

- C/C++
 - DevC++: 初学者的好朋友
 - Code::Blocks: 开源的全平台IDE
 - TDM-GCC: Windows平台比较好用的GCC
 - CLion: JetBrains出品的C/C++ IDE(个人强推)
 - VSCode——配合C/C++及Code Runner插件
 - VSCode——配合WSL2以及Remote WSL插件
 - Visual Studio
 - 解决方法一 - 单文件添加宏
 - 解决方法二 - 整个项目
- Python
 - 下载并安装Python
 - 通过Python官网下载
 - 通过其他镜像源下载
 - 通过Windows应用商店安装
 - pip切换为国内源
 - IDE——PyCharm
- Java
 - Java误区——必须添加环境变量
 - Java多版本问题
- Node.js
 - NVS - Node Version Switcher——Node版本管理
 - 直接安装Node.js
 - NPM国内源
 - npm-check管理包
- WSL(Windows-Subsystem-Linux)
 - 什么是WSL
 - 启用WSL和虚拟机平台功能
 - 方法一: 在控制面板开启相应功能
 - 方法二: 在Powershell中开启相应功能
 - 升级到WSL2
 - 安装Linux发行版
 - 通过其他方式登录到WSL
 - Windows Terminal
 - VS Code - Remote WSL
 - 配置Ubuntu
 - 修改为国内镜像源

- [Shell: zsh+ohmyzsh](#)
- [Cygwin](#)
 - [安装Cygwin](#)
 - [更换Shell为zsh+ohmyzsh](#)
 - [安装zsh](#)
 - [使用安装程序的GUI安装](#)
 - [使用安装程序的CLI安装](#)
 - [使用apt-cyg安装](#)
 - [设置zsh为默认shell](#)
 - [安装oh-my-zsh](#)
 - [下载zip包安装oh-my-zsh](#)
 - [使用Gitee安装oh-my-zsh](#)
 - [修改.zshrc\(可选\)](#)
 - [修改主题](#)
 - [去除行尾百分号](#)
 - [在Windows Terminal中添加Cygwin](#)

title: 开发环境搭建-Windows系统 author: LeoK77 categories: ComputerScience tags:

- Windows
- ComputerScience
- Cygwin
- WSL
- Linux description: ComputerScience/开发环境搭建-Windows系统 cover: /img/Windows_Logo.svg date: 2021-02-07 11:48:10 updated: 2022-01-04 11:18:14

C/C++

由于各种视频教程、相关书籍中的编译器多选择GCC，且Linux下的C/C++一般也是用GCC，为了满足跟着教材写就能运行的目的，所以这里建议用的环境是GCC(Linux-gcc、Cygwin-gcc、mingw64、TDM-GCC等)；如果唯独认准了Visual Studio作为IDE，可以看这一部分的最后，那里会阐述如何在Visual Studio下写C项目而不被诸如printf_s之类的MSVC自己独特的类型检查所烦恼。

DevC++：初学者的好朋友

初学者的需求不是花里胡哨的环境或各种配置之后才能跑一个不大不小的项目，所以我认为初学者所需要的是所见即所得，先啥都不配置，直接就是安装，然后写完一个C/C++源程序就可以一键编译，而恰好DevC++满足这几点。

1. 自 2011 年的 Dev-C++ 4.9.9.3 版本之后，你使用的版本均为 Orwell Dev-C++。目前最新版本为 2015 年 4 月 27 日的 Dev-C++ 5.11 版本，可于 [SourceForge](#) 下载。
2. 2020 年，Embarcadero 赞助并接手了原有的 Bloodshed Dev-C++ 项目，继续开发。项目地址位于 [GitHub](#) 和 [SourceForge](#)。
3. 在 2015 年停止更新后，因为教学需要，一位来自中国的个人开发者 [royqh1979](#) 决定继续开发他的 Dev-C++ 个人分支，命名为小熊猫 Dev-C++，集成了智能提示和高版本的 MINGW64，非常便于国内的个人使用和学习，项目官网位于 [小熊猫 Dev-C++](#)，源码地址位于 [Github](#)。

以上三种皆属于开盖即用的DevC++，不必纠结于哪个更好，因为在我看来DevC++是初学时期用的，后面做稍微大一些的项目的时候可以搭建更方便的环境。

Code::Blocks：开源的全平台IDE

Code::Blocks是一个开放源码的全功能的跨平台C/C++集成开发环境。Code::Blocks是开放源码软件。Code::Blocks由纯粹的C++语言开发完成，它使用了著名的图形界面库wxWidgets(3.x)版。对于追求完美的C++程序员，再也不必忍受Eclipse的缓慢。

以上内容摘自CodeBlocks百度百科：

<https://baike.baidu.com/item/Code::Blocks/C&C++/2131200>

硬挑缺点的话那就是他没有中文版(有民间汉化，但是使用民间汉化之后有时会导致代码自动补全功能失灵的状况)，很多人推荐的IDE，我们学校机房电脑上默认的IDE就有他。

Code::Blocks官网：<http://www.codeblocks.org/>

TDM-GCC：Windows平台比较好用的GCC

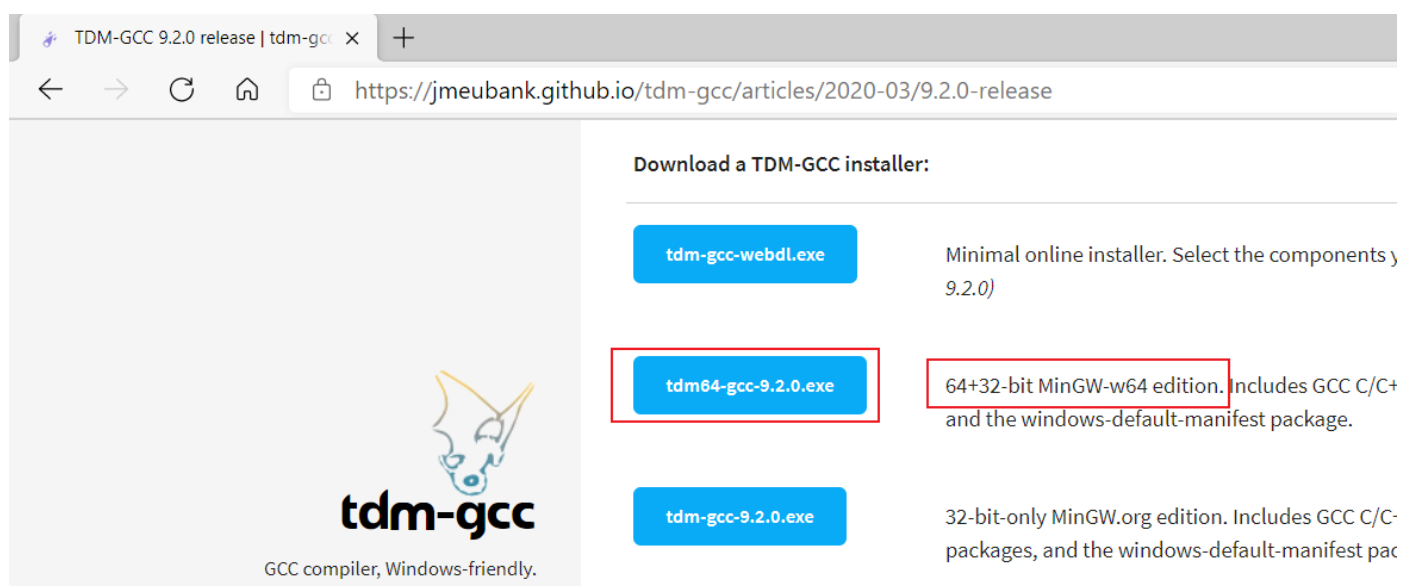
Windows系统下没有原生的GCC，所以需要MinGW、MinGW64、TDM-GCC之类的环境；除此之外还可以使用WSL2在Linux下开发(可配合VSCode+Remote WSL拓展)、使用Cygwin添加gcc组件开发(可配合CLion)，这两种使用方法有些许进阶性质，这里不详细展示 这里以TDM-GCC为例，主要是TDM-GCC的安装简单粗暴

且没啥幺蛾子，并不是说MinGW(有些老)、MinGW64(托管在sourceforge让人眼花缭乱，不能一眼让人搞明白咋下载安装)不好，萝卜咸菜各有所爱。

TDM-GCC is a compiler suite for Microsoft Windows. It is a commonly recommended compiler in many books, both for beginners and more experienced programmers. 译文：TDM-GCC是一款适用于微软Windows系统的编译器。他在很多书中被广泛提及，对于初学者和进阶的程序员都十分的友好。

以上内容摘自TDM-GCC Wiki: <https://en.wikipedia.org/wiki/TDM-GCC>

下载TDM-GCC直接去他的官网，点击官网首页的发行版，例如现在最新的发行版是“TDM-GCC 9.2.0 release”，点击之后可以看到下载按钮，选择“tdm64-gcc-9.2.0.exe”进行下载。



TDM-GCC 9.2.0 release | tdm-gcc x +

← → ↻ 🏠 🔒 <https://jmeubank.github.io/tdm-gcc/articles/2020-03/9.2.0-release>

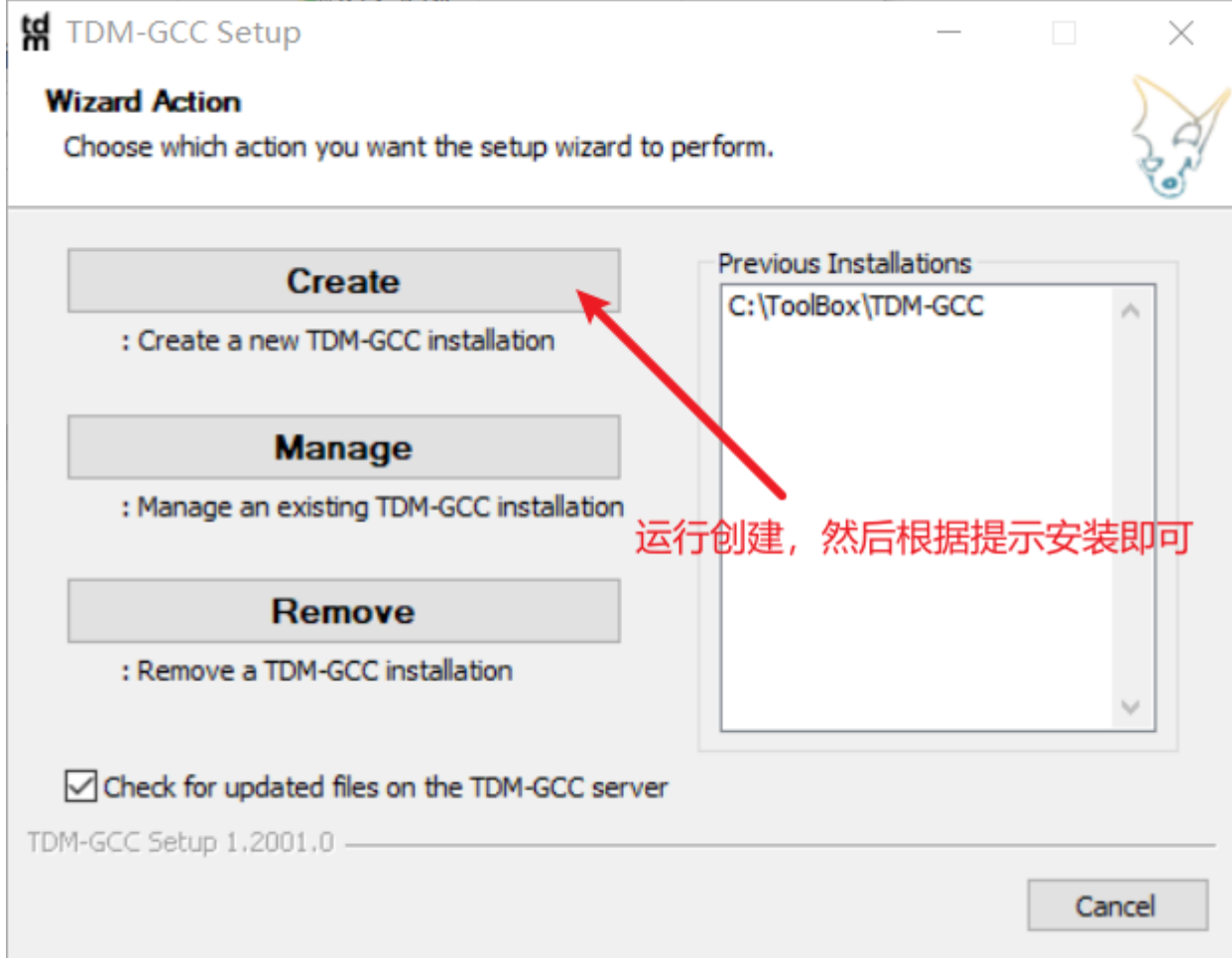
Download a TDM-GCC installer:

tdm-gcc-webdl.exe	Minimal online installer. Select the components y 9.2.0)
tdm64-gcc-9.2.0.exe	64+32-bit MinGW-w64 edition. Includes GCC C/C+ and the windows-default-manifest package.
tdm-gcc-9.2.0.exe	32-bit-only MinGW.org edition. Includes GCC C/C- packages, and the windows-default-manifest pac

tdm-gcc
GCC compiler, Windows-friendly.

TDM-GCC官网: <https://jmeubank.github.io/tdm-gcc/>

下载下来的“tdm64-gcc-9.2.0.exe”本质上是一个管理器，你可以使用它管理电脑中的TDM-GCC。



而且TDM-GCC会自动配置环境变量，在cmd下输入"gcc --version"，有如图所示内容就说明安装成功了。

```
Windows PowerShell
PS C:\Users\LeoK77> gcc --version
gcc.exe (tdm64-1) 9.2.0
Copyright (C) 2019 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
PS C:\Users\LeoK77> |
```

CLion: JetBrains出品的C/C++ IDE(个人强推)

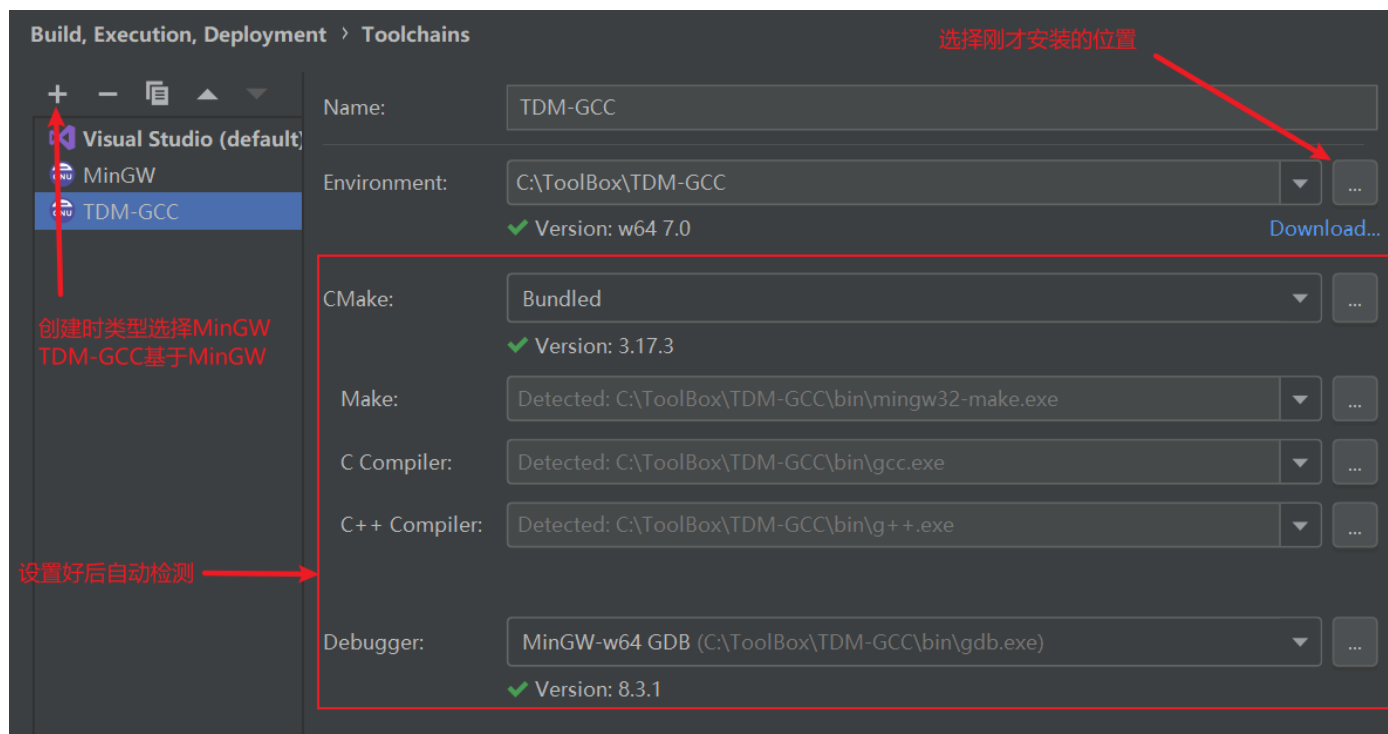
CLion本身不带编译器，支持MinGW(MinGW、MinGW64、TDM-GCC(个人推荐))、Visual Studio(MSVC)、CygWin(添加GCC编译器、GDB调试器、CMake项目管理)等工具链，需要单独安装。

