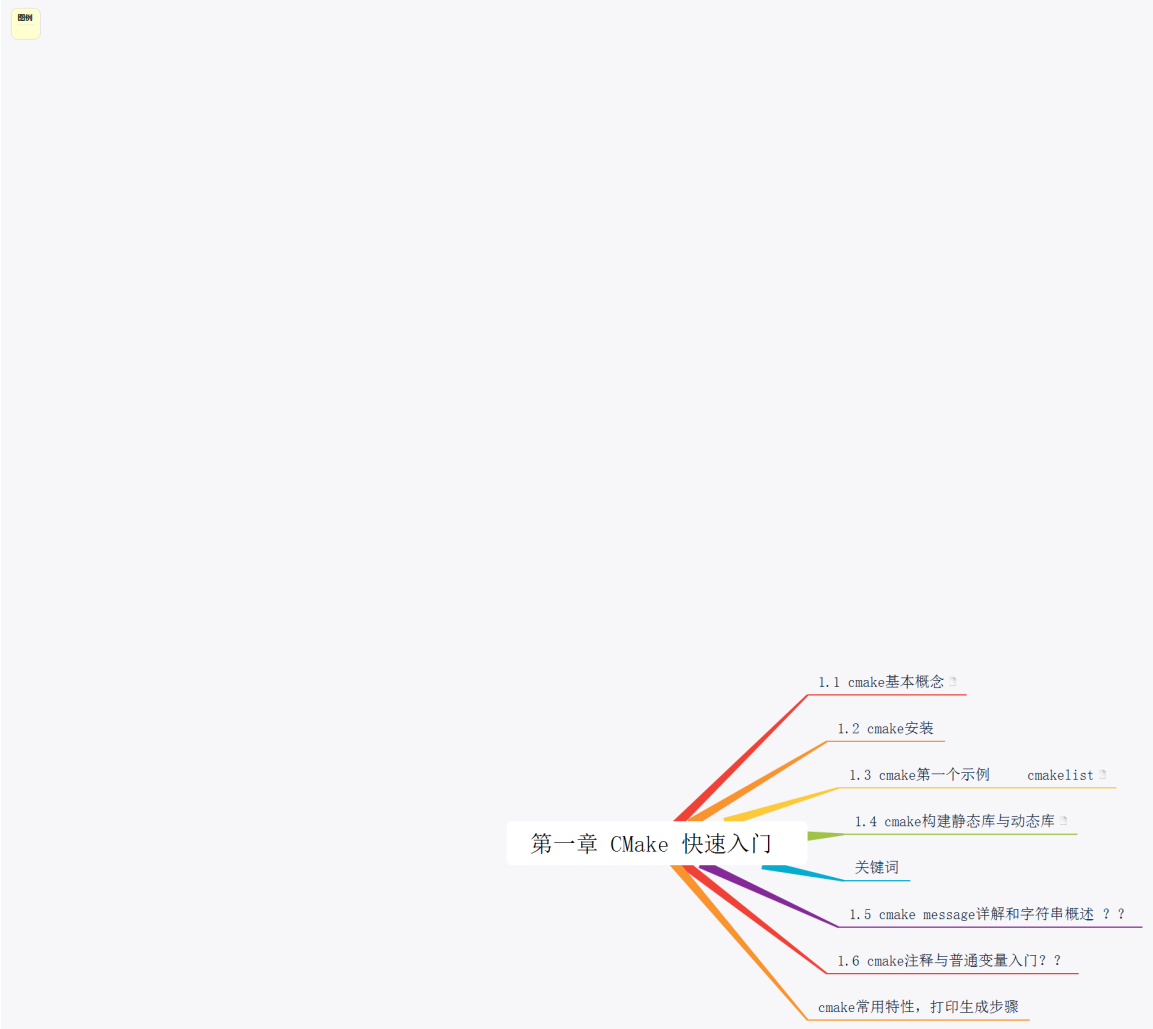
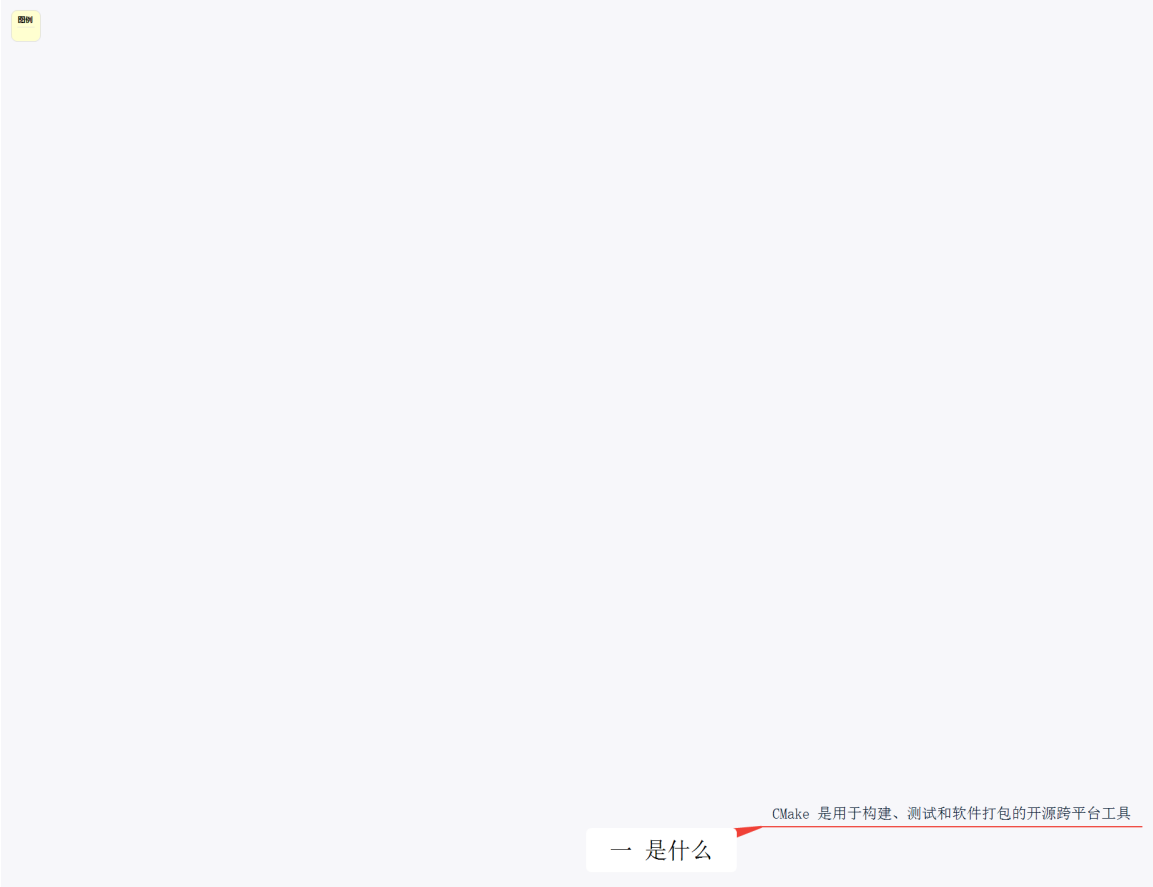
第一章 CMake 快速入门



1. 1.1 cmake基本概念

1.1. 一 是什么



1.1.1. CMake 是用于构建、测试和软件打包的开源跨平台工具

1.2. 二 为什么选用cmake

1.2.1. 为什么我需要一个好的构建系统

你想避免硬编码路径

您需要在多台计算机上构建一个包

你想使用 CI（持续集成）

你需要支持不同的操作系统

你想支持多个编译器

您想使用 IDE，但不是所有情况

你想描述你的程序的逻辑结构，而不是标志和命令

你想使用库

您想使用其他工具来帮助您编写代码 moc ProtoBuf

你想使用单元测试

1.2.2. 持续集成

每次集成都通过自动化的制造（包括提交、发布、自动化测试）来验证，准确地发现集成错误。

快速错误，每完成一点更新，就集成到主干，可以快速发现错误，定位错误也比较容易

各种不同的更新主干，如果不经常集成，会导致集成的成本变大

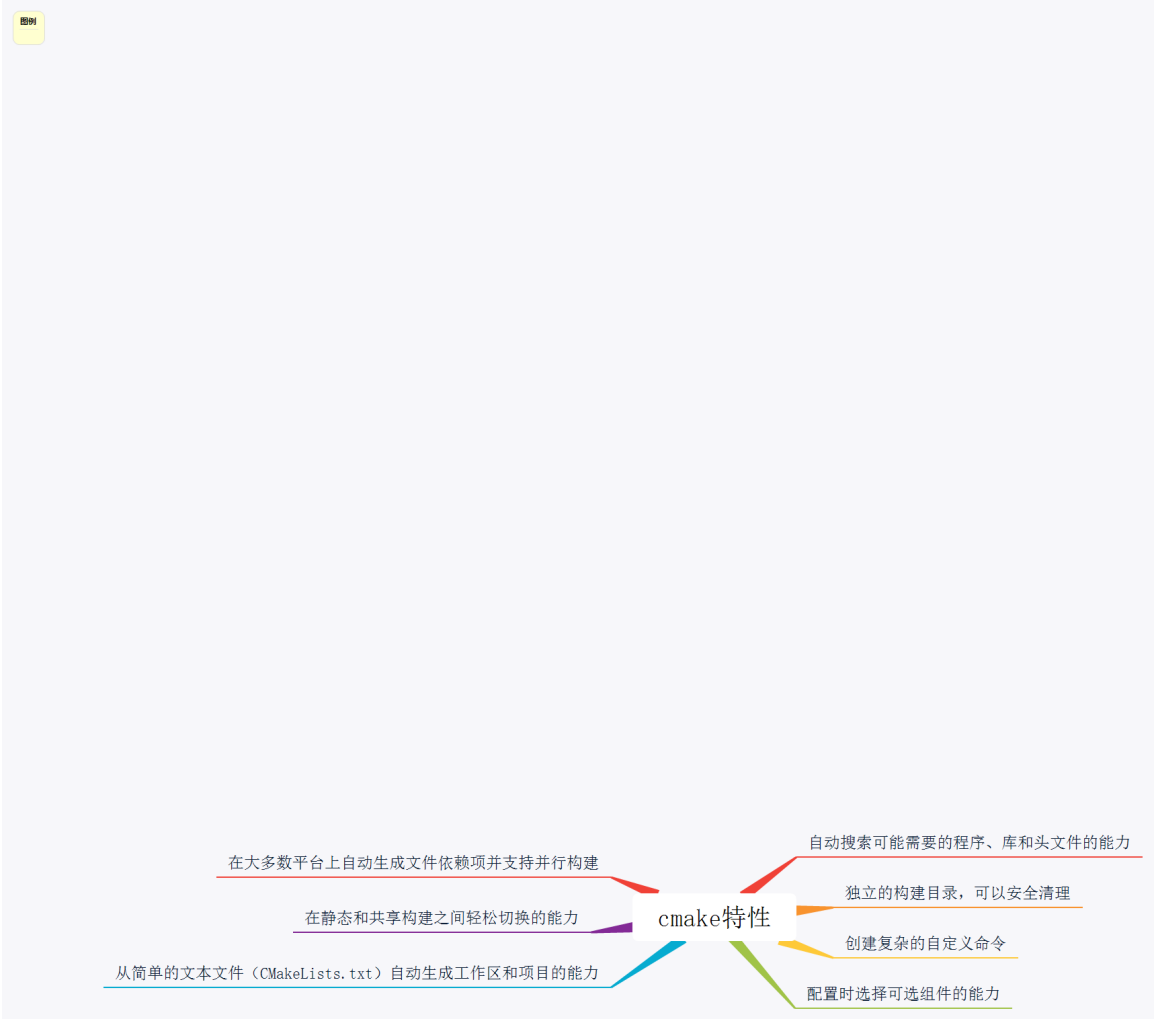
让产品可以快速地通过，同时保持关键测试合格

自动化测试，只要有一个测试用例不通过就不能集成

集成并不能删除发现的错误，而是让它们很容易和改正

1.2.3. 为什么是cmake

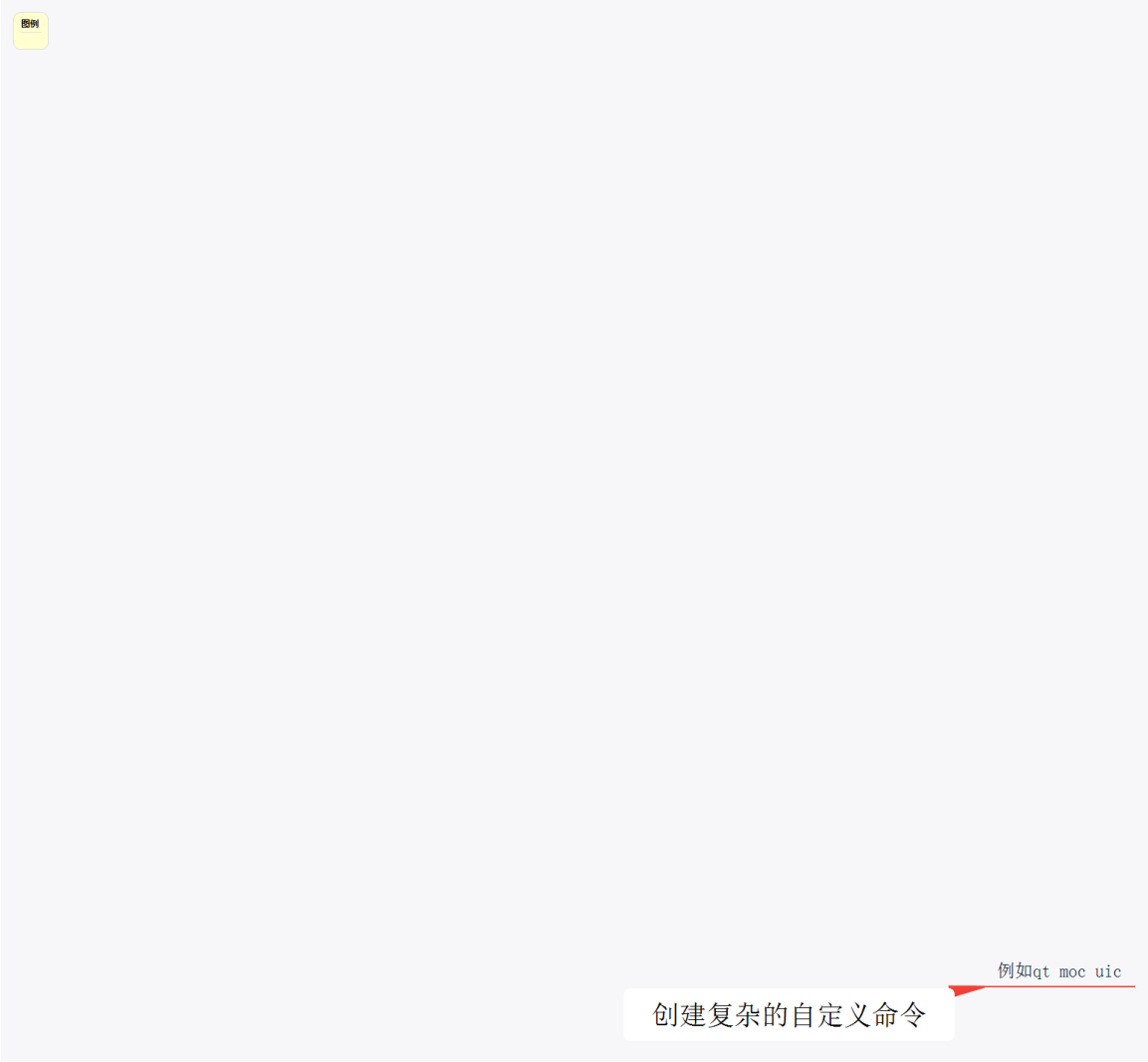
cmake特性



自动搜索可能需要的程序、库和头文件的能力

独立的构建目录，可以安全清理

创建复杂的自定义命令



例如qt moc uic

配置时选择可选组件的能力

从简单的文本文件（CMakeLists.txt）自动生成工作区和项目的能力

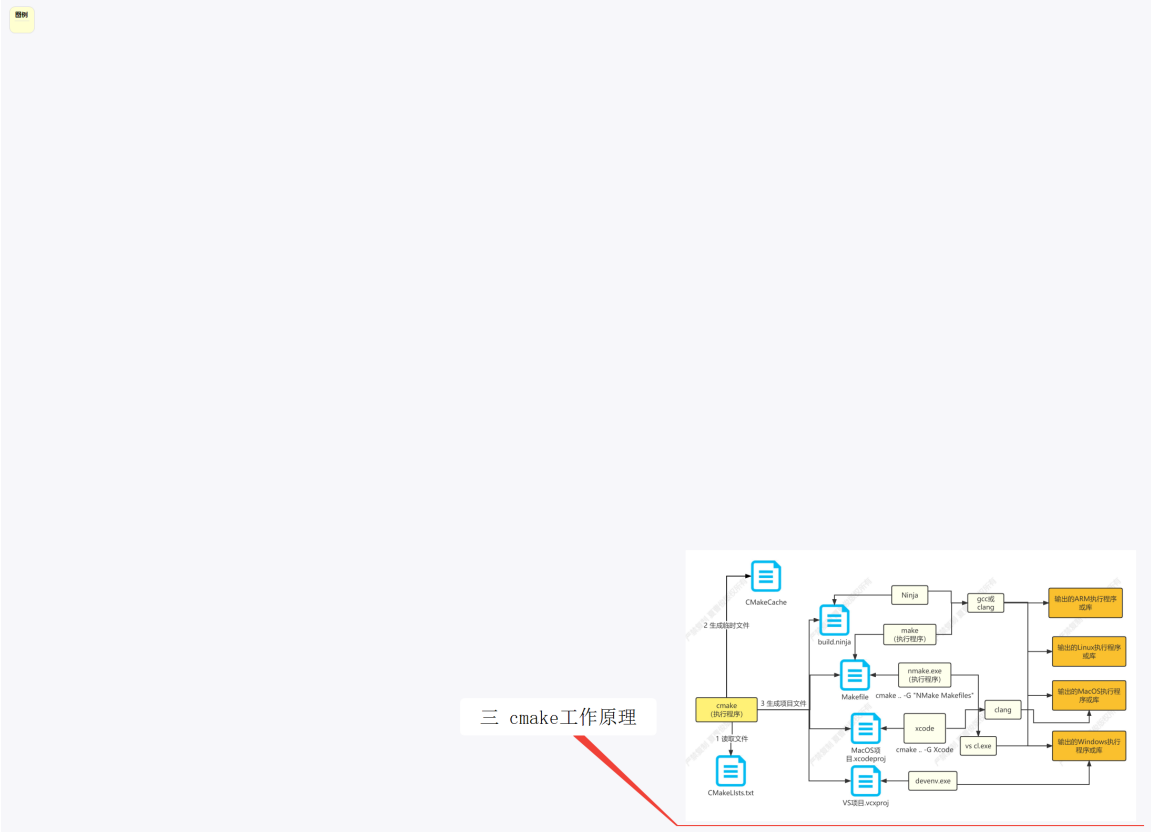
在静态和共享构建之间轻松切换的能力

在大多数平台上自动生成文件依赖项并支持并行构建

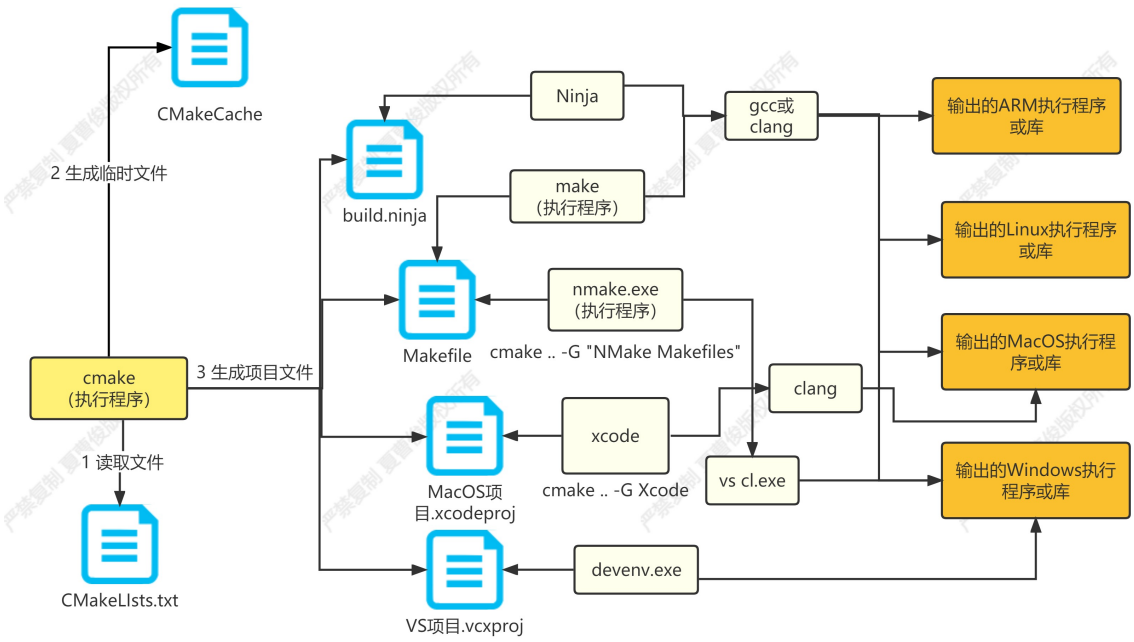
每个 IDE 都支持 CMake（ CMake 支持几乎所有IDE）

使用 CMake 的软件包比任何其他系统都多

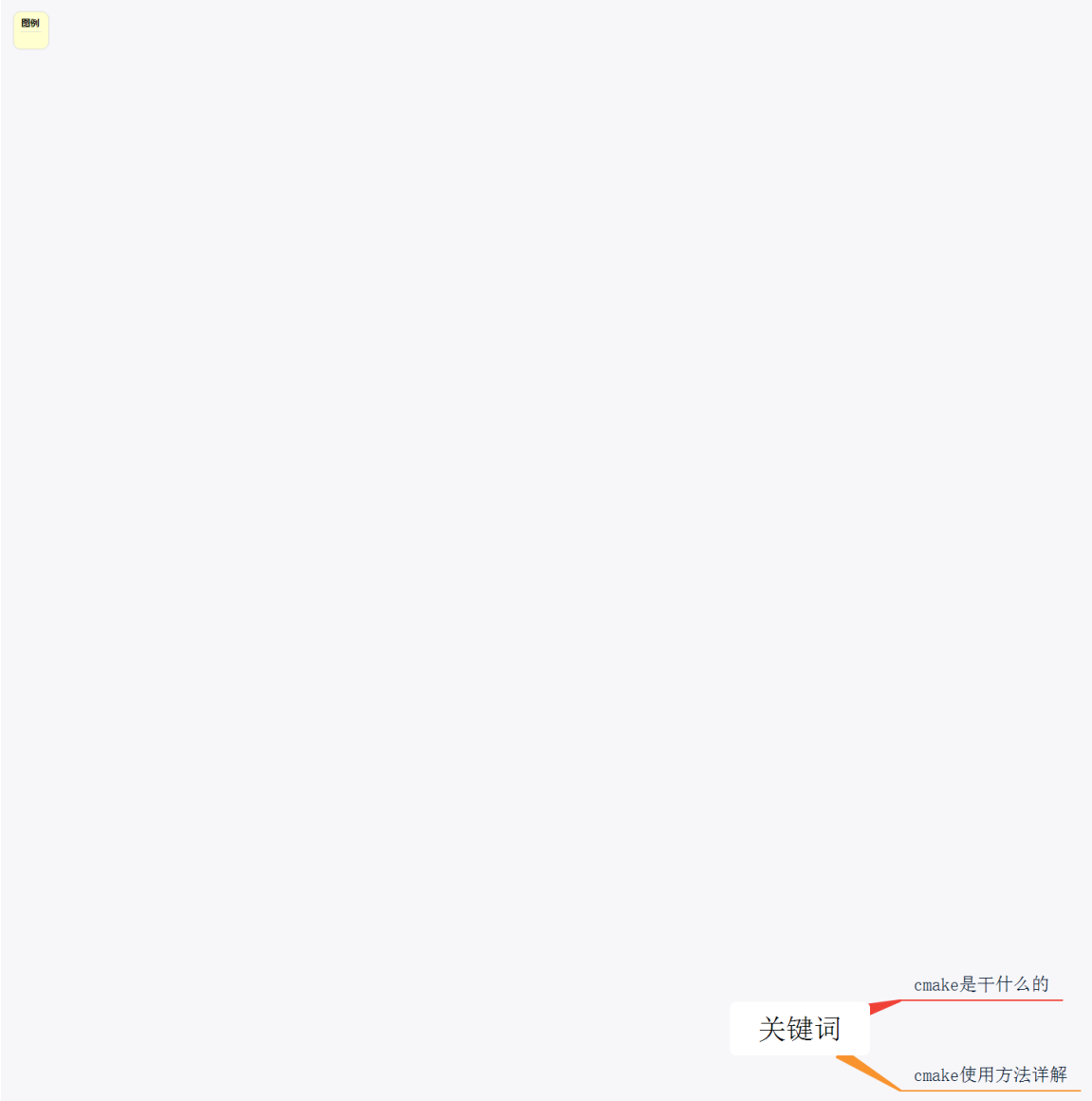
1.3. 三 cmake工作原理



1.3.1.



