vscode+qt+qmake开发环境搭建，史上最全最详细！

00. 前言

鉴于Qt官方 IDE 太过难用，VSCode + 各种插件功能强大，遂采用VSCode来写Qt项目。

01. 环境搭建

1. 需要安装的软件：

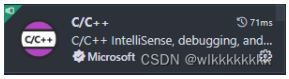
- VSCode，官方最新版就行
- Qt，版本随意，本文主要针对较老版本使用Qmake构建系统的项目

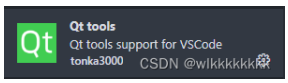
2. 环境变量：

- Qt环境变量，需要配置Qt库跟MinGW，尽量使用Qt安装时自带的MinGW

```
C:\Qt\5.15.2\mingw81_32\bin
C:\Qt\Tools\mingw810_32\bin
```


- VSCode插件 ，下面C/C++插件是必需的没意见吧

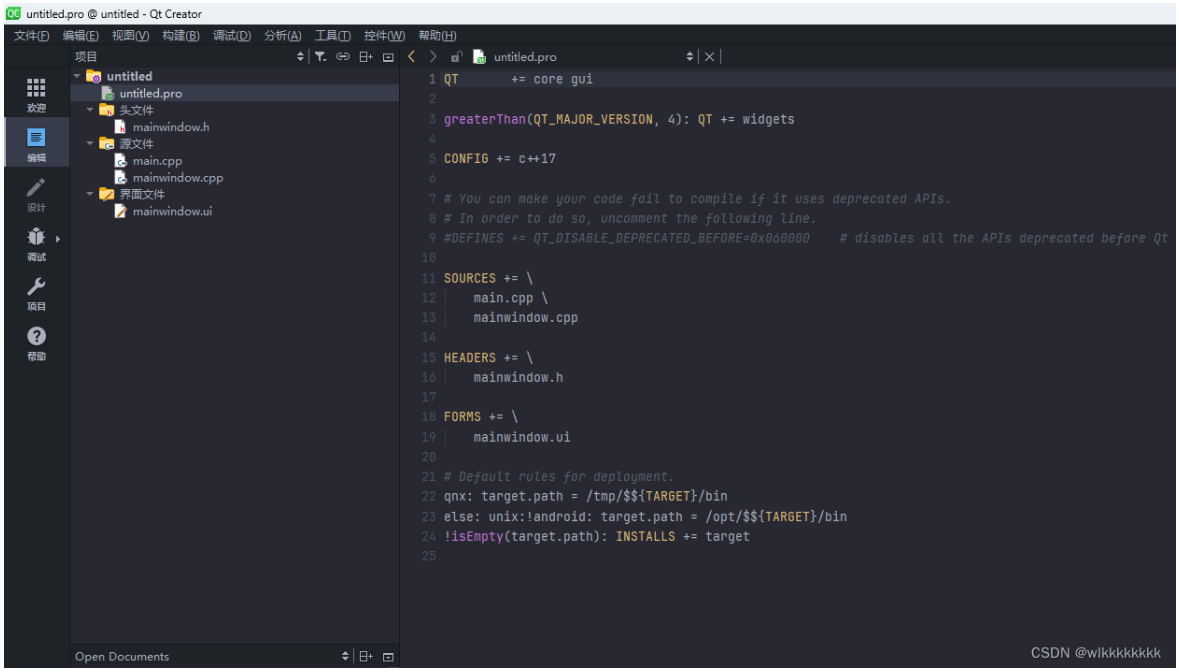




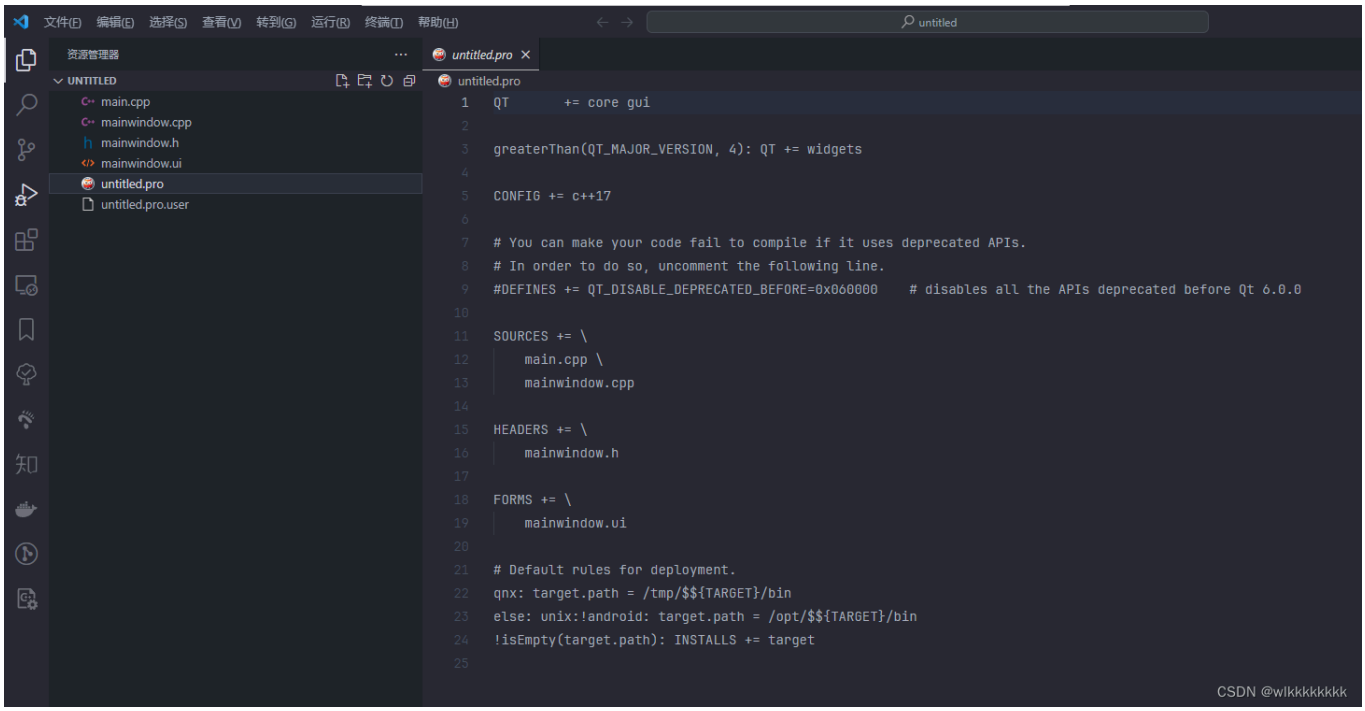