如果你使用过IDEA(Java)或者PyCharm(Python)，那么你一定知道JetBrains(不知道的话我就当你知道了，总之就是一个公司的同系列产品)，CLion就是JetBrains旗下的一款专为C/C++开发的IDE，快捷键与IDEA基本一致，代码补全功能可以用“丧心病狂”来形容

容，十分的强大，至少我习惯了JetBrains家的IDE之后，用其他的环境写代码没有代码补全的情况下我感觉十分难受。

只可惜这款IDE不是免费的，且没有社区版，但是不要气馁，对于学生用户可以申请免费许可证(需要有教育邮箱，高校学生咨询学长或者老师，我们学校是一入校就分配了教育邮箱的；现在还需要身份证明，学信网的网页身份认证即可)，只要用他写的程序不是用于商业用途，就可以随便用。

配置工具链：(Customize->All settings->)settings->Build->Toolchains，然后如图所示(虽然截图时我默认VS是帮别人debug受苦，我一般用Cygwin)：



JetBrains官网: <https://www.jetbrains.com/> CLion:

<https://www.jetbrains.com/clion/> JetBrains学生认证:

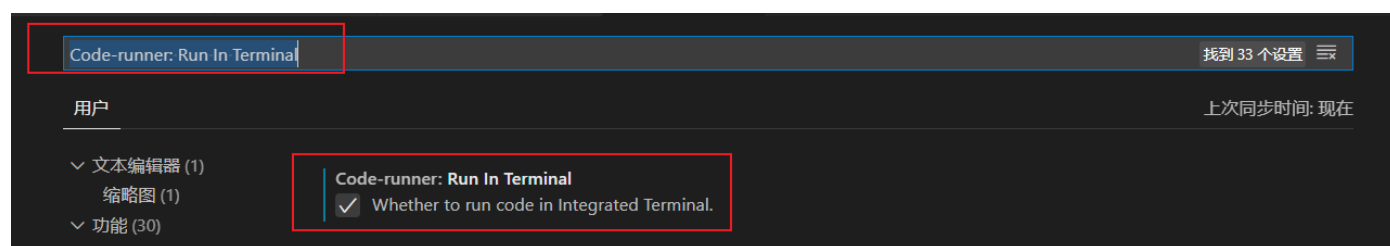
<https://www.jetbrains.com/zh-cn/community/education/#students>

VSCode——配合C/C++及Code Runner插件

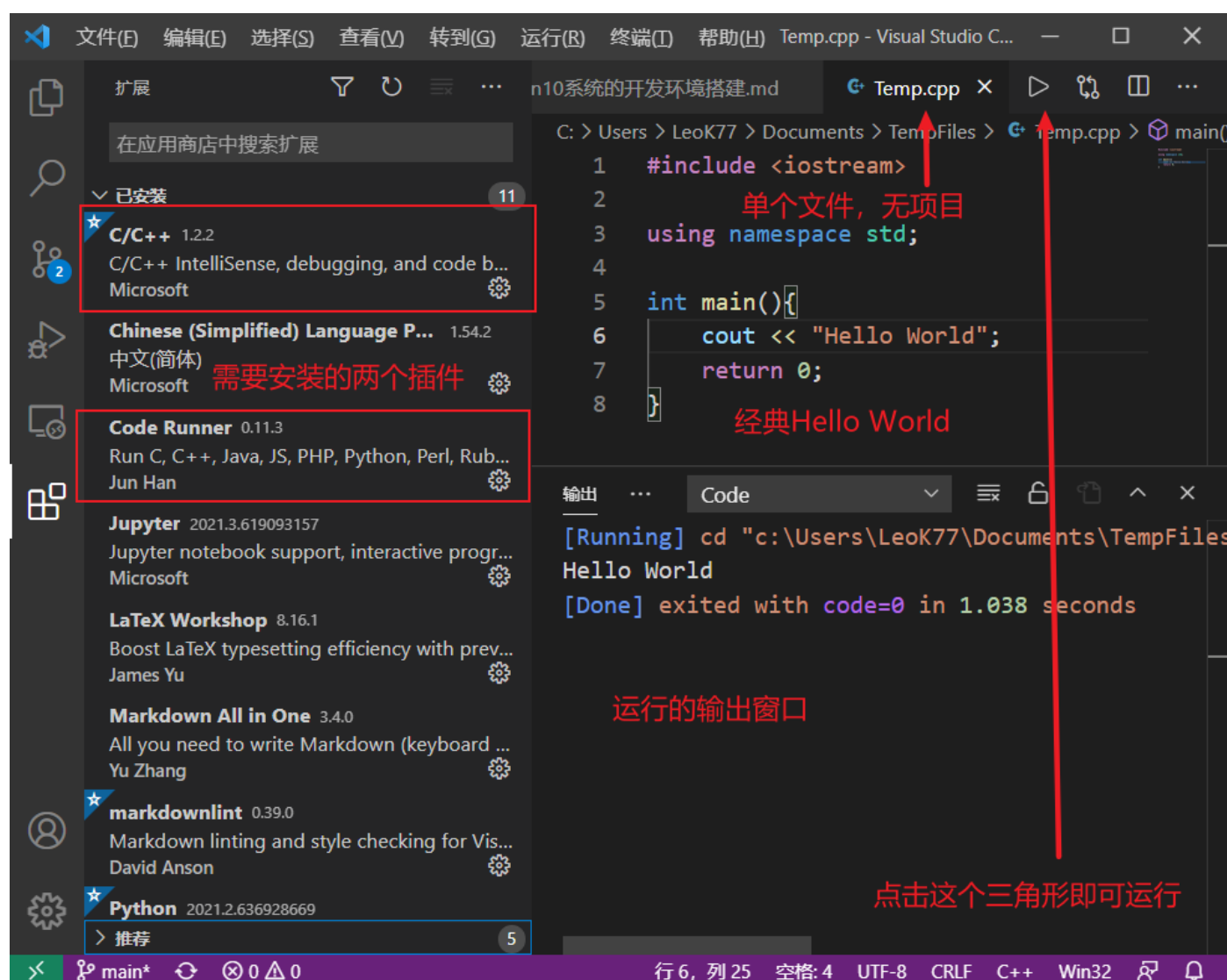
严格意义上来说VSCode似乎算不上一个IDE，但是它安装插件之后又好像是个IDE，但对于我(目前)来说他本质上只是一个文本编辑器以及SSH工具，当且仅当我实在是不想建个项目或者懒得打开CLion的时候我会用VSCode跑一些测试代码。

我知道有很多大佬会教如何详细的配置VSCode如何成为一个特别好用的C/C++IDE，但是我只是浅尝辄止，所以我只安装C/C++、Code Runner插件以及装好TDM-GCC就够了，当然我这样做不适合写项目，但是我写项目还真不用VSCode。

在设置里配置好"在终端运行"(先按下"**Ctrl+,"**，然后输入"**Code-runner: Run In Terminal**"，将如图所示的地方打钩)，这个选项默认是**false**，如果不启用的话那么在运行程序时是无法输入内容的，因为默认是如下面第二图所示在"输出窗口(只读编辑器)"下运行的。



默认在输出窗口执行程序，按照上述设置好"**Code-runner: Run In Terminal**"后就可以在终端中输入输出了，我就不再单独截图演示了。

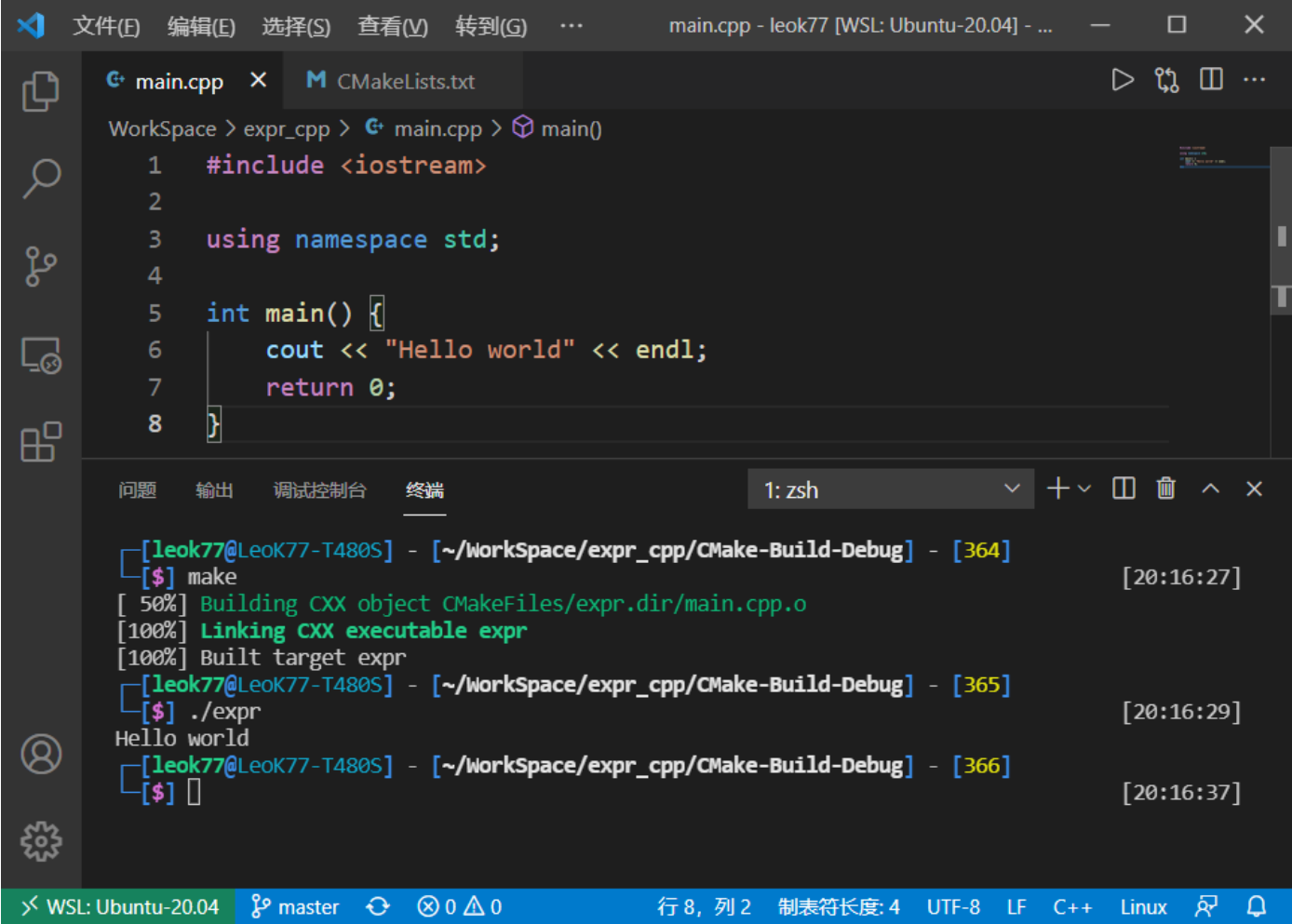


VSCode——配合WSL2以及Remote WSL插件

WSL2的安装见本文后半部分

这种情况下需要一些Linux命令基础，即调试运行等动作不再是简单的按一个按钮就可以了，包括项目也是有一个宏观的认知，而不是手动添加，虽然也可以一个文件就直接编译，但是换到这种工具链之后还是建议通过手写CMake来管理项目，而不是只有一个文件。

```
sudo apt-get install gcc gdb cmake make
```



Visual Studio

微软出品的宇宙第一IDE，Win10环境下最强大的IDE没有之一(虽然我好像只有学C#的时候一直用他)，但是有些过于前卫——默认情况下写C程序的时候，`scanf`等函数是被禁用的(有安全漏洞)，需要自己关闭安全性检查，VS2015的时候可直接在创建项目的时候关闭，后来就需要自己设置了。

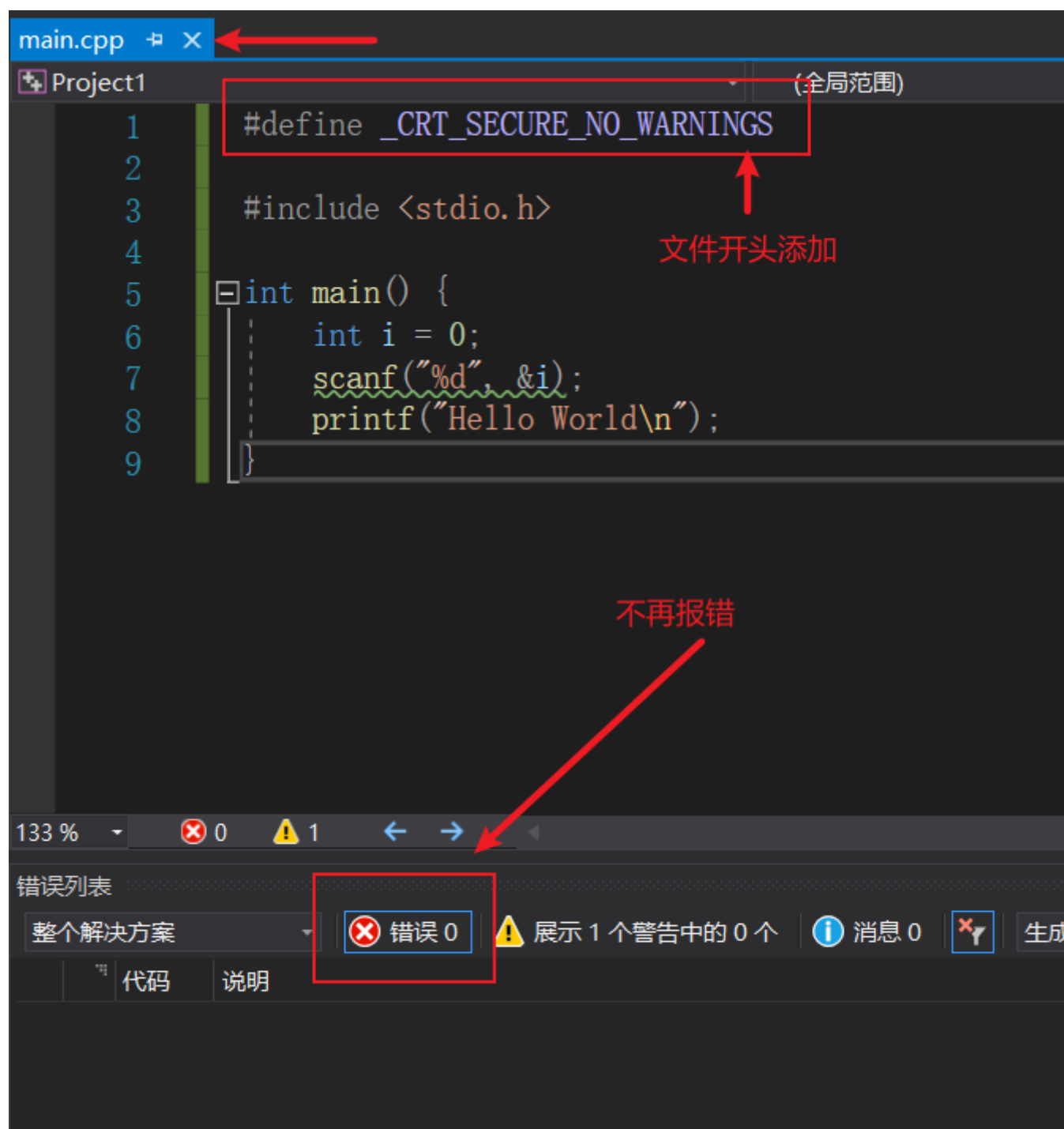
“scanf”函数报错如图所示：

	代码	说明
	C4996	'scanf': This function or variable may be unsafe. Consider using scanf_s instead. To disable deprecation, use _CRT_SECURE_NO_WARNINGS. See online help for details.