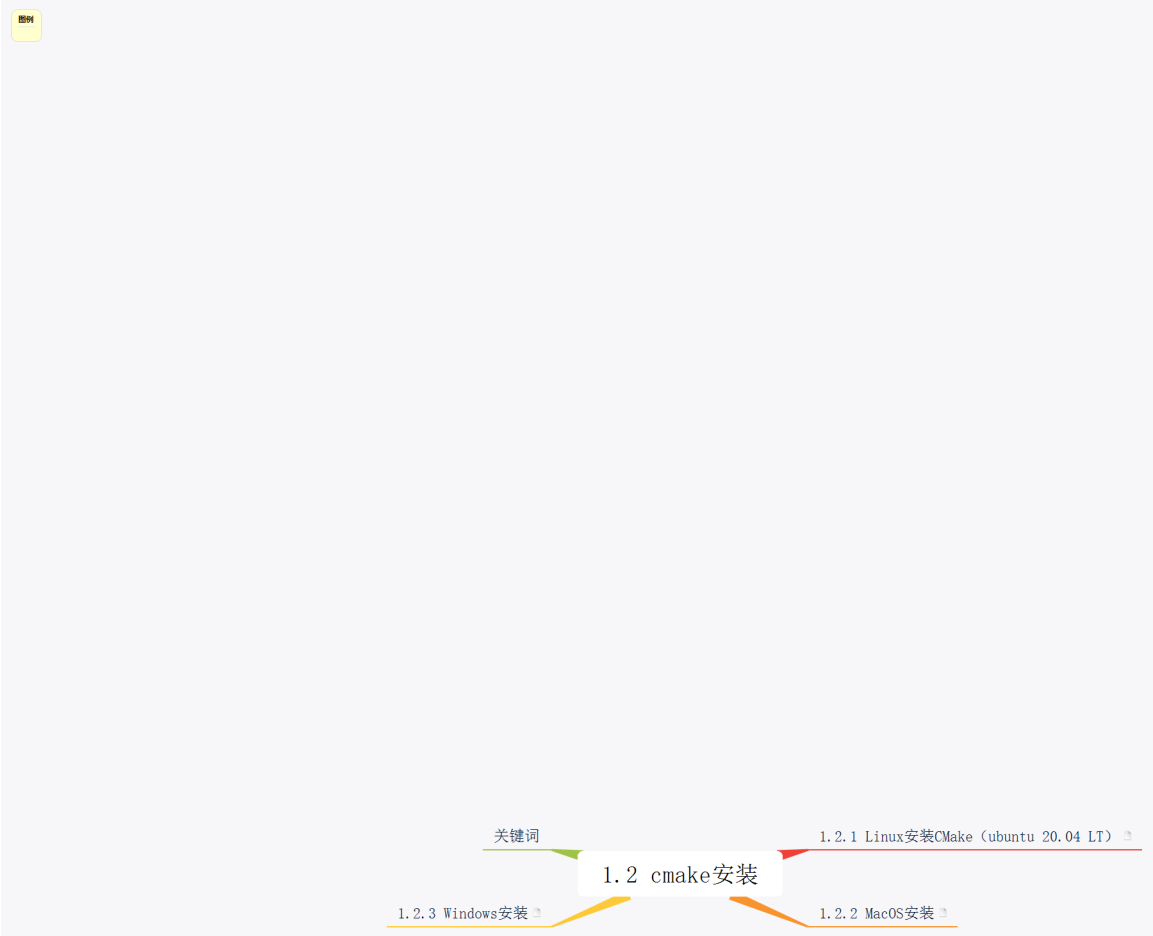
1.4. 关键词



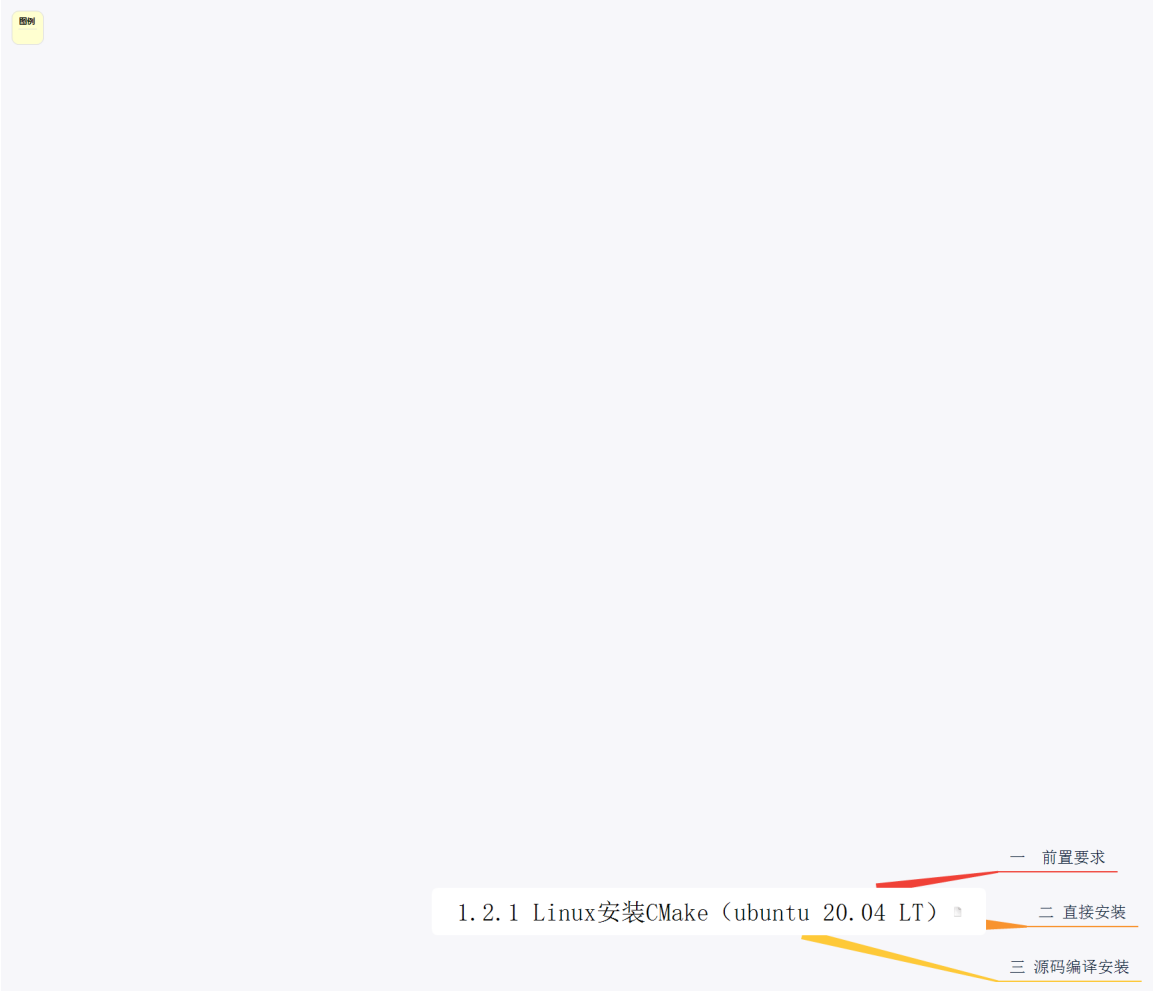
1.4.1. cmake是干什么的

1.4.2. cmake使用方法详解

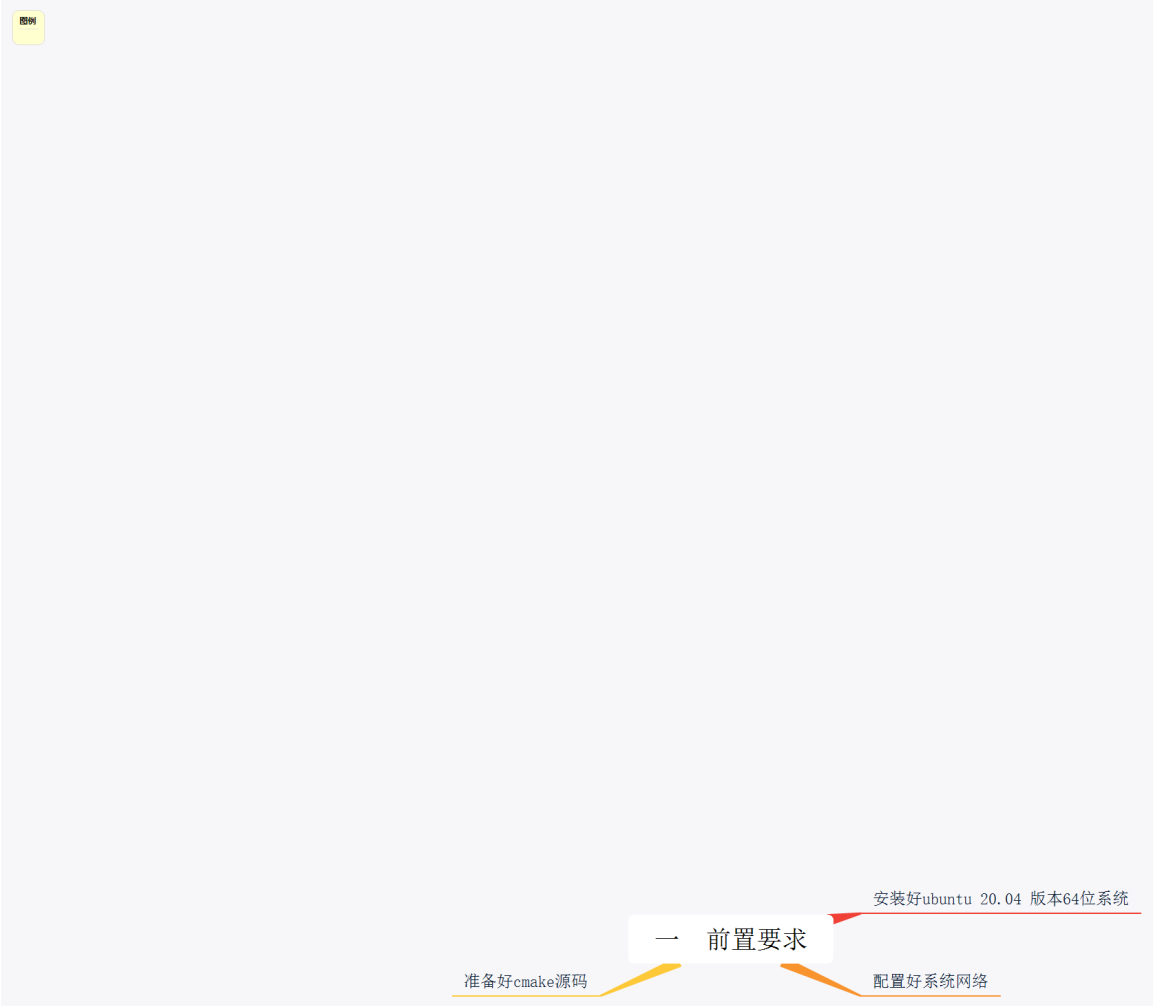
2. 1.2 cmake安装



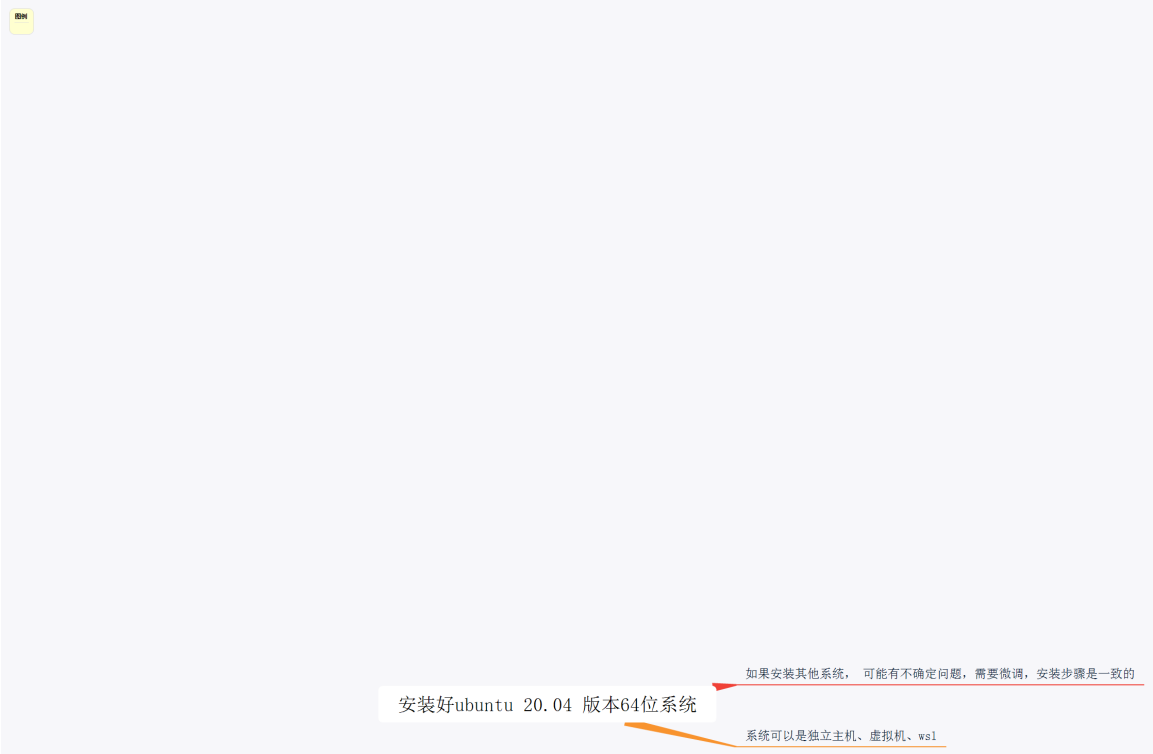
2.1. 1.2.1 Linux安装CMake（ubuntu 20.04 LT）



2.1.1. 一 前置要求



安装好ubuntu 20.04 版本64位系统



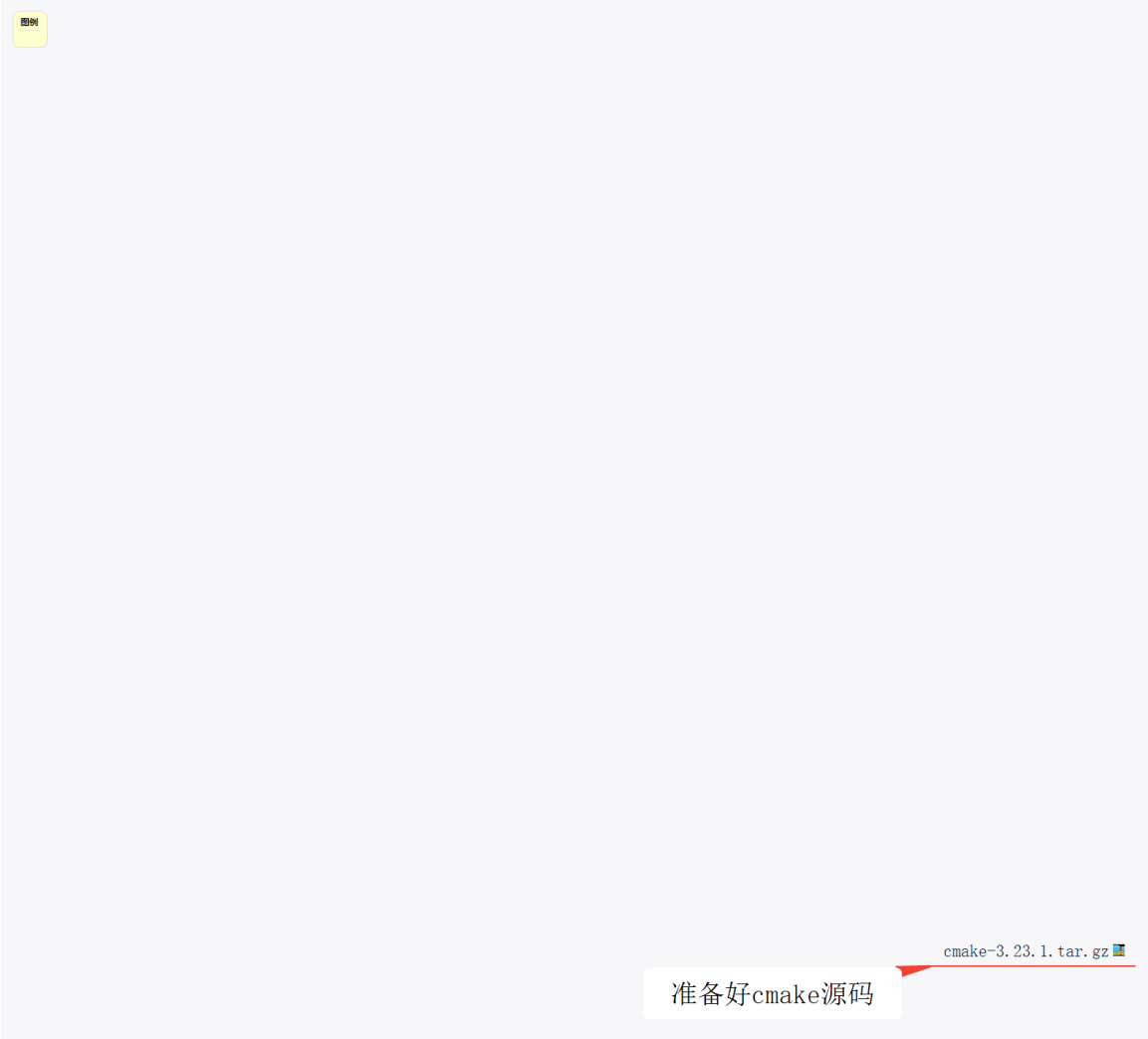
如果安装其他系统， 可能有不确定问题，需要微调，安装步骤是一致的

系统可以是独立主机、虚拟机、wsl

配置好系统网络

需要在线安装编译工具

准备好cmake源码



[cmake-3.23.1.tar.gz]

提取码：1234 链接：https://pan.baidu.com/s/1AAfC b3oTA8cguIRg8wR-zg

2.1.2. 二 直接安装

apt install cmake

二进制安装

2.1.3. 三 源码编译安装

1 安装编译工具和依赖库

sudo apt install g++

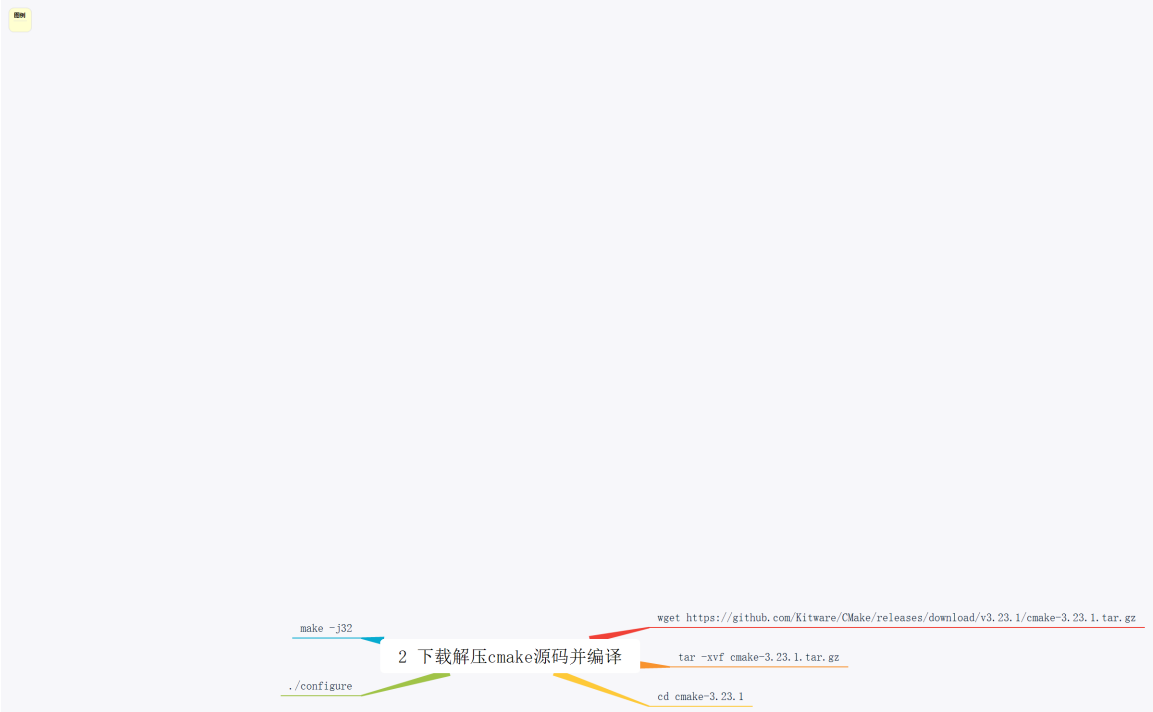
sudo apt install make

apt install ninja-build

apt install unzip

apt install libssl-dev

2 下载解压cmake源码并编译



wget https://github.com/Kitware/CMake/releases/download/v3.23.1/cmake-3.23.1.tar.gz