02. 开始配置

1. 创建项目项目

先用Qt Creator创建个Qmake项目，最简单的空白窗口，项目结构如下



2. 使用VSCode打开项目



3. 配置C/C++插件

VSCode快捷键 `ctrl+shift+p` 打开命令面板，输入 `c++`，选择编辑配置

4. 编辑C/C++设置

- 编译器路径:下拉有得选就选你配置环境变量的Qt版本，没有就自己复制路径过来

- IntelliSense 模式：选择安装的gcc的架构，我安装的是64位Qt，上面自带的gcc编译器也是64位，就选择 `windows-gcc-x64`

- 头文件路径：这个主要是实现头文件识别，要不然Qt的头文件一直飘红，也没法自动跳转头文件，第一行是当前像目录下所有，第二行是安装的Qt库的头文件路径

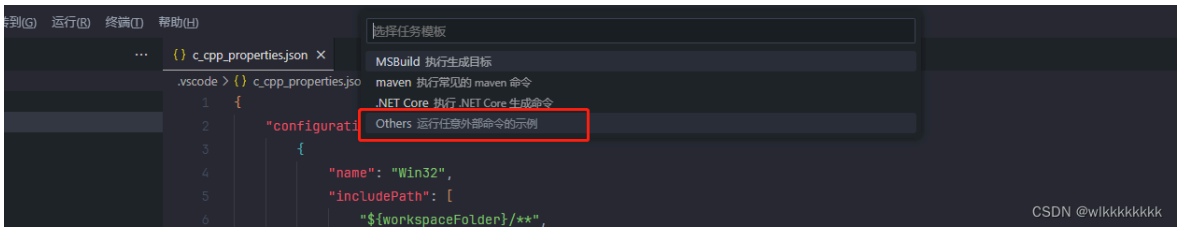
- 剩下的配置按需配置，不配置也不影响。选择编辑json可以查看C/C++插件json版本的配置