解决方法一 - 单文件添加宏

在要使用“scanf”等函数的文件首部添加如下语句：

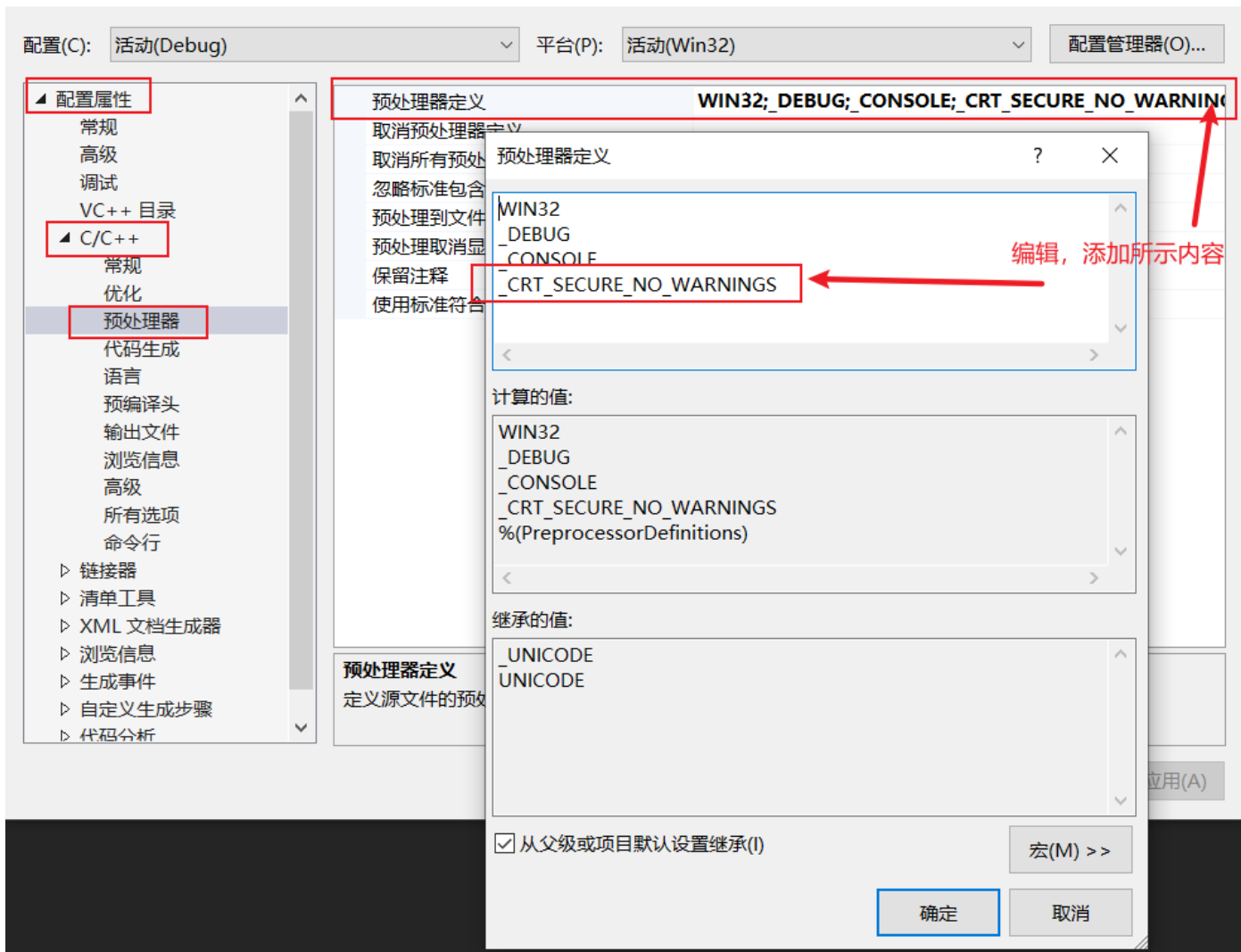
```
#define _CRT_SECURE_NO_WARNINGS
```

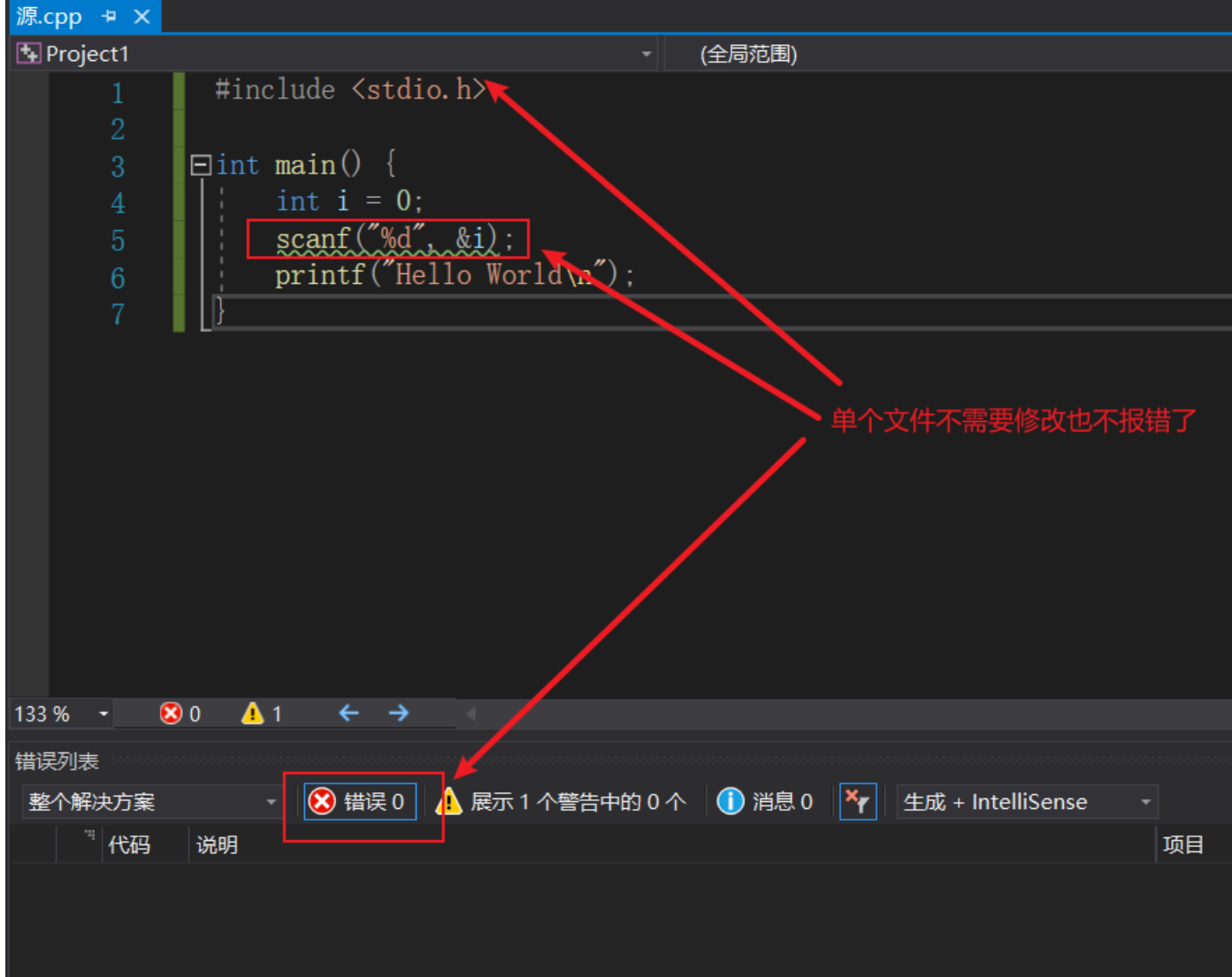


解决方法二 - 整个项目

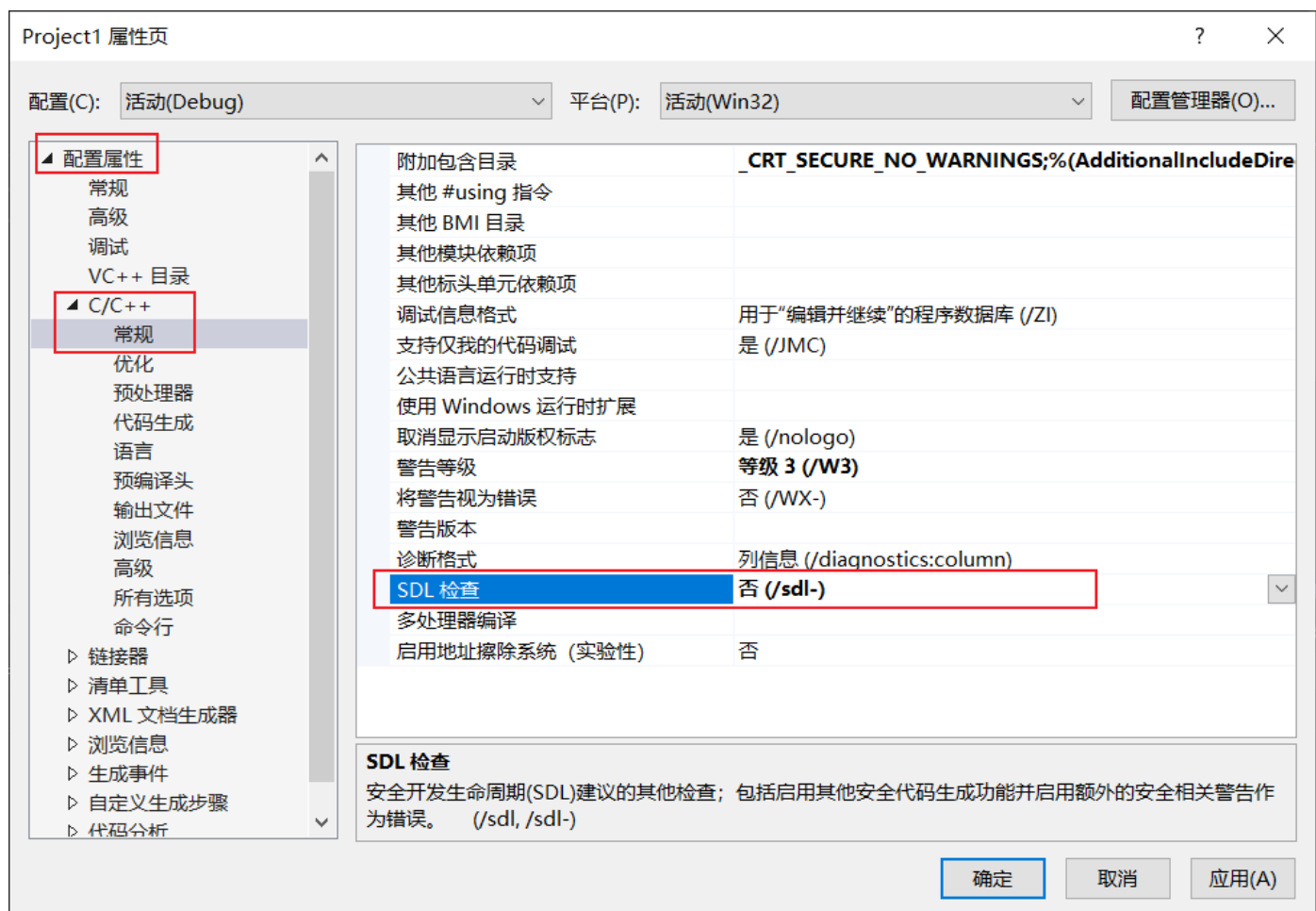
如果整个项目都是C语言项目，又不想用“scanf_s”替换“scanf”，那么可以在项目属性中添加规则，或者直接禁用SDL。

方法一、将“_CRT_SECURE_NO_WARNINGS”添加到预处理器中





方法二、直接关闭“SDL检查”



Python

Python不像C/C++，因为Python既是标准的制定者又是标准的实现者，所以Python解释器不需要像C/C++那样在GCC、Clang、MSVC等一众环境中选择，认准[Welcome to Python.org](https://www.python.org/)(Python官网)即可

下载并安装Python

下载Python可以通过微软应用商店、Python官网以及各大镜像源。我使用的是在Python官网下载。

安装的时候建议选择"Add Python to PATH"，不然还得自己手动添加到PATH



通过Python官网下载

Python - Windows: <https://www.python.org/downloads/windows/>

以Python 3.7.9 为例，我下载的时候会选择“x86_64”即64位，“executable installer”版，安装时不需要额外下载内容。

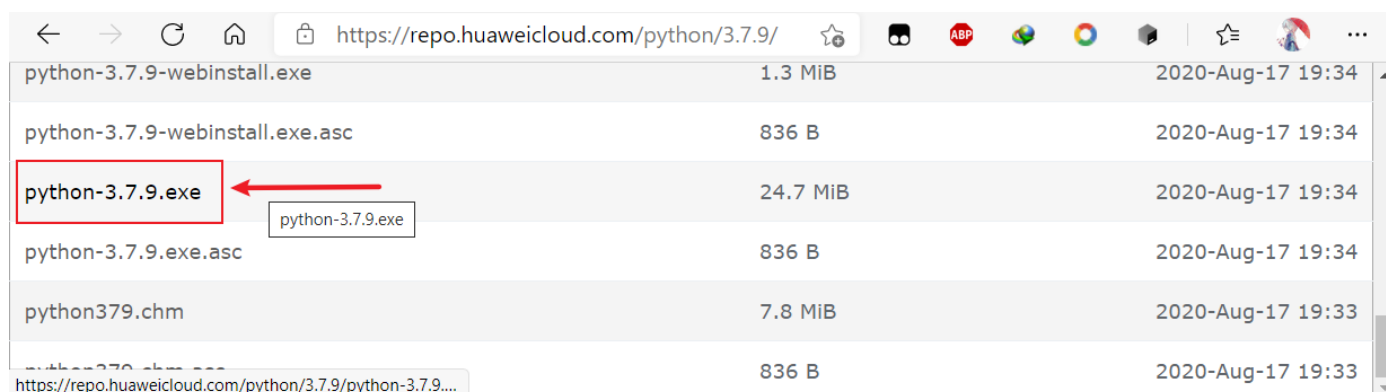
- [Python 3.7.9 - Aug. 17, 2020](#)

Note that Python 3.7.9 *cannot* be used on Windows XP or earlier.

- Download [Windows help file](#)
- Download [Windows x86-64 embeddable zip file](#)
- Download [Windows x86-64 executable installer](#)
- Download [Windows x86-64 web-based installer](#)
- Download [Windows x86 embeddable zip file](#)
- Download [Windows x86 executable installer](#)
- Download [Windows x86 web-based installer](#)

通过其他镜像源下载

以华为云为例，进入进入华为云的Python镜像源，找到要安装的Python版本，如Python3.7.9，则点击3.7.9，选择详细版本，对于华为云镜像源选择“python-3.7.9.exe”。



python-3.7.9-webinstall.exe	1.3 MiB	2020-Aug-17 19:34
python-3.7.9-webinstall.exe.asc	836 B	2020-Aug-17 19:34
python-3.7.9.exe	24.7 MiB	2020-Aug-17 19:34
python-3.7.9.exe.asc	836 B	2020-Aug-17 19:34
python379.chm	7.8 MiB	2020-Aug-17 19:33
python379.chm.asc	836 B	2020-Aug-17 19:33

华为云 - Python: <https://repo.huaweicloud.com/python/>

通过Windows应用商店安装

一般情况下，除非系统只允许安装应用商店的程序，或者实在是懒得去官网/镜像源下载，我不会从这儿下载python。



pip切换为国内源

pip源的替换并不总是最好的，因为有时候国内源可能存在更新不及时或其他问题，我使用国内源的时候偶尔会无法完成导包，所以如果自身网络环境不是十分差劲的话可以选择不替换pip源

```
#通过默认源升级pip
pip install pip -U
#通过TUNA清华源更新pip
pip install -i https://pypi.tuna.tsinghua.edu.cn/simple pip -U
#设置pip的默认源为TUNA清华源
pip config set global.index-url https://pypi.tuna.tsinghua.edu.cn/simple
#设置pip的默认源为华为云
pip config set global.index-url https://repo.huaweicloud.com/repository/pypi/simple
```

IDE——PyCharm

虽然Python自带的IDLE好像功能就够用了，但是谁会嫌弃更好用的IDE(PyCharm)呢

只需要知道PyCharm很好用就够了，而且PyCharm有可以免费使用的社区版(不涉及商业行为，学生还可以用教育邮箱申请使用专业版)，无论是项目管理还是代码补全等一系列小工具，JetBrains家的IDE永远都是首屈一指的。

Java

Java误区——必须添加环境变量

很多教程都说Win10里Java需要配置环境变量才能使用，但实际上只要没有在终端里用Java的要求，没必要加环境变量。

- 以与开发无关的一件事为例：当我在玩Minecraft Java Edition的时候，毫无疑问我是需要Java环境的，但我不一定需要在终端中使用Java
 - 客户端：作为一个玩家，我只需要在客户端中配置好Java路径即可，因为在启动的时候不需要终端的全局Java
 - 服务端：作为一个服务器的维护者，开服指令往往需要Java，其实这时候也不是必须使用全局的Java，如下面示例中的`java -version`命令示意，我既可以找到他的路径，在使用的时候打好全部的路径然后使用Java，也可以添加环境变量使用全局的Java，仅是使用命令时候的方便与否

而且实测安装jre8他会自动配置好环境变量。

当电脑中有多个Java环境的时候，只要在IDE里选对应的Java环境就好了，这是跟路径有关的，而不是“唯一性”。环境变量实际上就是省去了输入父路径的功夫，如果使用的时候输入父路径，那么就相当于有这个环境变量了。

而“电脑中只可以有一个Java”，是只可以有一个“`java.exe`”在环境变量里，如果我既把“Java8”的路径添进入了，又把“Java11”的路径填进去了，那么这两个“`java.exe`”在执行的时候就按照顺序执行，如果环境变量中Java8在前面，那么先扫描到了Java8，执行时使用的就是Java8。

比如我的电脑中，我将“`C:\Program Files (x86)\Common Files\Oracle\Java\javapath`”添加到了环境变量里，那么这个目录下的“`java.exe`”我就可以直接引用了。这时候，我是直接以“`java`”的形式调用，还是以“`& 'C:\Program Files (x86)\Common Files\Oracle\Java\javapath\java.exe'`”的形式(因为路径中有空格，所以看起来不太一样)调用这个“`java.exe`”，实际使用上是完全一致的，只是一个方便一个不方便罢了。

