

72 Precompiled Headers in C++

本节会讲C++的

```
precompiled header
```

（预编译头文件），首先讨论这是什么，为什么你应该使用它。然后进入VS和gcc、g++，了解如何在现实世界中设置它们并通过一个简单例子看看实际的好处。

1. 什么是预编译头文件

预编译的头文件实际上是让你抓取一堆头文件，并将它们转换成编译器可以使用的格式，而不必一遍又一遍地读取这些头文件。举个例子，每次在C++文件中`#include <vector>`的时候，它需要读取整个Vector头文件并编译它，而且Vector还包含一堆其它的包含文件，这些文件也一样需要读取，预处理器必须把这些复制到这个Vector文件，这就有10w+行代码了，它们需要被解析并以某种形式标记并编译，在你想要编译main文件之前，因为你的main文件包含Vector文件的话，Vector必须复制并粘贴到main文件中，然后所有代码每次都需要被解析和编译。重点是每次你对C++文件进行修改，哪怕只是加了个空格，整个文件都要重新编译，所以Vector文件必须被复制并粘贴到你的C++文件中，从头开始重新解析并编译。不仅如此，你的项目中有多个文件它们又都包含了Vector，你不得不持续一遍遍地解析同样的代码，这需要大量时间。

所以你可以用一个叫做**预编译头文件**的东西来代替，它的作用是接受一堆你告诉它要接收的头文件（基本上是一堆代码）它只编译一次，以二进制格式存储，这对编译器来说比单纯的文本处理要快得多。这样就不需要解析整个Vector文件，每次它只需要看预编译的头文件，which此时已经是非常快速且容易使用的、对编译器来说很容易使用的二进制格式。这意味着它会大幅加快编译时间，特别是你的项目越来越大，你会有越来越多的C++文件。越来越多的头文件，诸如此类，你可以在预编译头文件中添加更多内容，你也有更多使用了共同头文件的源文件需要编译，它会指数级地加速，好的多得多。

所以如果你关心编译时间，你一定要使用预编译头文件。

不过，还有些你不应该用预编译头文件做的事：

到目前为止提到的预编译头文件，其本质还是头文件，which包含了一堆其它头文件。因此你可能会想把项目中所有的东西都放在预编译头文件中，如果这样做的话是不是构建速度飞快。

是这样，但是如果你把东西放到预编译头文件中，而这些东西会发生变化，在实际的项目中我们在处理项目所以它很有可能会变化，显然必须重新构建预编译的头文件，而这要花费时间，这也可能会导致编译速度变慢。所以不要把会频繁更改的文件放入预编译头文件中。

尽管预编译头文件很有用，而且把你自己的项目文件当进去也没问题，比如把一个不会需要修改的Log.h文件放进去就很好，因为这个文件很常用，也方便使用，你不需要再手动地将Log包含到项目中的每个C++文件中。但只要这个Log会修改，就不适合放入预编译头文件中，否则每次都要重新编译。

预编译头文件真正有用的是外部依赖，本质上它主要用于不是你写的那些代码，比如STL、Windows api等，如果你要`#include <windows.h>`，which is a 巨大的的头文件，包含了非常多的其它头文件，你不回去修改windows.h或者STL，所以它没有理由不被你放在预编译头文件中，因为它们的代码可能比你的实际项目代码多很多倍，每个C++文件每次都要编译它们可想是一件多么恐怖的事情，你可能永远也不会去修改它们。因此直接把它们放入到预编译头文件中就不用管了。

2. 依赖关系

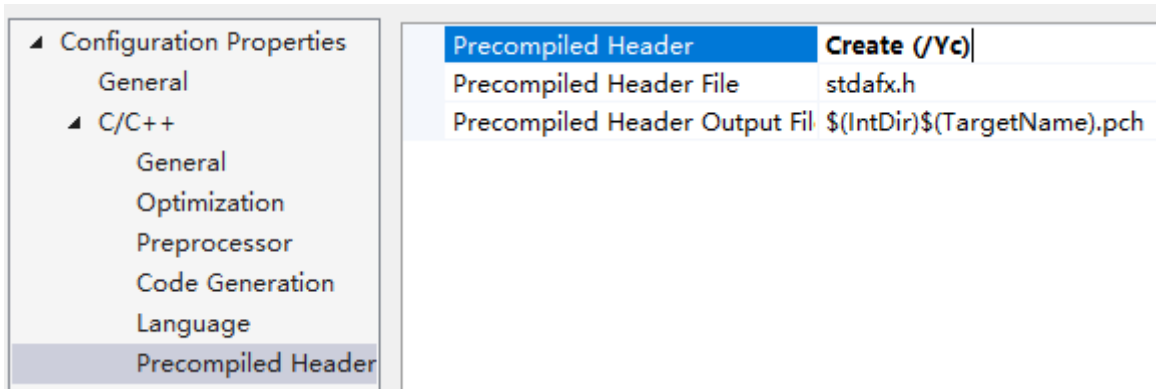
PCH（就是预编译头文件）实际上做的事是把所有东西都塞进来，它可能会隐藏现在实际正在使用的东西，会影响可读性。比如只有个别文件需要使用一个窗口库GLFW，那就没必要把所有的依赖项都放在PCH中，如果你只看一个单独的cpp文件你并不知道它需要什么依赖，再把它导入其它文件时就不好理解它依赖的东西了。但如果你通过实际的`include`包含它们就很清晰了，可以看到每个文件需要什么文件。但是如果你只包含PCH，然后PCH中放很多包含文件，就会比较麻烦了。