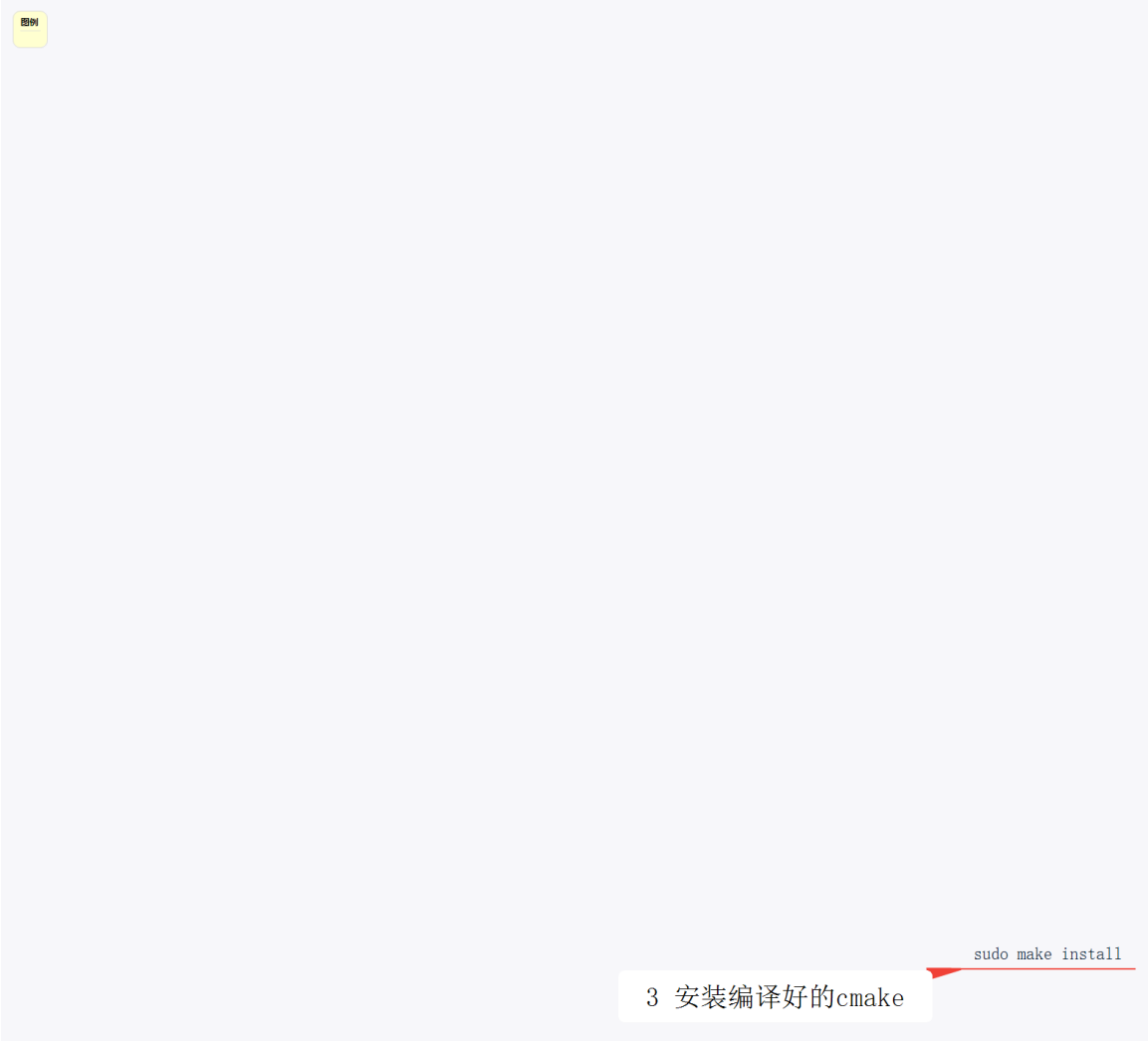
tar -xvf cmake-3.23.1.tar.gz

cd cmake-3.23.1

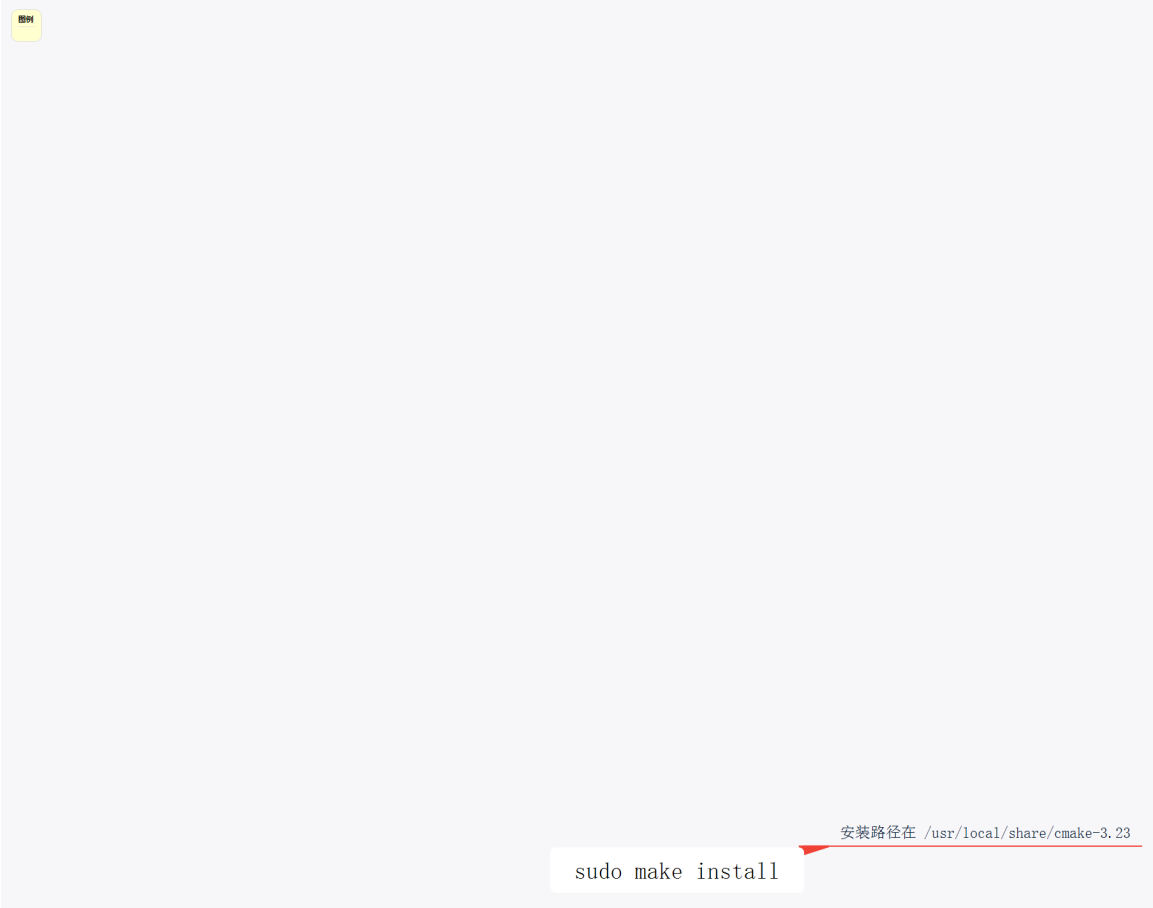
./configure

make -j32

3 安装编译好的cmake

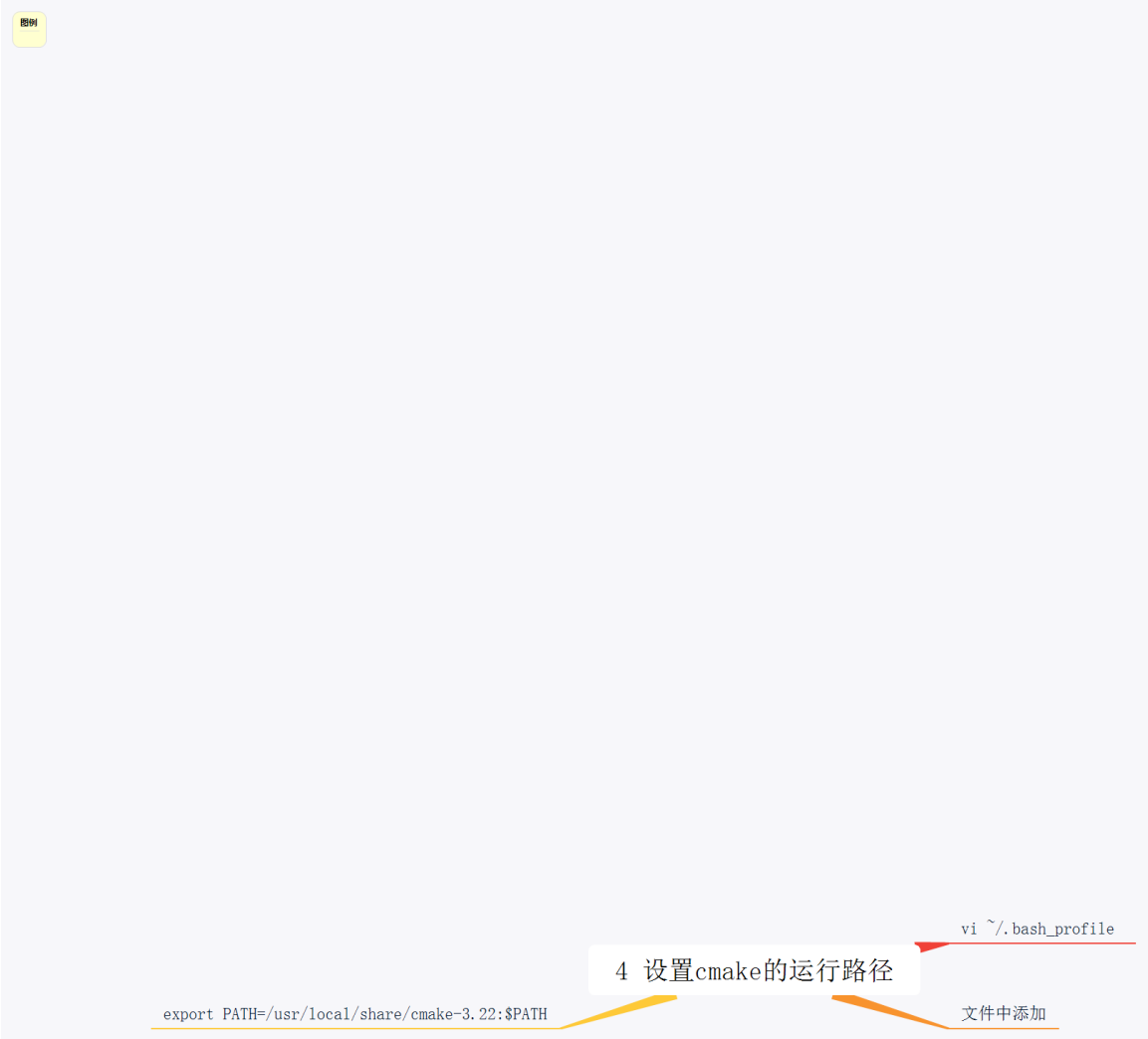


sudo make install



安装路径在 /usr/local/share/cmake-3.23

4 设置cmake的运行路径

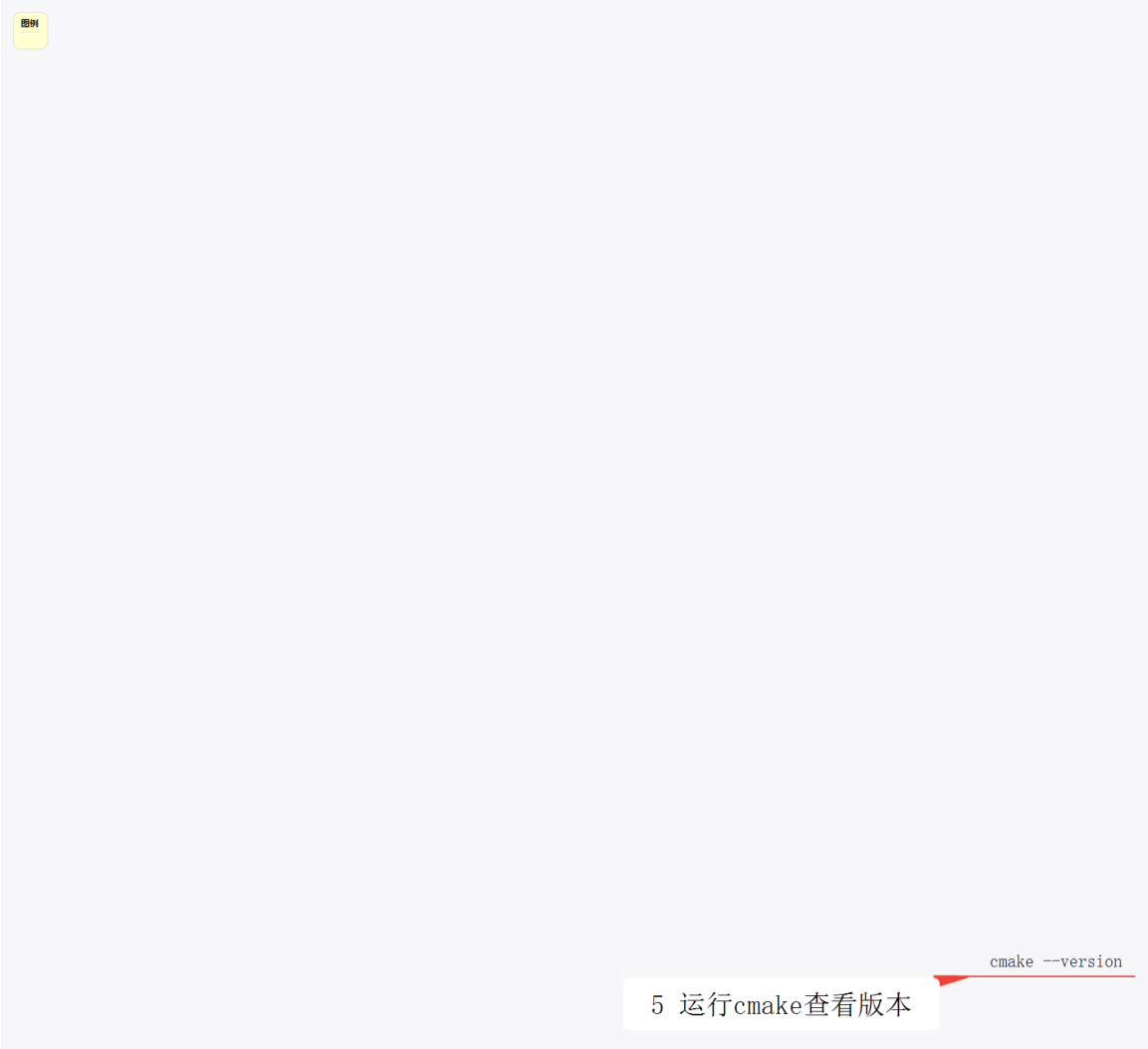


vi ~/.bash_profile

文件中添加

export PATH=/usr/local/share/cmake-3.22:$PATH

5 运行cmake查看版本



cmake --version

cmake version 3.23.1

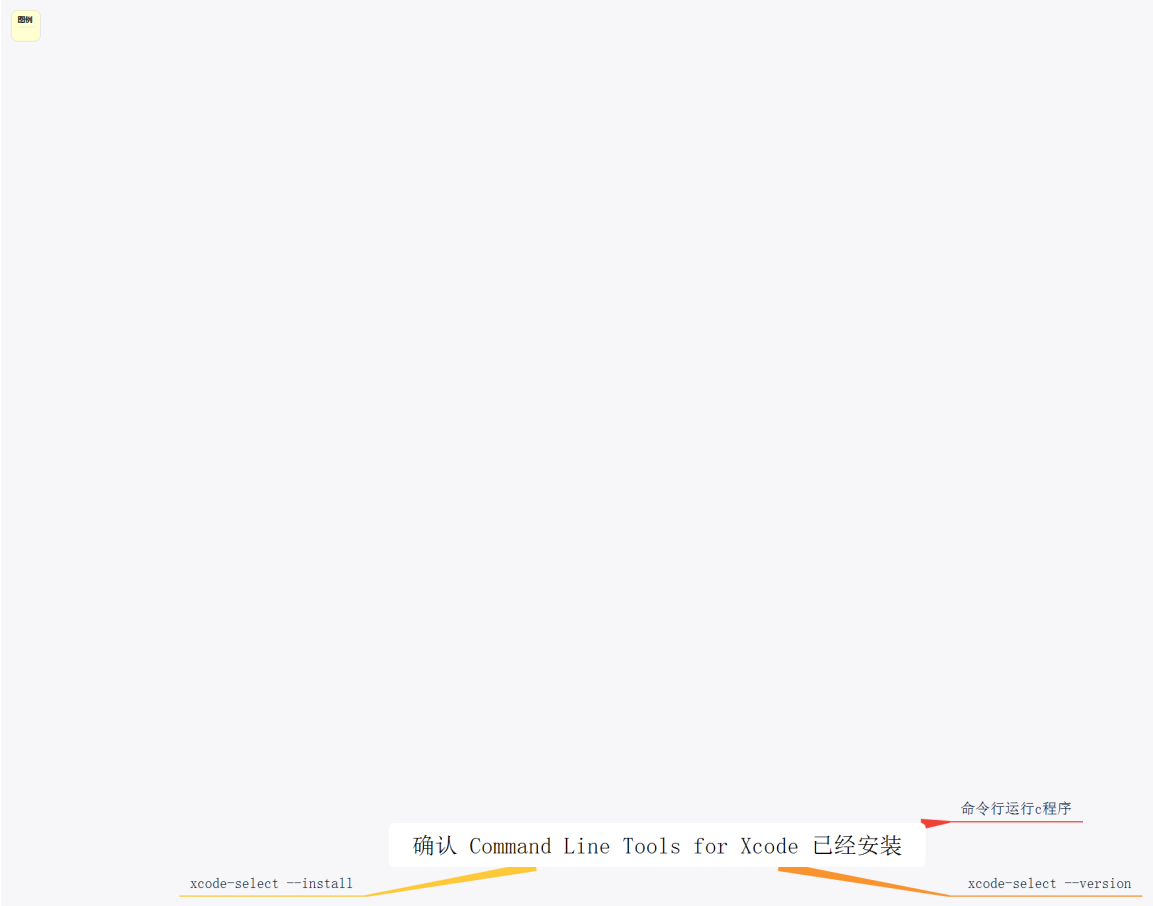
CMake suite maintained and supported by Kitware (kitware.com/cmake).

2.2. 1.2.2 MacOS安装

2.2.1. 一 前置要求

安装好xcode开发工具 （clang）

确认 Command Line Tools for Xcode 已经安装

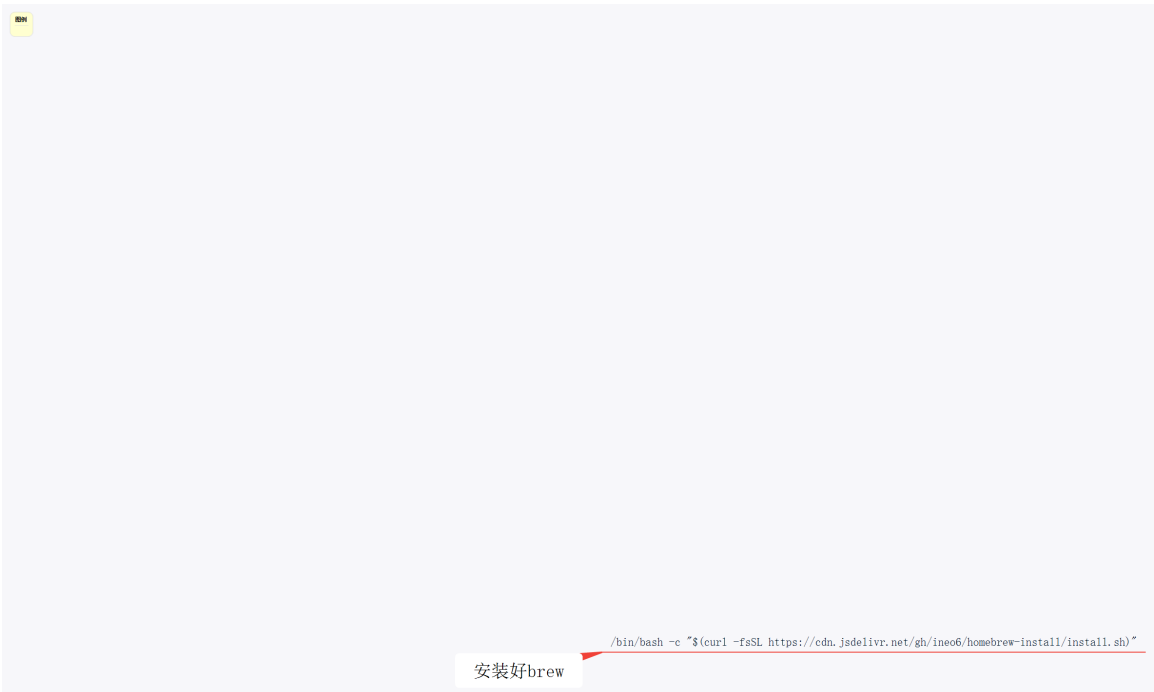


命令行运行c程序

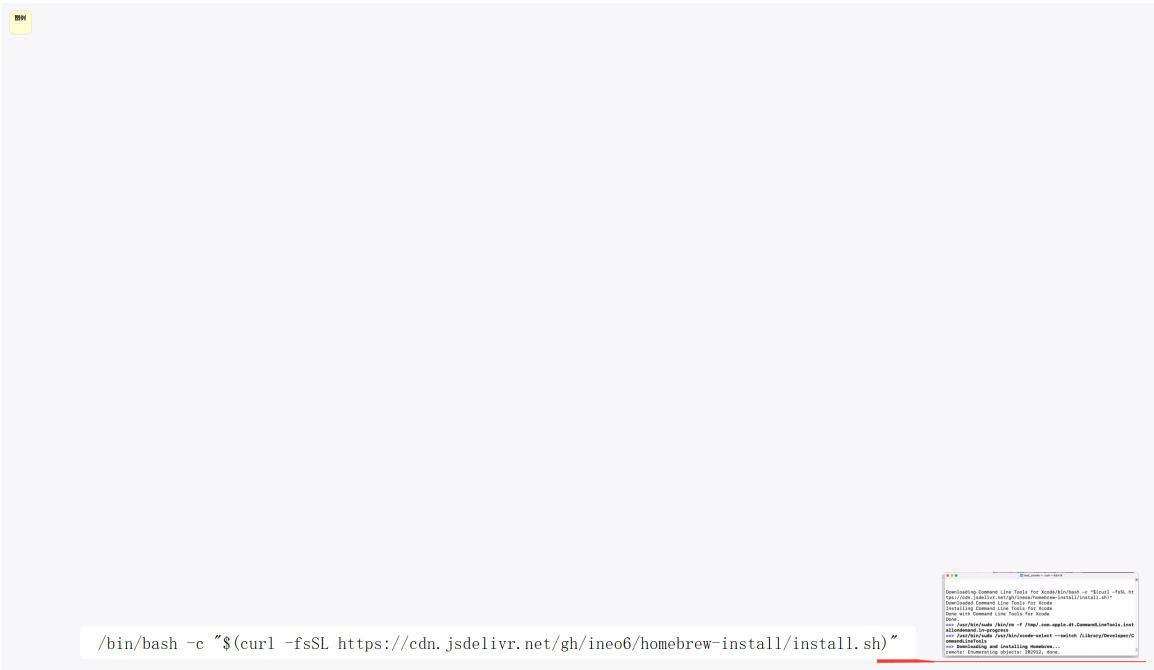
xcode-select --version

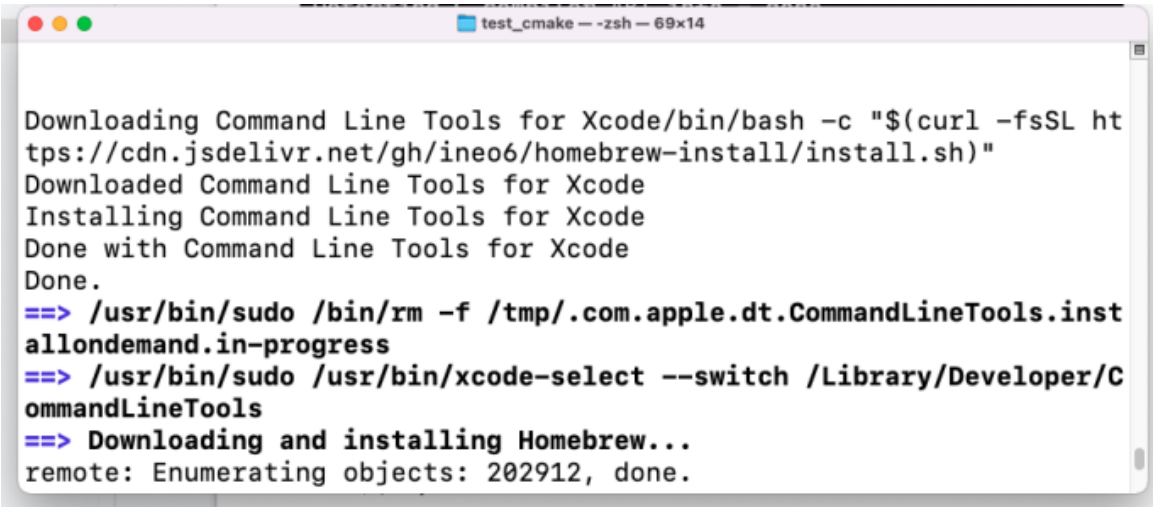
xcode-select --install

安装好brew

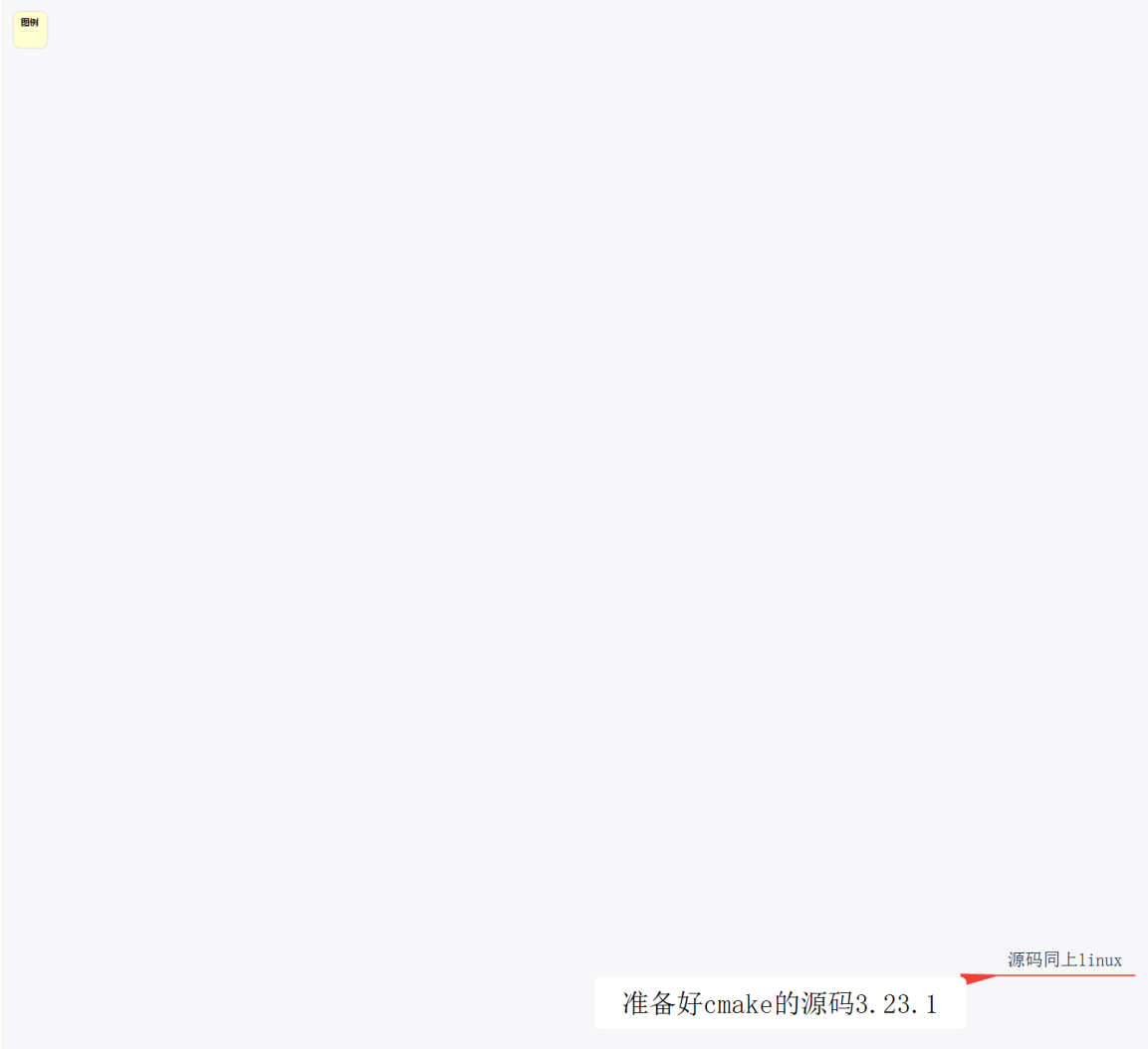


/bin/bash -c "$(curl -fsSL https://cdn.jsdelivr.net/gh/ineo6/homebrew-install/install.sh)"





