Contents

Part I

基本语言特性

这一部分介绍了C++17中新的核心语言特性，但不包括那些专为泛型编程（即template）设计的特性。这些新增的特性对于应用程序员的日常编程非常有用，因此每一个使用C++17的C++程序员都应该了解它们。

专为模板编程设计的新的核心语言特性在Part II中介绍。

Part II
模板特性

这一部分介绍了 C++17 为泛型编程（即 template）提供的新的语言特性。

我们首先从类模板参数推导开始，这一特性只影响模板的使用。之后的章节会介绍为编写泛型代码（函数模板，类模板，泛型库等）的程序员提供的新特性。

14 扩展的 using 声明

using 声明扩展之后可以支持逗号分隔的名称，也可以支持参数包。

例如，你现在可以这么写：

```
class Base {
  public:
    void a();
    void b();
    void c();
};

class Derived : private Base {
  public:
    using Base::a, Base::b, Base::c;
};
```

在 C++17 之前，你需要使用 3 个 using 声明分别进行声明。

14.1 使用变长的 using 声明

逗号分隔的 using 声明允许你用泛型代码从可变数量的所有基类中派生同一种运算。

这项技术的一个很酷的应用是创建一个重载的 lambda 的集合。通过如下定义：

tmpl/overload.hpp

```
// 继承所有基类里的函数调用运算符
template<typename... Ts>
struct overload : Ts...
{
    using Ts::operator()...;
};

// 基类的类型从传入的参数中推导
template<typename... Ts>
overload(Ts...) -> overload<Ts...>;
```

你可以像下面这样重载两个 lambda：

```
auto twice = overload {
                [](std::string& s) { s += s; },
                [](auto& v) { v *= 2; }
             };
```

这里，我们创建了一个 `overload` 类型的对象，并且提供了推导指引 来根据 lambda 的类型推导出 `overload` 的基类的类型。并且我们使用了聚合体初始化 来调用每个 lambda 生成的闭包类型的拷贝构造函数来初始化基类子对象。

上例中的using声明使得overload类型可以同时访问所有子类中的函数调用运算符。如果没有这个using声明，两个基类会产生同一个成员函数operator()的重载，这将会导致歧义。[4]

最后，如果你传递一个字符串参数将会调用第一个重载，其他类型（操作符*=有效的类型）将会调用第二个重载：

```
int i = 42;
twice(i);
std::cout << "i: " << i << '\n';    // 打印出：84
std::string s = "hi";
twice(s);
std::cout << "s: " << s << '\n';    // 打印出：hihi
```

这项技术的另一个应用是std::variant访问器。

14.2 使用变长using声明继承构造函数

除了逐个声明继承构造函数之外，现在还支持如下的方式：你可以声明一个可变参数类模板Multi，让它继承每一个参数类型的基类：

tmpl/using2.hpp

```
template<typename T>
class Base {
    T value{};
  public:
    Base() {
        ...
    }
    Base(T v) : value{v} {
        ...
    }
    ...
};

template<typename... Types>
class Multi : private Base<Types>...
{
  public:
    // 继承所有构造函数：
    using Base<Types>::Base...;
    ...
};
```

有了所有基类构造函数的using声明，你可以继承每个类型对应的构造函数。

现在，当使用不同类型声明Multi<>时：

```
using MultiISB = Multi<int, std::string, bool>;
```

[4]clang和Visual C++都不会把不同基类中不同类型的同名函数当作歧义处理，所以这个例子中其实不需要using。然而，这段代码如果没有using声明的将不具备可移植性。

你可以使用每一个相应的构造函数来声明对象：

```
MultiISB m1 = 42;
MultiISB m2 = std::string("hello");
MultiISB m3 = true;
```

根据新的语言规则，每一个初始化会调用匹配基类的相应构造函数和所有其他基类的默认构造函数。因此：

```
MultiISB m2 = std::string("hello");
```

会调用 `Base<int>` 的默认构造函数，`Base<std::string>` 的字符串构造函数，`Base<bool>` 的默认构造函数。

原则上讲，你也可以通过如下声明来支持 `Multi<>` 进行赋值操作：

```
template<typename... Types>
class Multi : private Base<Types>...
{
    ...
    // 派生所有赋值运算符
    using Base<Types>::operator=...;
};
```

14.3 后记

逗号分隔的 using 声明列表由 Robert Haberlach 在 https://wg21.link/p0195r0 中首次提出。最终被接受的提案由 Robert Haberlach 和 Richard Smith 发表于 https://wg21.link/p0195r2。

关于继承构造函数有一些核心的问题。最终修复这些问题的提案由 Richard Smith 发表于 https://wg21.link/n4429。

还有一个由 Vicente J.Botet Escriba 提出的提案。除了 lambda 之外，它还支持重载普通函数、成员函数来实现泛型的 `overload` 函数。然而，这个提议并没有进入 C++17 标准。详情请见 https://wg21.link/p0051r1。

Part III
新的标准库组件

这一部分介绍C++17中新的标准库组件。

Part IV
已有标准库的拓展和修改

这一部分介绍C++17对已有标准库组件的拓展和修改

Part V
专家的工具

这一部分介绍了普通应用程序员通常不需要知道的新的语言特性和库。它主要包括了为编写基础库和语言特性的程序员准备的用来解决特殊问题语言特性（例如修改了堆内存的管理方式）。

Part VI
一些通用的提示

这一部分介绍了一些有关C++17的通用的提示，例如对C语言和废弃特性的兼容性更改。