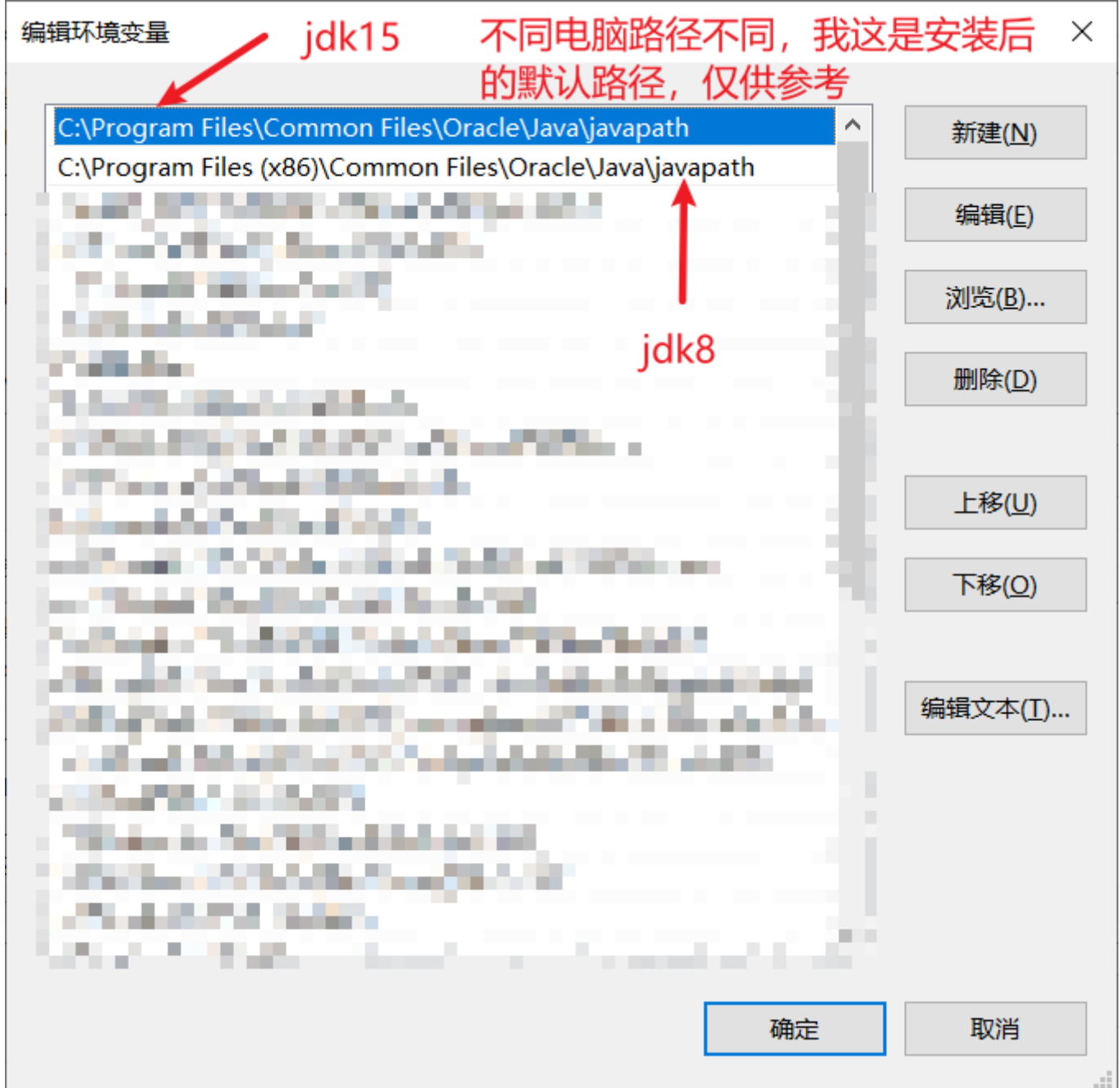

```
Windows PowerShell
PS C:\Users\LeoK77> & 'C:\Program Files (x86)\Common Files\Oracle\Java\javapath\java.exe' -version
java version "1.8.0_281"
Java(TM) SE Runtime Environment (build 1.8.0_281-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.281-b09, mixed mode)
PS C:\Users\LeoK77> java -version
java version "1.8.0_281"
Java(TM) SE Runtime Environment (build 1.8.0_281-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.281-b09, mixed mode)
PS C:\Users\LeoK77>
```

Java多版本问题

Java8: <https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html> 新版本Java: <https://www.oracle.com/java/technologies/java-se-glance.html>

前面已经提到了JDK在安装的时候是会自动添加到环境变量里的，当电脑里有多個Java环境的时候，只需要把自己想要使用的那个Java环境调动到最前面就可以了(命令行里使用"java"命令的时候是在环境变量里从头开始找，找到第一个包含"Java"的就直接用，不考虑后面的)。

比如在如图所示的情况下，我在cmd下"java -version"的结果就是jdk15的结果，而如果我吧jdk8放在第一个，那么就是默认jdk8了。



如图所示这两个路径里面的"javapath"目录下的"java.exe、javac.exe、javew.exe、jshell.exe"在我的理解下本质上是指向安装的jdk目录下的链接，所以完全可以自己建一个文件夹，然后在这个文件夹里创建自己想用的jdk的这四个应用程序的链接，然后把把这个文件夹添加到环境变量PATH的首位，这样就可以直接调用自己想用的jdk了。

Node.js

其实我本人并没有写过Node.js的项目，但是用的Hexo、PicGo等软件需要使用Node.js。而Hexo使用新版的Node.js的时候会出现一些Warning，虽然影响不大，但是使用的时候终端全是Warning着实让人恼火，所以我需要使用旧版的Node.js，以免我以后需要学习的时候环境受到影响，所以我使用NVS(Node Version Swithcer)来管理我电脑中的Node.js。

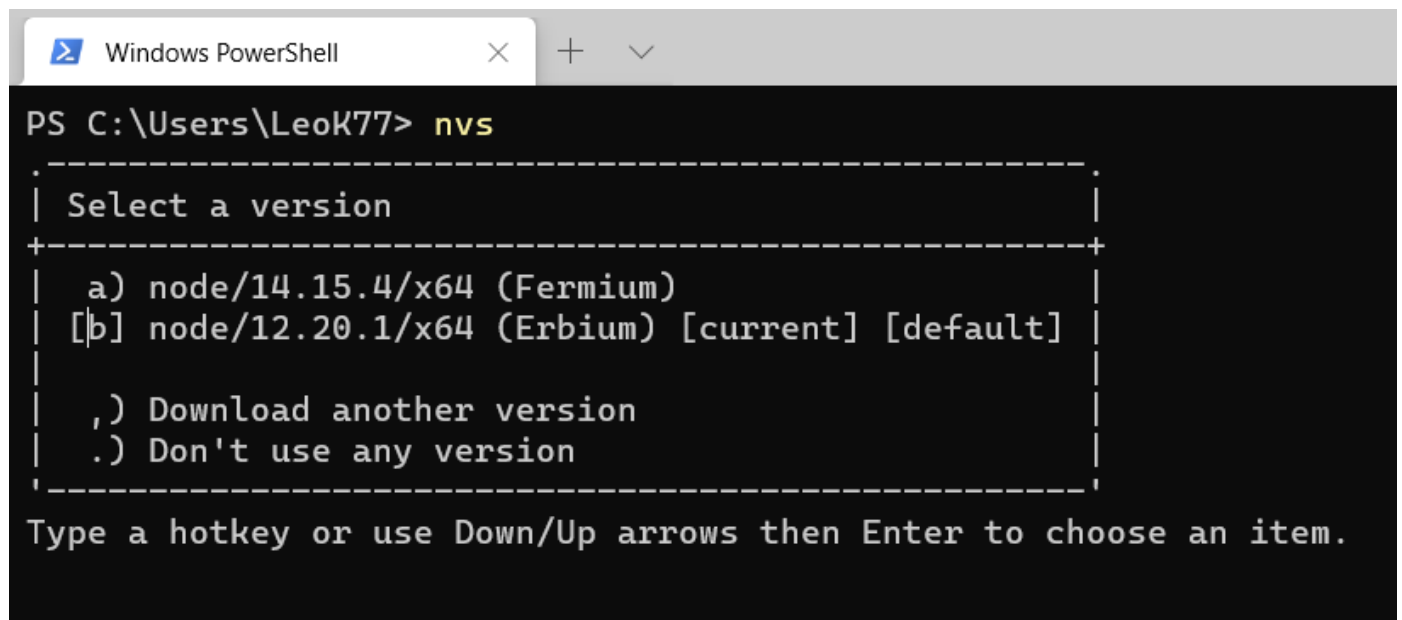
NVS - Node Version Switcher——Node版本管理

NVS是Github上的一个开源项目，允许电脑中存在多个Node.js环境，并在使用时决定选择哪个作为默认环境，下载、安装及使用方法请参考官方文档，其实不用记住太多的命令，在终端里输入“nvs”就可以以一种友好的方式通过“nvs”管理不同版本的Node.js了，但是部分命令确实是仅仅用“nvs”无法解决的。

NVS - Github: <https://github.com/jasongjin/nvs>

下面是我认为较为常用或者需要进行设置的地方：

```
nvs                #不带参数运行，有交互界面
nvs remote         #查看nvs下载时使用的仓库
nvs remote node-taobao https://npm.taobao.org/mirrors/node #添加淘宝镜像源到nvs中
nvs remote node-huawei https://repo.huaweicloud.com/nodejs/ #添加华为云镜像源到nvs中
nvs add 12         #安装Node.js 12
nvs use 12         #使用Node.js 12
nvs link 12        #将Node.js 12设置为默认的Node.js
#设置为默认的版本之后需要再nvs use 12一下
nvs ls            #查看nvs安装的Node.js版本、正在用的版本、默认的版本
```



```
Windows PowerShell
PS C:\Users\LeoK77> nvs
Select a version
+-----+
| a) node/14.15.4/x64 (Fermium) |
| [b] node/12.20.1/x64 (Erbium) [current] [default] |
| ,) Download another version |
| .) Don't use any version |
+-----+
Type a hotkey or use Down/Up arrows then Enter to choose an item.
```

直接安装Node.js

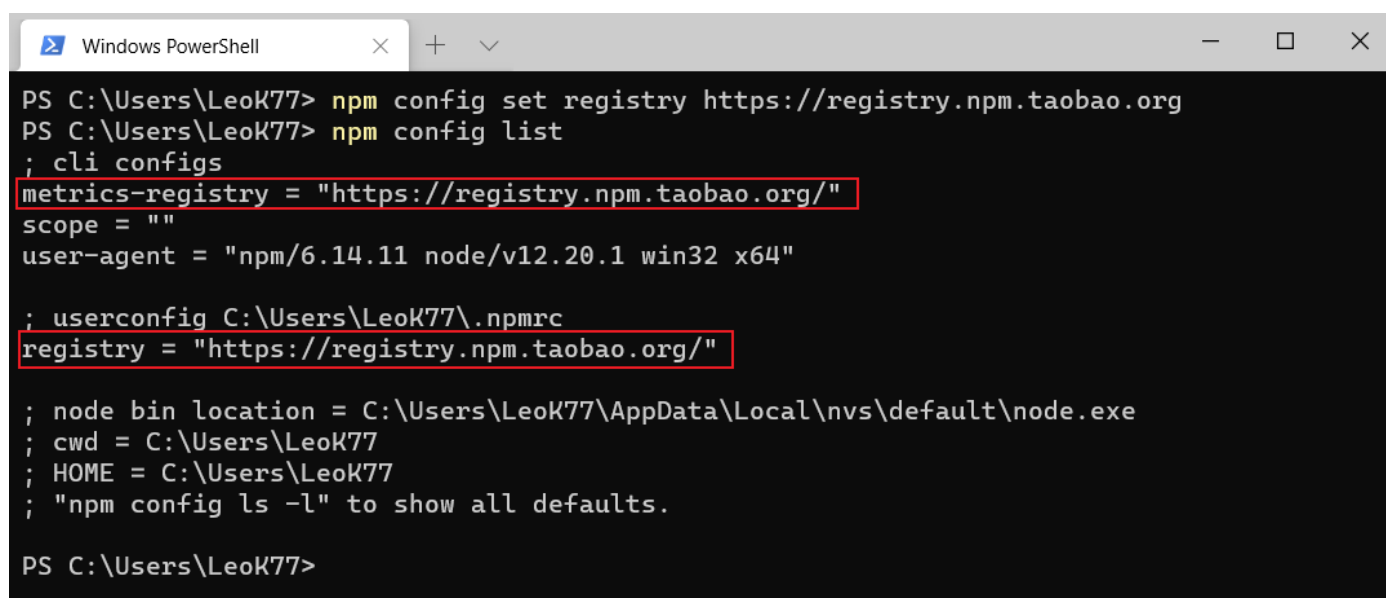
使用NVS管理Node.js的话就不需要手动下载安装包然后安装Node.js了。

Node.js官网: <https://nodejs.org> Node.js - 淘宝镜像:
<https://npm.taobao.org/mirrors/node> Node.js - 华为云:

NPM国内源

npm的默认下载仓库是<https://registry.npmjs.org>，访问速度不是那么的快，国内用户没有科学上网工具的话可以切换为淘宝仓库镜像源或者华为云镜像源等国内镜像源。

```
npm config set registry https://registry.npm.taobao.org #设置为淘宝仓库镜像源
npm config set registry https://repo.huaweicloud.com/repository/npm/ #设置为华为云仓库镜像源
npm config set registry https://registry.npmjs.org #切换回默认源
npm config list #查看设置是否成功
```



```
Windows PowerShell
PS C:\Users\Leok77> npm config set registry https://registry.npm.taobao.org
PS C:\Users\Leok77> npm config list
; cli configs
metrics-registry = "https://registry.npm.taobao.org/"
scope = ""
user-agent = "npm/6.14.11 node/v12.20.1 win32 x64"

; userconfig C:\Users\Leok77\.npmrc
registry = "https://registry.npm.taobao.org/"

; node bin location = C:\Users\Leok77\AppData\Local\nvs\default\node.exe
; cwd = C:\Users\Leok77
; HOME = C:\Users\Leok77
; "npm config ls -l" to show all defaults.

PS C:\Users\Leok77>
```

npm-check管理包

对于我来说使用Node.js时，我没了解npm怎么才能十分优雅的查看自己正在使用的包有没有可以升级的，所以我安装了npm-check用来检测包是否有可用升级。

```
npm install npm-check -g --save # 全局安装 npm-check
npm-check -u # 在使用node.js的地方进行检查
```

比如我使用的Hexo框架是需要nodejs的，那么在我的Hexo博客文件夹中，我可以使用上述命令检查是否有可升级的包，如图所示：

```
Windows PowerShell
PS C:\Users\LeoK77\Documents\WorkSpace\Blog-Hexo-LeoK77> npm-check -u
♥ Your modules look amazing. Keep up the great work. ♥
PS C:\Users\LeoK77\Documents\WorkSpace\Blog-Hexo-LeoK77> |
```

WSL(Windows-Subsystem-Linux)

有其他虚拟机需求的请看: [VirtualBox基础使用教程 | LeoK77](#) 注: 启用WSL2后, 由于使用了Hyper-V虚拟机功能, 其他的虚拟机平台如VirtualBox、VMware都将无法使用。WSL官方文档: <https://docs.microsoft.com/zh-cn/windows/wsl/install-win10>

什么是WSL

适用于 Linux 的 Windows 子系统可让开发人员按原样运行 GNU/Linux 环境 - 包括大多数命令行工具、实用工具和应用程序 - 且不会产生传统虚拟机或双启动设置开销。

- 在 Microsoft Store 中选择你偏好的 GNU/Linux 分发版。
- 运行常用的命令行软件工具（例如 grep、sed、awk）或其他 ELF-64 二进制文件。
- 运行 Bash shell 脚本和 GNU/Linux 命令行应用程序，包括：
 - 工具: vim、emacs、tmux
 - 语言: NodeJS、Javascript、Python、Ruby、C/ C++、C# 与 F#、Rust、Go 等。
 - 服务: SSHD、MySQL、Apache、lighttpd、MongoDB、PostgreSQL。
- 使用自己的 GNU/Linux 分发包管理器安装其他软件。
- 使用类似于 Unix 的命令行 shell 调用 Windows 应用程序。
- 在 Windows 上调用 GNU/Linux 应用程序。

按照我的理解, WSL本质上就是一个虚拟机(这样描述不准确), 但是相较于VMWare Workstation或者VirtualBox创建的虚拟机来说, 它更兼容Windows, 性能更好(特指WSL2), 但是不提供GUI界面, 所以如果想使用带图形界面的Linux, 还是老老实实双系统或者装一个正经的虚拟机吧。