准备好cmake的源码3.23.1

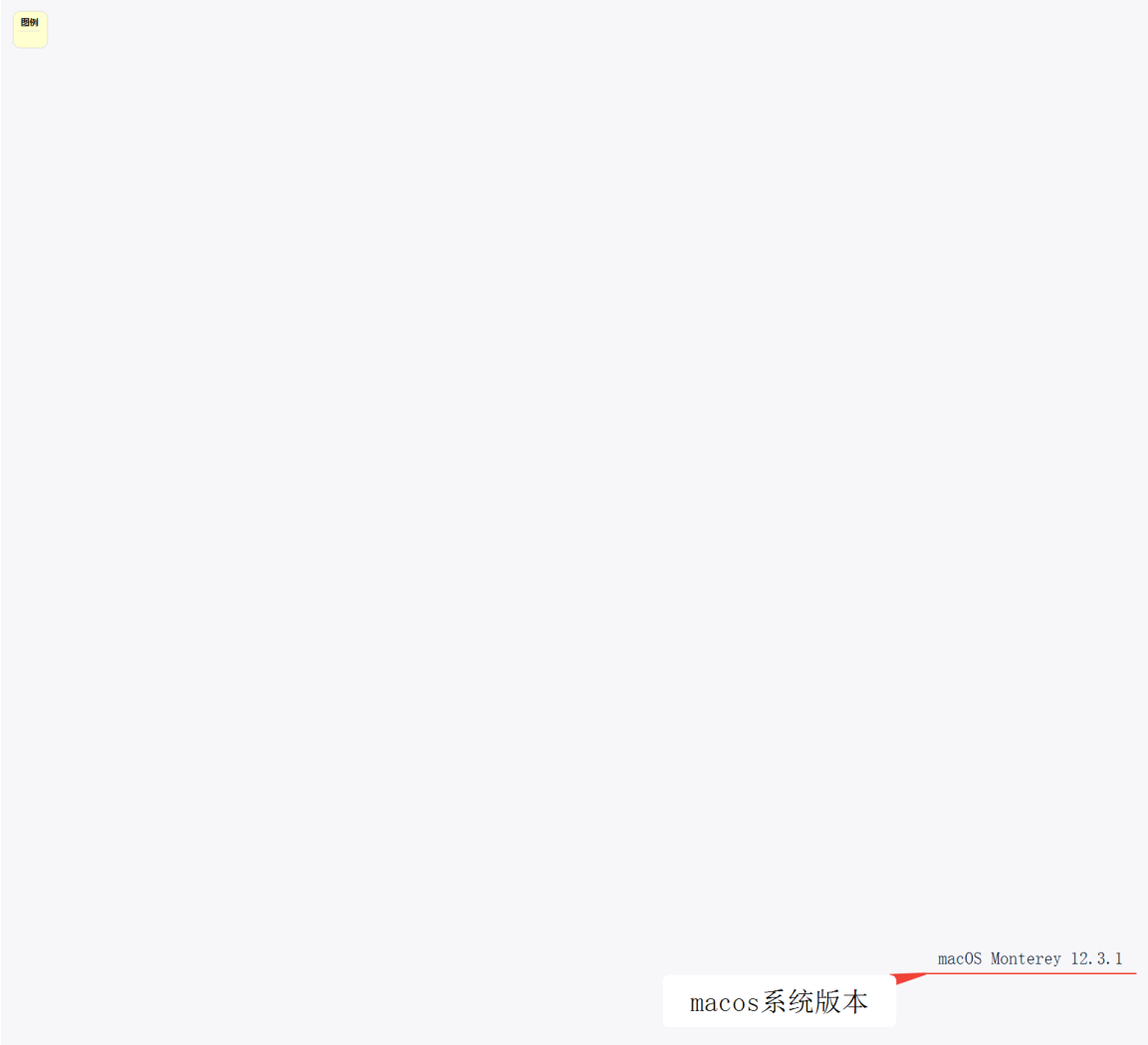


源码同上linux

提取码：1234 链接：https://pan.baidu.com/s/1AAfC b3oTA8cguIRg8wR-zg

cmake.org.cn

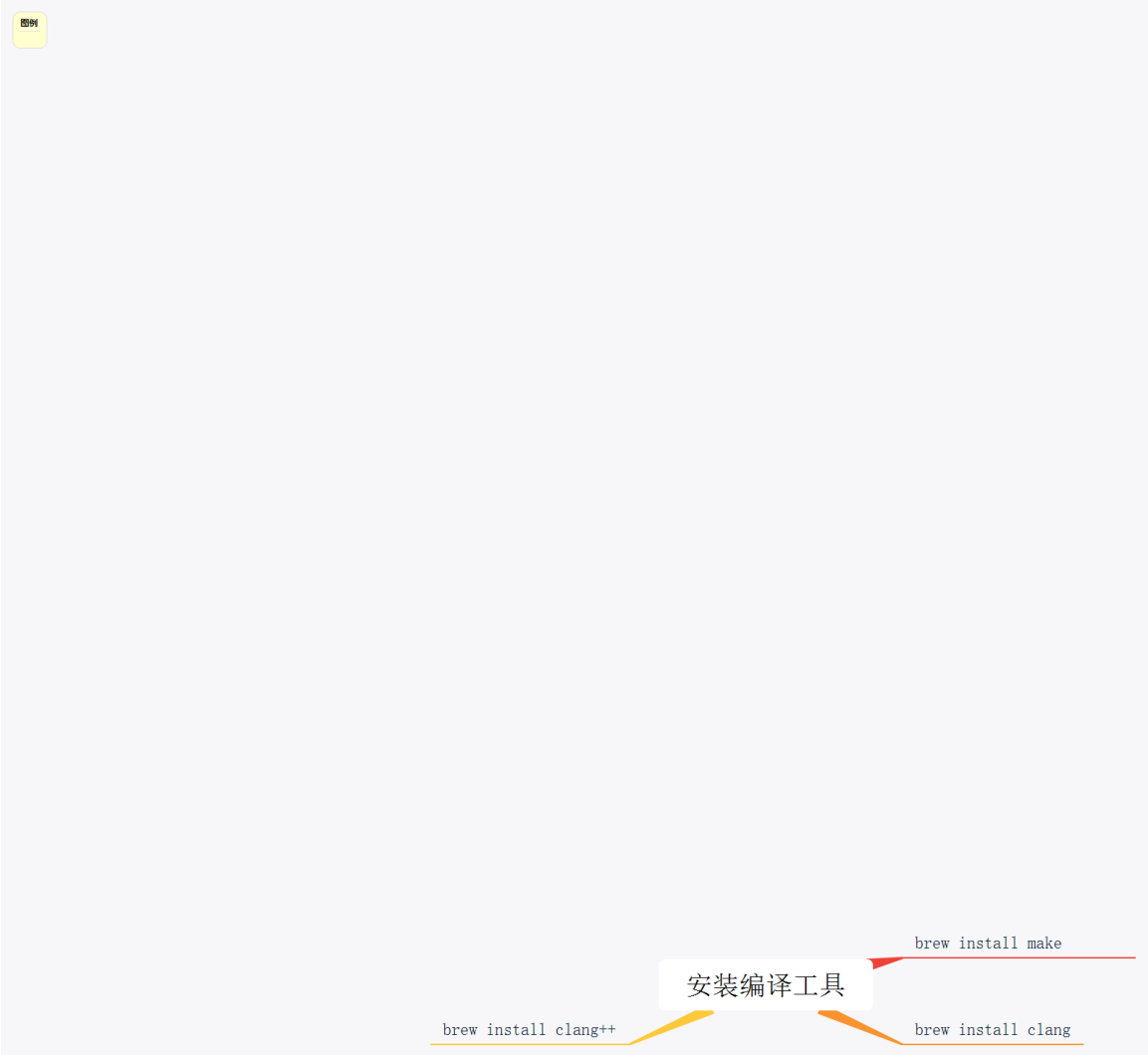
macos系统版本



macOS Monterey 12.3.1

2.2.2. 二 源码编译安装

安装编译工具

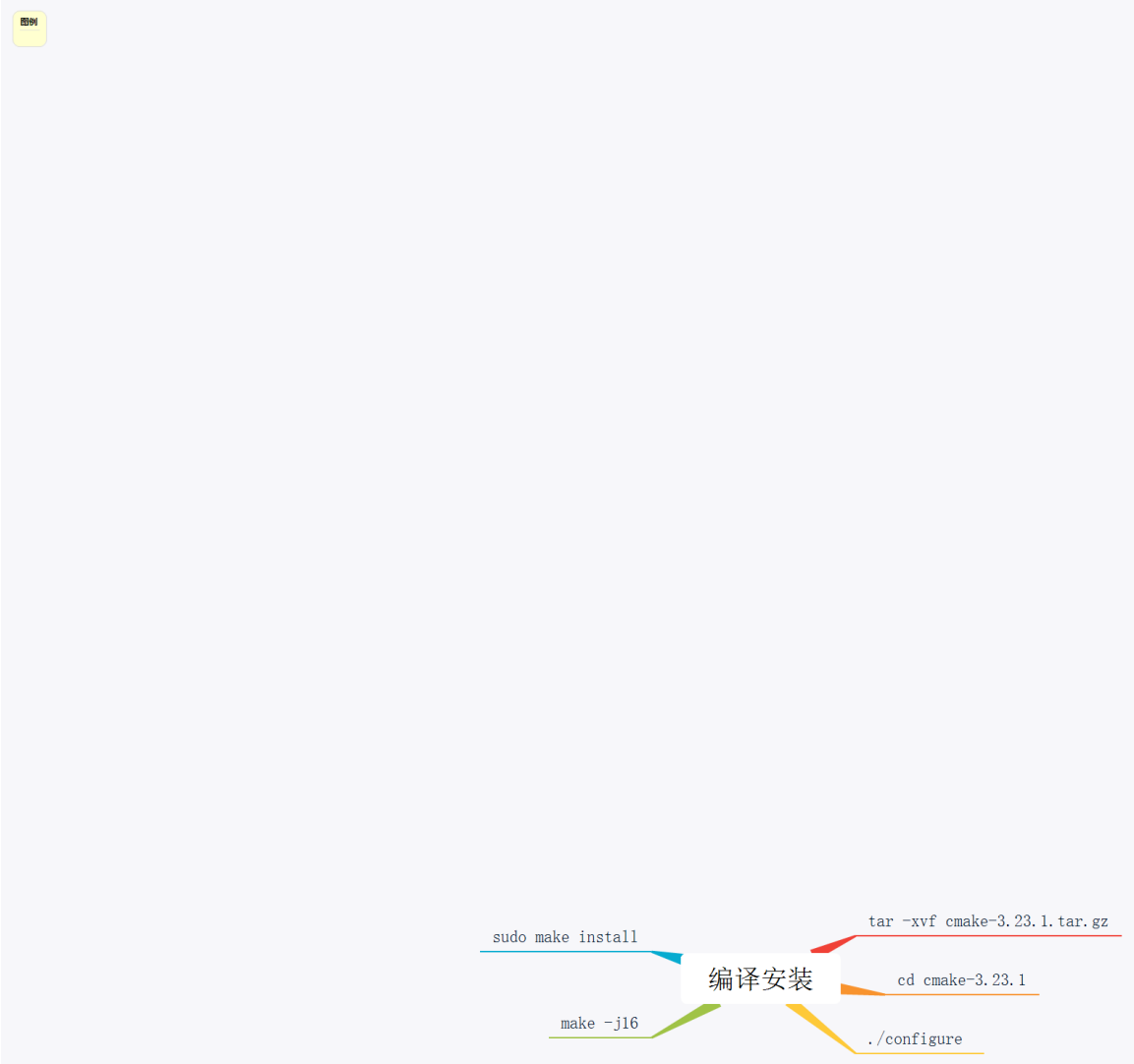


brew install make

brew install clang

brew install clang++

编译安装



tar -xvf cmake-3.23.1.tar.gz

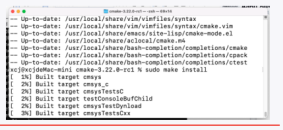
cd cmake-3.23.1

./configure

make -j16

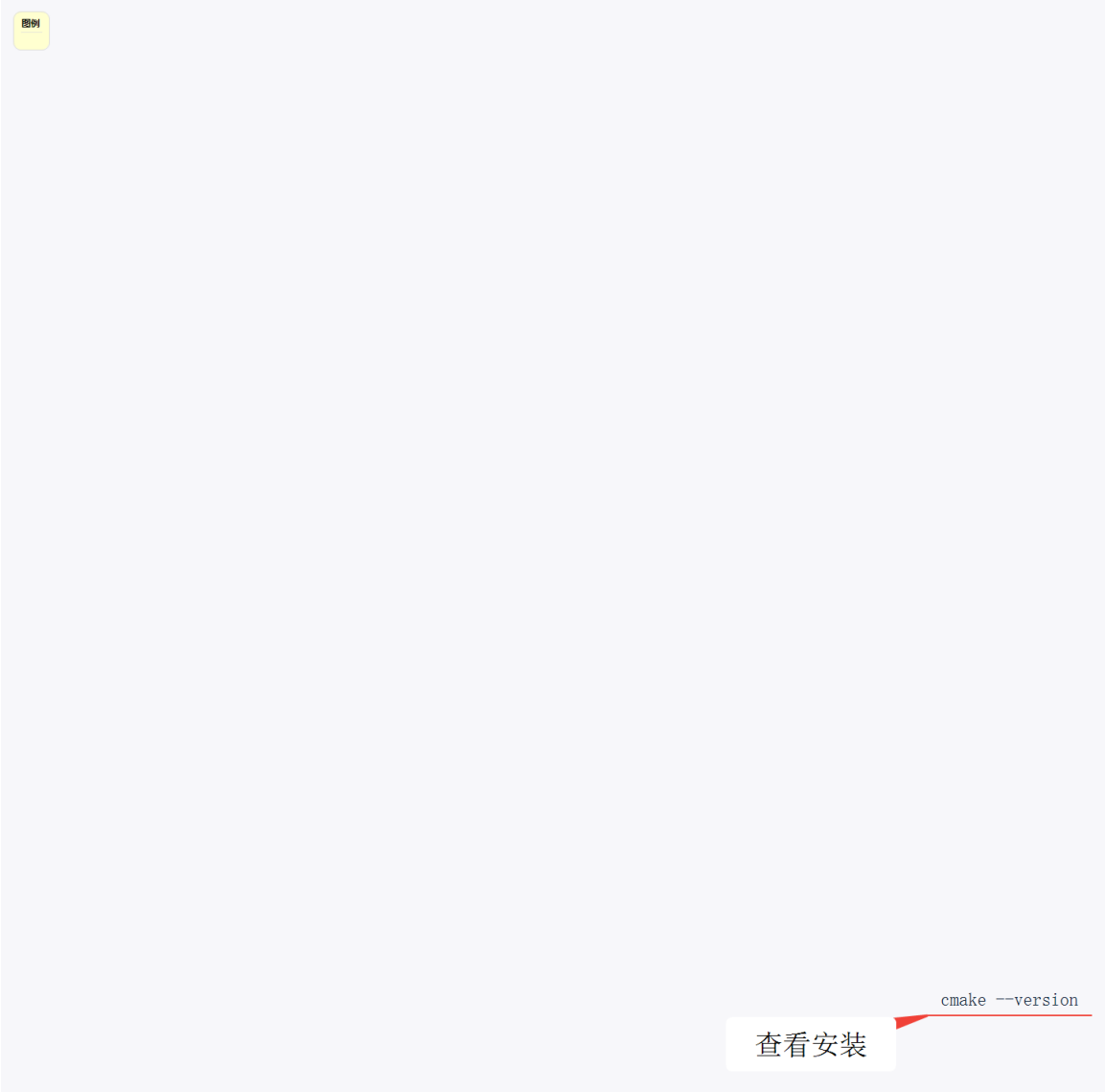
sudo make install

sudo make install

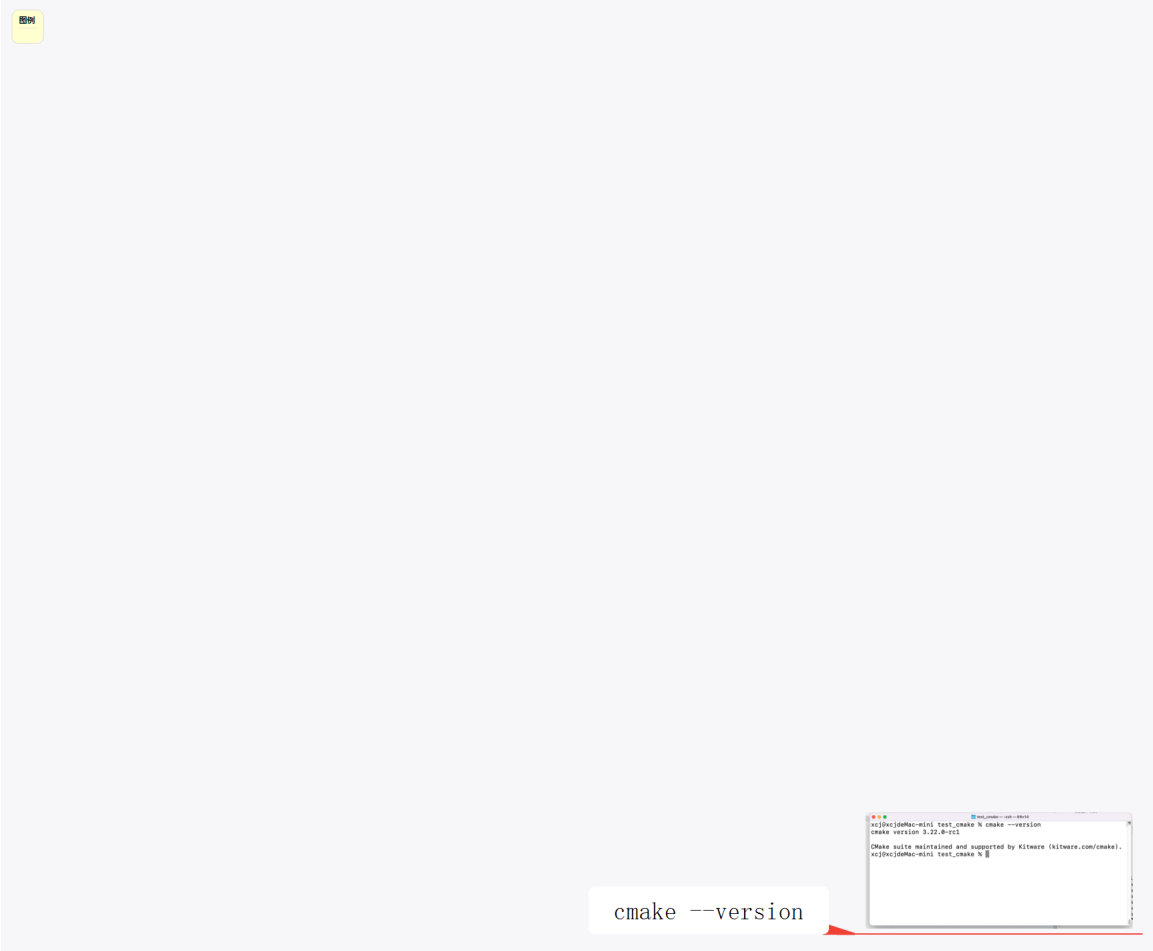


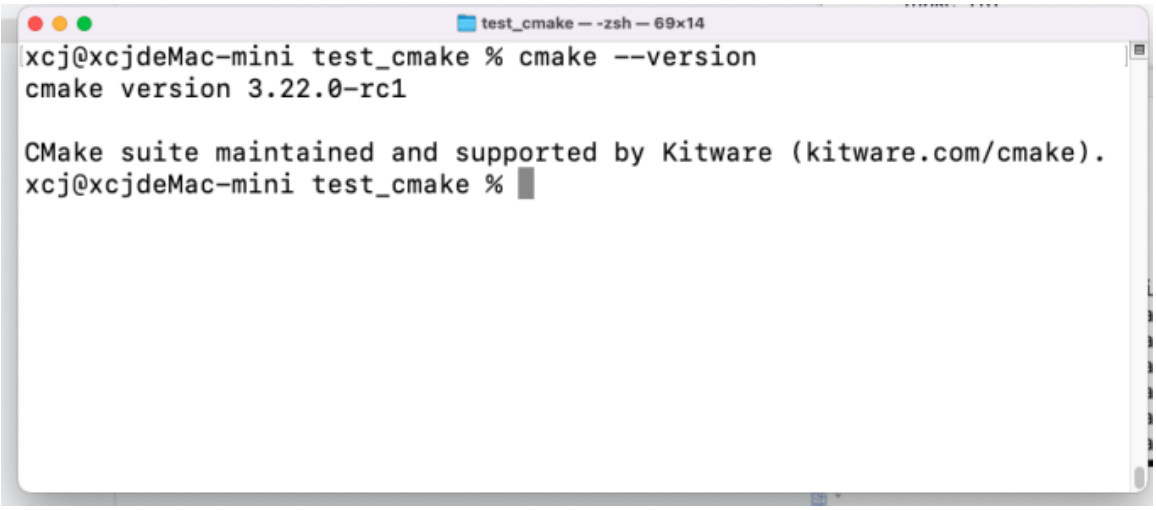
```
cmake-3.22.0-rc1 — -zsh — 69×14
-- Up-to-date: /usr/local/share/vim/vimfiles/syntax
-- Up-to-date: /usr/local/share/vim/vimfiles/syntax/cmake.vim
-- Up-to-date: /usr/local/share/emacs/site-lisp/cmake-mode.el
-- Up-to-date: /usr/local/share/aclocal/cmake.m4
-- Up-to-date: /usr/local/share/bash-completion/completions/cmake
-- Up-to-date: /usr/local/share/bash-completion/completions/cpack
-- Up-to-date: /usr/local/share/bash-completion/completions/ctest
xcj@xcjdeMac-mini cmake-3.22.0-rc1 % sudo make install
[  1%] Built target cmsys
[  2%] Built target cmsys_c
[  2%] Built target cmsysTestsC
[  2%] Built target testConsoleBufChild
[  2%] Built target cmsysTestDynload
[  3%] Built target cmsysTestsCxx
```

