**概述**

本文主要介绍VSCode作为C++主力开发工具的使用环境，VSCode具有丰富的扩展插件，每个人根据喜好可以安装各式各样的插件，本文仅根据作者本地的环境进行介绍。本文总共分为四部分，前三部分是独立的几个开发工具的介绍，包括Git、CMake、Doxygen、clang-format、clang-tidy等，第五部分则介绍如何使用VSCode融合前面的几个开发工具，实现一个完整的开发环境。

**第一部分 Git**

Git作为版本控制软件，基础的拉取代码、提交代码、推送代码这里不再详细介绍，本次主要介绍两个常用的多分支操作，即变基和摘取，可参考文档《Git变基和摘取示例（可视化工具）.docx》。

**变基（rebase）**

默认的git pull实际对应的行为是git fetch + git merge，即拉取远端并将远端合并到本地分支，当本地分支与远端分支各有提交时，会自动形成一个merge commit，当一个分支使用的人很多时，这种merge commit会有很多，导致分支图比较混乱，看不出代码路线图。

实际上在安装Git时，可以选择git pull的行为，除了git fetch + git merge以外，还可以选择git fetch + git rebase。合理的使用变基，可以带来美观的分支图，同时，对于也可以让冲突在合并请求前解决。单个分支开发时，Git变基使用起来就跟SVN差不多，总是先获取远端的改动，然后将本地的改动接在后面。修改全局配置将git pull的默认行为修改为git fetch + git rebase，指令如下：

git config --global --add pull.rebase true

**摘取（cherry-pick）**

摘取就是将一个分支中某些提交在另一个分支上再次提交，避免重复操作，主要在涉及多分支开发的情况下，同一个改动，能够快速在各个分支上提交。减少分支合并时需要合并的代码量。

**第二部分 CMake**

CMake是一个跨平台的构建工具，能够输出各种各样的makefile或者project文件，这里主要介绍在Windows平台，使用MSVC作为开发环境下的CMake的使用。使用CMake的优点：

CMakeLists.txt是文本文件，容易处理冲突，可以减少工程配置冲突的修改难度

CMake添加工程、修改工程属性、修改工程名字等操作非常方便，且能够自动解析工程依赖关系

VSCode支持，有简易还用的插件

更容易引入第三方库或者框架，因为大部分C++开源项目都支持CMake

跨平台开发准备

CMake内容非常庞大，这里贴一个简单的dll + exe的CMakeList示例：

# 第一个CMakeLists.txt

cmake\_minimum\_required(VERSION 3.16)

# 添加一个动态库，名字叫testlib，源文件是testlib.cpp

add\_library(testlib SHARED testlib.cpp)

# 动态库在Debug编译时自动添加后缀‘d’

set\_target\_properties(testlib PROPERTIES DEBUG\_POSTFIX d)

# 指定头文件搜索目录

target\_include\_directories(testlib PUBLIC ${CMAKE\_CURRENT\_LIST\_DIR})

target\_include\_directories(testlib PUBLIC ${CMAKE\_SOURCE\_DIR}/include)

# 编译预处理宏

target\_compile\_definitions(testlib

    PRIVATE

    PLUGINUIA\_LIB

)

--------------------------------------------------------------

# 第二个CMakeLists.txt

cmake\_minimum\_required(VERSION 3.16)

# 添加一个可执行文件，名字叫execute，源文件是main.cpp

add\_executable(execute main.cpp)

# 链接到testlib库

target\_link\_libraries(execute PRIVATE testlib)

**第三部分 其它工具**

**doxygen**

doxygen是一个用于规范化注释，并支持将注释生成API文档的工具。参考文档《doxygen代码注释规范.docx》

**clang-format**

clang-format是LLVM系列里面的一个代码格式化工具，通过clang-format.exe和.clangformat配置文件，实现代码格式统一，包括换行、空格、缩进等。

**clang-tidy**

clang-tidy是LLVM系列里面的一个代码静态检查工具，通过clang-tidy.exe和.clang-tidy配置文件，实现代码的静态检查，在编译前检查可能出现问题的代码，提高代码质量。

