C++ 语言学习笔记

Roger Young

2017 年 9 月 19 日

目录

第一章　C++ 11 更新

1.1　auto

For variables, specifies that the type of the variable that is being declared will be automatically deduced from its initializer. For functions, specifies that the return type is a trailing return type or will be deduced from its return statements (since C++14). for non-type template parameters, specifies that the type will be deduced from the argument (since C++17)

1.1.1　基本语法

```
auto variable initializer;
auto function -> return type;
auto function;
decltype(auto) variable initializer;
decltype(auto) function;
auto :: ;
cv(optional) auto ref(optional) parameter;
template < auto Parameter >;
cv(optional) auto ref(optional) [ identifier-list ] initializer;
```

1.1.2　解释

1. When declaring variables in block scope, in namespace scope, in initialization statements of for loops, etc., the keyword `auto` may be used as the type specifier.

   Once the type of the initializer has been determined, the compiler determines the type that will replace the keyword `auto` using the rules for template argument deduction from a function call (see template argument deduction#Other contexts for details). The keyword `auto` may be accompanied by modifiers, such as `const` or `&`, which will participate in the type deduction. For example, given `const auto& i = expr;`, the type of i is exactly the type of the argument u in an imaginary template `template<class U> void f(const U& u)` if the function call f(expr) was compiled. Therefore, `auto&&` may be deduced either as an lvalue reference or rvalue reference according to the initializer, which is used in range-based for loop.

   If `auto` is used to declare multiple variables, the deduced types must match. For example, the declaration `auto i = 0, d = 0.0;` is ill-formed, while the declaration `auto i = 0, *p = &i;`

is well-formed and the auto is deduced as int.

2. In a function declaration that uses the trailing return type syntax, the keyword `auto` does not perform automatic type detection. It only serves as a part of the syntax.

3. In a function declaration that does not use the trailing return type syntax, the keyword auto indicates that the return type will be deduced from the operand of its return statement using the rules for template argument deduction.

4. If the declared type of the variable is `decltype(auto)`, the keyword `auto` is replaced with the expression (or expression list) of its initializer, and the actual type is deduced using the rules for `decltype`.

5. If the return type of the function is declared `decltype(auto)`, the keyword `auto` is replaced with the operand of its return statement, and the actual return type is deduced using the rules for `decltype`.

6. A nested-name-specifier of the form `auto::` is a placeholder that is replaced by a class or enumeration type following the rules for constrained type placeholder deduction.

7. A parameter declaration in a lambda expression. (since C++14) A function parameter declaration. (concepts TS)

8. If a template parameter is declared auto, its type is deduced from the corresponding argument.

9. A structured binding declaration

1.2 decltype

1.3 lambda expression

第二章　C++17 新功能

2.1 Inline Variables

Before C++17, if your class had any non-const static data members, you had to allocate memory for them. For example, suppose you have the following class definition:

Before C++17

```
class MyClass
{
private:
    static int s_anInt;
    static std::string s_aString;
};
```

Then your source file should contain the following:

Before C++17

```
int MyClass::s_anInt = 42;
std::string MyClass::s_aString = "Hello World!";
```

This is annoying.

C++17 now supports inline variables which allow you to write the MyClass definition as follows:

Before C++17

```
class MyClass
{
private:
    static inline int s_anInt = 42;
    static inline std::string s_aString = "Hello World!";
};
```

This feature makes it easier to write header only classes that contain non-const static data members.

At the time of this writing, Microsoft Visual C++ 2017 does not yet support inline variables.

附录