刚才配置的都在这里面了，现在引用Qt头文件已经不报错了 c_cpp_properties.json

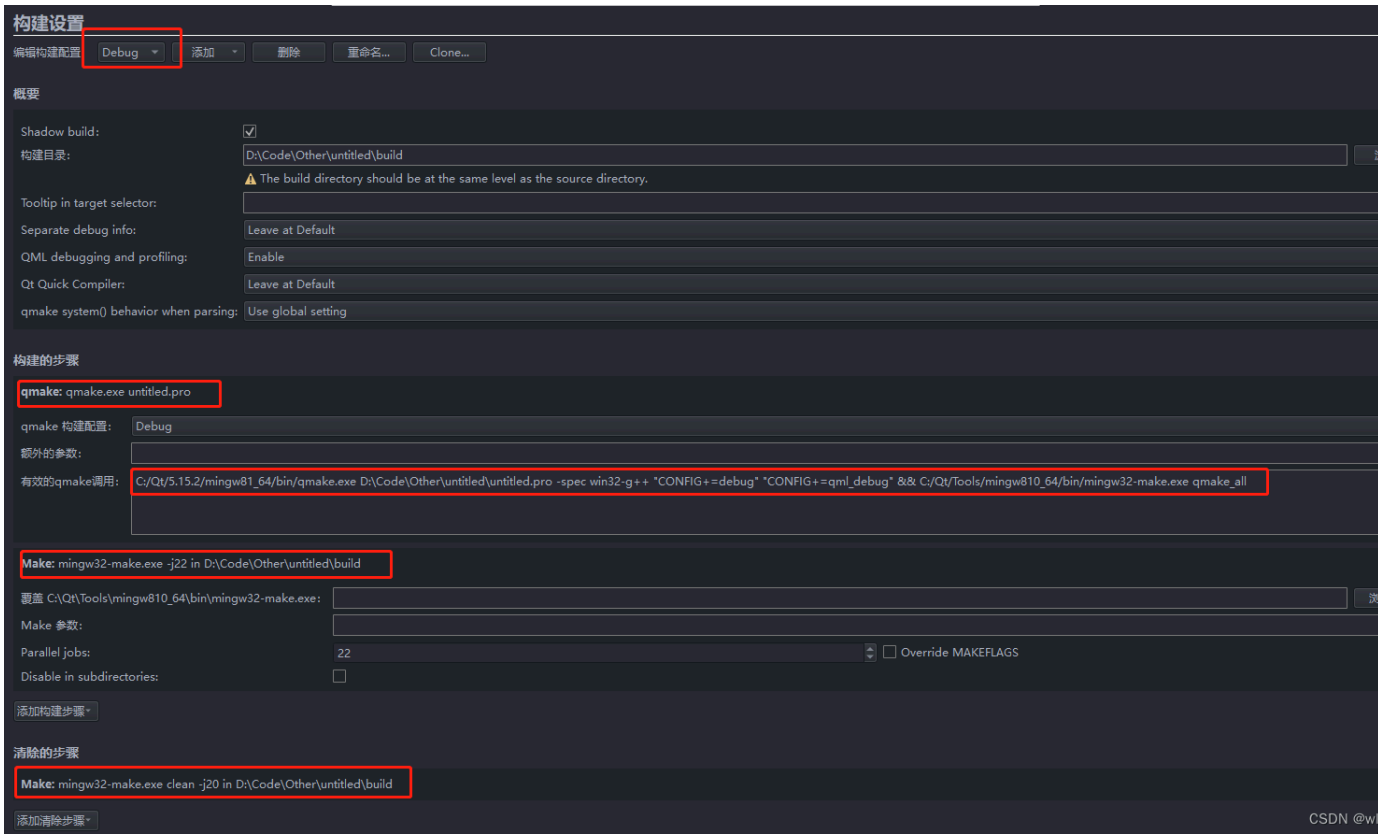
```
{
    "configurations": [
        {
            "name": "Win32",
            "includePath": [
                "${workspaceFolder}/**",
                "C:/Qt/5.15.2/mingw81_64/include/**"
            ],
            "defines": [
                "_DEBUG",
                "UNICODE"
```