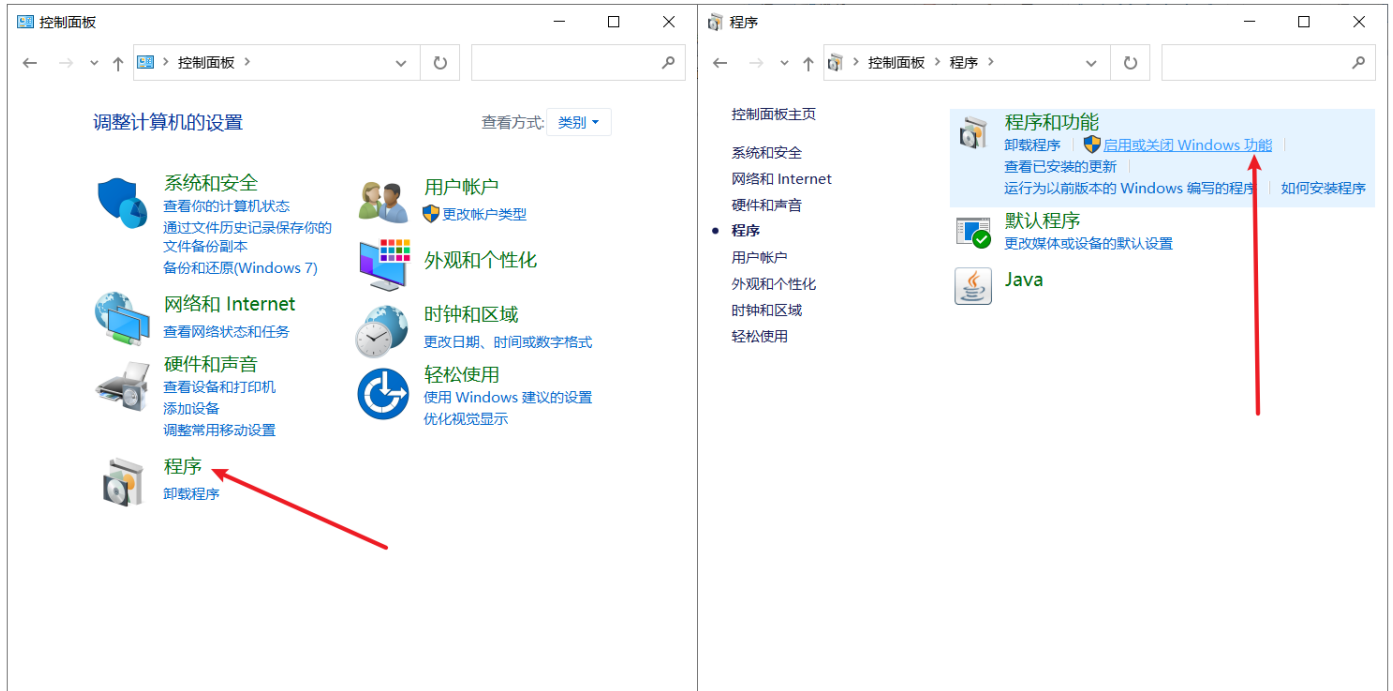
WSL和WSL2在本质上不是一个东西, 但是鄙人才疏学浅, 不是很明白其中的区别, 所以本文中所有两者比较的观点都有很强的主观性, 本人目前使用的是WSL2

启用WSL和虚拟机平台功能

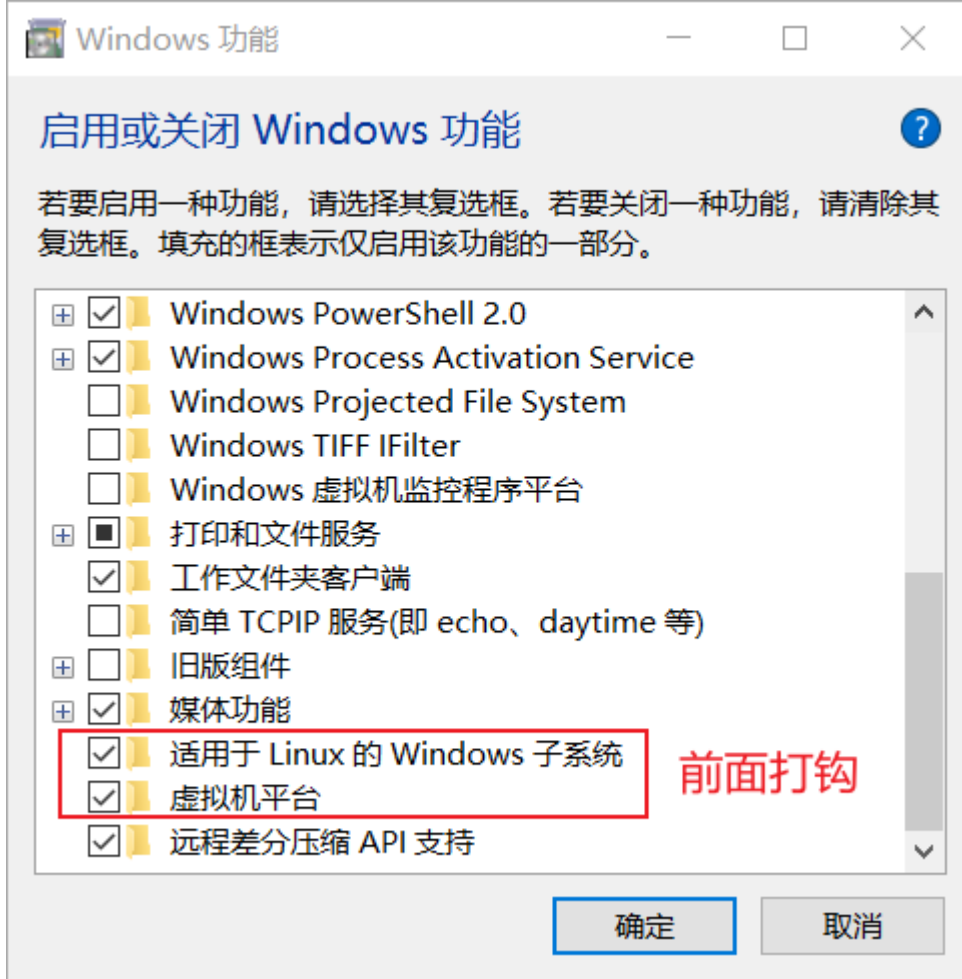
- 适用于Linux的Windows子系统：此功能必须开启
- 虚拟机平台：如果要使用WSL2的话，需开启此功能

方法一：在控制面板开启相应功能

Win+Q打开搜索框，输入"控制面板"以打开控制面板；在控制面板窗口点击"程序"，之后点击"启用或关闭Windows功能"。



在Windows功能页下拉到最下面，点击"适用于Linux的Windows子系统"和"虚拟机平台"后点击确定。



在cmd下执行如下语句，若输出开端为如图所示，则说明WSL启用成功：

```
wsl --help
```

```
PS C:\Users\LeoK77> wsl --help
版权所有 (c) Microsoft Corporation。保留所有权利。

用法: wsl.exe [参数] [选项...] [命令行]

用于运行 Linux 二进制文件的参数:

    如果未提供命令行, wsl.exe 将启动默认的 shell。

--exec, -e <命令行>
    执行指定的命令而不使用默认的 Linux shell。

--
    按原样传递剩余的命令行。
```

方法二：在Powershell中开启相应功能

以管理员身份打开Powershell并执行下述命令：(第一条是开启WSL，第二条是开启WSL2需要的虚拟机平台功能)

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

功能成功开启之后需要重启计算机！！！！

升级到WSL2

WSL与WSL2的功能比较表如图所示，可以看出除了跨操作系统文件系统的性能外，WSL2体系结构在多个方面比WSL1更具优势，学习和使用Linux的情况下，如果没有必须在Linux中访问Windows文件系统的需求，使用WSL2都是更好的选择，且WSL2的性能更好并提供100%的系统调用兼容性。

功能	WSL 1	WSL 2
Windows 和 Linux 之间的集成	✓	✓
启动时间短	✓	✓
占用的资源量少	✓	✓
可以与当前版本的 VMware 和 VirtualBox 一起运行	✓	✓
托管 VM	✗	✓
完整的 Linux 内核	✗	✓
完全的系统调用兼容性	✗	✓
跨 OS 文件系统的性能	✓	✗

下载并安装适用于x64计算机的WSL2Linux内核更新包，安装时会提示你提供提升的权限，选择"是"以批准此安装。

安装完成之后，在cmd执行如下语句，将WSL2设置为默认版本：

```
wsl --set-default-version 2
```

安装Linux发行版

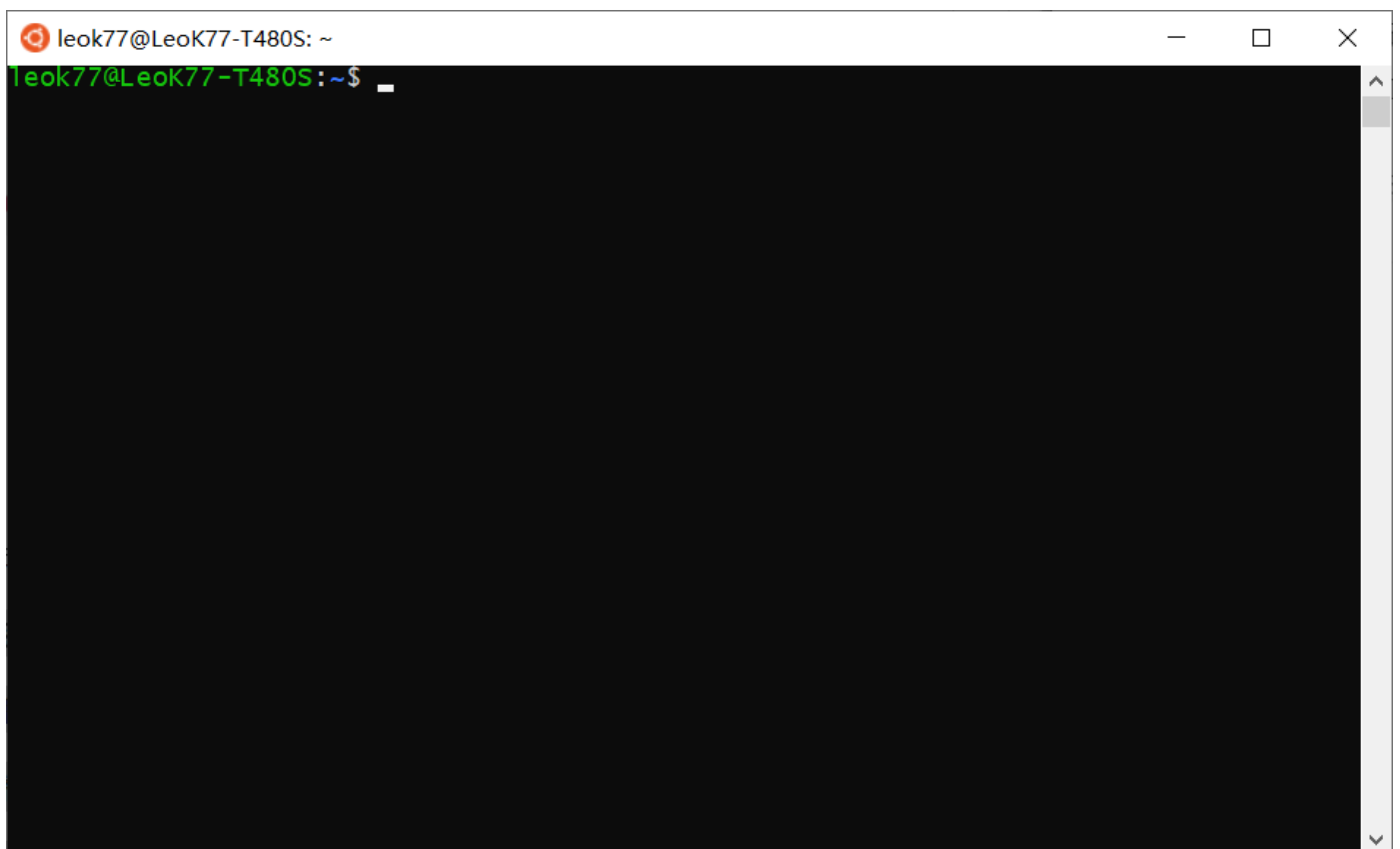
打开[Microsoft Store](#)，选择偏好的Linux发行版。这里列出的并不是所有的可用的Linux发行版，不满足于这几个发行版可以前去查看WSL官方文档，此处我使用的是[Ubuntu 20.04 LTS](#)。

首次启动新安装的Linux发行版时，将打开一个控制台窗口，系统会要求等待一段时间，以便文件解压缩并存储到电脑上，未来的启动时间会很快。

新安装的Linux发行版会要求创建用户账户(非root)和密码，此用户名和密码拥有以下属性：

- 此用户名和密码特定于安装的每个单独的 Linux 分发版，与 Windows 用户名无关。
- 创建用户名和密码后，该帐户将是分发版的默认用户，并将在启动时自动登录。
- 此帐户将被视为 Linux 管理员，能够运行 `sudo (Super User Do)` 管理命令。
- 在适用于 Linux 的 Windows 子系统上运行的每个 Linux 分发版都有其自身的 Linux 用户帐户和密码。每当添加分发版、重新安装或重置时，都必须配置一个 Linux 用户帐户。

安装结束后的Ubuntu 20.04 LTS窗口如下：

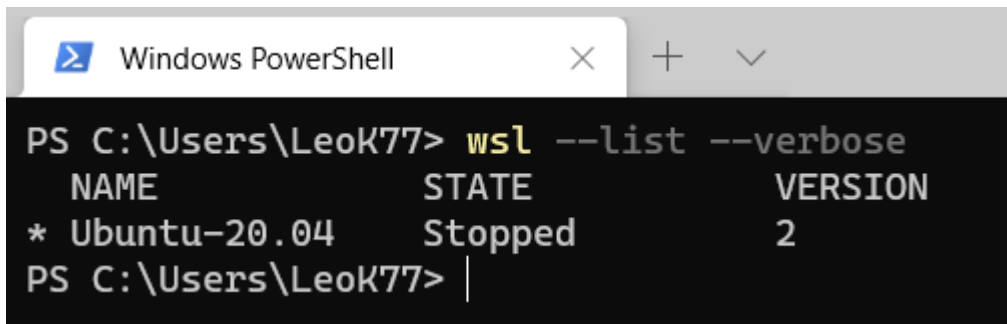


在此窗口下使用"`exit`"可以退出窗口，但是WSL会在后台运行，彻底关闭的方式是在Powershell下使用如下命令：

```
wsl --shutdown
```

查询wsl运行状态需使用如下命令：

```
wsl --list -verbose # 完整命令  
wsl -l -v          # 简写
```



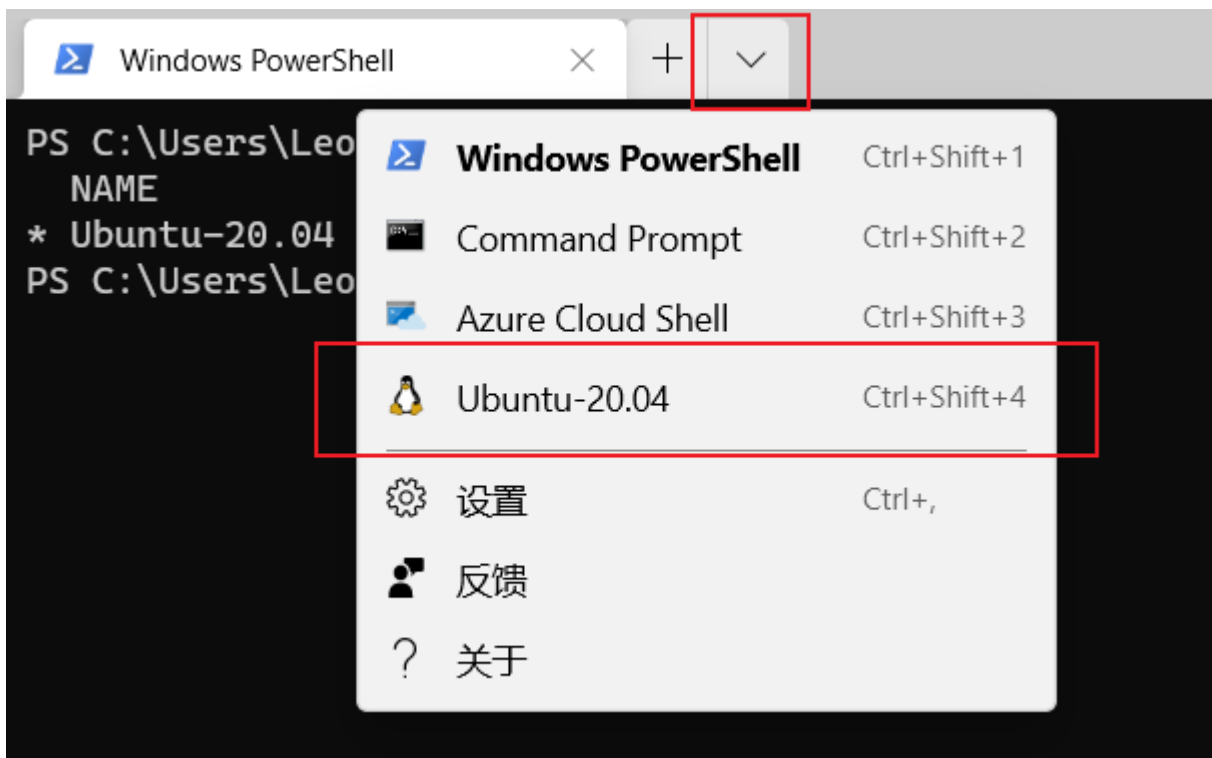
```
Windows PowerShell  
PS C:\Users\LeoK77> wsl --list --verbose  
NAME                STATE              VERSION  
* Ubuntu-20.04      Stopped            2  
PS C:\Users\LeoK77> |
```

