C++ 20 的模块对老式头文件的兼容性和提升

原创 RainyCRH 日常记录 2019/08/12 15:03 阅读数 807

唠叨

根据标准，模块兼容老式头文件（必须的）……

如果写下 `import <some_header.h>;` 或者 `import "some_header.h";` 这样的 `import` 语句，那么针对同一个头文件来说，它们会作为同一个隐式的独立模块来编译，并且是原子的。

而，现在的预处理器中， `#include <some_header.h>` 或者 `#include "some_header.h"` 则是无视任何 C++ 语义，暴力把文件内容复制粘贴进源文件……（逃

#include 方式

```
// hello.cpp

#include "some_header.h"
#include <vector>

int main()
{
    std::vector<int> numbers = { 1, 2, 3, 4, 5 };

    cao_dan::fuck(numbers);
}
```

针对如上源文件，如果我修改了这里面的任何内容，都会导致头文件 `<vector>` 和 `some_header.h` 的内容被暴力 `#include` 进 hello.cpp 然后重新编译一次。

如果头文件的内容，是一堆模板元的话……这编译速度，你懂的……（逃

import 方式

```
// hello.cpp

import "some_header.h";
```

```
import <vector>;

int main()
{
    std::vector<int> numbers = { 1, 2, 3, 4, 5 };

    cao_dan::fuck(numbers);
}
```

这种方式，很好地兼容了老式头文件。不同的是，它把同一个头文件当成一个模块单元来编译，并生成 BMI（标准中叫 Binary Module Interface，即二进制模块接口）文件，缓存起来。这样在后续编译中，如果没改变头文件 `some_header.h` 和 `<vector>` 的内容的话，构建系统就会直接加载这些 BMI，并链接，而不用重新编译。（666

有宏控制的情况

问题来了：如果老式头文件中有宏控制怎么bang?

```
// legacy.h

#pragma once

#ifdef SOME_OPTION_1
// ...
#elif defined(SOME_OPTION_2)
//...
#endif
```

针对以上头文件，采用 `import "legacy.h"` 的方式无法导入宏，也就无法让用户设置生效。这种情况，大佬们也考虑到了。推荐的做法是用包装头文件实现。

```
// legacy_some_option_1.h

#pragma once

#define SOME_OPTION_1

#include <legacy.h>


// legacy_some_option_2.h

#pragma once
```

```
#define SOME_OPTION_2

#include <legacy.h>
```

这样 `legacy_some_option_1.h` 就是针对 `SOME_OPTION_1` 的设置， `legacy_some_option_2.h` 就是针对 `SOME_OPTION_2` 的设置。

使用如下：

```
// main.cpp

import "legacy_some_option_1.h";
// import "legacy_some_option_2.h";

// ...
```

废话

针对这个 BMI 以及其派生出来的 C++ 生态系统技术报告，准备在 C++ 20 发布之前拟定。主要包括一些亟待统一的地方：

- 模块 ABI
- 基于模块编译的二进制库的发布和使用
- 针对入门者的友好的模块化 Hello World
- 构建系统（如 CMake、MSBuild 等等）
- 编译器实现（如 MSVC、Clang、GCC 等）
- 名称查找规则（比如一个模块 `boost.asio` 怎么对应到具体的 BMI 文件或源文件，这个规则是什么？）
- 模块世界的包管理工具（急需！