5. 配置task.json

- 选择**终端-运行任务-配置任务-使用模板创建task.json文件-Others**；这一步无所谓，就是搞个模板，用我下面贴的一样



- 配置编译Qt项目的task，这一步其实是跟Qt Creator相同的，我们打开Qt Creator，选择项目，查看刚才的项目的项目构建配置



Qt Creator构建步骤分析如下：

- 设置构建目录，也就是编译出来的中间文件目录
- `qmake`，这一步其实是用 `qmake` 将 `.pro` 配置文件编译成 `makefile`，并且将其中涉及的.ui、.qrc等编译成cpp，都在构建目录中，下面是切换Debug/Release时不同的qmake编译命令

```
#Debug
C:/Qt/5.15.2/mingw81_64/bin/qmake.exe D:\Code\Other\untitled\untitled.pro -spec win32-g++ "CONFIG+=debug" "CONFIG+=qml_debug" && C:/Qt/Tools/mingw810_64/bin/mingw32

#Release
C:/Qt/5.15.2/mingw81_64/bin/qmake.exe D:\Code\Other\untitled\untitled.pro -spec win32-g++ "CONFIG+=qml_debug" && C:/Qt/Tools/mingw810_64/bin/mingw32-make.exe qmake_
```

- `make`，真正的编译cpp，MinGW使用的式 `mingw32-make.exe`

```
mingw32-make.exe -j22 in D:\Code\Other\untitled\build
```

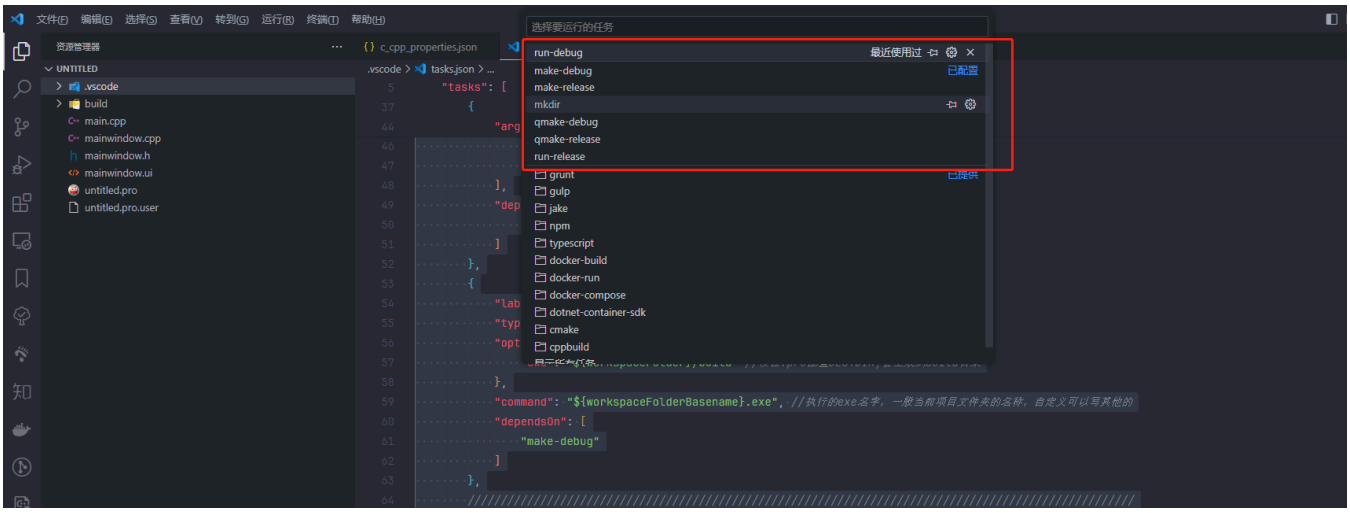
- `clean`，这里使用还是 `mingw32-make.exe`

```
mingw32-make.exe clean -j20 in D:\Code\Other\untitled\build
```

- 将上面Qt Creator构建步骤转换为 `task.json` 就行了，我把debug、relese全流程都加上了，非常简单！

```
{
    // See https://go.microsoft.com/fwlink/?LinkId=733558
    // for the documentation about the tasks.json format
    "version": "2.0.0",
    "tasks": [
        {
            //在当前项目目录创建build文件夹
            "label": "mkdir", //任务名称
            "type": "shell", //任务类型，定义任务是被作为进程运行还是在 shell 中作为命令运行。
            "options": {
                "cwd": "${workspaceFolder}" //已执行程序或脚本的当前工作目录。设置当前项目文件夹
```