通过其他方式登录到WSL

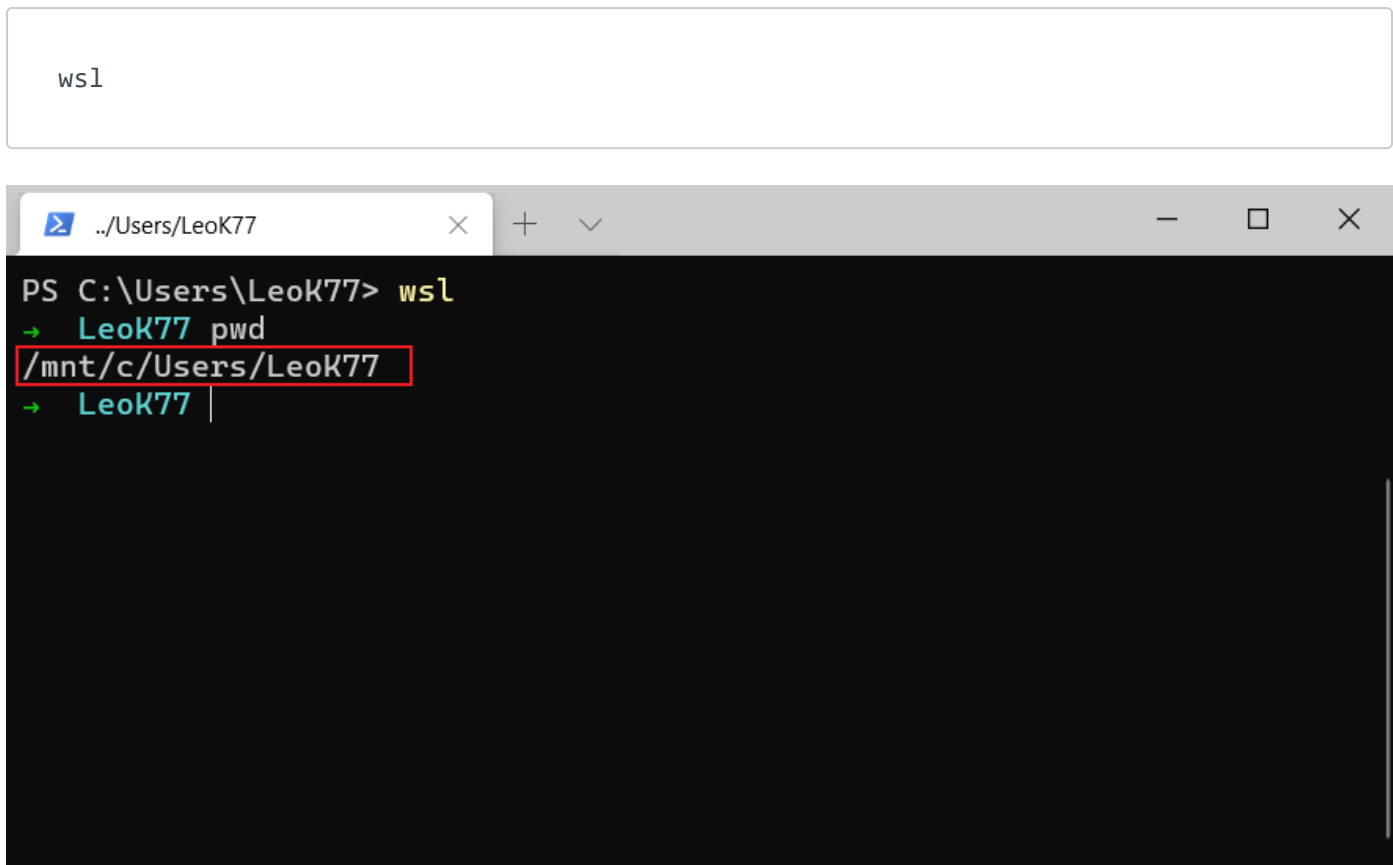
如果觉着Ubuntu的默认终端窗口有些丑的话，可以使用其他方式登录到WSL(其实就是我自己习惯下面这俩工具了)。

Windows Terminal

在Windows Terminal窗口点击如图所示位置可以直接进入WSL(需等待其启动)：



在Windows Terminal窗口下输入wsl也可直接进入：



需要注意的是，这两种方式进入WSL的默认路径都不是**/home/username**，而是Windows系统的用户文件夹的挂载位置**/mnt/c/Users/username**，执行操作时需注意。

VS Code - Remote WSL

安装Remote WSL插件，在插件栏搜索即可：



安装好之后侧边栏会多出一个"远程资源管理器图标"，点击之后选择"WSL Targets"，在点击如图所示位置(会自动搜寻本机中拥有的WSL主机)，就可以在VS Code中访问WSL了。



配置Ubuntu

修改为国内镜像源

初始镜像源是官方源，在国内访问会比较慢，我之前用的是清华源，结果install gcc的时候提示有个包找不到，所以就换成华为源了。

如果之前没有换过镜像源，那么可以直接使用下列命令进行换源(第一条是备份，第二条是换源)：

```
sudo cp -a /etc/apt/sources.list /etc/apt/sources.list.bak
sudo sed -i "s@http://.*archive.ubuntu.com@http://repo.huaweicloud.com@g"
/etc/apt/sources.list
sudo sed -i "s@http://.*security.ubuntu.com@http://repo.huaweicloud.com@g"
/etc/apt/sources.list
```

换好源之后update、upgrade一下，舒服多了。

```
sudo apt-get update
sudo apt-get upgrade -y
```

Shell: zsh+ohmyzsh

Shell是否更换视自己习惯而定，我主要是喜欢zsh的补全功能，然后习惯用ohmyzsh的主题了(ohmyzsh在资源占用方面好像有所诟病，但日常使用区别并不大)。

```
sudo apt-get install zsh -y    #安装zsh
chsh -s /bin/zsh username    #修改用户的默认终端为zsh
#下载并安装 oh-my-zsh
wget https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh
sh install.sh
```

修改"/home/username/.zshrc"中的ZSH_THEME为"duellj":

```
ZSH_THEME="duellj"
```

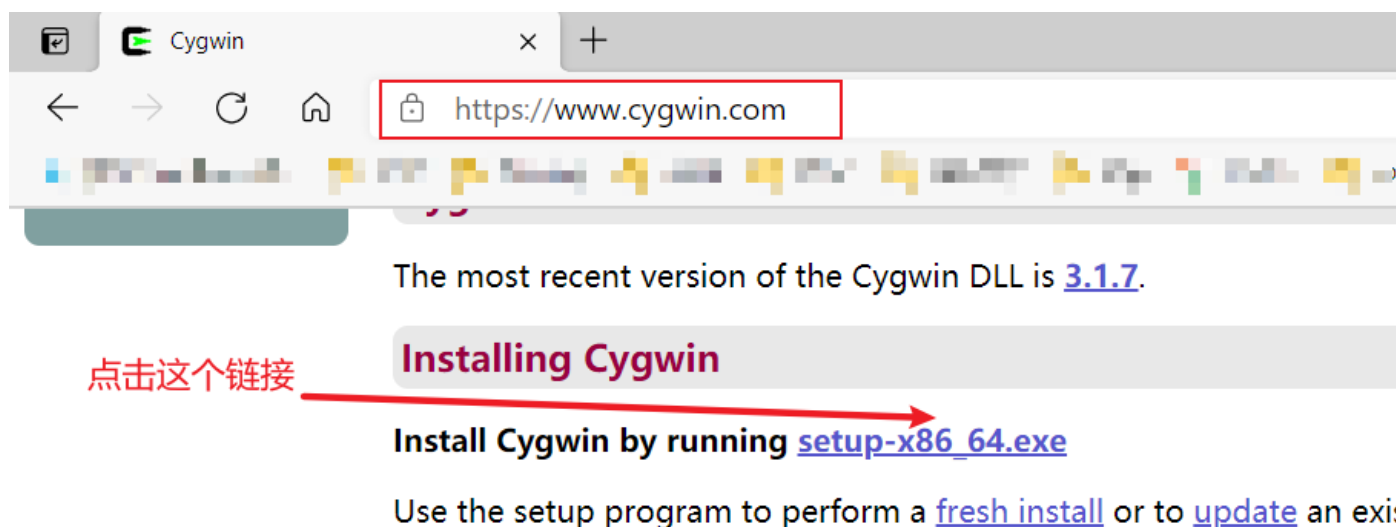
在.zshrc文件末尾追加如下内容，这样就不会出现末尾的空行显示成%的情况了：

```
# 去除末尾的百分号
setopt PROMPT_CR
setopt PROMPT_SP
export PROMPT_EOL_MARK=""
```

Cygwin

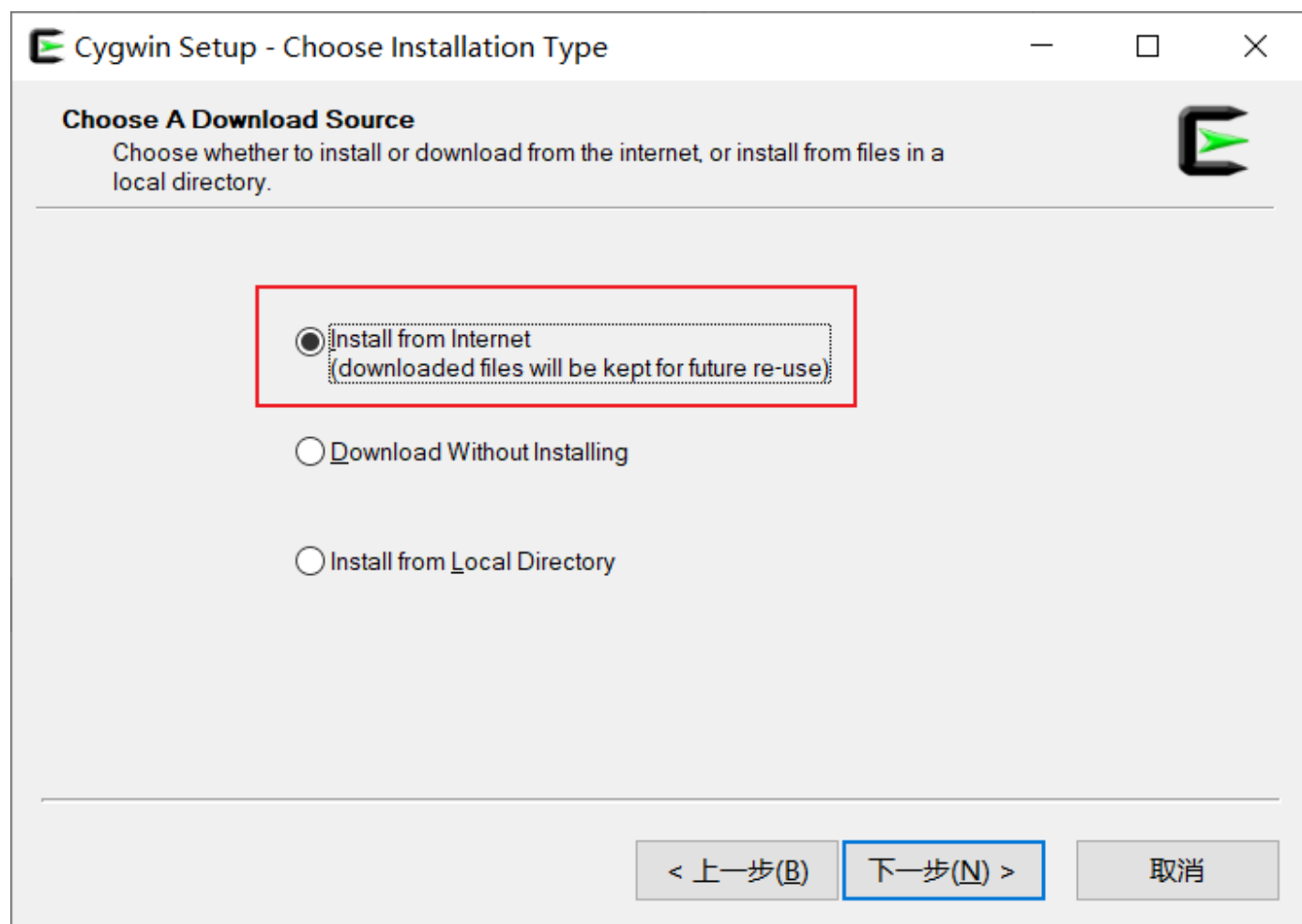
安装Cygwin

下载安装Cygwin没啥可说的，进入[Cygwin官网](https://www.cygwin.com)找到下载链接(在官网如图所示的位置，不要点击这个图片啊喂)点击即可下载。



安装过程的注意事项除了选择好安装组件外也没啥好说的，也就是选择"Install from Internet"(另外那俩是要搭配在一起使用的，一个下载一个安装，这一个直接下载并安

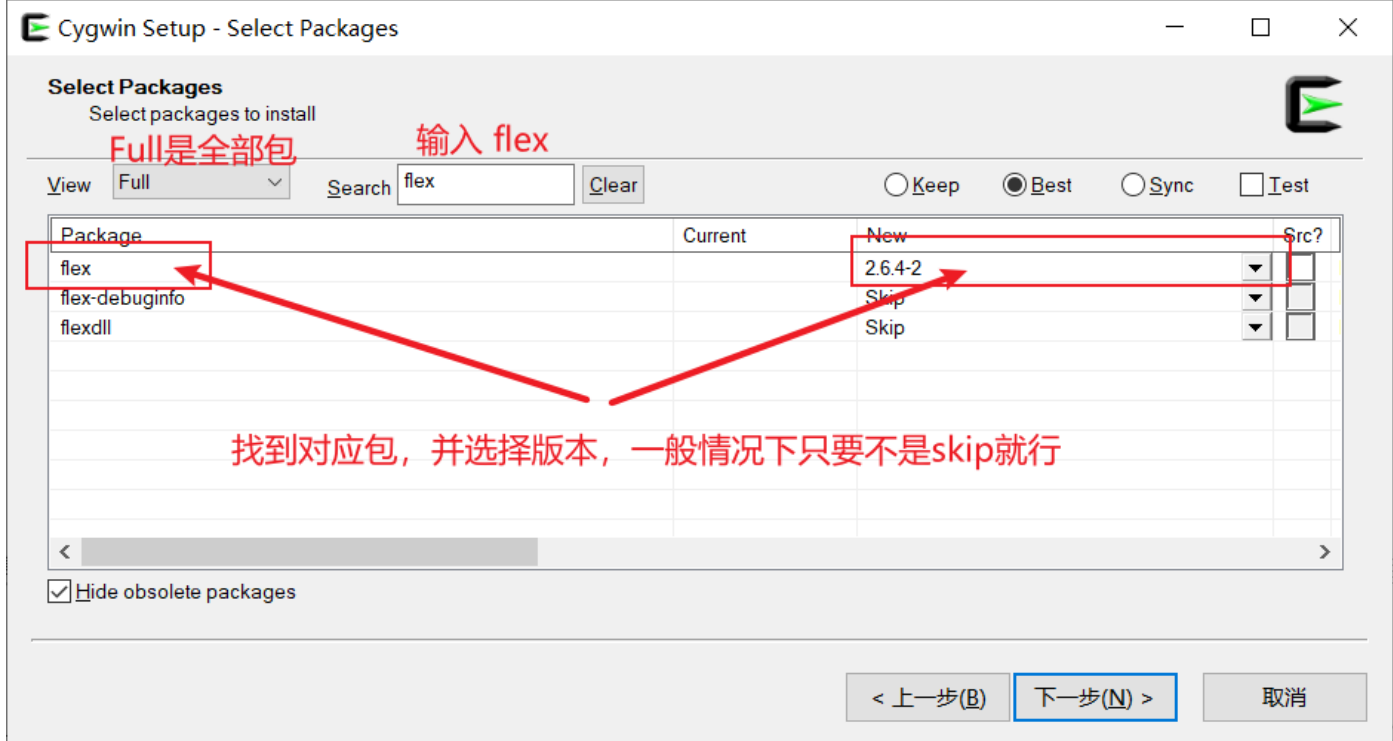
装), 以及下载的时候选国内镜像, 我选的阿里云镜像。



这三个镜像是我一眼看到的, 后面应该还有, 我感觉这仨里面随便选一个就行了。



下一步就是选择需要安装的组件了, 这里以flex作为示例, 其余组件按照如图所示方法安装即可, 建议安装**wget**:



安装完之后桌面上应该会有一个Cygwin的图标，双击打开就可以使用了；如果安装的时候没选择创建桌面图标，那么就打开搜索("Win+Q")然后搜索Cygwin，名字叫"Cygwin64 Terminal"的就是了，如果搜不到的话，好好反思一下自己是不是选了下载没安装。