查看安装



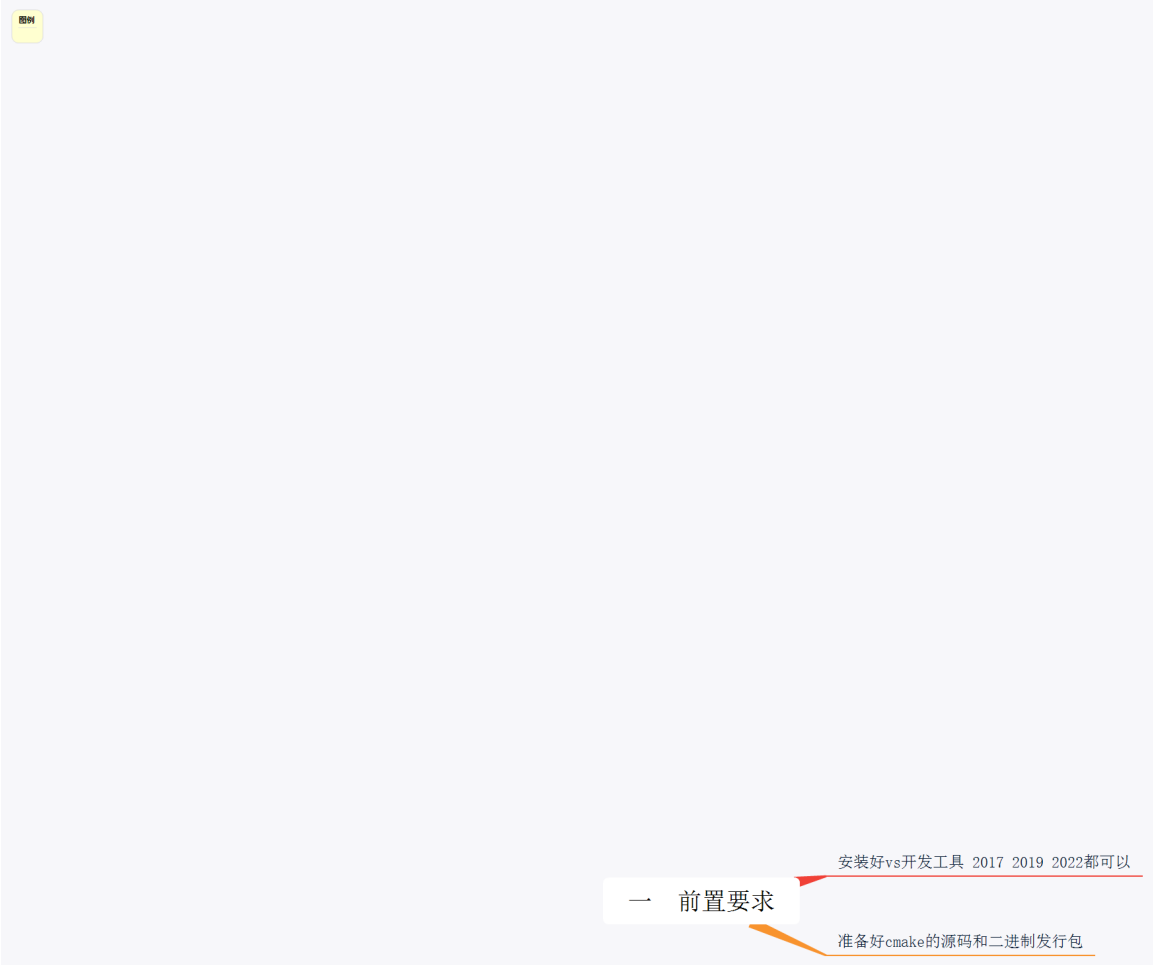
cmake --version





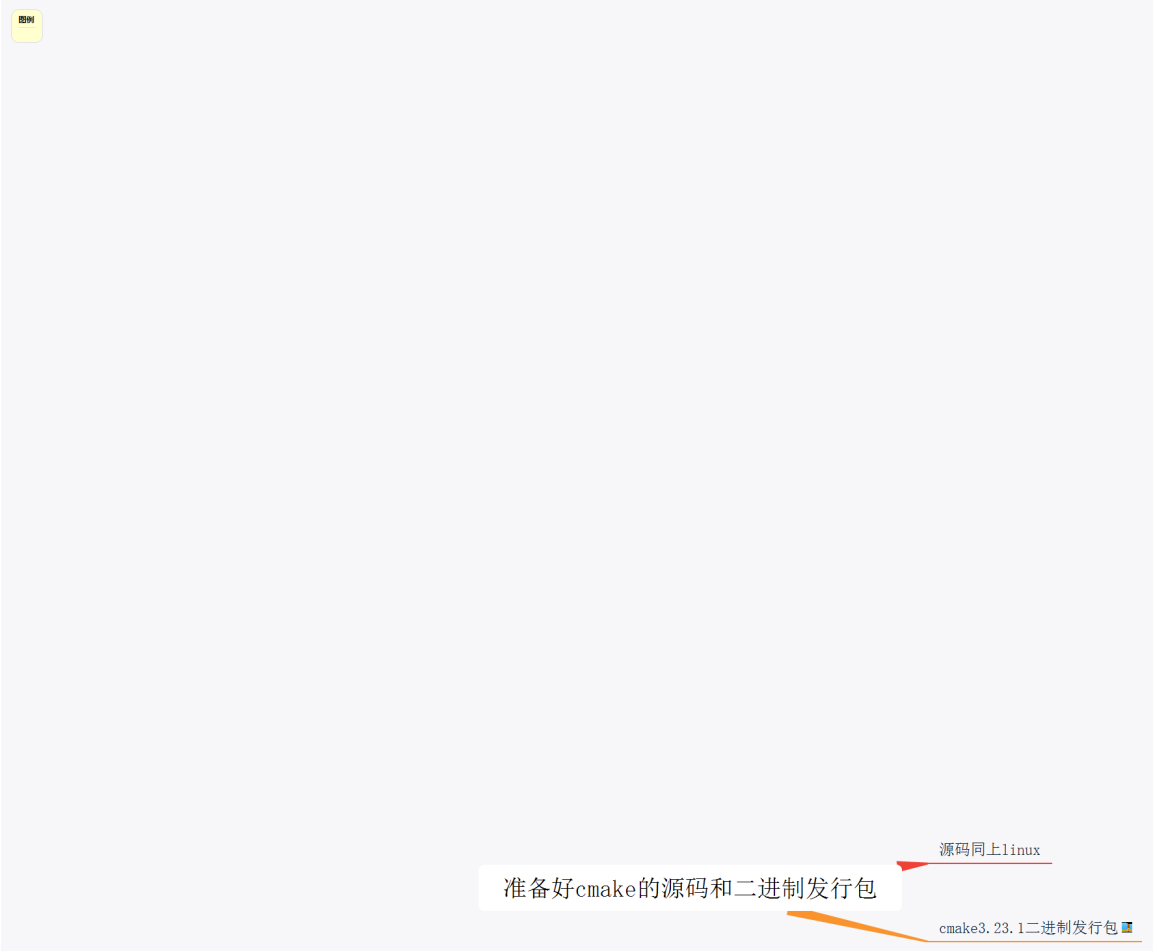
2.3. 1.2.3 Windows安装

2.3.1. 一 前置要求



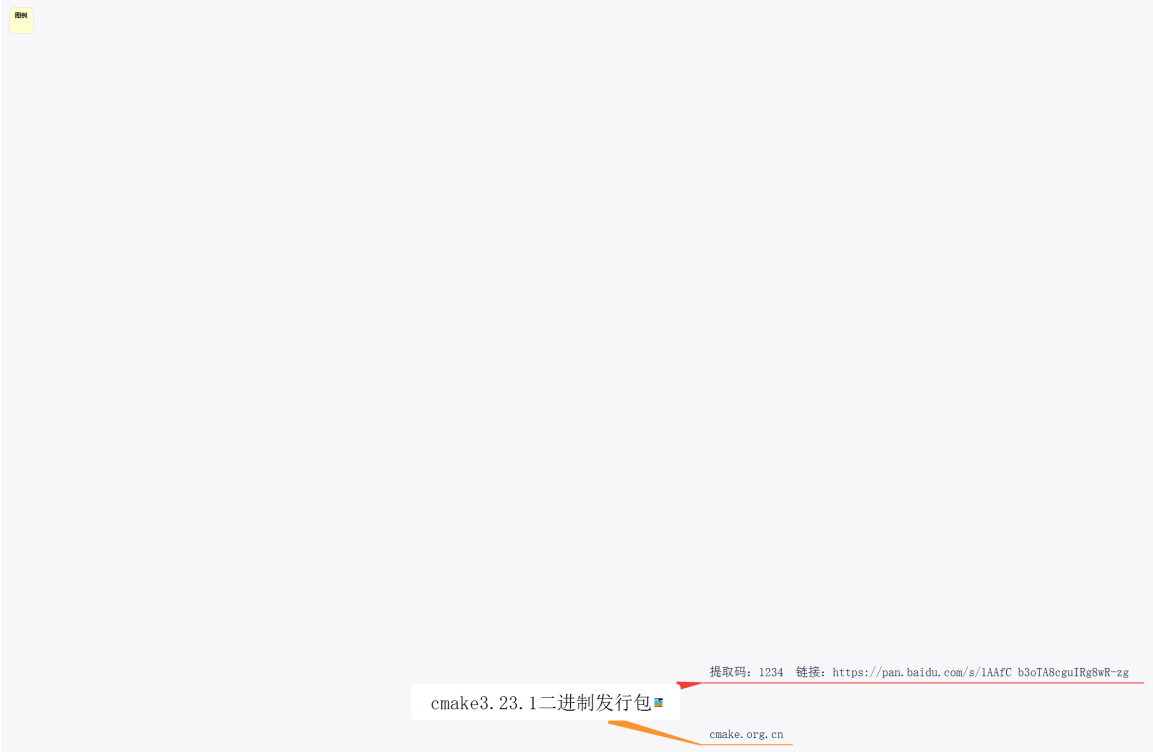
安装好vs开发工具 2017 2019 2022都可以

准备好cmake的源码和二进制发行包



源码同上linux

cmake3.23.1二进制发行包

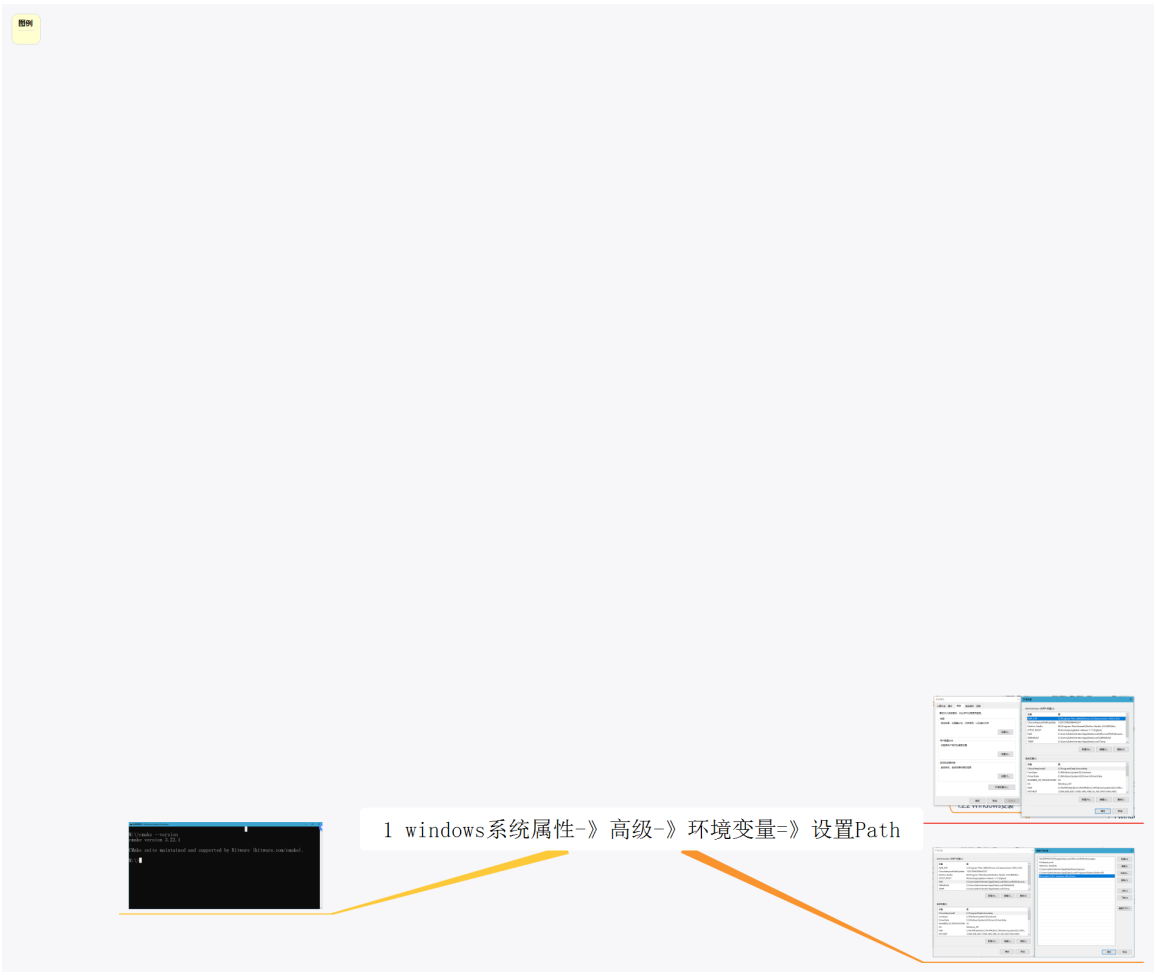


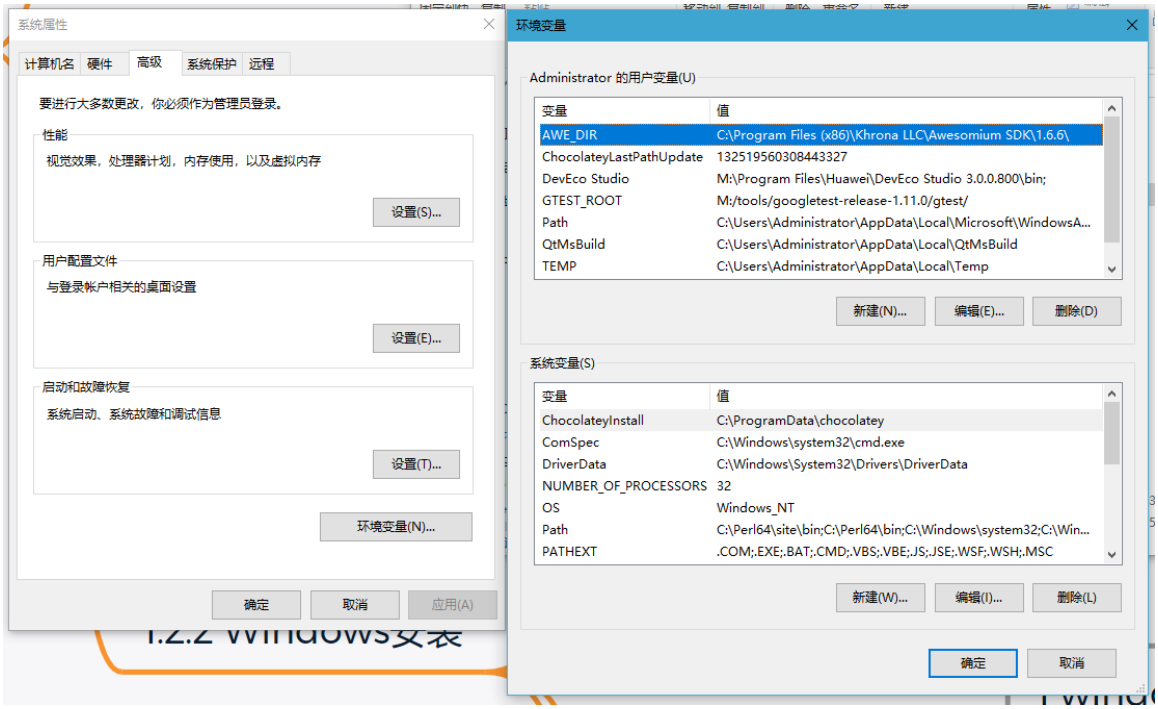
提取码：1234 链接：https://pan.baidu.com/s/1AAfC b3oTA8cguIRg8wR-zg

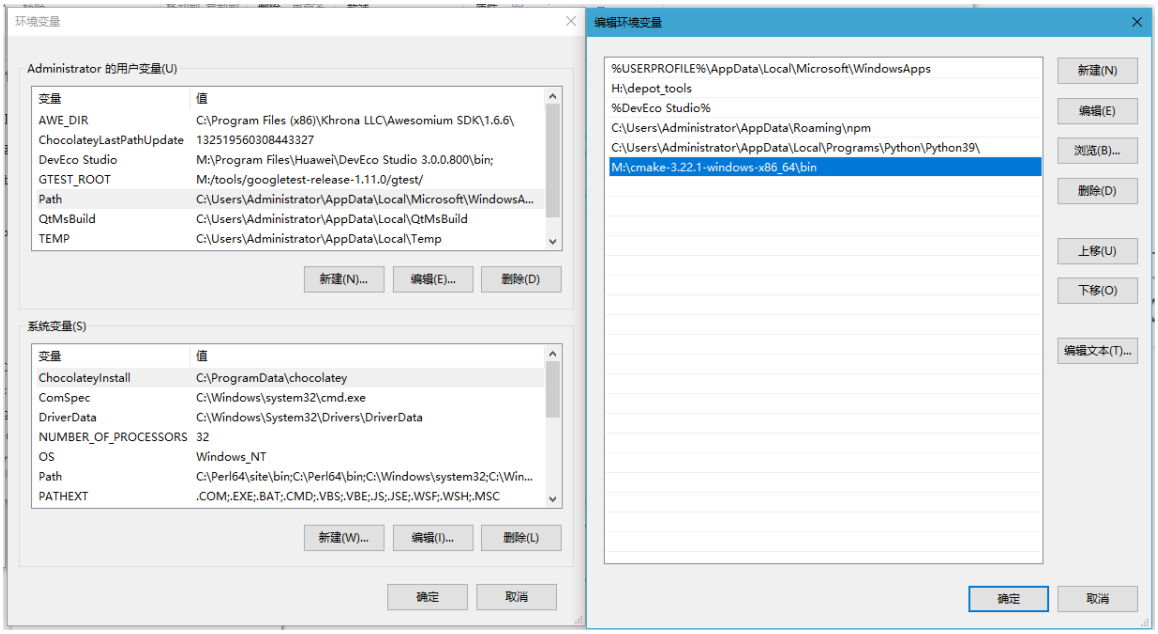
cmake.org.cn

2.3.2. 二 发布文件安装

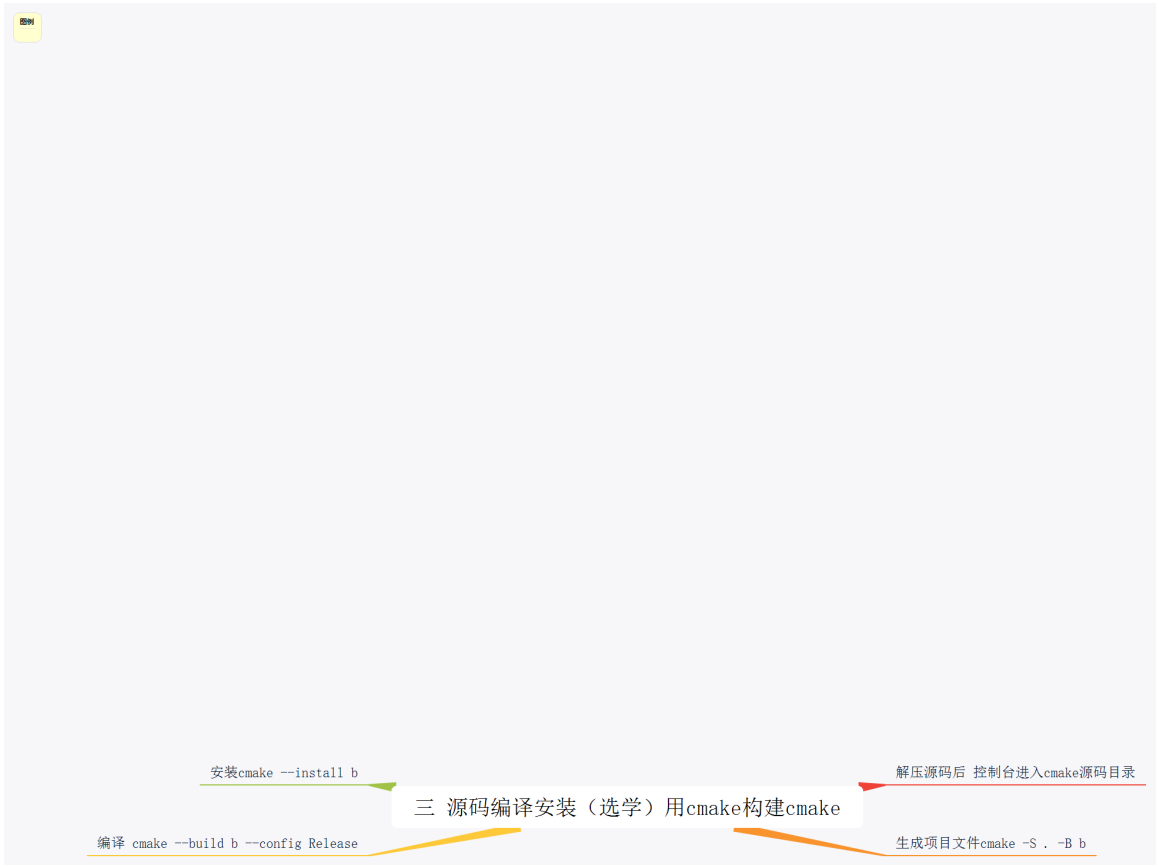
1 windows系统属性-》高级-》环境变量=》设置Path


图例
1 windows系统属性-》高级-》环境变量=》设置Path


系统属性
计算机名 硬件 高级 系统保护 远程
要进行大多数更改，你必须作为管理员登录。
性能
视觉效果，处理器计划，内存使用，以及虚拟内存
设置(S)...
用户配置文件
与登录帐户相关的桌面设置
设置(E)...
启动和故障恢复
系统启动、系统故障和调试信息
设置(T)...
环境变量(N)...
确定 取消 应用(A)
环境变量
Administrator 的用户变量(U)
变量 值
AWE_DIR C:\Program Files (x86)\Khrona LLC\Awesomium SDK\1.6.6\
ChocolateyLastPathUpdate 132519560308443327
DevEco Studio M:\Program Files\Huawei\DevEco Studio 3.0.0.800\bin;
GTEST_ROOT M:/tools/googletest-release-1.11.0/gtest/
Path C:\Users\Administrator\AppData\Local\Microsoft\WindowsA...
QtMsBuild C:\Users\Administrator\AppData\Local\QtMsBuild
TEMP C:\Users\Administrator\AppData\Local\Temp
新建(N)... 编辑(E)... 删除(D)
系统变量(S)
变量 值
ChocolateyInstall C:\ProgramData\chocolatey
ComSpec C:\Windows\system32\cmd.exe
DriverData C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS 32
OS Windows_NT
Path C:\Perl64\site\bin;C:\Perl64\bin;C:\Windows\system32;C:\Win...
PATHEXT .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
新建(W)... 编辑(I)... 删除(L)
确定 取消

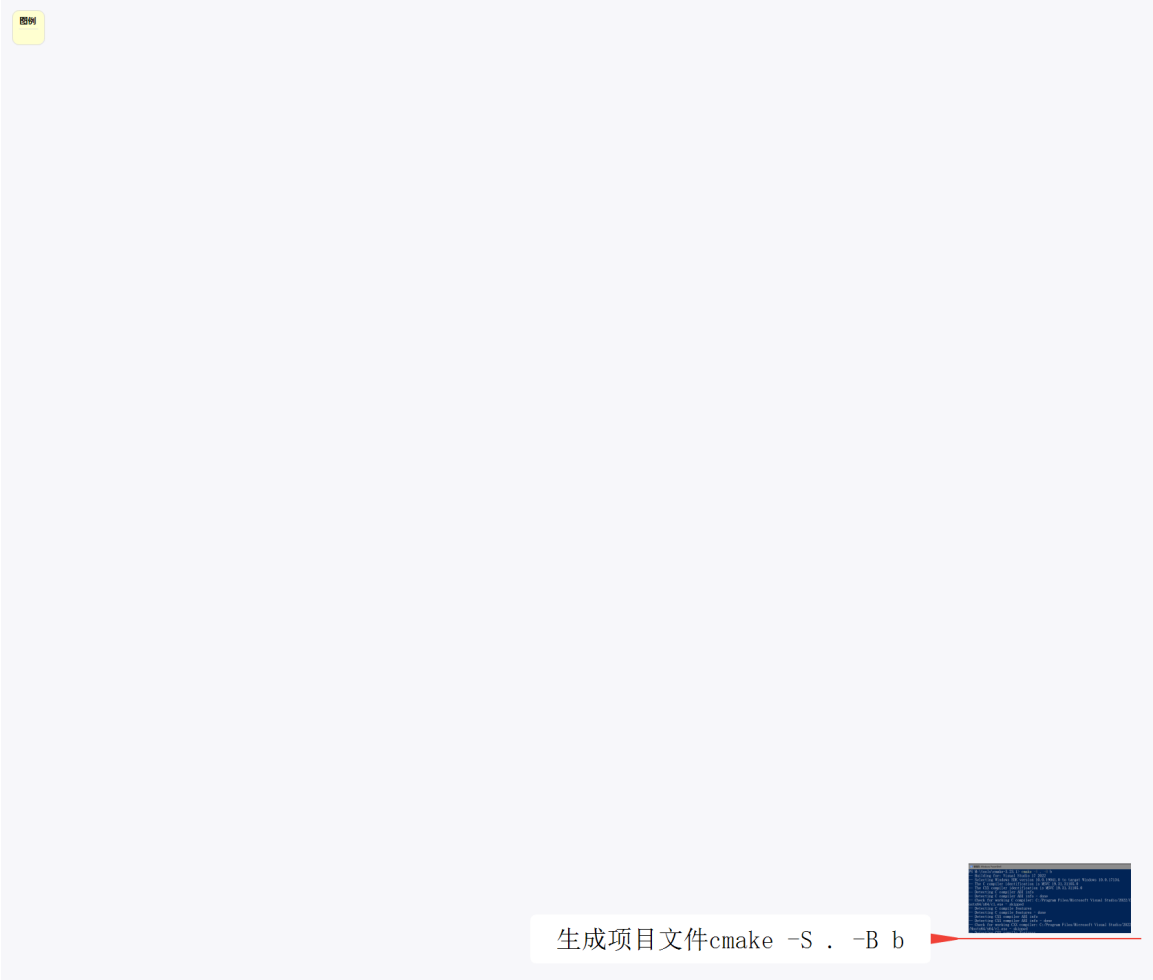

环境变量
Administrator 的用户变量(U)
变量 值
AWE_DIR C:\Program Files (x86)\Khrona LLC\Awesomium SDK\1.6.6\
ChocolateyLastPathUpdate 132519560308443327
DevEco Studio M:\Program Files\Huawei\DevEco Studio 3.0.0.800\bin;
GTEST_ROOT M:/tools/googletest-release-1.11.0/gtest/
Path C:\Users\Administrator\AppData\Local\Microsoft\WindowsA...
QtMsBuild C:\Users\Administrator\AppData\Local\QtMsBuild
TEMP C:\Users\Administrator\AppData\Local\Temp
新建(N)... 编辑(E)... 删除(D)
系统变量(S)
变量 值
ChocolateyInstall C:\ProgramData\chocolatey
ComSpec C:\Windows\system32\cmd.exe
DriverData C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS 32
OS Windows_NT
Path C:\Perl64\site\bin;C:\Perl64\bin;C:\Windows\system32;C:\Win...
PATHEXT .COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
新建(W)... 编辑(I)... 删除(L)
确定 取消
编辑环境变量
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
H:\depot_tools
%DevEco Studio%
C:\Users\Administrator\AppData\Roaming\npm
C:\Users\Administrator\AppData\Local\Programs\Python\Python39\
M:\cmake-3.22.1-windows-x86_64\bin
新建(N)
编辑(E)
浏览(B)...
删除(D)
上移(U)
下移(O)
编辑文本(T)...
确定 取消

2.3.3. 三 源码编译安装（选学）用cmake构建cmake



解压源码后 控制台进入cmake源码目录

生成项目文件cmake -S . -B b



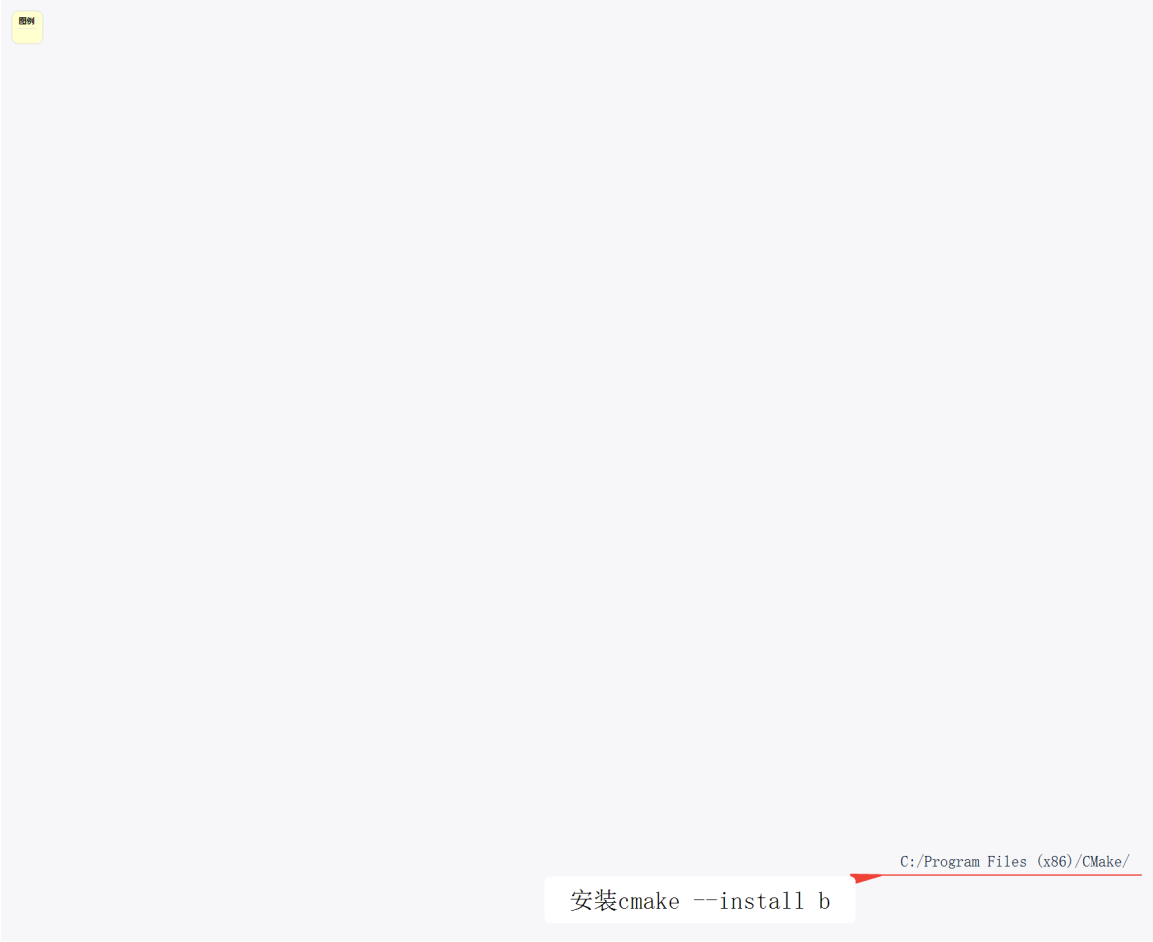

```
管理员: Windows PowerShell
PS M:\tools\cmake-3.23.1> cmake -S . -B b
-- Building for: Visual Studio 17 2022
-- Selecting Windows SDK version 10.0.19041.0 to target Windows 10.0.17134.
-- The C compiler identification is MSVC 19.31.31105.0
-- The CXX compiler identification is MSVC 19.31.31105.0
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: C:/Program Files/Microsoft Visual Studio/2022/C
ostx64/x64/cl.exe - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: C:/Program Files/Microsoft Visual Studio/2022
/Hostx64/x64/cl.exe - skipped
-- Detecting CXX compile features
```

编译 cmake --build b --config Release

编译 cmake --build b --config Release

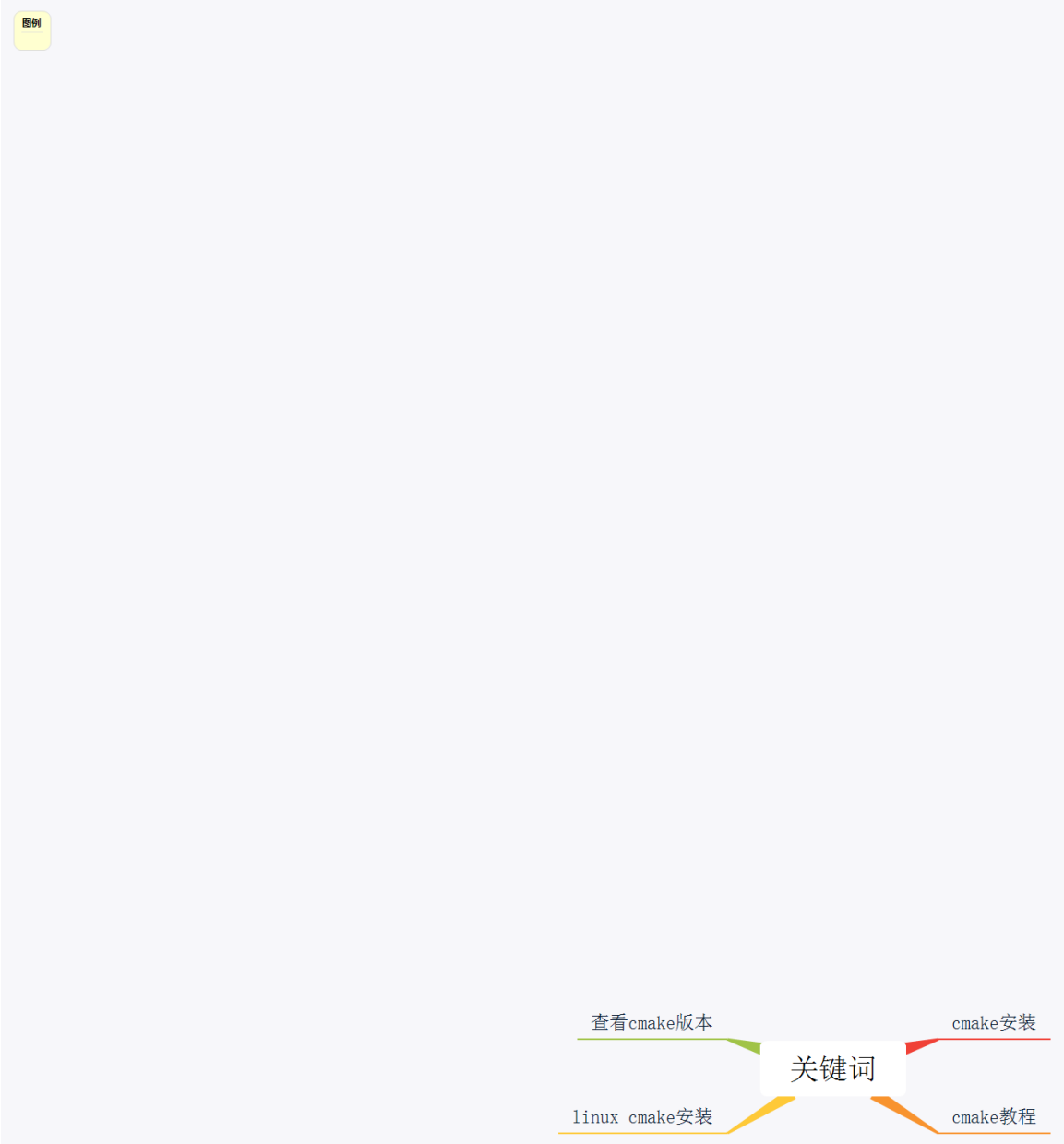
```
PS M:\tools\cmake-3.23.1> cmake --build b
用于 .NET Framework 的 Microsoft (R) 生成引擎版本 17.1.0+ae57d105c
版权所有(C) Microsoft Corporation。保留所有权利。

  Checking Build System
  Building Custom Rule M:/tools/cmake-3.23.1/Source/CMakeLists.txt
  CMakeVersion.vcxproj -> M:\tools\cmake-3.23.1\b\Source\CMakeVersion.dir\Debug\CMakeVersion.lib
  Building Custom Rule M:/tools/cmake-3.23.1/Source/kwsys/CMakeLists.txt
  ProcessWin32.c
  Base64.c
  MD5.c
  Terminal.c
  EncodingC.c
  System.c
  String.c
  Directory.cxx
  Glob.cxx
  RegularExpression.cxx
  CommandLineArguments.cxx
```

