C++ 20常用新特性

目录

模块

从 C 语言中，C++ 继承了 #include 机制，依赖从头文件使用文本形式包含 C++ 源代码，这些头文件中包含了接口的文本定义。一个流行的头文件可以在大型程序的各个单独编译的部分中被 #include 数百次。基本问题是：

(1) 不够卫生：一个头文件中的代码可能会影响同一翻译单元中包含的另一个 #include 中的代码的含义，因此 #include 并非顺序无关。宏是这里的一个主要问题，尽管不是唯一的问题。

(2) 分离编译的不一致性：两个翻译单元中同一实体的声明可能不一致，但并非所有此类错误都被编译器或链接器捕获。

(3) 编译次数过多：从源代码文本编译接口比较慢。从源代码文本反复地编译同一份接口非常慢。

所以，在 C++ 程序中改进模块化是一个迫切的需求

模块

```
#include <iostream>
int main() {
  std::cout << "Hello, C++!" << std::endl;
}
```

这段标准代码有 70 个左右的字符，但是在 #include 之后，它会产生 419909 个字符需要编译器来消化。尽管现代 C++ 编译器已有傲人的处理速度，但模块化问题已经迫在眉睫。

模块

模块化是什么意思？顺序独立性：import X; import Y; 应该与 import Y; import X; 相同。

换句话说，任何东西都不能隐式地从一个模块“泄漏”到另一个模块。

这是 #include 文件的一个关键问题。

#include 中的任何内容都会影响所有后续的 #include。

顺序独立性是“代码卫生”和性能的关键。

模块

C++ 20 中正式引入了模块的概念，模块是一个用于在翻译单元间分享声明和定义的语言特性。它们可以在某些地方替代使用头文件。

其主要优点如下：

1. 没有头文件。
2. 声明实现仍然可分离, 但非必要。
3. 可以显式指定导出哪些类或函数。
4. 不需要头文件重复引入宏（include guards）。
5. 模块之间名称可以相同，并且不会冲突。
6. 模块只处理一次，编译更快（头文件每次引入都需要处理，需要通过 pragma once 约束）。
7. 预处理宏只在模块内有效。
8. 模块的引入与引入顺序无关。

模块

创建模块

源文件->添加->新建项->Module

创建***.ixx文件

模块

```
//创建模块
// mymodule.ixx  //模块名和文件名没有强制要求，一般会相同
export module helloworld;  // 模块声明
import <iostream>;        // 导入声明 注意；号


export void hello() {     // 导出声明
    std::cout << "Hello world!\n";
}
```

模块

```
//导入模块
// main.cpp
import helloworld;  // 导入声明


int main() {
    hello();
}
```

协程

协程就是一个可以挂起（suspend）和恢复（resume）的函数（不能是 main 函数）。你可以暂停协程的执行，去做其他事情，然后在适当的时候恢复到暂停的位置继续执行。协程让我们使用同步方式写异步代码。

C++ 提供了三个方法挂起协程：co_await， co_yield 和 co_return。

C++20协程只是提供协程机制，而不是提供协程库。C++20的协程是无栈协程，无栈协程是一个可以挂起/恢复的特殊函数，是函数调用的泛化，且只能被线程调用，本身并不抢占内核调度。

C++20 提供了三个新关键字(co_await、co_yield 和 co_return)，如果一个函数中存在这三个关键字之一，那么它就是一个协程。

co_yield some_value: 保存当前协程的执行状态并挂起，返回some_value给调用者

co_await some_awaitable: 如果some_awaitable没有ready，就保存当前协程的执行状态并挂起

co_return some_value: 彻底结束当前协程，返回some_value给协程调用者

<=> 三向比较运算符

也叫: 三路比较运算符

三路比较结果如下

(a <=> b) < 0 // 如果 a < b 则为 true

(a <=> b) > 0 // 如果 a > b 则为 true

(a <=> b) == 0 // 如果 a 与 b 相等或者等价 则为 true

类似于C的strcmp 函数返回-1, 0, 1

一般情况: 自动生成所有的比较操作符, 如果对象是结构体则逐个比较, 可以用下面代码代替所有的比较运算符

auto X::operator<=>(const Y&) = default;

高级情况: 指定返回类型(支持6种所有的比较运算符)

<=> 三向比较运算符

```
int num1 = 100, num2 = 100;
if ((num1 <=> num2) < 0) {
        cout << "num1 < num2" << endl;
}
else if ((num1 <=> num2) > 0) {
        cout << "num1 > num2" << endl;
}
else {
        cout << "num1 = num2" << endl;
}
```

范围 ranges

范围库始于 Eric Niebler 对 STL 序列观念的推广和现代化的工作。它提供了更易于使用、更通用及性能更好的标准库算法。

例如，C++20 标准库为整个容器的操作提供了期待已久的更简单的表示方法。

```
void func1(vector<string>& s) {
    sort(s);        // 而不是 sort(vs.begin(), vs.end());
}
```

范围 ranges

```
#include <vector>
#include <ranges>
#include <iostream>
using namespace std;
int main()
{
    auto ints = views::iota(0, 10);//生成0-9
    auto even = [](int i) { return 0 == i % 2; };
    auto square = [](int i) { return i * i; };

    for (int i : ints | views::filter(even) | views::transform(square))
        cout << i << ' ';

    return 0;
}
```