- 既然配置好了，那就可以开心的运行代码，有两种方式
  - **终端-运行任务-选择任务**，我们配置的task都在里面了，选择 `run-debug/run-release` 就行，因为配置过 `depend[]`，前面qmake、make都被一条龙调用



- 既然是VSCode，不用快捷键怎么可以，超级强大的 `ctrl+shift+p` ，输入 `run` ，和手点流程一样，回车，方向键选择 `run-debug/run-release` 回车执行就好了

一条龙调用的命令在**控制台**就会有输出了，exe启动后 `log` 也会在这输出

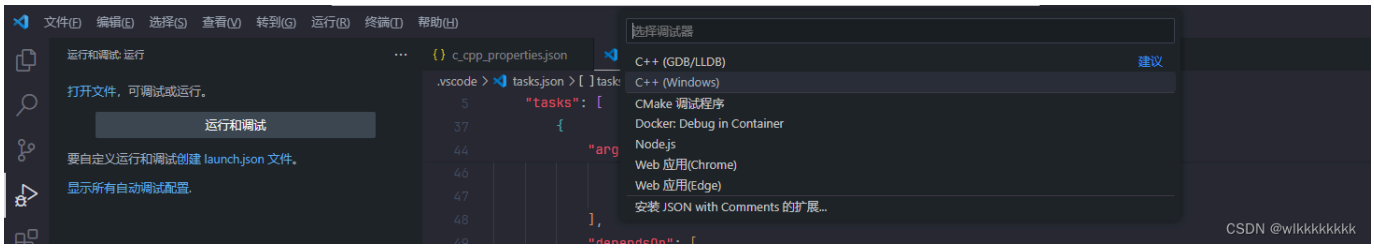
03. 配置断点调试

上面配置的是以 `debug/release` 方式运行程序，那么要打断点调试怎么办呢?

