e)（其中e的类型为T）的推导规则有五条：
  1. 如果e是一个未加括号的标识符表达式（结构化绑定除外）或者未加括号的类成员访问，则decltype(e)推断出的类型是e的类型T。如果并不存在这样的类型，或者e是一组重载函数，则无法进行推导。
  2. 如果e是一个函数调用或者仿函数调用，那么decltype(e)推断出的类型是其返回值的类型。
  3. 如果e是一个类型为T的左值，则decltype(e)是T&。
  4. 如果e是一个类型为T的将亡值，则decltype(e)是T&&。
  5. 除去以上情况，则decltype(e)是T。

关于推导规则一个有趣的问题：C++中为什么对lambda引用捕获变量进行decltype后得到的类型为非引用类型？
我的回答：https://www.zhihu.com/question/605296860/answer/3067543229

推导规则

```
const int&& foo();
int i;
struct A {
    double x;
};
const A* a = new A();
decltype(foo());          // decltype(foo())推导类型为const int&&
decltype(i);              // decltype(i)推导类型为int
decltype(a->x);           // decltype(a->x)推导类型为double
decltype((a->x));         // decltype((a->x))推导类型为const double&
```

cv限定符的推导

- 通常情况下，decltype(e)所推导的类型会同步e的cv限定符

```
const int i = 0;
decltype(i);    // decltype(i)推导类型为const int
```

- 当e是未加括号的成员变量时，父对象表达式的cv限定符会被忽略

```
struct A {
    double x;
};
const A* a = new A();
decltype(a->x); // decltype(a->x)推导类型为double, const属性被忽略
```

decltype(auto)

- 用decltype的推导表达式规则来推导auto

```
auto x1 = (i);                  // x1推导类型为int
decltype(auto) x2 = (i);        // x2推导类型为int&
```

- 为非类型模板形参占位符

```
template<decltype(auto) N>
void f()
{
    std::cout << N << std::endl;
}
```

感谢聆听
欢迎关注