日期和时区

日期库是日期库是多年工作和实际使用的结果,它基于 chrono 标准库的时间支持。在 2018 年，它进入了 C++20，并和旧的时间工具一起放在 <chrono> 中。

日期和时区

```
#include <iostream>
#include <chrono>
using namespace std;
using namespace std::chrono;
int main()
{

    // creating a year
    auto y1 = year{ 2019 };
    auto y2 = 2019y;
    // creating a mouth
    auto m1 = month{ 9 };
    auto m2 = September;
    // creating a day
    auto d1 = day{ 18 };
    auto d2 = 18d;

    year_month_day date1{ 2022y,July, 21d };
    auto date2 = 2022y / July / 21d;
    chrono::year_month_day date3{ Monday[3] / July / 2022 };

    cout << date1 << endl;
    cout << date2 << endl;
    cout << date3 << endl;

    return 0;
}
```

格式化

iostream 库提供了类型安全的 I/O 的扩展，但是它的格式化工具比较弱。

另外，还有的人不喜欢使用 << 分隔输出值的方式。

格式化库提供了一种类 printf 的方式去组装字符串和格式化输出值，同时这种方法类型安全、快捷，并能和 iostream 协同工作。

类型中带有 << 运算符的可以在一个格式化的字符串中输出。

```
string s1 = "C++";
cout << format("The string '{}' has {} characters", s1, s1.size());
cout << format("The string '{0}' has {1} characters", s1, s1.size()) << endl;
cout << format("The string '{1}' has {0} characters", s1.size(), s1) << endl;
```

跨度

越界访问，有时也称为缓冲区溢出，从 C 的时代以来就一直是一个严重的问题。考虑下面的例子：

```
void func1(int* p, int n) {   // n 是什么？
    for (int i = 0; i < n; ++i) {
        p[i] = 7;             // 是否可行？
    }
}
```

跨度

span<T> 类模板就这样被放到 C++ 核心指南的支持库中。

```
void func(span<int> a) {   // span 包含一个指针和一条大小信息
  for (int& x : a) {
    x = 7;               // 可以
  }
}
```

范围 for 从跨度中提取范围，并准确地遍历正确数量的元素（无需代价高昂的范围检查）。这个例子说明了一个适当的抽象可以同时简化写法并提升性能。对于算法来说，相较于挨个检查每一个访问的元素，明确地使用一个范围（比如 span）要容易得多，开销也更低。

并发

C++ 20 并发编程 std::promise

std::promise和std::future是一对, 通过它们可以进行更加灵活的任务控制

promise通过函数set_value()传入一个值, 异常, 或者通知, 并异步的获取结果

并发

```
#include <iostream>
#include <future>
#include <format>
using namespace std;

void product(promise<int>&& intPromise, int v1, int v2)
{
	intPromise.set_value(v1 * v2);
}

int main()
{
	int num1 = 200;
	int num2 = 300;
	promise<int> productPromise;

	future<int> productResult = productPromise.get_future();

	jthread productThread(product, move(productPromise), num1, num2);

	cout << format("product is {}\n", productResult.get());
}
```

并发

std::future:

从promise获取值

询问值是否可用

等待通知

创建shared_future

并发

(构造函数)	**构造 future 对象 (公开成员函数)**
(析构函数)	析构 future 对象 (公开成员函数)
operator=	移动future对象 (公开成员函数)
share	从 `*this` 转移共享状态给 `shared_future` 并返回它 (公开成员函数)
获取结果	
get	返回结果 (公开成员函数)
状态	
valid	检查 future 是否拥有共享状态 (公开成员函数)
wait	等待结果变得可用 (公开成员函数)
wait_for	等待结果，如果在指定的超时间隔后仍然无法得到结果，则返回。 (公开成员函数)
wait_until	等待结果，如果在已经到达指定的时间点时仍然无法得到结果，则返回。 (公开成员函数)

并发

std::future_status

调用后wait_for或者wait_until返回的结果

```
enum class future_status
{
        ready,     //成功
        timeout,   //超时
        deferred   //延迟
};
```

并发

```
#include <iostream>
#include <future>
#include <format>
using namespace std;
void getAnswer(promise<int> intPromise)
{
	this_thread::sleep_for(2s);
	intPromise.set_value(100);
}
int main()
{
	promise<int> answerPromise;
	auto fut = answerPromise.get_future();
	jthread productThread(getAnswer, move(answerPromise));
	future_status status{};

	do
	{
		status = fut.wait_for(0.5s);
		cout << "结果未准备完成 " << endl;
	} while (status != future_status::ready);
	cout << format("answer is {}\n ", fut.get());
}
```

知识点总结

1.模块

2.协程

3.<=> 三向比较运算符

4.范围

5.日期和时区

6.格式化

7.跨度

8.并发

作业

1. C++ 20 新特性 模块(Modules) 有哪些优点？

优点:

1）没有头文件；

2）声明实现仍然可分离, 但非必要；

3）可以显式指定导出哪些类或函数；

4）不需要头文件重复引入宏 (include guards)；

5）模块之间名称可以相同，并且不会冲突；

6）模块只处理一次, 编译更快 (头文件每次引入都需要处理)；

7）预处理宏只在模块内有效；

8）模块的引入与引入顺序无关。