**第四部分 基于VSCode的融合开发环境**

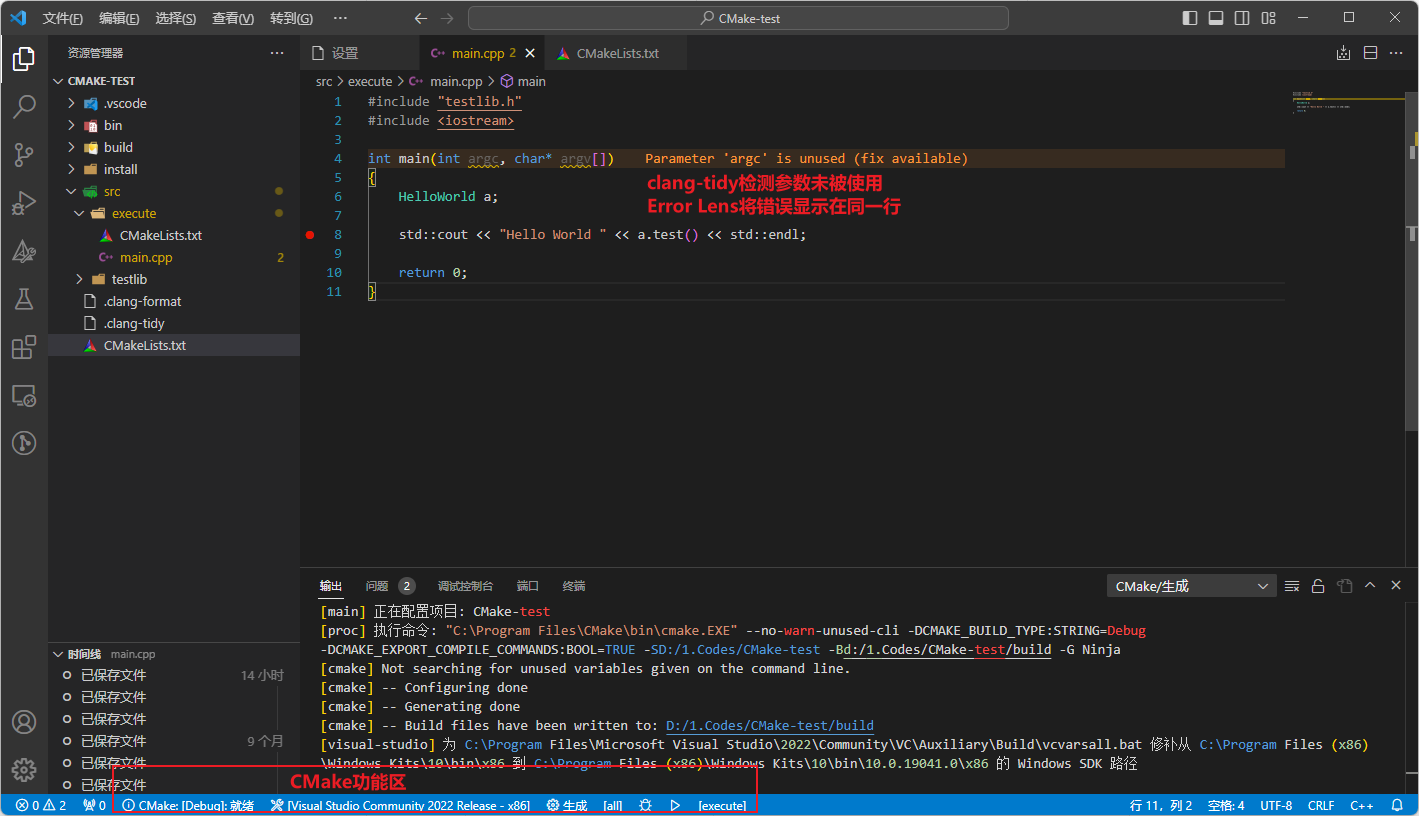
**安装内容**

|  |  |
| --- | --- |
| **安装项** | **说明** |
| Visual Studio 2022 | MSVC编译器 |
| VS Code | VSCode主程序 |
| CMake | CMake主程序 |
| LLVM | 用于使用clang-format、clang-tidy、clangd等功能 |
| Doxygen | 用于生成接口API文档 |
| Git | 用于版本管理 |

**VSCode插件**

|  |  |  |
| --- | --- | --- |
|  | **插件** | **说明** |
| 必要安装 | C/C++ | VSCode进行C++开发必不可少的插件，是一个C++语言服务，包括代码智能感知、调试、代码编辑等功能。 |
| Clangd | 除了代码调试，几乎可以替换C/C++插件。Clangd插件依赖LLVM的clangd语言服务（包含C++），同时可以直接支持clang-format和clang-tidy，推荐安装clangd并且停用C/C++的语言服务。  C/C++和Clangd的语言服务是冲突的，只能选择一个，但是在VSCode上调试C++代码必须要安装C/C++插件。 |
| CMake Tools | VSCode上使用CMake的插件。 |
| Doxygen Documentation Generator | 生成Doxygen注释的插件。 |
| 推荐安装 | CMake + CMake Language Support | 方便编辑CMakeLists.txt文件，提供CMake语言服务。依赖 Dot Net 6.0，可以去官网下载安装 |
| vscode-icons | 高对比度的资源管理器，方便查看文件结构 |
| GitLens | 方便管理GIT分支和查看代码改动 |
| Git Graph | 方便查看Git分支图 |
| Error Lens | 方便查看错误提示 |