更换Shell为zsh+ohmyzsh

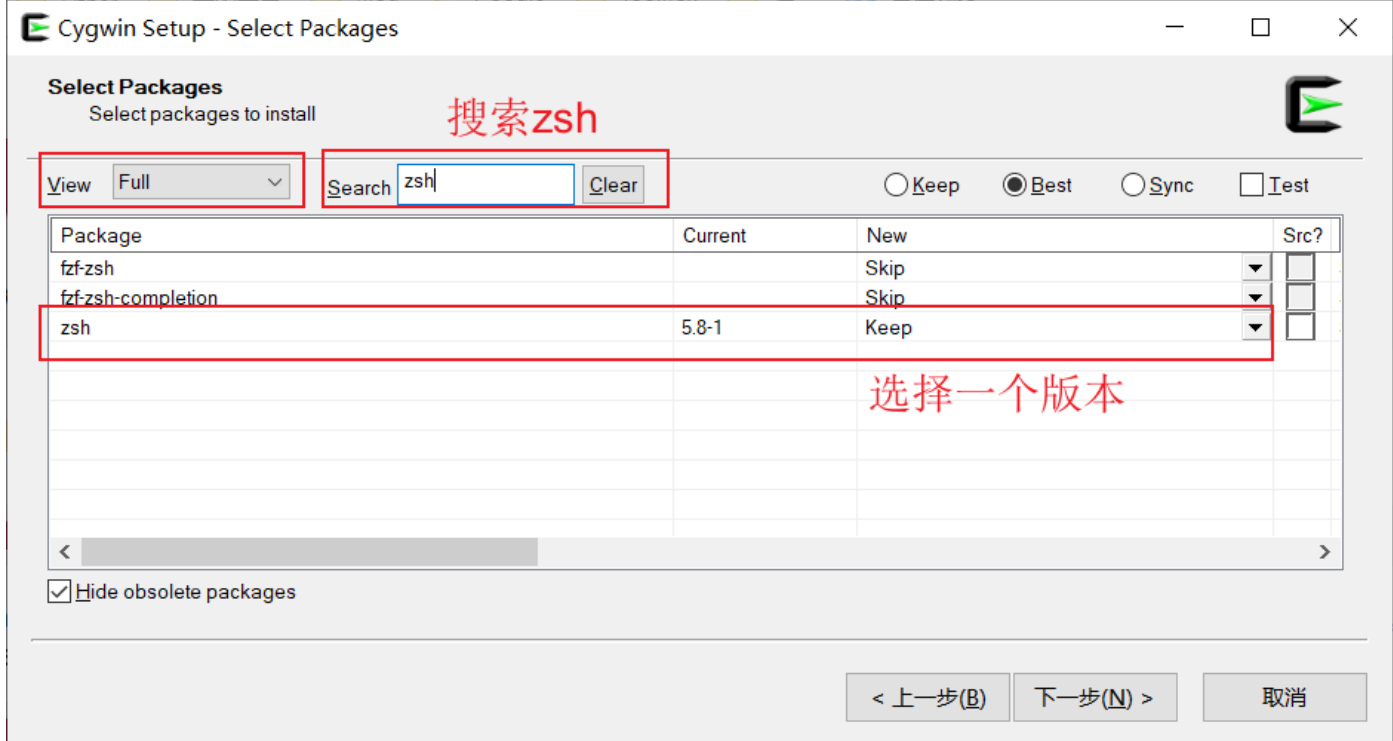
安装zsh

安装zsh有多个方法，这里展示三种：①使用安装程序的GUI安装；②使用安装程序的CLI安装；③安装包管理器apt-cyg，并用apt-cyg安装。在这几种方法里面任选其一即可

需要注意的是在使用前两种方法的时候，Cygwin不能处于运行状态，或者说不可以有正在运行的Cygwin窗口，无论是Cygwin Terminal还是在Windows Terminal中使用Cygwin的bash/zsh等shell都不可以

使用安装程序的GUI安装

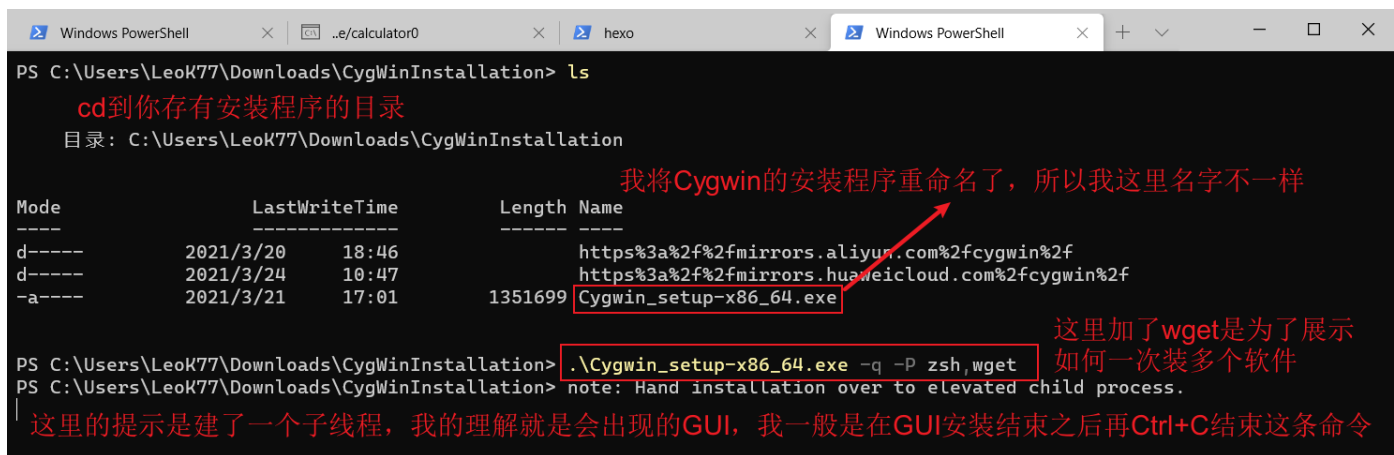
所谓安装程序就是从官网下载下来的Cygwin安装程序，即"setup-x86_64.exe"，不断下一步直到出现如图所示这个页面，然后根据图片中的信息在“Full”视图中搜索“zsh”并选择自己想安装的版本，我这里是“keep”是因为我已经装过了，一般情况下需要选择一串数字版本号，比如我现在的版本是“5.8-1”，即前面的Current——当前版本。



使用安装程序的**CLI**安装

安装程序除了可以在窗口执行以外，也提供命令行操作，不过执行命令的时候还是会出现一个窗口，然后安装结束Ctrl+C一下就好了。

```
.\Cygwin_setup-x86_64.exe -q -P zsh,wget
```



使用**apt-cyg**安装

apt-cyg GitHub: <https://github.com/transcode-open/apt-cyg>

apt-cyg是一个类似于apt-get的包管理器，适用于Cygwin，可以使用其GitHub中的说明直接安装，可能是我的网络问题，一直失败，所以我采用先自行下载到本地再执行安装的方式：

1、打开上述apt-cyg的GitHub连接，找到仓库中的apt-cyg文件，直接点击即可，然后在此文件的页面点击那个"Raw"，进入一个新的页面，复制这个页面的链接然后下载此连接内容，保存为apt-cyg文件(没有拓展名)，并将其放在/home/username目录下，效果如图所示：

```
LeoK77@LeoK77-T480S ~  
$ pwd  
/home/LeoK77      /home/username  
  
LeoK77@LeoK77-T480S ~  
$ ls  
apt-cyg          apt-cyg
```

2、安装apt-cyg，执行完下述语句之后需要重启Cygwin Terminal，不然无法使用apt-cyg命令

```
install apt-cyg /bin
```

3、使用apt-cyg安装zsh

```
apt-cyg install zsh
```

跑码结束且无ERROR即为安装成功，可以通过在shell中输入zsh来判断是否可用(可能需要重开一下Cygwin Terminal)

设置zsh为默认shell

在Cygwin的安装目录中，打开/etc/nsswitch.conf文件，并在其中添加"db_shell: /bin/zsh"；比如我的Cygwin安装在"C:\ToolBox\CygWin"目录下，我需要打开的文件就是"C:\ToolBox\CygWin\etc\nsswitch.conf"，然后在其中添加上述内容

```
为Cygwin安装oh-my-zsh.md 2, U  nsswitch.conf X
C: > ToolBox > CygWin > etc > nsswitch.conf
1 # /etc/nsswitch.conf
2 #
3 # This file is read once by the first process in a Cygwin process tree.
4 # To pick up changes, restart all Cygwin processes. For a description
5 # see https://cygwin.com/cygwin-ug-net/ntsec.html#ntsec-mapping-nsswitch
6 #
7 # Defaults:
8 # passwd:  files db
9 # group:   files db
10 # db_enum: cache builtin
11 # db_home: /home/%U
12 # db_shell: /bin/bash
13 db_shell: /bin/zsh
14 # db_gecos: <empty>
15
```

这样在打开Cygwin Terminal应用程序的时候就会默认打开zsh而不是bash了

安装oh-my-zsh

对于我来说，我习惯用oh-my-zsh(没错，我是一个颜狗)，而且我并不会用到什么插件，我需要的仅仅是shell的美化和好看，所以我不在意别人说的ohmyzsh资源浪费问题，既然已经将Cygwin的shell换成了zsh，那么安装oh-my-zsh就是我接下来要做的事情。

oh-my-zsh GitHub: <https://github.com/ohmyzsh/ohmyzsh>

除了官方GitHub提供的安装方式外，这里提供两种安装方式，第一种是下载zip包然后安装，第二种是使用Gitee作为镜像源安装

下载zip包安装oh-my-zsh

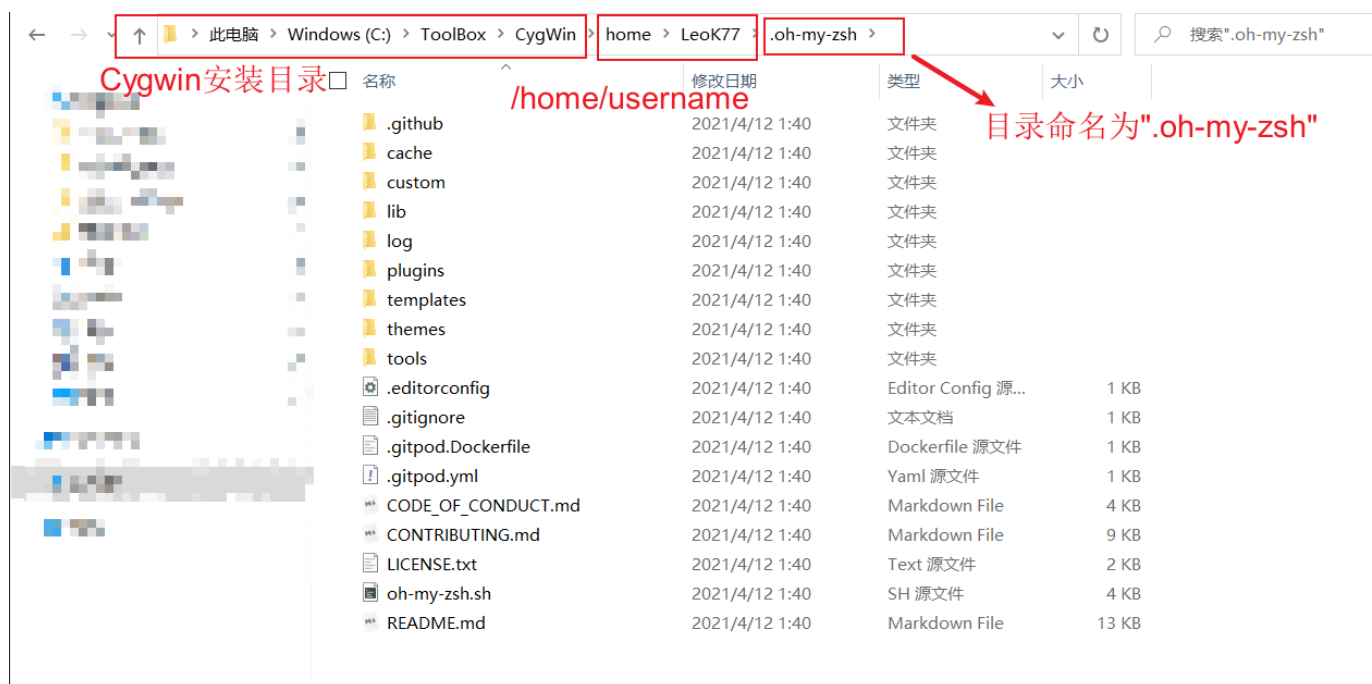
1、下载oh-my-zsh，打开oh-my-zsh的GitHub仓库，然后下载zip包即可(由于我git clone一直失败，所以只好手动下载zip了，如果对自己网络有信息，完全可以git clone)，就不图示说明如何操作了，挺简单的操作

2、解压oh-my-zsh到Cygwin下的/home/username文件夹下，需要注意的是这里如果在Windows下解压此文件可能需要管理员权限，为了防止权限问题，这里展示如何在Cygwin下使用命令行解压缩此zip文件：

```
pwd # 查看当前路径，此操作需要当前目录是 /home/username，如果不是的话可以使用 cd ~ 操作
到达 /home/username
cp /cygdrive/c/Users/LeoK77/Downloads/Compressed/ohmyzsh-master.zip ./ # 将下载下来的
ohmyzsh-master.zip文件复制到当前目录(/home/username)，此操作需要将前面这个路径替换为自
己的ohmyzsh-master.zip文件的路径
```

```
unzip ohmyzsh-master.zip -d ./ # 执行解压操作, 直接解压到当前路径下  
mv ohmyzsh-master ./oh-my-zsh # 重命名为 .oh-my-zsh 文件夹
```

解压之后, 目录结构应该如下:



3、将.oh-my-zsh目录下的templates/zshrc.zsh-template中的内容复制到home/username目录下的.zshrc中, 可以在Cygwin中使用如下命令

```
cp ../oh-my-zsh/templates/zshrc.zsh-template ../zshrc
```

使用Gitee安装oh-my-zsh

可以将下方内容保存为install.sh, 然后执行"sh ./install.sh"来安装oh-my-zsh; 相较于官方install.sh, 此文件仅将下载源及更新源修改为了Gitee的镜像源:

<https://gitee.com/mirrors/oh-my-zsh>; 如果想自己修改的话可以下载官方install.sh然后修改文件的第43行开始的段落, 与本文件等效

```
#!/bin/sh  
#  
# This script should be run via curl:  
# sh -c "$(curl -fsSL  
https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"  
# or via wget:  
# sh -c "$(wget -qO-  
https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"  
# or via fetch:  
# sh -c "$(fetch -o -  
https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)"  
#  
# As an alternative, you can first download the install script and run it
```

```

afterwards:
#   wget https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh
#   sh install.sh
#
# You can tweak the install behavior by setting variables when running the script.
For
# example, to change the path to the Oh My Zsh repository:
#   ZSH=~/.zsh sh install.sh
#
# Respects the following environment variables:
#   ZSH      - path to the Oh My Zsh repository folder (default: $HOME/.oh-my-zsh)
#   REPO     - name of the GitHub repo to install from (default: ohmyzsh/ohmyzsh)
#   REMOTE   - full remote URL of the git repo to install (default: GitHub via
HTTPS)
#   BRANCH   - branch to check out immediately after install (default: master)
#
# Other options:
#   CHSH      - 'no' means the installer will not change the default shell
(default: yes)
#   RUNZSH    - 'no' means the installer will not run zsh after the install
(default: yes)
#   KEEP_ZSHRC - 'yes' means the installer will not replace an existing .zshrc
(default: no)
#
# You can also pass some arguments to the install script to set some these options:
#   --skip-chsh: has the same behavior as setting CHSH to 'no'
#   --unattended: sets both CHSH and RUNZSH to 'no'
#   --keep-zshrc: sets KEEP_ZSHRC to 'yes'
# For example:
#   sh install.sh --unattended
# or:
#   sh -c "$(curl -fsSL
https://raw.githubusercontent.com/ohmyzsh/ohmyzsh/master/tools/install.sh)" "" --
unattended
#
set -e

# Track if $ZSH was provided
custom_zsh=${ZSH:+yes}

# Default settings
# https://gitee.com/mirrors/oh-my-zsh
ZSH=${ZSH:-~/.oh-my-zsh}
REPO=${REPO:-mirrors/oh-my-zsh}
REMOTE=${REMOTE:-https://gitee.com/${REPO}.git}
BRANCH=${BRANCH:-master}

# Other options
CHSH=${CHSH:-yes}
RUNZSH=${RUNZSH:-yes}
KEEP_ZSHRC=${KEEP_ZSHRC:-no}

command_exists() {
  command -v "$@" >/dev/null 2>&1
}

# The [ -t 1 ] check only works when the function is not called from
# a subshell (like in `$(...)` or `(...)` , so this hack redefines the

```

```

# function at the top level to always return false when stdout is not
# a tty.
if [ -t 1 ]; then
    is_tty() {
        true
    }
else
    is_tty() {
        false
    }
fi

# This function uses the logic from supports-hyperlinks[1][2], which is
# made by Kat Marchán (@zkat) and licensed under the Apache License 2.0.
# [1] https://github.com/zkat/supports-hyperlinks
# [2] https://crates.io/crates/supports-hyperlinks
#
# Copyright (c) 2021 Kat Marchán
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
supports_hyperlinks() {
    # $FORCE_HYPERLINK must be set and be non-zero (this acts as a logic bypass)
    if [ -n "$FORCE_HYPERLINK" ]; then
        [ "$FORCE_HYPERLINK" != 0 ]
        return $?
    fi

    # If stdout is not a tty, it doesn't support hyperlinks
    is_tty || return 1

    # DomTerm terminal emulator (domterm.org)
    if [ -n "$DOMTERM" ]; then
        return 0
    fi

    # VTE-based terminals above v0.50 (Gnome Terminal, Guake, ROXTerm, etc)
    if [ -n "$VTE_VERSION" ]; then
        [ $VTE_VERSION -ge 5000 ]
        return $?
    fi

    # If $TERM_PROGRAM is set, these terminals support hyperlinks
    case "$TERM_PROGRAM" in
        Hyper|iTerm.app|terminology|WezTerm) return 0 ;;
    esac

    # kitty supports hyperlinks
    if [ "$TERM" = xterm-kitty ]; then
        return 0
    fi