安装cmake --install b



C:/Program Files (x86)/CMake/

2.4. 关键词



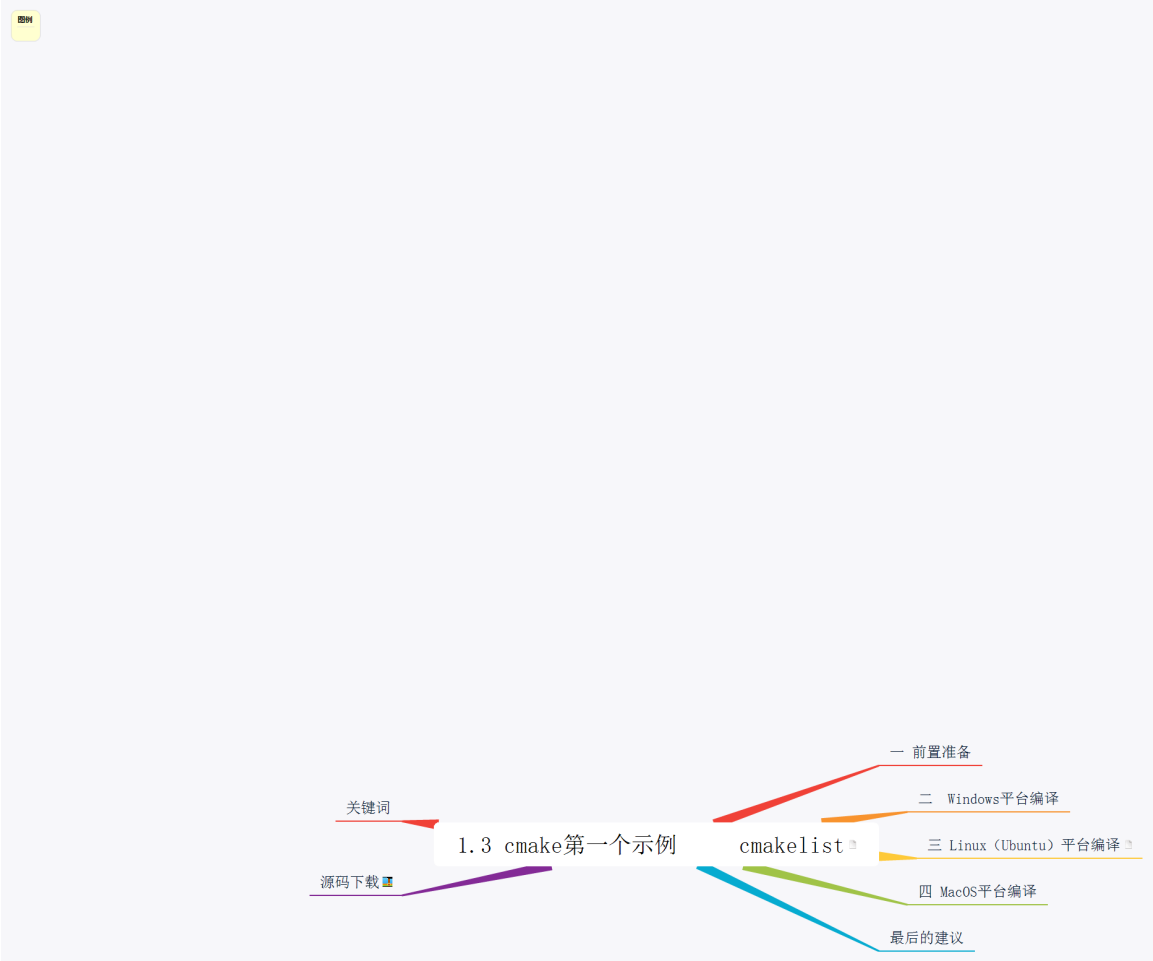
2.4.1. cmake安装

2.4.2. cmake教程

2.4.3. linux cmake安装

2.4.4. 查看cmake版本

3. 1.3 cmake第一个示例cmakelist


图例
一 前置准备
关键词
二 Windows平台编译
1.3 cmake第一个示例
cmakelist
三 Linux（Ubuntu）平台编译
源码下载
四 MacOS平台编译
最后的建议

3.1. 一 前置准备

3.1.1. 准备测试的c++程序文件first_cmake.cpp


```
//first_cmake.cpp
#include <iostream>
using namespace std;
int main(int argc,char *argv[])
{
    cout<<"first cmake c++"<<endl;
    return 0;
}
```

3.1.2. 在源码的同目录下编写第一个CMakeLists.txt

 # CMakeLists.txt

 # 指定cmake的最低版本

cmake_minimum_required (VERSION 3.20)

 # 构建的项目名称

project (first_cmake)

 # 构建执行程序

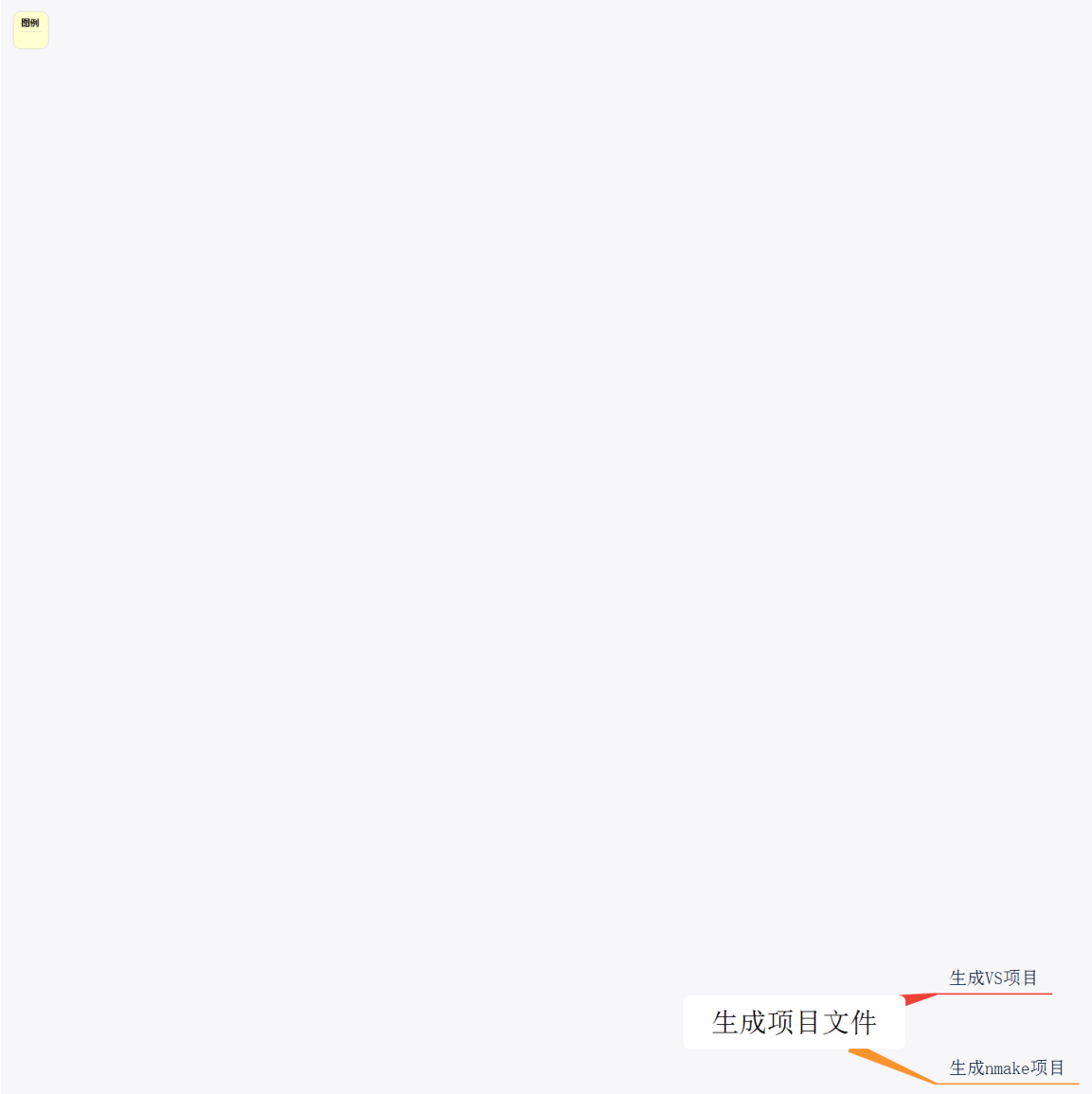
add_executable(first_cmake first_cmake.cpp)

3.2. 二 Windows平台编译

3.2.1. CMake=》vs项目=》cl编译

3.2.2. 4 自动创建构建目录

3.2.3. 生成项目文件



生成VS项目

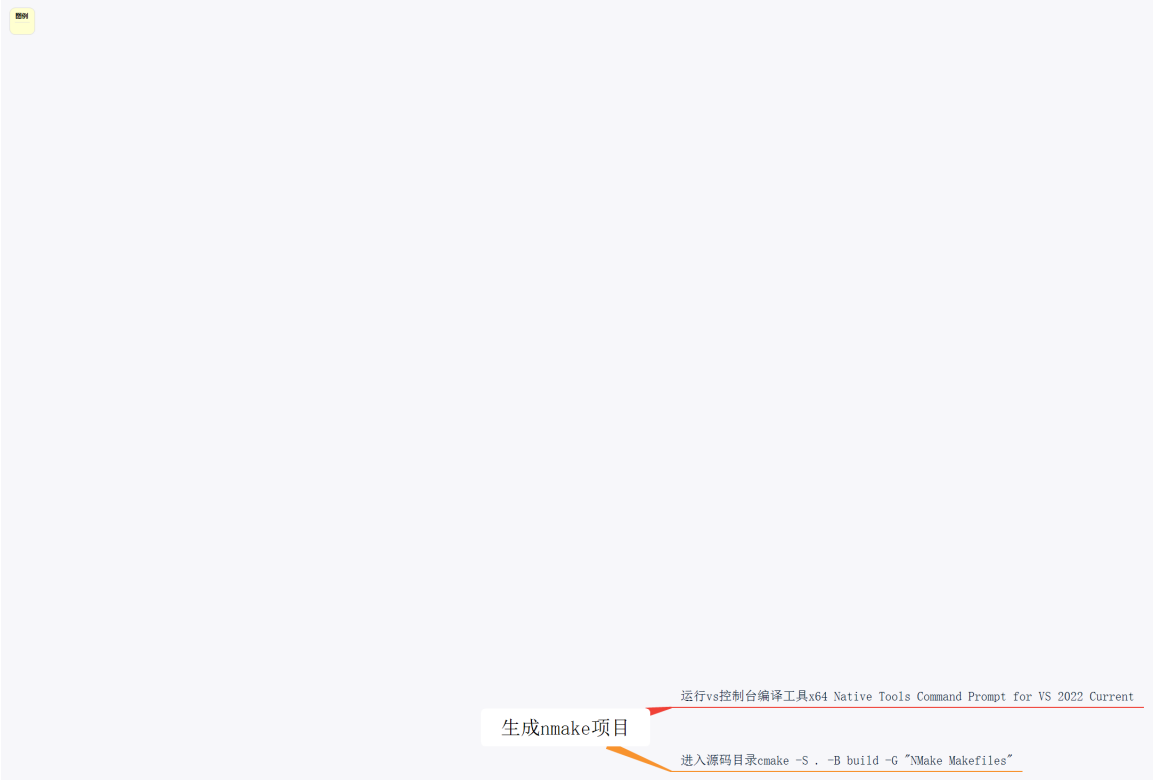
1

源码目录下面创建一个编译目录build，用于生成cmake的临时文件和项目文件，放在独立的目录方便清理和查看

2 进入编译目录build，直接运行cmake ..

使用默认生成项目文件，下图生成的是vs2022的64位项目（文件所在的路径和目录层次不能太深，太深会找不到编译器）

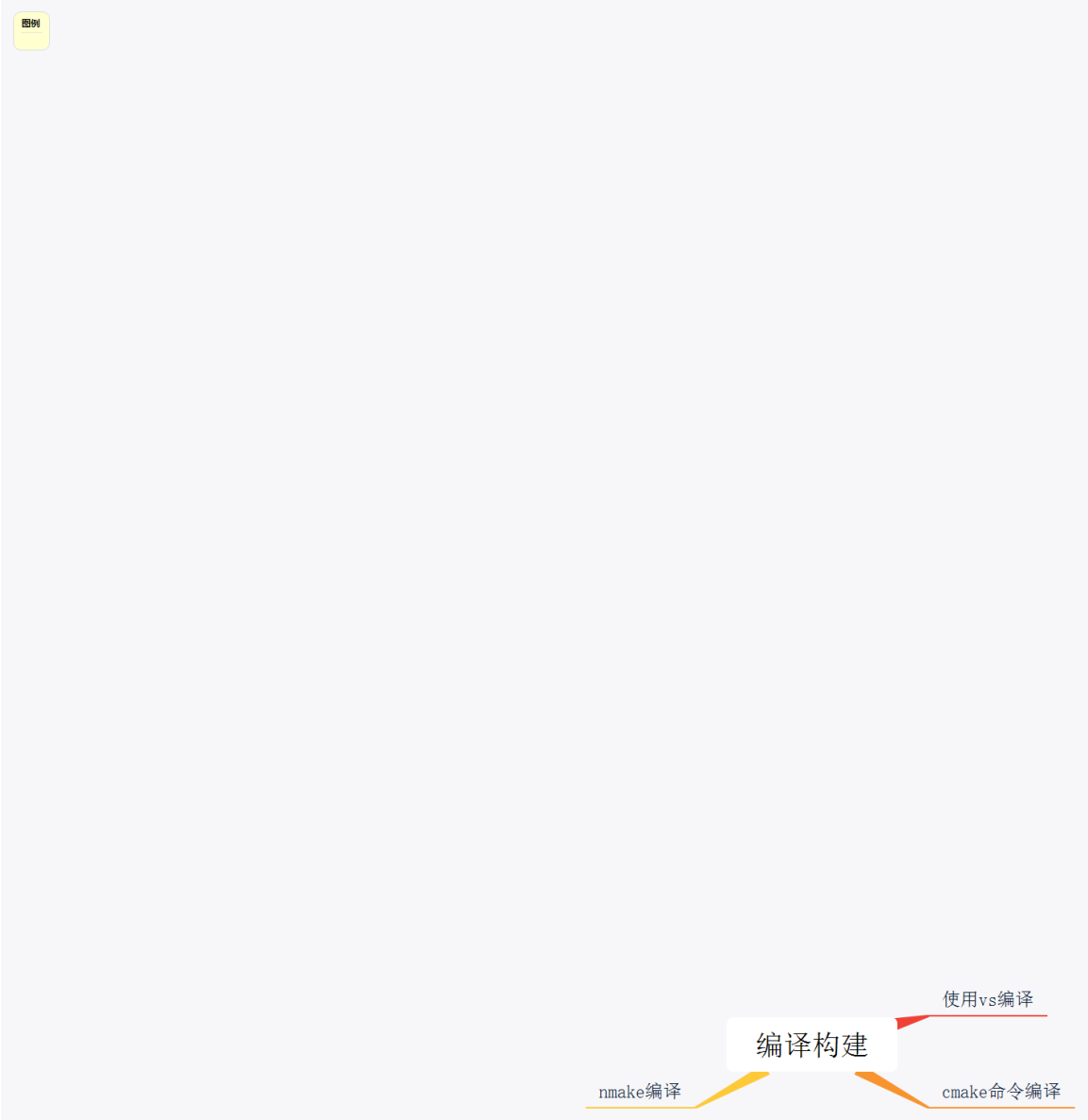
生成nmake项目



运行vs控制台编译工具x64 Native Tools Command Prompt for VS 2022 Current

进入源码目录cmake -S . -B build -G "NMake Makefiles"