安装上述插件后，一个简易的C++开发环境如下：



**clangd插件配置**

clangd配置中，主要是需要开启clang-tidy，其它配置可以根据个人喜好添加，clangd支持配置的内容可以通过命令行获取：

# 摘取部分clangd命令行说明

PS D:\> clangd --help

clangd compilation flags options:

--compile-commands-dir=<string> - Specify a path to look for compile\_commands.json. If path is invalid, clangd will look in the current directory and parent paths of each source file

--query-driver=<string> - Comma separated list of globs for white-listing gcc-compatible drivers that are safe to execute. Drivers matching any of these globs will be used to extract system includes. e.g. /usr/bin/\*\*/clang-\*,/path/to/repo/\*\*/g++-\*

clangd feature options:

--all-scopes-completion - If set to true, code completion will include index symbols that are not defined in the scopes (e.g. namespaces) visible from the code completion point. Such completions can insert scope qualifiers

--background-index - Index project code in the background and persist index on disk.

--clang-tidy - Enable clang-tidy diagnostics

--completion-style=<value> - Granularity of code completion suggestions

=detailed - One completion item for each semantically distinct completion, with full type information

=bundled - Similar completion items (e.g. function overloads) are combined. Type information shown where possible

--fallback-style=<string> - clang-format style to apply by default when no .clang-format file is found

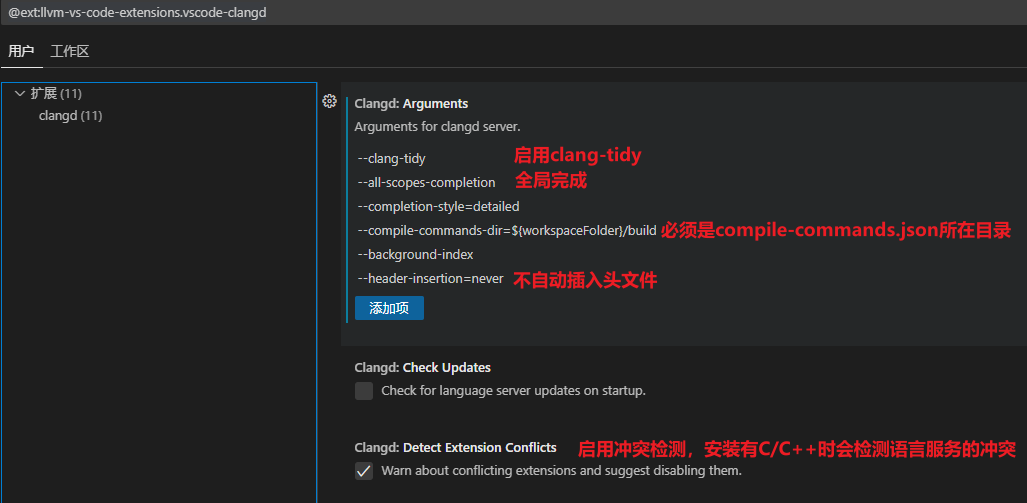
--function-arg-placeholders - When disabled, completions contain only parentheses for function calls. When enabled, completions also contain placeholders for method parameters

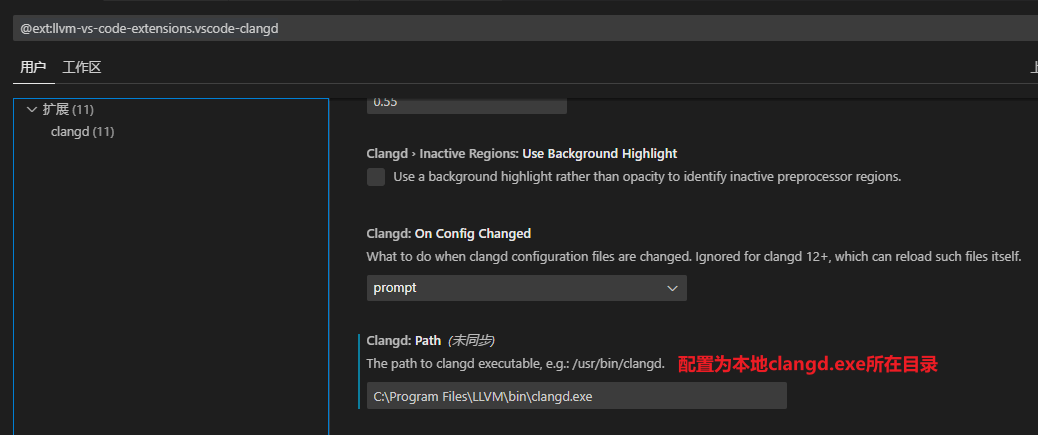
--header-insertion=<value> - Add #include directives when accepting code completions