所以**不要把所有依赖都放在PCH中**，因为包含实际的依赖会更容易阅读。应该放进PCH的东西是像STL这样的，因为string、vectors、std::cout是许多地方都要用到的，你不希望每次都编译它们，而GLFW可能就只需要编译一次。

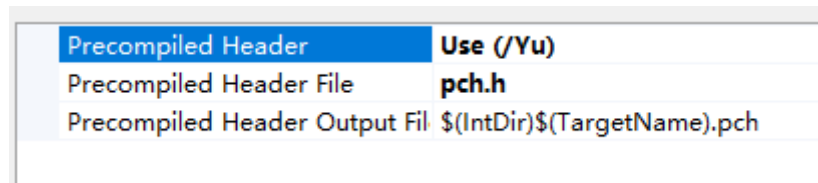
3. 案例

对于Visual Studio

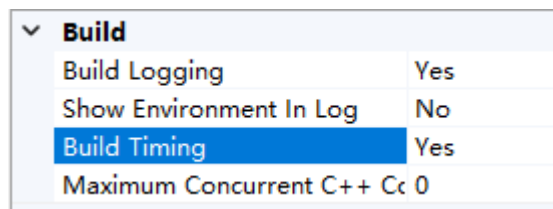
我们需要创建一个包含头文件的cpp文件，然后编辑它的属性（并且把第二行的值清空）：



然后到整个项目下，选择项目属性，指定为Use并且指定文件名（注意要选择全平台）：



然后build整个项目，可以去Tools的设置里面修改，让VS显示build耗时：



不使用PCH的build耗时和后续构建耗时：

```
1>Project Performance Summary:
1> 2871 ms C:\Dev\HelloWorld\HelloWorld\HelloWorld.vcxproj 1 calls
1>
```

```
1>Project Performance Summary:
1> 1471 ms C:\Dev\HelloWorld\HelloWorld\HelloWorld.vcxproj 1 calls
1>
```

使用PCH后的build耗时和后续构建耗时：

```
1>Project Performance Summary:
1> 1705 ms C:\Dev\HelloWorld\HelloWorld\HelloWorld.vcxproj 1 calls
1>
```

```
1>Project Performance Summary:
1> 485 ms C:\Dev\HelloWorld\HelloWorld\HelloWorld.vcxproj 1 calls
1>
```

可以看到用pch的构建速度快很多。

对于g++

Windows用户可以用Cygwin模拟Unix环境，或直接使用Windows的计时功能：






```
Measure-Command { g++ -std=c++11 Main.cpp }
```

```
Seconds      : 1  
Milliseconds : 231  
Ticks        : 12313404
```

创建预编译文件:

```
g++ -std=c++11 pch.h
```

生成了这个pch.h.gch文件, 是个多达100MB的文件

 a.exe	2023/7/29 14:19	应用程序	56 KB
 Main.cpp	2023/7/29 14:04	C++ Source	1 KB
 pch.cpp	2023/7/29 13:51	C++ Source	1 KB
 pch.h	2023/7/29 13:45	H 文件	1 KB
 pch.h.gch	2023/7/29 14:20	GCH 文件	117,217 KB

删掉a.exe, 相当于clean build, 再计时:

```
Seconds      : 0  
Milliseconds : 229  
Ticks        : 2299135
```

可以发现快了六倍左右。

总结

希望这些例子可以带来些启发。

回顾一下为什么要用预编译头文件: 加速编译时间, 也使实际编写代码更加方便, 因为你不需要再一遍遍地包含常用的头文件。除非是非常小的sandbox项目, 每个项目都应该使用PCH, 不过你在里面放什么就是另一个故事了。