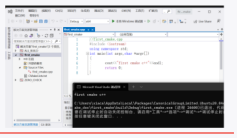
3.2.4. 编译构建

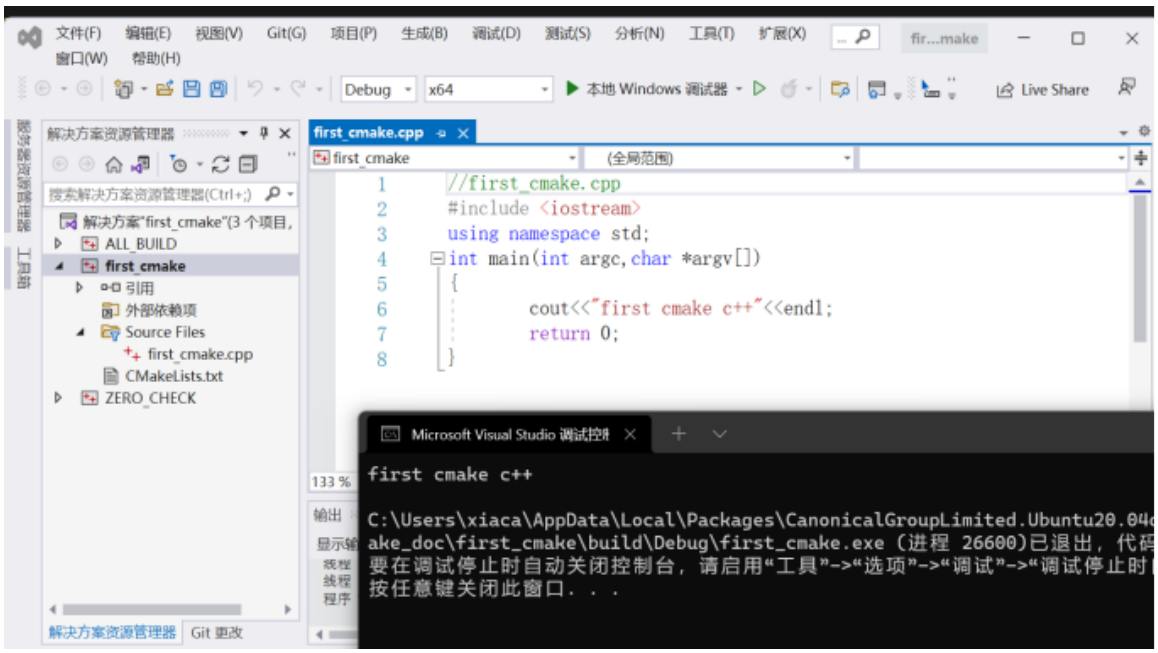


使用vs编译

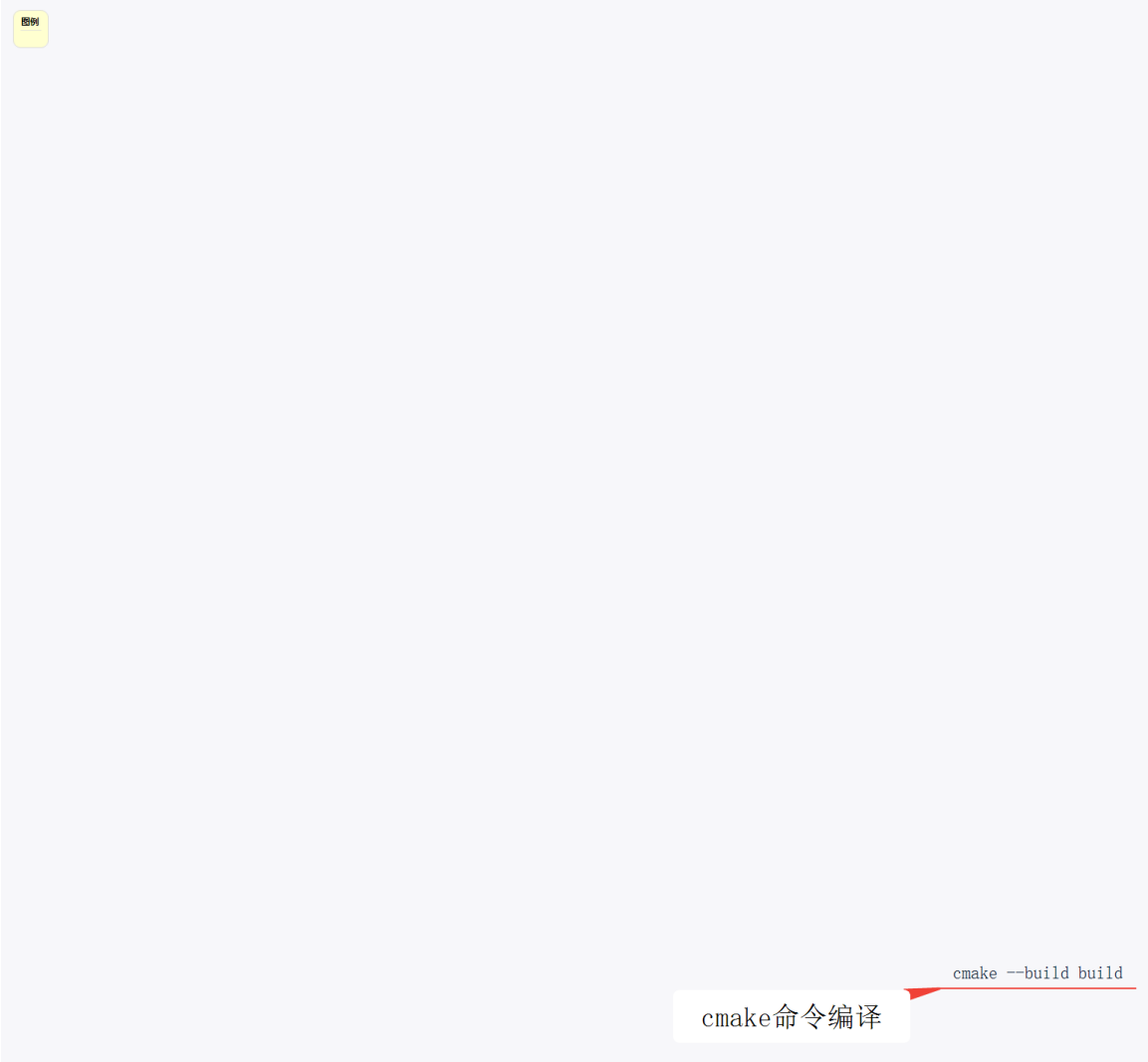
3 进入build目录打开解决方案

3 进入build目录打开解决方案



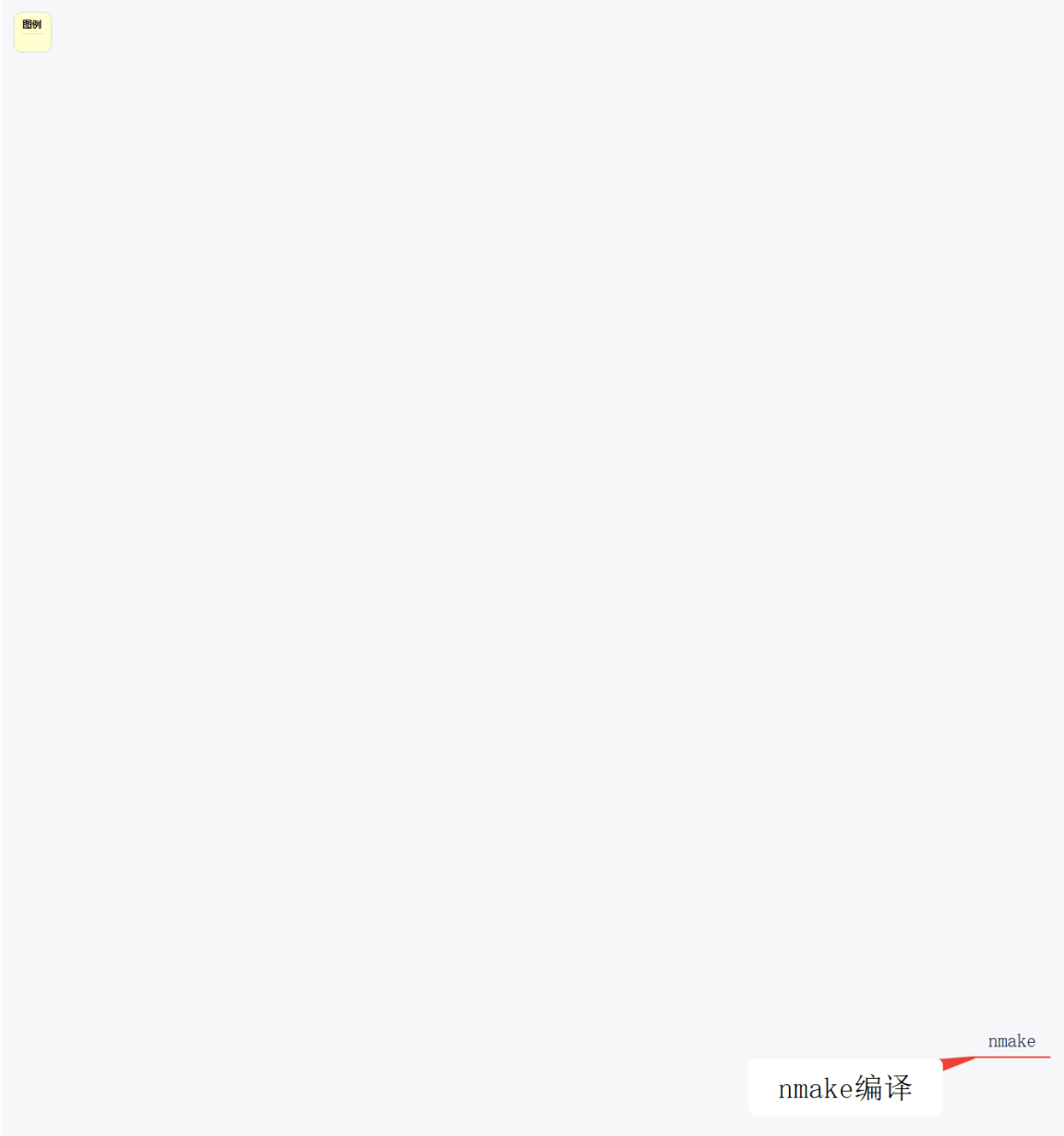


cmake命令编译



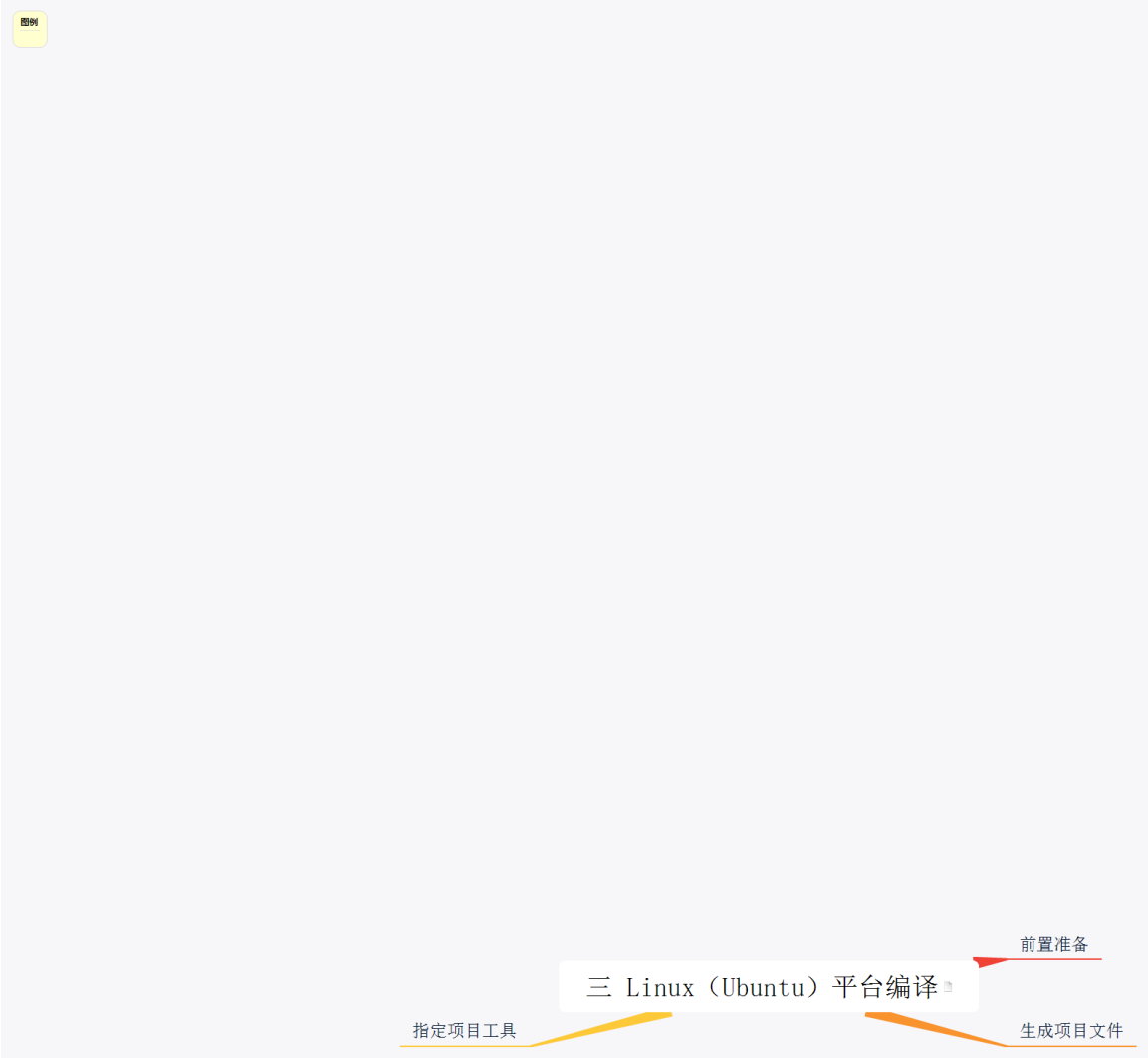
cmake --build build

nmake编译

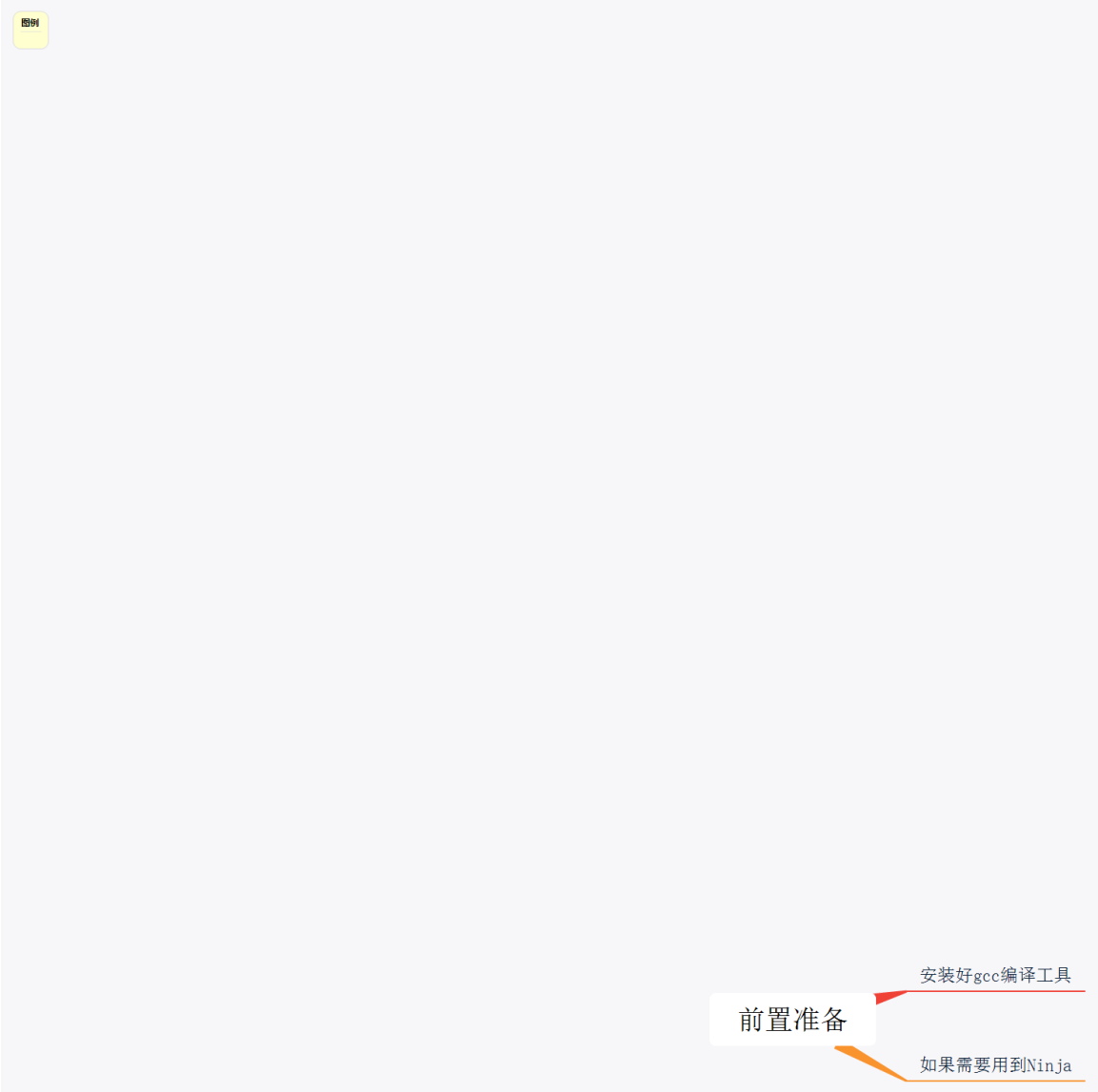


nmake

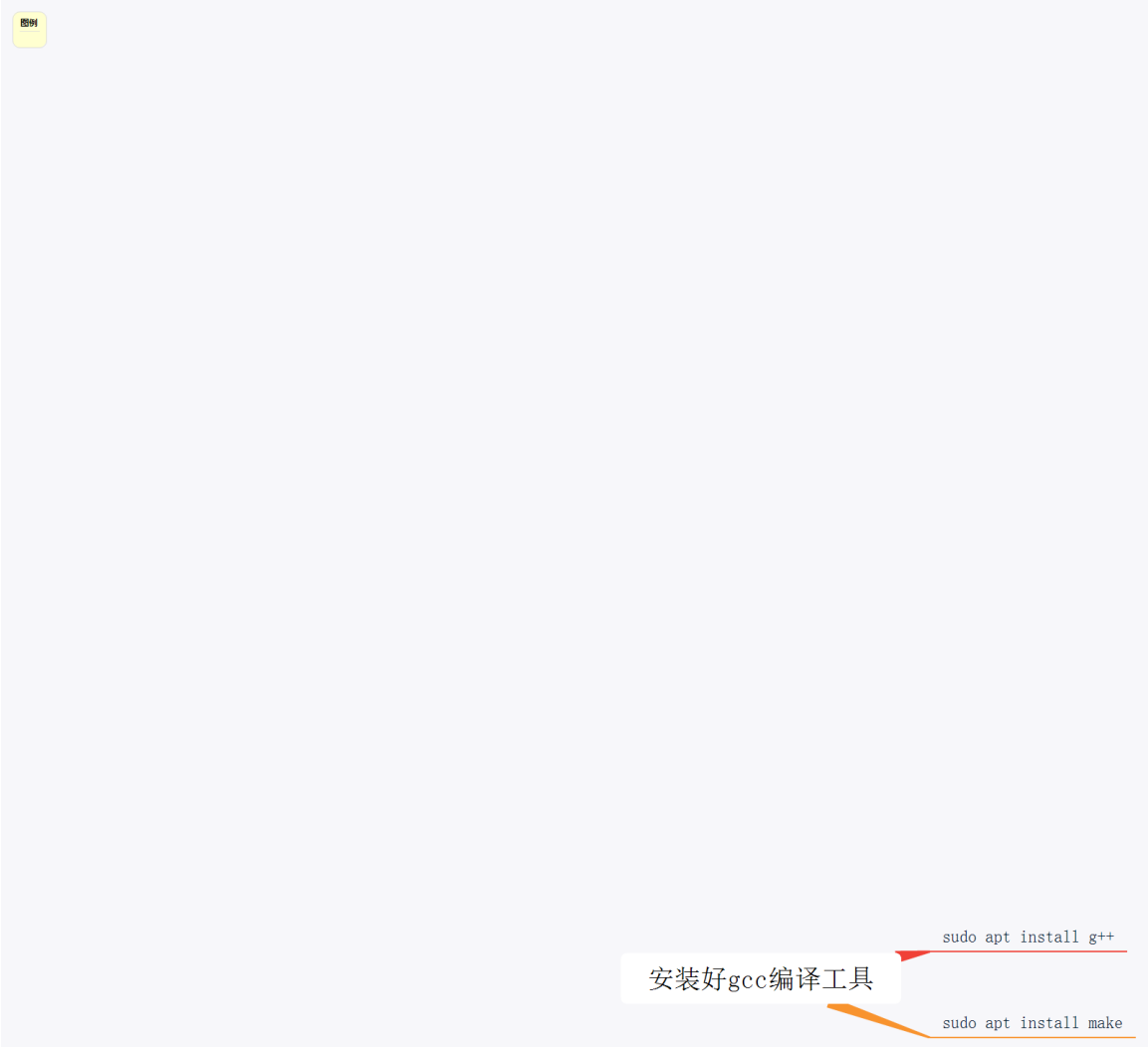
3.3. 三 Linux（Ubuntu）平台编译



3.3.1. 前置准备



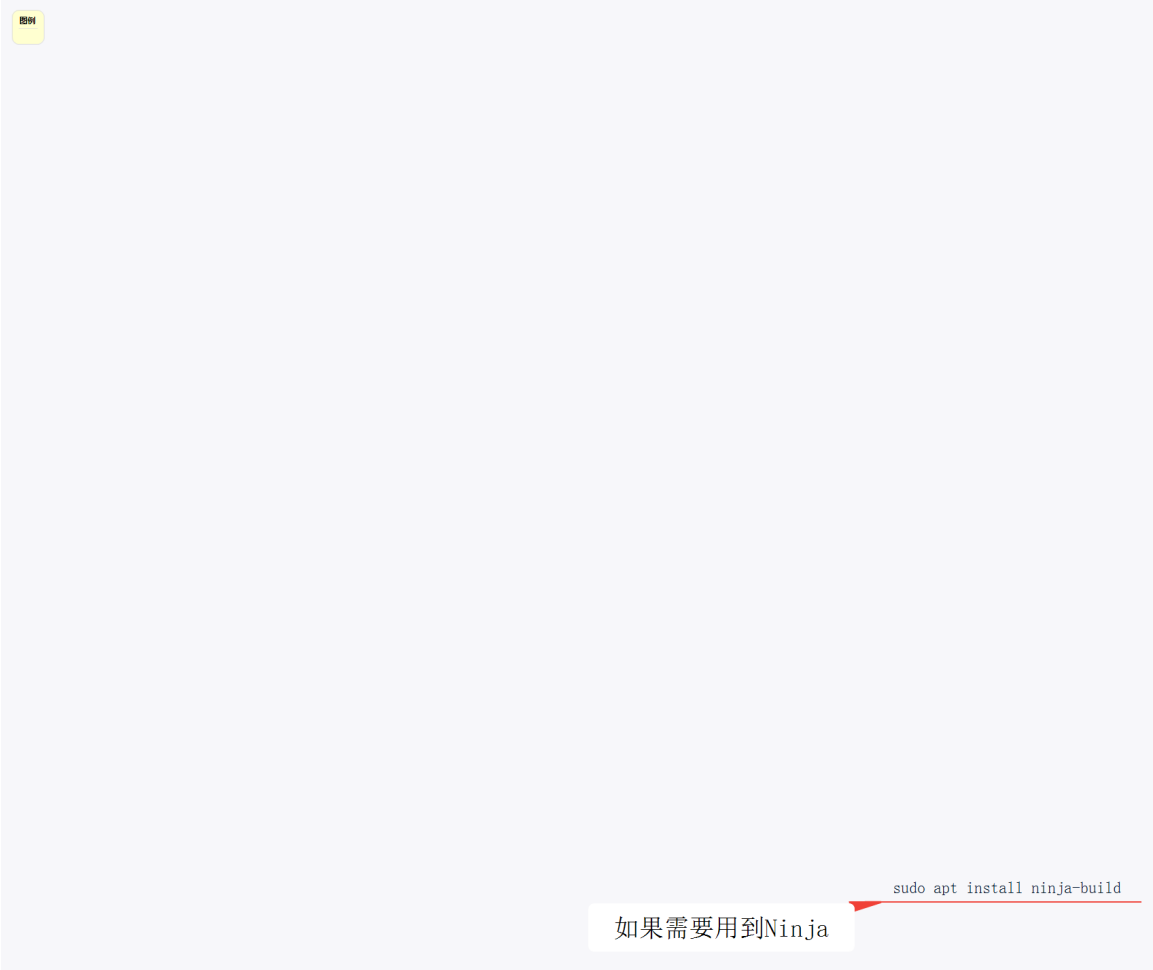
安装好gcc编译工具



sudo apt install g++

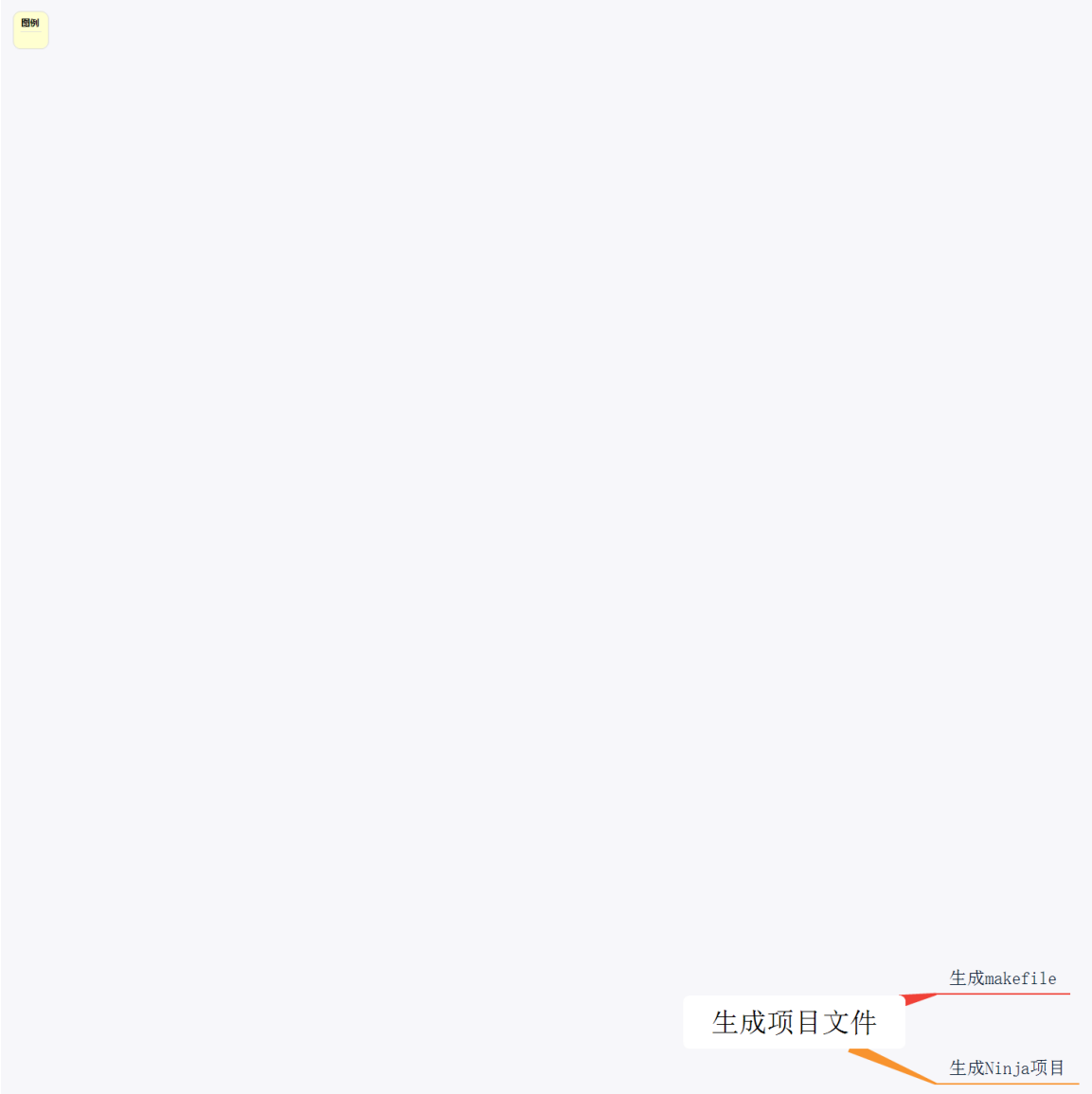
sudo apt install make

如果需要用到Ninja



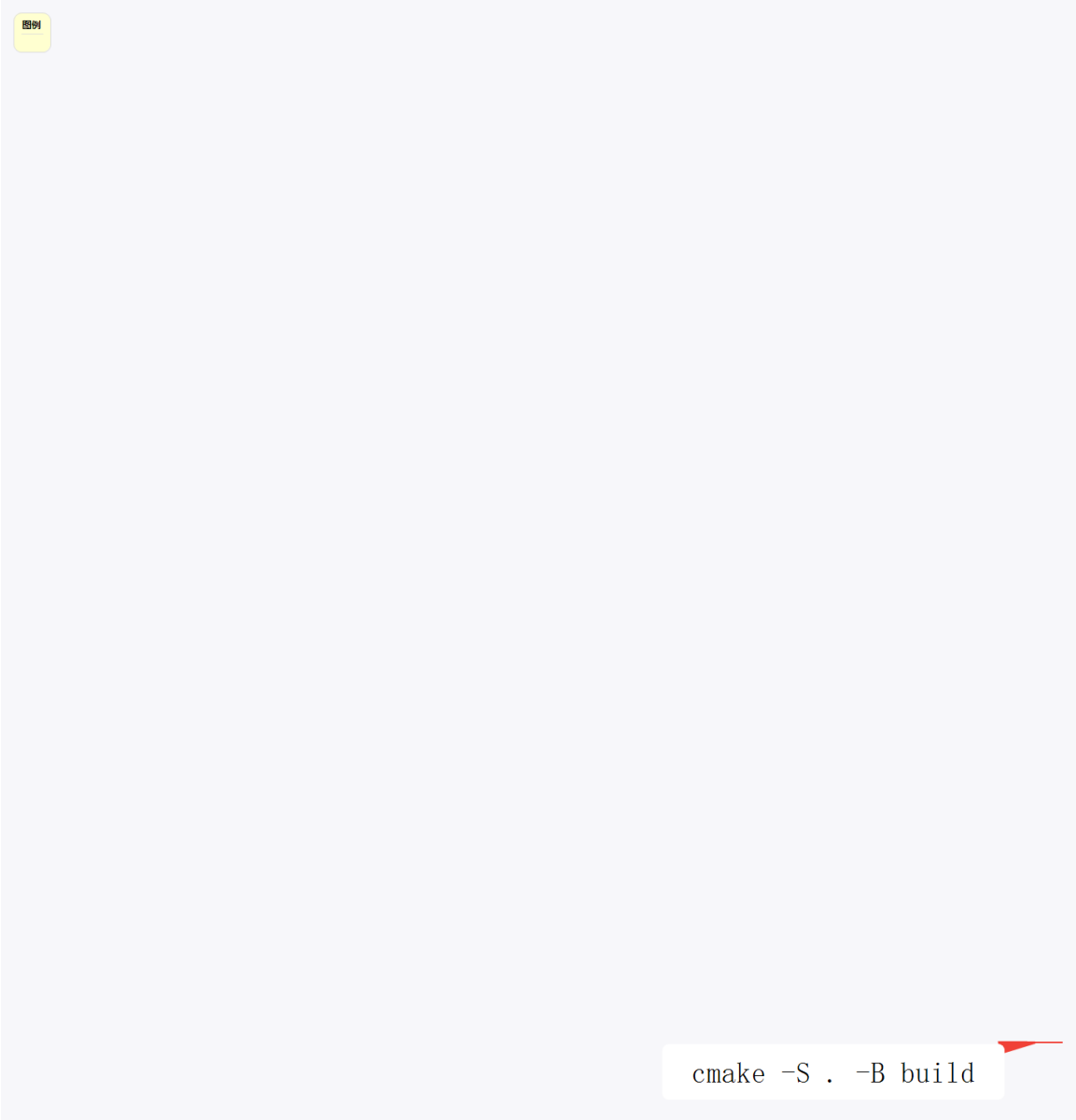
sudo apt install ninja-build

3.3.2. 生成项目文件

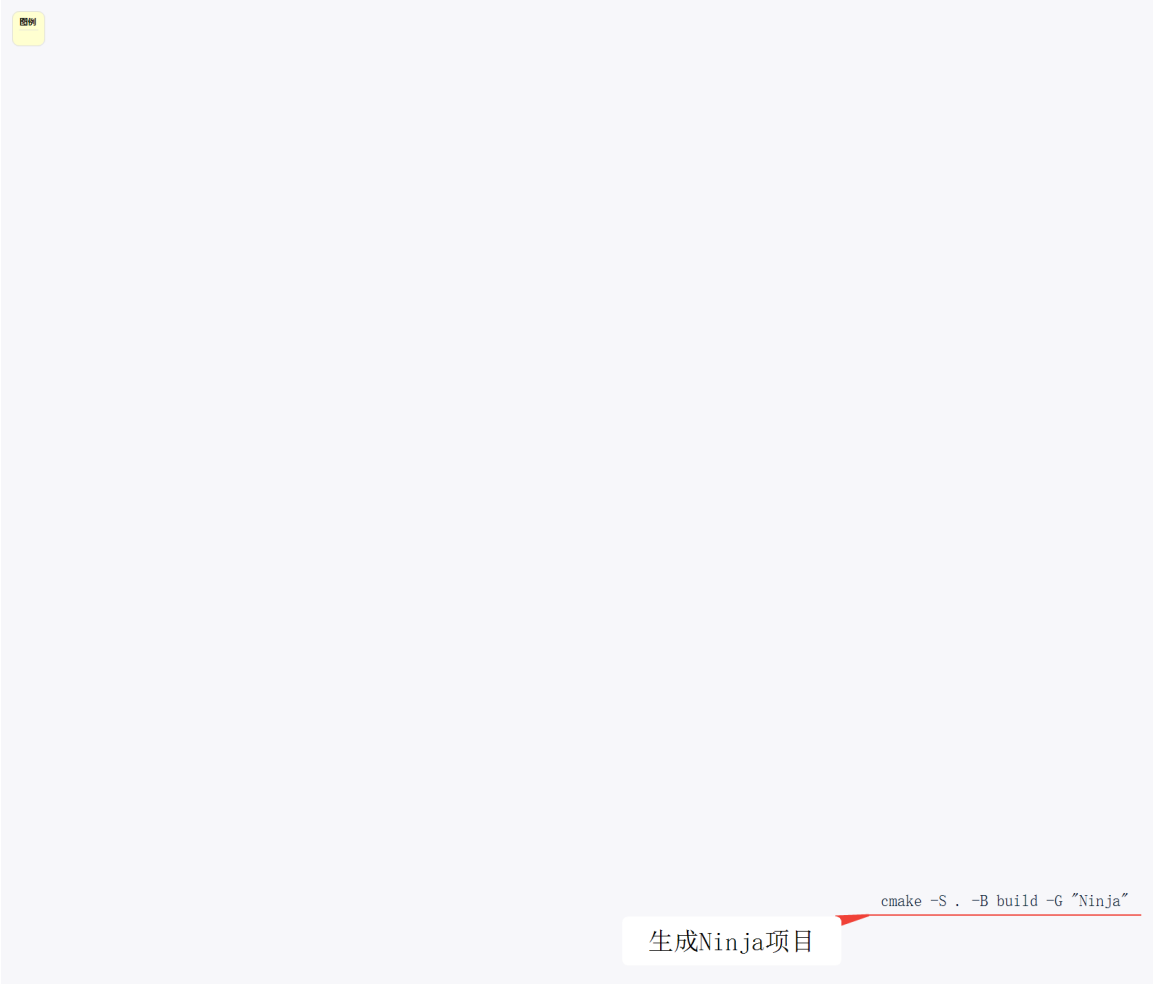


生成makefile

cmake -S . -B build



生成Ninja项目



cmake -S . -B build -G "Ninja"

3.3.3. 指定项目工具

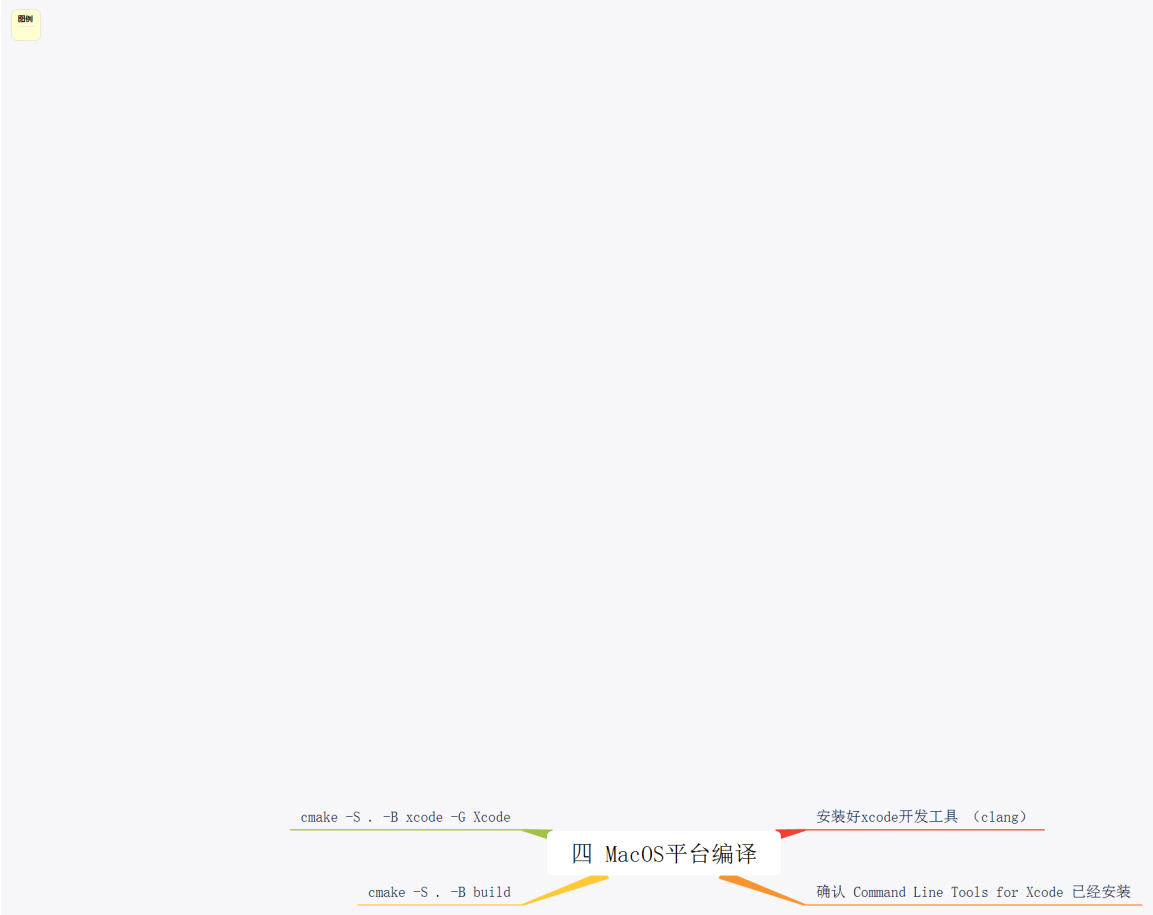
在linux主要有两种，一种是生成make的makefile一种是生成Ninja的build.ninja