=iwyu - Include what you use. Insert the owning header for top-level symbols, unless the header is already directly included or the symbol is forward-declared

=never - Never insert #include directives as part of code completion

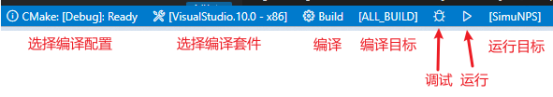
本地目前的配置如下：





**CMake插件**

CMake功能区的按钮功能如下：



**编译配置**

|  |  |
| --- | --- |
| Debug | 禁用优化，包括调试信息 |
| Release | 进行优化，提高速度，排除调试信息 |
| RelWithDebInfo | 进行优化，提高速度，包括调试信息 |
| MinSizeRel | 优化最小二进制大小，排除调试信息 |

**编译套件和调试运行**

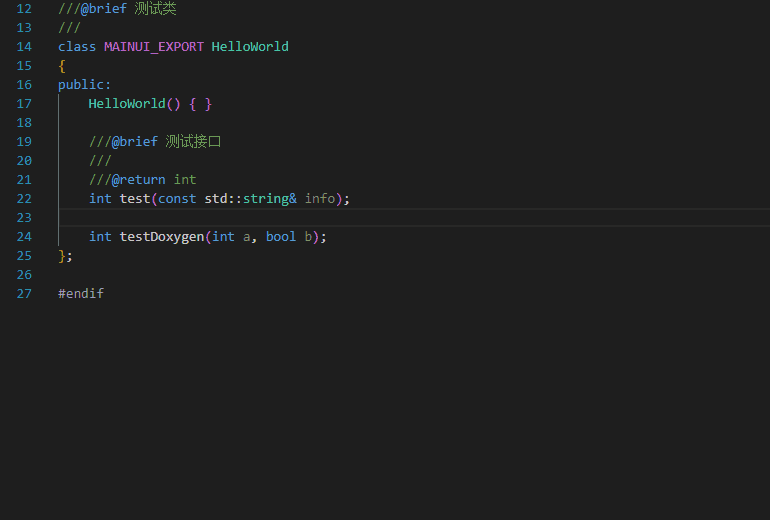
编译套件取决于本地电脑所安装的编译器，如MSVC、MinGW、Clang等。编译目标和和运行目标取决于项目中编译和运行的对象个数。

**CMake Install**

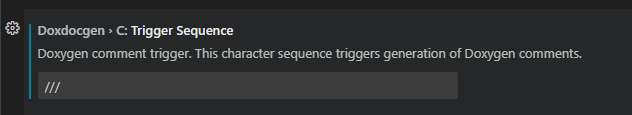
CMake Install是将编译后的文件进行安装，可以预先设计每个对象的安装位置，解决拷贝依赖库等问题。

**Doxygen注释生成：**

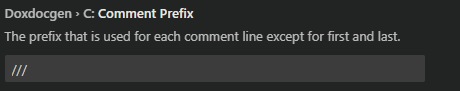
在函数声明的上方，输入触发符号（本地为///）后，按回车即可生成该函数的注释：



配置触发注释的方式：

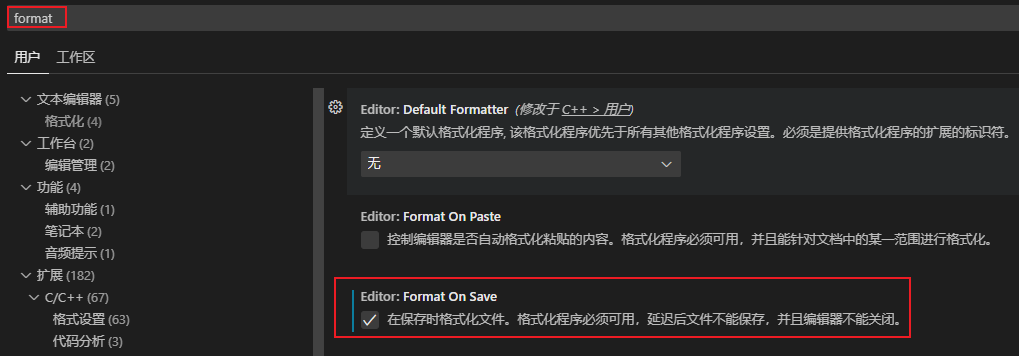


配置注释前缀：



**Clang-Format格式化**

一版来说，配置为保存时自动触发格式化即可：



**Clang-Tidy自动修复**

clangd插件带来的一个好处是可以自动修复clang-tidy检测出来的代码问题（标注为fix available），以下为演示效果：