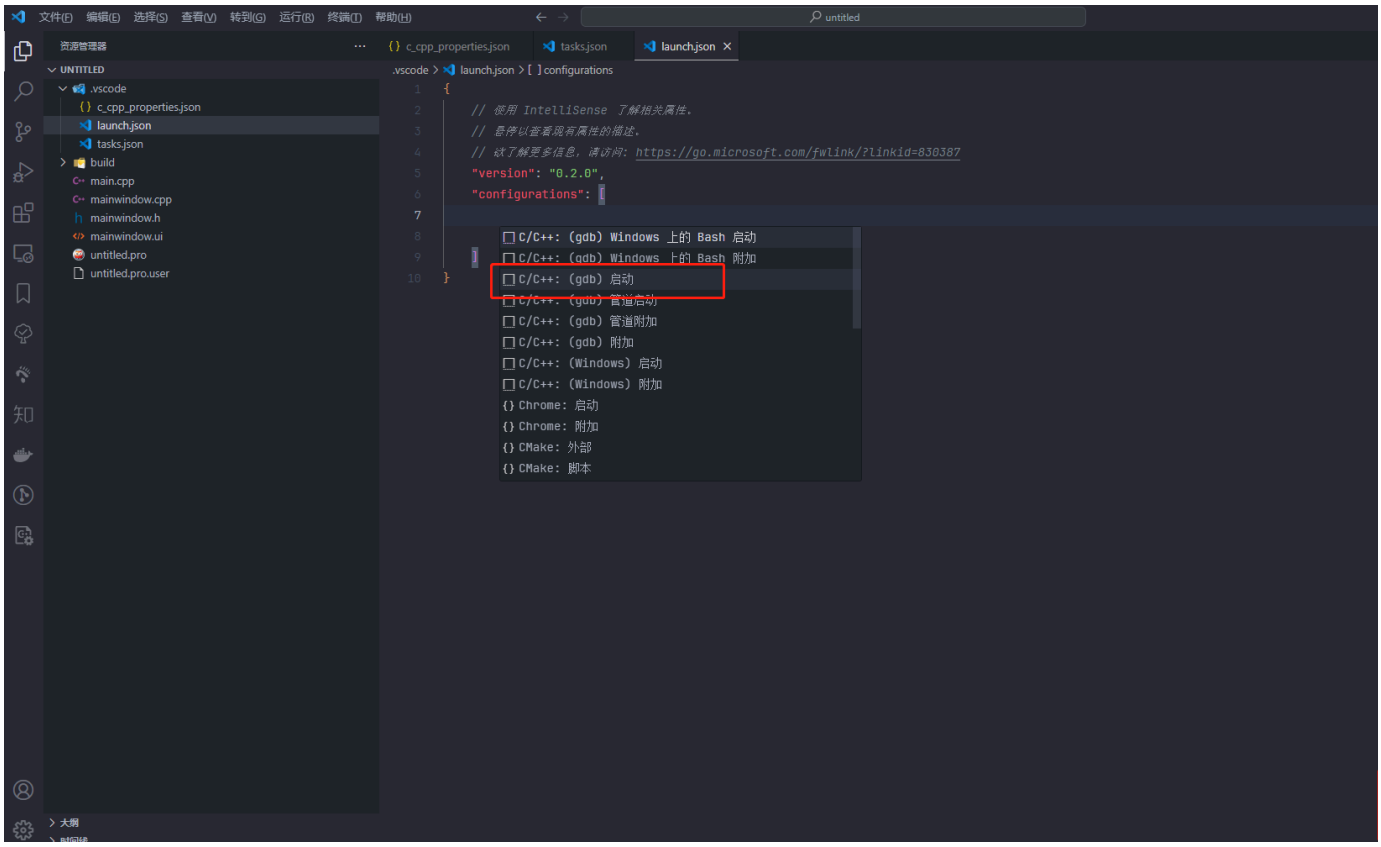
1.配置launch.json

- 侧边栏选择调试，点击创建 `launch.json` 文件

- 选择c++就行，第一个第二个没关系，反正生成的几乎空白的模板



- 选择创建配置，选择**gdb启动**作为基础模板，来在上面修改

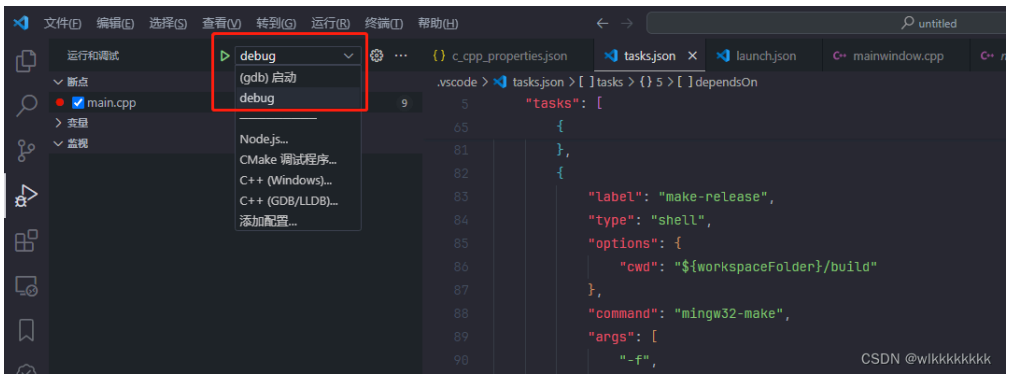


- 修改内容如下，我放了自动生成的模板跟修改后的，可以对比看下修改了那些地方，实际使用记得删除哈

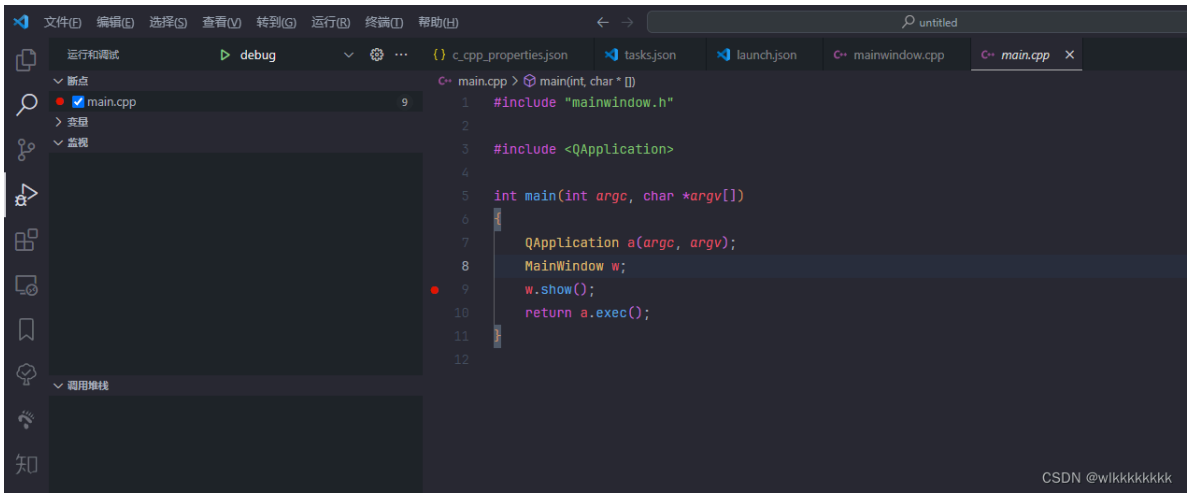
```
{
    // 使用 IntelliSense 了解相关属性。
    // 悬停以查看现有属性的描述。
    // 欲了解更多信息，请访问: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "name": "(gdb) 启动", //模板，实际使用记得删除
            "type": "cppdbg",
            "request": "launch",
            "program": "输入程序名称，例如 ${workspaceFolder}/a.exe"
```