```



```

fi

# Windows Terminal or Konsole also support hyperlinks
if [ -n "$WT_SESSION" ] || [ -n "$KONSOLE_VERSION" ]; then
    return 0
fi

return 1
}

fmt_link() {
    # $1: text, $2: url, $3: fallback mode
    if supports_hyperlinks; then
        printf '\033]8;;%s\a%s\033]8;;\a\n' "$2" "$1"
        return
    fi

    case "$3" in
        --text) printf '%s\n' "$1" ;;
        --url|*) fmt_underline "$2" ;;
    esac
}

fmt_underline() {
    is_tty && printf '\033[4m%s\033[24m\n' "$*" || printf '%s\n' "$*"
}

# shellcheck disable=SC2016 # backtick in single-quote
fmt_code() {
    is_tty && printf '\033[2m%s\033[22m\n' "$*" || printf '\033[2m%s\n' "$*"
}

fmt_error() {
    printf '%sError: %s%s\n' "$BOLD$RED" "$*" "$RESET" >&2
}

setup_color() {
    # Only use colors if connected to a terminal
    if is_tty; then
        RAINBOW="
$(printf '\033[38;5;196m')
$(printf '\033[38;5;202m')
$(printf '\033[38;5;226m')
$(printf '\033[38;5;082m')
$(printf '\033[38;5;021m')
$(printf '\033[38;5;093m')
$(printf '\033[38;5;163m')
"
        RED=$(printf '\033[31m')
        GREEN=$(printf '\033[32m')
        YELLOW=$(printf '\033[33m')
        BLUE=$(printf '\033[34m')
        BOLD=$(printf '\033[1m')
        RESET=$(printf '\033[m')
    else
        RAINBOW=""
        RED=""
        GREEN=""
        YELLOW=""
    fi
}

```

```

BLUE=""
BOLD=""
RESET=""
fi
}

setup_ohmyzsh() {
# Prevent the cloned repository from having insecure permissions. Failing to do
# so causes compinit() calls to fail with "command not found: compdef" errors
# for users with insecure umasks (e.g., "002", allowing group writability). Note
# that this will be ignored under Cygwin by default, as Windows ACLs take
# precedence over umasks except for filesystems mounted with option "noacl".
umask g-w,o-w

echo "${BLUE}Cloning Oh My Zsh...${RESET}"

command_exists git || {
    fmt_error "git is not installed"
    exit 1
}

ostype=$(uname)
if [ -z "${ostype%CYGWIN*}" ] && git --version | grep -q msysgit; then
    fmt_error "Windows/MSYS Git is not supported on Cygwin"
    fmt_error "Make sure the Cygwin git package is installed and is first on the
\\$PATH"
    exit 1
fi

git clone -c core.eol=lf -c core.autocrlf=false \
    -c fsck.zeroPaddedFilemode=ignore \
    -c fetch.fsck.zeroPaddedFilemode=ignore \
    -c receive.fsck.zeroPaddedFilemode=ignore \
    -c oh-my-zsh.remote=origin \
    -c oh-my-zsh.branch="$BRANCH" \
    --depth=1 --branch "$BRANCH" "$REMOTE" "$ZSH" || {
    fmt_error "git clone of oh-my-zsh repo failed"
    exit 1
}

echo
}

setup_zshrc() {
# Keep most recent old .zshrc at .zshrc.pre-oh-my-zsh, and older ones
# with datestamp of installation that moved them aside, so we never actually
# destroy a user's original zshrc
echo "${BLUE}Looking for an existing zsh config...${RESET}"

# Must use this exact name so uninstall.sh can find it
OLD_ZSHRC=~/.zshrc.pre-oh-my-zsh
if [ -f ~/.zshrc ] || [ -h ~/.zshrc ]; then
    # Skip this if the user doesn't want to replace an existing .zshrc
    if [ "$KEEP_ZSHRC" = yes ]; then
        echo "${YELLOW}Found ~/.zshrc.${RESET} ${GREEN}Keeping...${RESET}"
        return
    fi
    if [ -e "$OLD_ZSHRC" ]; then
        OLD_OLD_ZSHRC="${OLD_ZSHRC}-${date +%Y-%m-%d_%H-%M-%S}"

```

```

    if [ -e "$OLD_OLD_ZSHRC" ]; then
        fmt_error "$OLD_OLD_ZSHRC exists. Can't back up ${OLD_ZSHRC}"
        fmt_error "re-run the installer again in a couple of seconds"
        exit 1
    fi
    mv "$OLD_ZSHRC" "${OLD_OLD_ZSHRC}"

    echo "${YELLOW}Found old ~/.zshrc.pre-oh-my-zsh." \
        "${GREEN}Backing up to ${OLD_OLD_ZSHRC}${RESET}"
    fi
    echo "${YELLOW}Found ~/.zshrc.${RESET} ${GREEN}Backing up to
${OLD_ZSHRC}${RESET}"
    mv ~/.zshrc "$OLD_ZSHRC"
    fi

    echo "${GREEN}Using the Oh My Zsh template file and adding it to
~/.zshrc.${RESET}"

    sed "/^export ZSH=/ c\\
export ZSH=\"\$ZSH\"
\" \$ZSH/templates/zshrc.zsh-template\" > ~/.zshrc-omztemp
mv -f ~/.zshrc-omztemp ~/.zshrc

echo
}

setup_shell() {
    # Skip setup if the user wants or stdin is closed (not running interactively).
    if [ "$CHSH" = no ]; then
        return
    fi

    # If this user's login shell is already "zsh", do not attempt to switch.
    if [ "$(basename -- "$SHELL")" = "zsh" ]; then
        return
    fi

    # If this platform doesn't provide a "chsh" command, bail out.
    if ! command_exists chsh; then
        cat <<EOF
I can't change your shell automatically because this system does not have chsh.
${BLUE}Please manually change your default shell to zsh${RESET}
EOF
        return
    fi

    echo "${BLUE}Time to change your default shell to zsh:${RESET}"

    # Prompt for user choice on changing the default login shell
    printf '%sDo you want to change your default shell to zsh? [Y/n]%s ' \
        "${YELLOW}" "${RESET}"
    read -r opt
    case $opt in
        y*|Y*|") echo "Changing the shell..." ;;
        n*|N*) echo "Shell change skipped."; return ;;
        *) echo "Invalid choice. Shell change skipped."; return ;;
    esac

    # Check if we're running on Termux

```

```

case "$PREFIX" in
    *com.termux*) termux=true; zsh=zsh ;;
    *) termux=false ;;
esac

if [ "$termux" != true ]; then
    # Test for the right location of the "shells" file
    if [ -f /etc/shells ]; then
        shells_file=/etc/shells
    elif [ -f /usr/share/defaults/etc/shells ]; then # Solus OS
        shells_file=/usr/share/defaults/etc/shells
    else
        fmt_error "could not find /etc/shells file. Change your default shell
manually."
        return
    fi

    # Get the path to the right zsh binary
    # 1. Use the most preceding one based on $PATH, then check that it's in the
shells file
    # 2. If that fails, get a zsh path from the shells file, then check it actually
exists
    if ! zsh=$(command -v zsh) || ! grep -qx "$zsh" "$shells_file"; then
        if ! zsh=$(grep '^./.*zsh$' "$shells_file" | tail -1) || [ ! -f "$zsh" ];
then
            fmt_error "no zsh binary found or not present in '$shells_file'"
            fmt_error "change your default shell manually."
            return
        fi
    fi
fi

# We're going to change the default shell, so back up the current one
if [ -n "$SHELL" ]; then
    echo "$SHELL" > ~/.shell.pre-oh-my-zsh
else
    grep "^$USERNAME:" /etc/passwd | awk -F: '{print $7}' > ~/.shell.pre-oh-my-zsh
fi

# Actually change the default shell to zsh
if ! chsh -s "$zsh"; then
    fmt_error "chsh command unsuccessful. Change your default shell manually."
else
    export SHELL="$zsh"
    echo "${GREEN}Shell successfully changed to '$zsh'.${RESET}"
fi

echo
}

```

```

# shellcheck disable=SC2183 # printf string has more %s than arguments ($RAINBOW
expands to multiple arguments)
print_success() {
    printf '%s      %s__    %s          %s          %s          %s          %s__    %s\n'
$RAINBOW $RESET
    printf '%s  ____  %s/ /_    %s  ____  ____  %s__  ____  %s  ____  %s____%s/ /_    %s\n'
$RAINBOW $RESET
    printf '%s /  __ \ %s/  __ \    %s /  __ \  ____ \ %s/ / / / %s /_  / %s/  ____/ %s  __ \ %s\n'
$RAINBOW $RESET

```

```

printf '%s/ /_/ /%s / / / %s / / / / / %s /_/ / %s / /_/%s(____)%s / / / %s\n'
$RAINBOW $RESET
printf '%s\____/%s_/ /_/ %s /_/ /_/ /_/ %s\__, / %s /____/%s____/%s_/ /_/ %s\n'
$RAINBOW $RESET
printf '%s %s %s %s %s /____/ %s %s %s'
%s....is now installed!%s\n' $RAINBOW $GREEN $RESET
printf '\n'
printf '\n'
printf "%s %s %s\n" "Before you scream ${BOLD}${YELLOW}Oh My Zsh!${RESET} look
over the" \
    "${(fmt_code "${(fmt_link ".zshrc" "file://$HOME/.zshrc" --text)})}" \
    "file to select plugins, themes, and options."
printf '\n'
printf '%s\n' "• Follow us on Twitter: ${(fmt_link @ohmyzsh
https://twitter.com/ohmyzsh)"
printf '%s\n' "• Join our Discord community: ${(fmt_link "Discord server"
https://discord.gg/ohmyzsh)"
printf '%s\n' "• Get stickers, t-shirts, coffee mugs and more: ${(fmt_link "Planet
Argon Shop" https://shop.planetargon.com/collections/oh-my-zsh)"
printf '%s\n' $RESET
}

```

```

main() {
    # Run as unattended if stdin is not a tty
    if [ ! -t 0 ]; then
        RUNZSH=no
        CHSH=no
    fi

    # Parse arguments
    while [ $# -gt 0 ]; do
        case $1 in
            --unattended) RUNZSH=no; CHSH=no ;;
            --skip-chsh) CHSH=no ;;
            --keep-zshrc) KEEP_ZSHRC=yes ;;
        esac
        shift
    done

    setup_color

    if ! command_exists zsh; then
        echo "${YELLOW}Zsh is not installed.${RESET} Please install zsh first."
        exit 1
    fi

    if [ -d "$ZSH" ]; then
        echo "${YELLOW}The \ZSH folder already exists ($ZSH).${RESET}"
        if [ "$custom_zsh" = yes ]; then
            cat <<EOF

```

You ran the installer with the `\ZSH` setting or the `\ZSH` variable is exported. You have 3 options:

1. Unset the ZSH variable when calling the installer:
`$(fmt_code "ZSH= sh install.sh")`
2. Install Oh My Zsh to a directory that doesn't exist yet:
`$(fmt_code "ZSH=path/to/new/ohmyzsh/folder sh install.sh")`
3. (Caution) If the folder doesn't contain important information,

```
you can just remove it with $(fmt_code "rm -r $ZSH")
```

```
EOF
```

```
else
    echo "You'll need to remove it if you want to reinstall."
fi
exit 1
fi
```

```
setup_ohmyzsh
setup_zshrc
setup_shell
```

```
print_success
```

```
if [ $RUNZSH = no ]; then
    echo "${YELLOW}Run zsh to try it out.${RESET}"
    exit
fi
```

```
exec zsh -l
```

```
}
```

```
main "$@"
```

修改.zshrc(可选)

修改主题

修改主题，如果喜欢默认主题的话可以不执行此操作，如果不喜欢的话，可以自行修改主题，我比较喜欢的主题是"duellj"，所以我需要编辑".zshrc"文件，将其中的"ZSH_THEME="robbyrussell""改为"ZSH_THEME="duellj""

去除行尾百分号

在".zshrc"文件尾部追加如下三行内容，主题效果及需要修改的地方都如图所示：

```
# 去除末尾的百分号
setopt PROMPT_CR
setopt PROMPT_SP
export PROMPT_EOL_MARK=""
```

```
Windows PowerShell x | hexo x ~ x + v - □ x
[LeoK77@LeoK77-T480S] - [~] - [23]
[~]$ cat .zshrc [15:47:20]
# ZSH_DISABLE_COMPFIX="true"
# If you come from bash you might have to change your $PATH.
# export PATH=$HOME/bin:/usr/local/bin:$PATH

# Path to your oh-my-zsh installation.
# export ZSH=$HOME/.oh-my-zsh
export ZSH=/home/LeoK77/.oh-my-zsh

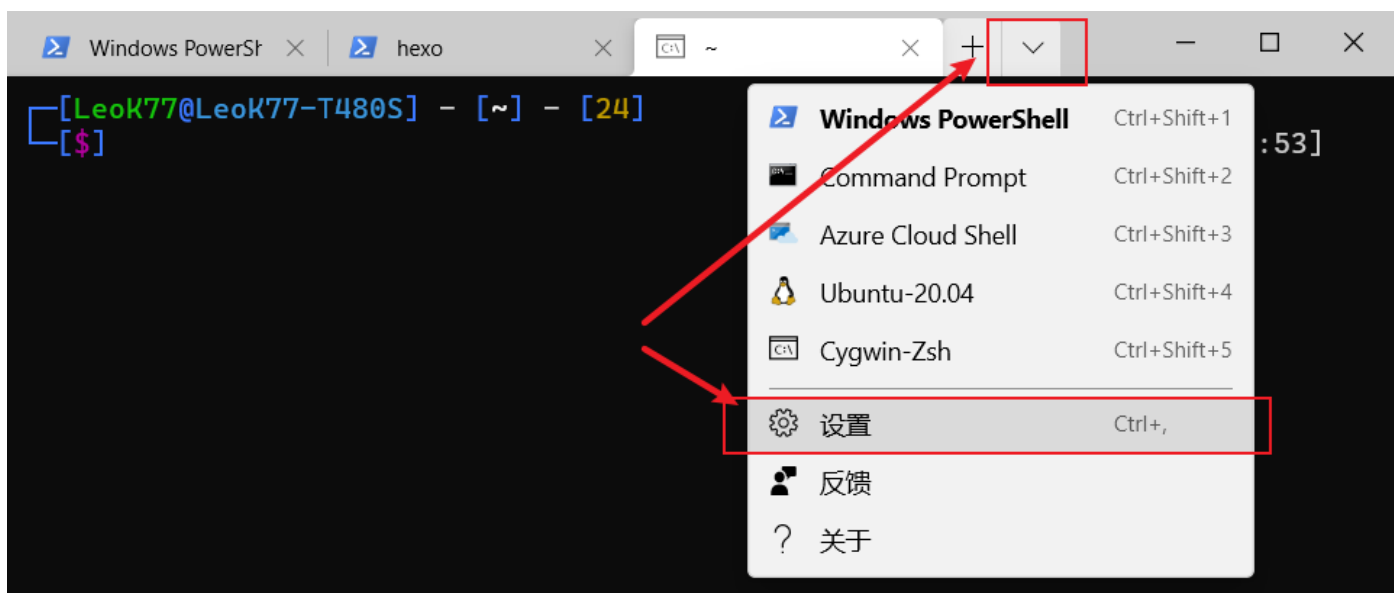
# Set name of the theme to load --- if set to "random", it will
# load a random theme each time oh-my-zsh is loaded, in which case,
# to know which specific one was loaded, run: echo $RANDOM_THEME
# See https://github.com/ohmyzsh/ohmyzsh/wiki/Themes
# ZSH_THEME="robbyrussell"
ZSH_THEME="duellj"
```

至此，Cygwin下的oh-my-zsh已经配置好了(没错，我不改其他配置，因为我只是一个颜狗，对于我来说这些就够了)，此时再打开Cygwin Terminal的时候就是一个有着oh-my-zsh的黑框框了；但对于一个颜狗来说，这显然还是不够好看，所以我要把这个黑框框加入到Windows Terminal里面，虽然同样是黑框框，但我还是喜欢Windows Terminal

在Windows Terminal中添加Cygwin

不用Windows Terminal的可以跳过这一部分了，如果没安装但是想用的可以自行去Win10应用商店下载一个(免费应用，而且是一个相对好看的黑框框)，然后按照下述步骤操作

打开Windows Terminal，找到设置，然后点击即可打开配置文件。



然后可以照着前面的配置自己添加一个Cygwin zsh的中断配置，guid可以自己造一个，也可以在Cygwin Terminal中使用uuidgen命令随机生成一个，只要不和前面的那几项重复即可，name可以自己随便命名，比如我这里直接写成了"Cygwin-Zsh"，commandline是zsh所在地址，在你的Cygwin安装目录下的/bin/zsh.exe位置，"-i -l"是为了刷新终端，不然会有些问题(参考自[这篇文章](#))，然后就可以在Windows Terminal中选择使用Cygwin zsh了。

```
{
  "guid": "{7090d3b0-57a7-4078-b6aa-3a495228e8aa}",
  "hidden": false,
  "name": "Cygwin-Zsh",
  "commandline": "C:\\ToolBox\\CygWin\\bin\\zsh.exe -i -l"
}
```

```
{
  "guid": "{07b52e3e-de2c-5db4-bd2d-ba144ed6c273}",
  "hidden": false,
  "name": "Ubuntu-20.04",
  "source": "Windows.Terminal.Wsl"
},
{
  "guid": "{7090d3b0-57a7-4078-b6aa-3a495228e8aa}",
  "hidden": false,
  "name": "Cygwin-Zsh",
  "commandline": "C:\\ToolBox\\CygWin\\bin\\zsh.exe -i -l"
}
```