2. 运行调试

- 重新打开调试，刚才配置的调试配置已经存在了

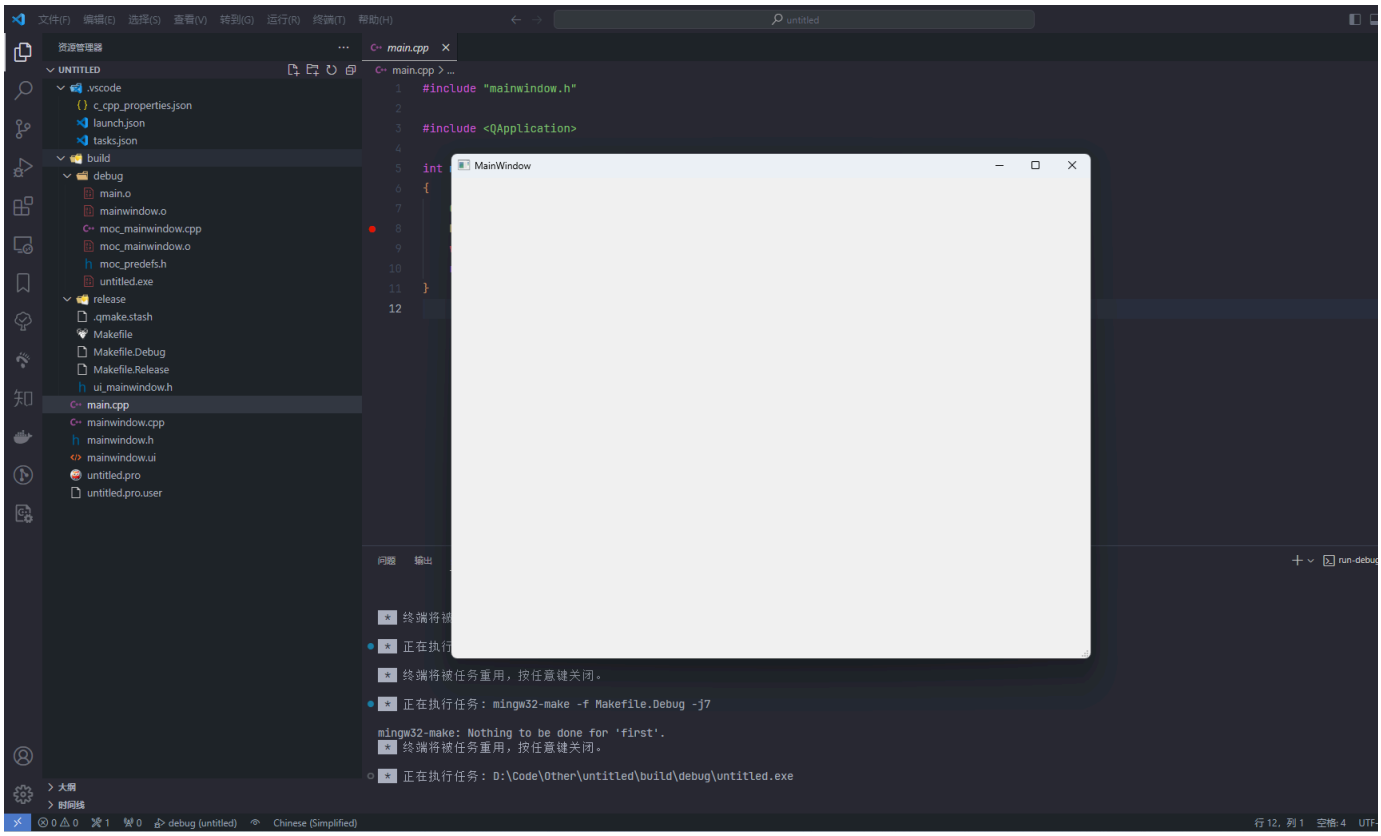


- 打个断点，点击绿色三角，运行下看看效果

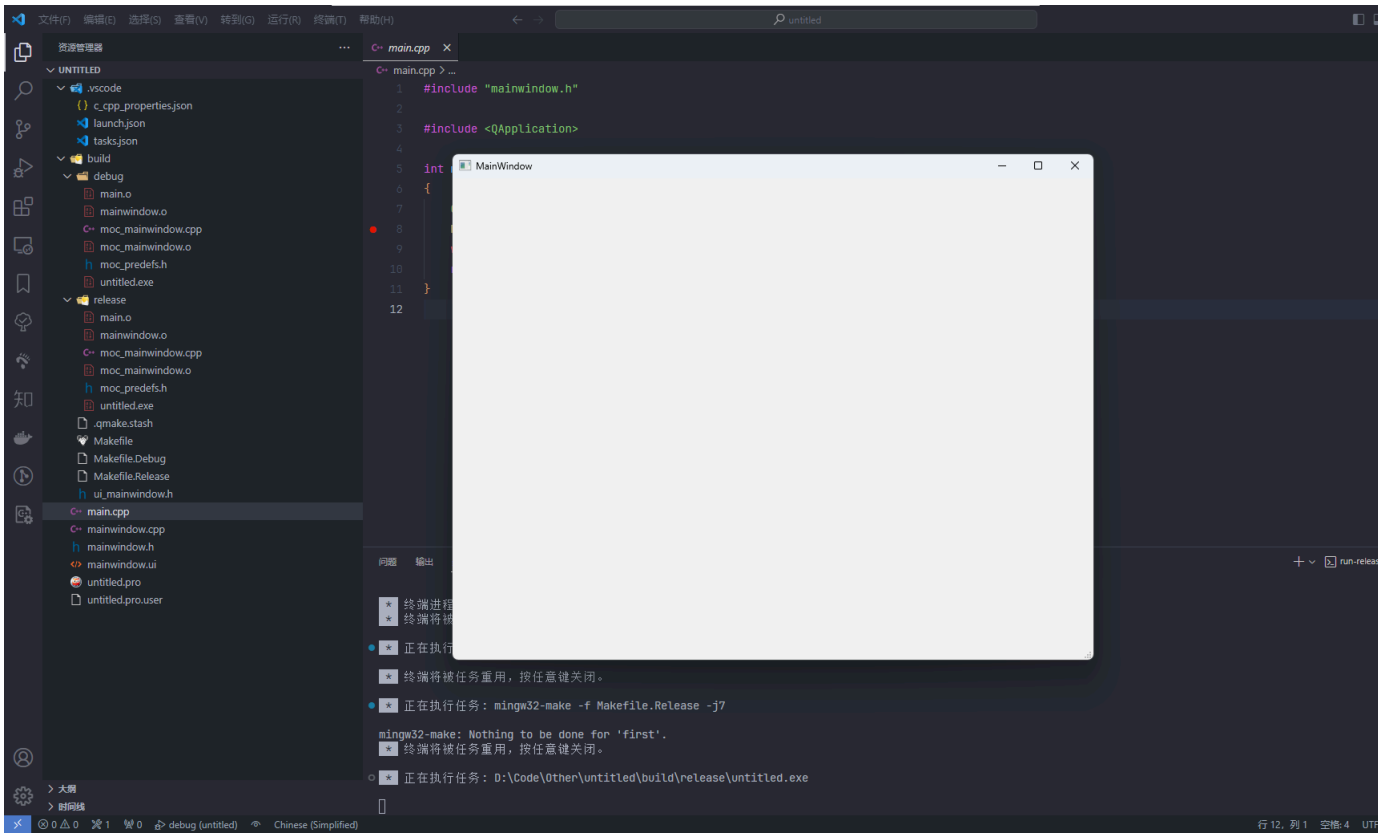


04. 运行效果

- run-debug 及自动生成的目录结构、控制台log，ctrl+c结束程序



- run-release



- **debug断点调试**