生成makefile见上面示例

生成Ninja

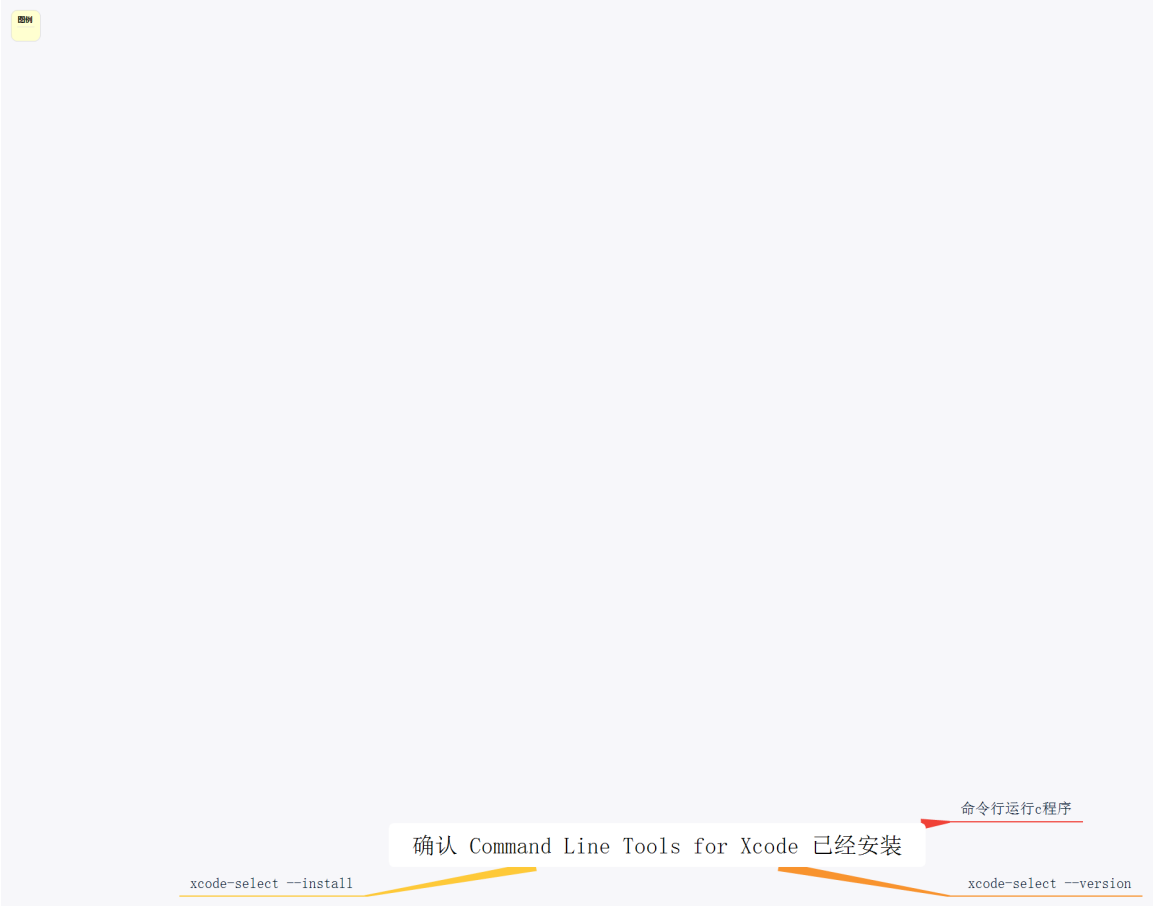
cmake .. -G Ninja

3.4. 四 MacOS平台编译



3.4.1. 安装好xcode开发工具 （clang）

3.4.2. 确认 Command Line Tools for Xcode 已经安装

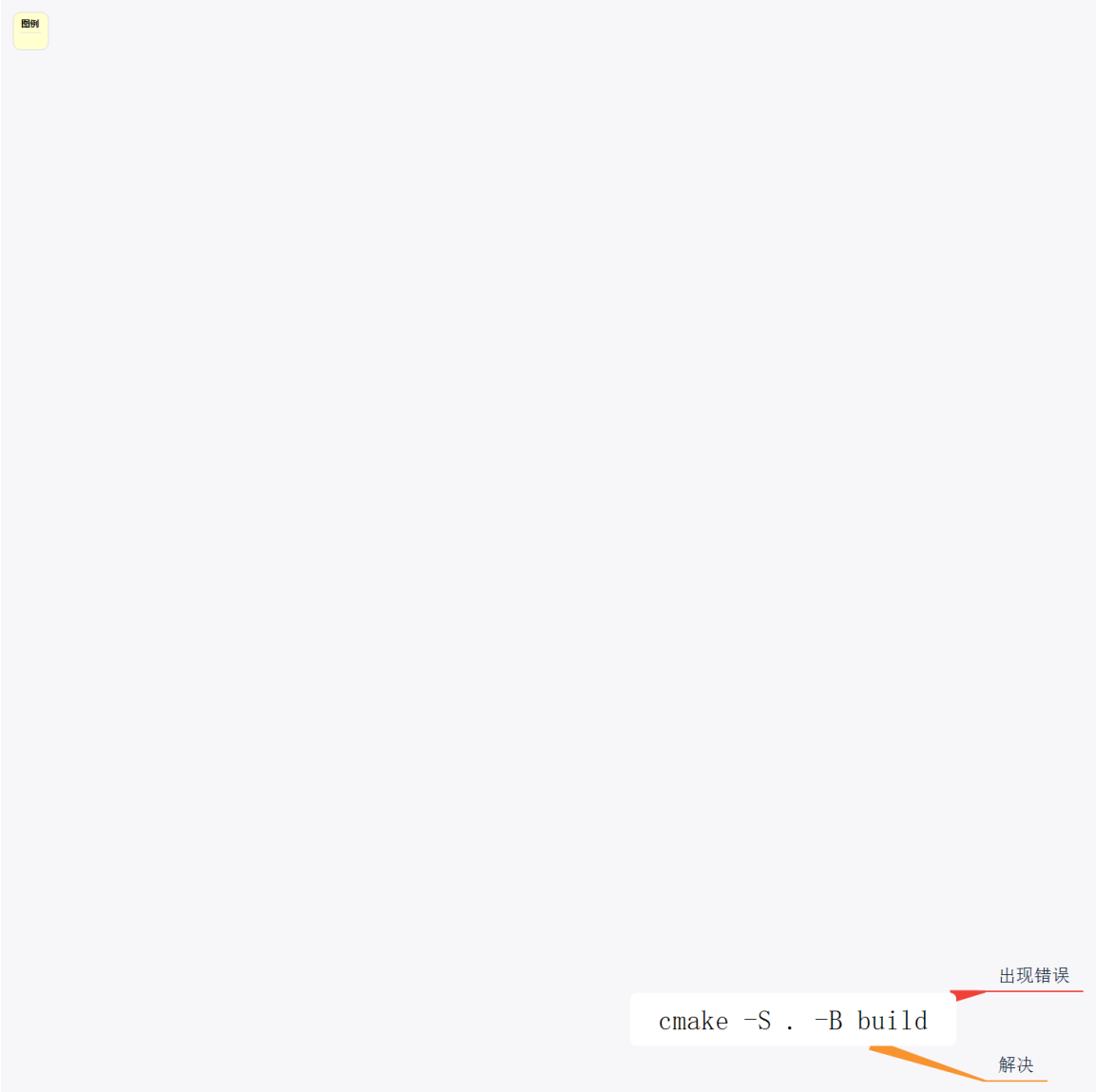


命令行运行c程序

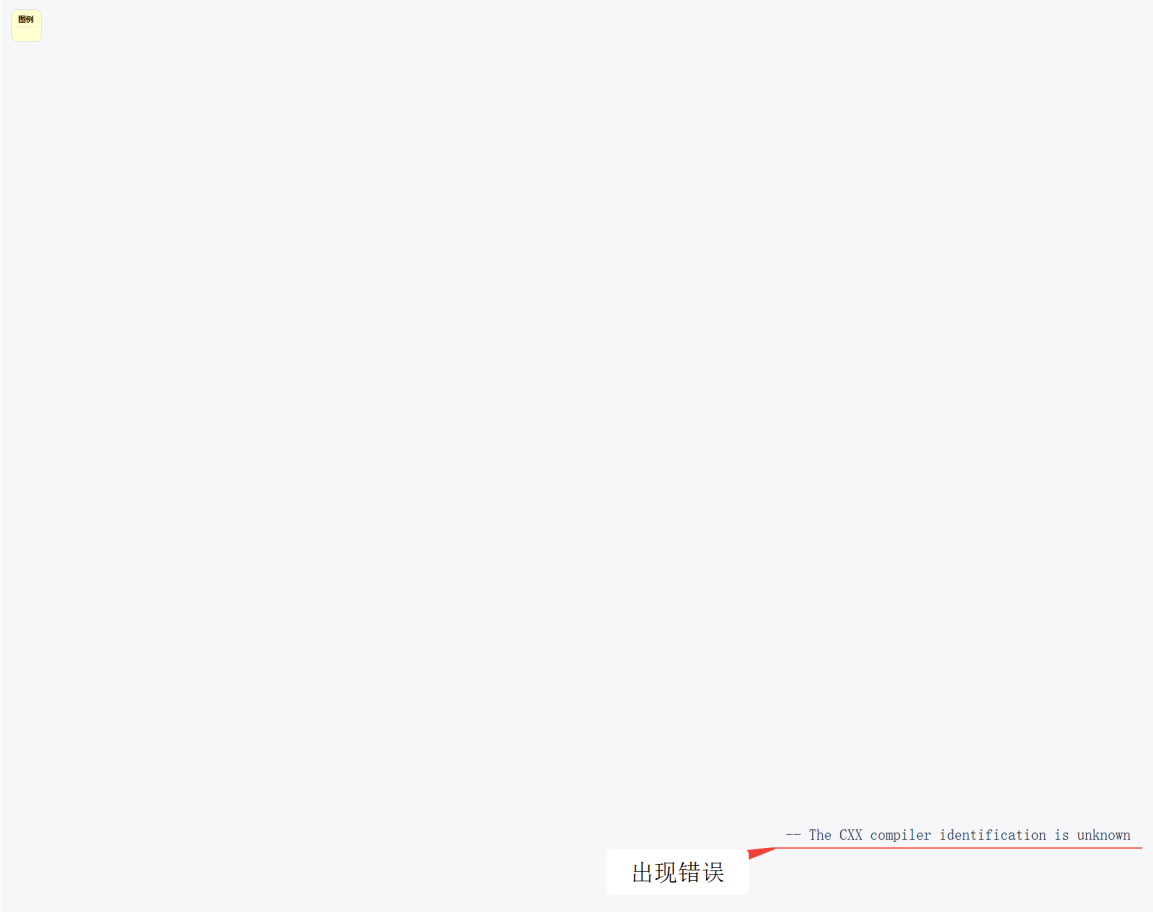
xcode-select --version

xcode-select --install

3.4.3. cmake -S . -B build

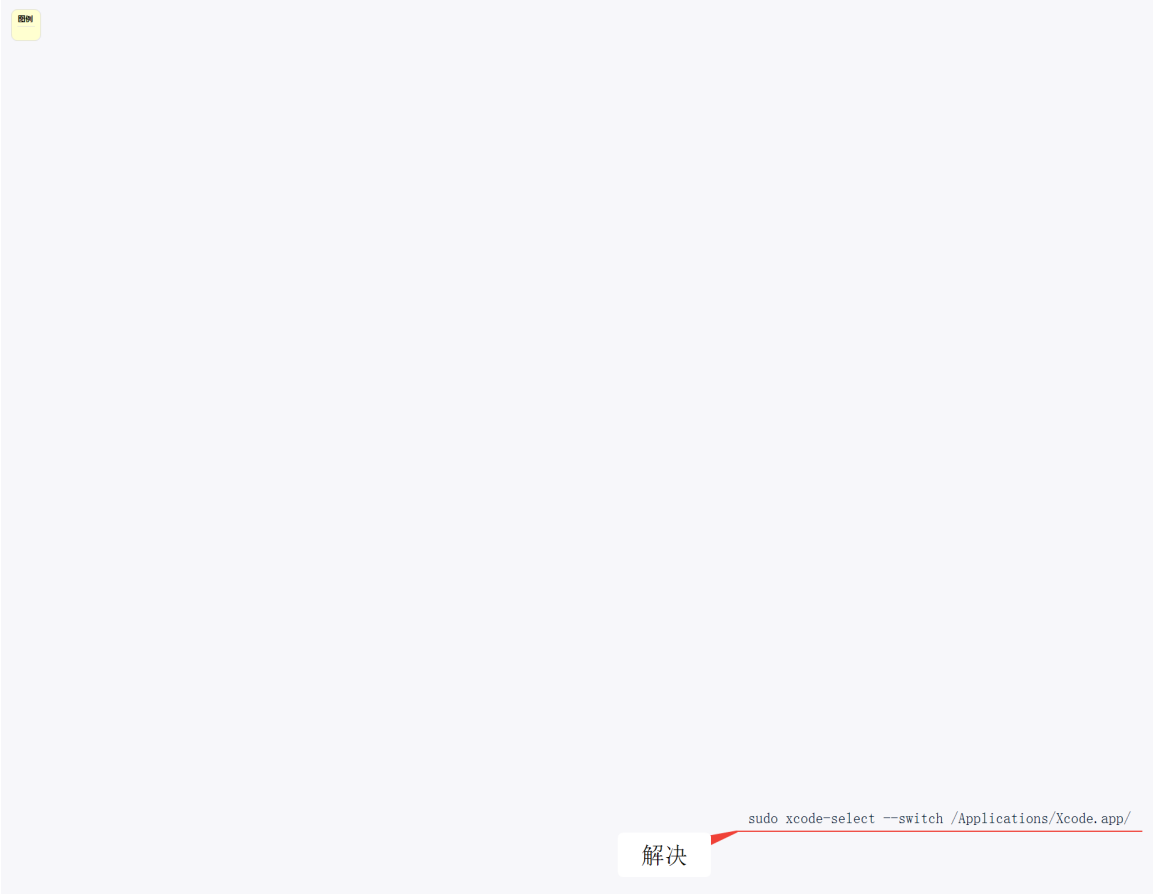


出现错误



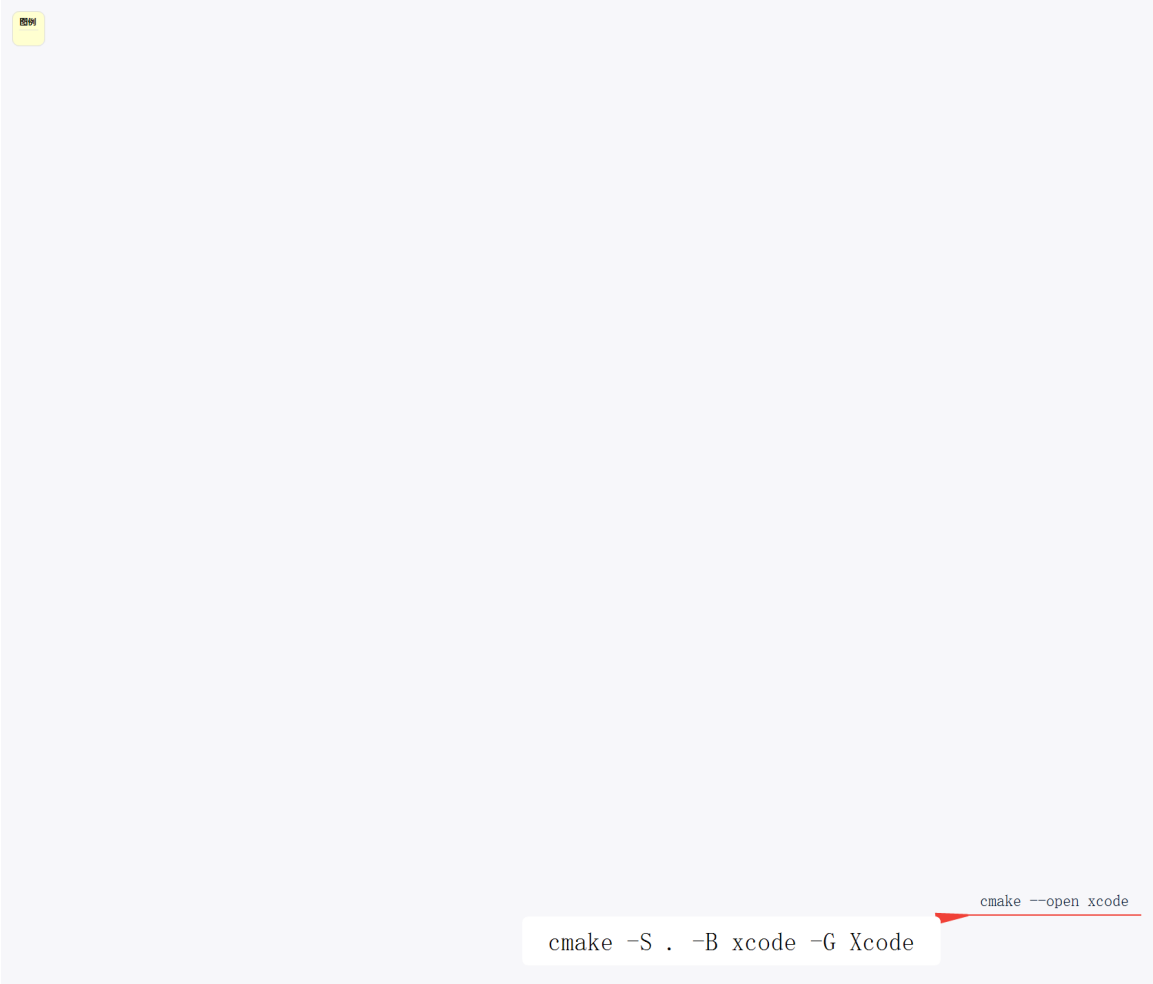
-- The CXX compiler identification is unknown

解决



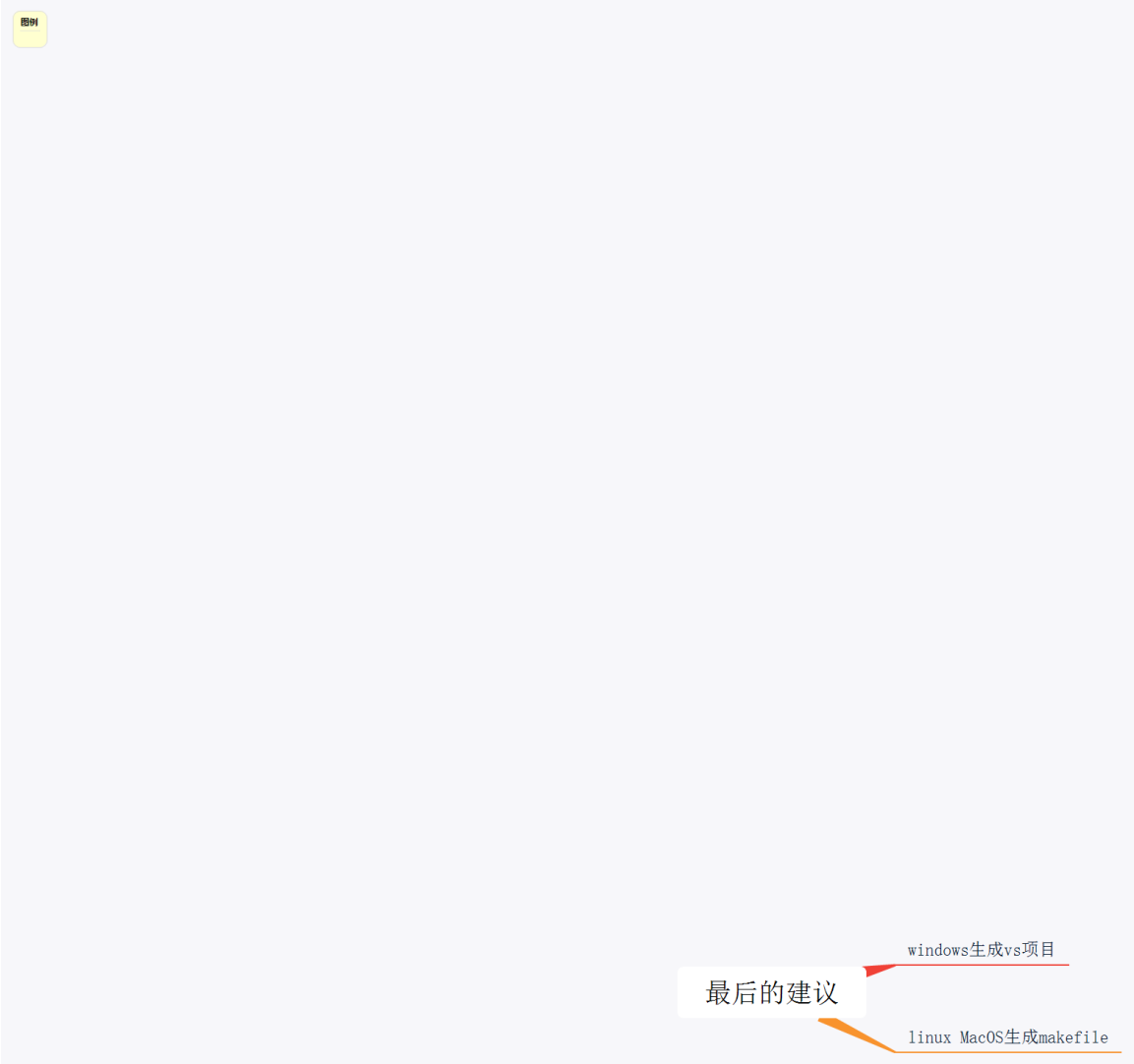
sudo xcode-select --switch /Applications/Xcode.app/

3.4.4. cmake -S . -B xcode -G Xcode



cmake --open xcode

3.5. 最后的建议

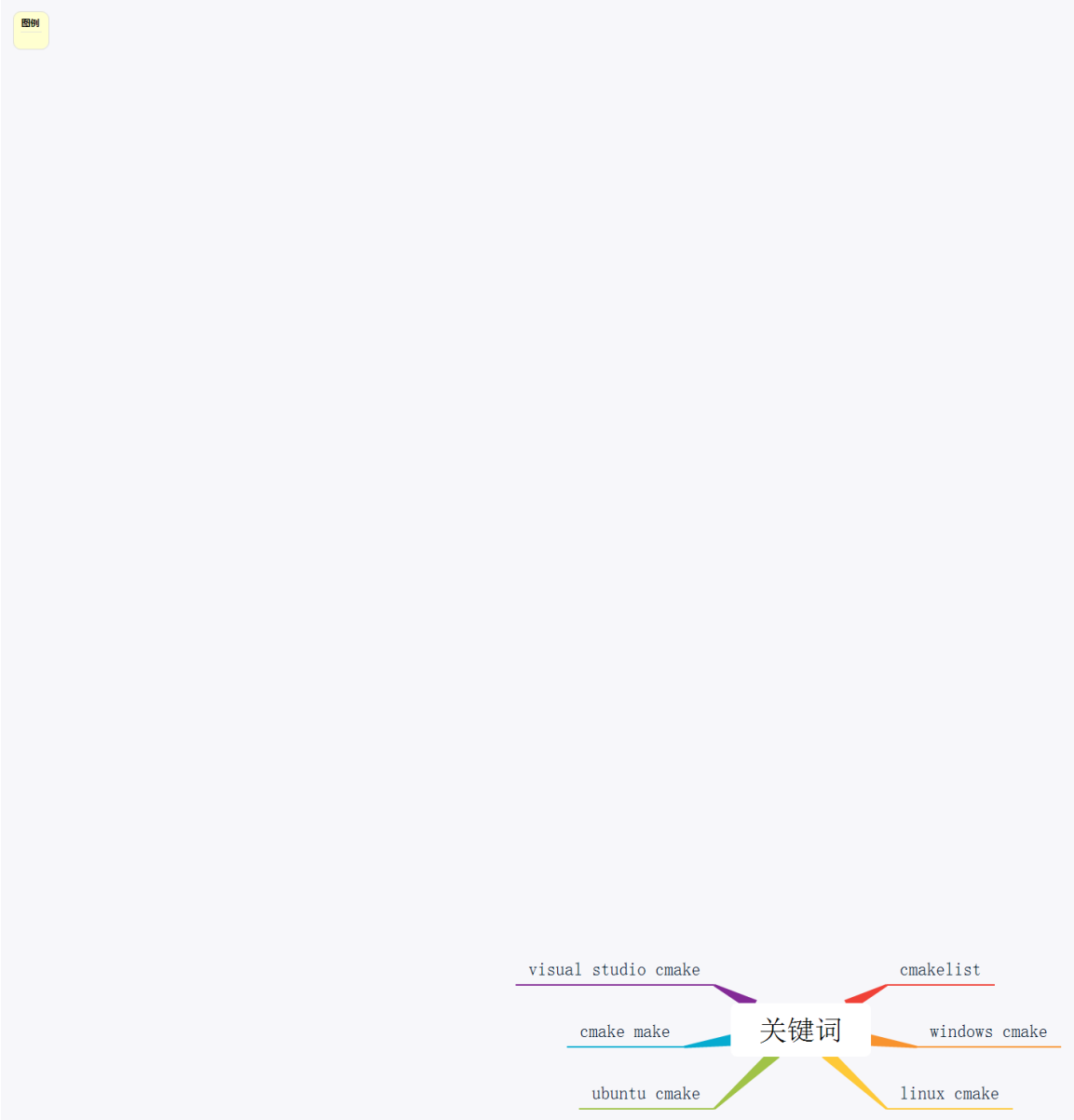


3.5.1. windows生成vs项目

3.5.2. linux MacOS生成makefile

3.6. 源码下载

3.7. 关键词



3.7.1. cmakelist

3.7.2. windows cmake

3.7.3. linux cmake

3.7.4. ubuntu cmake

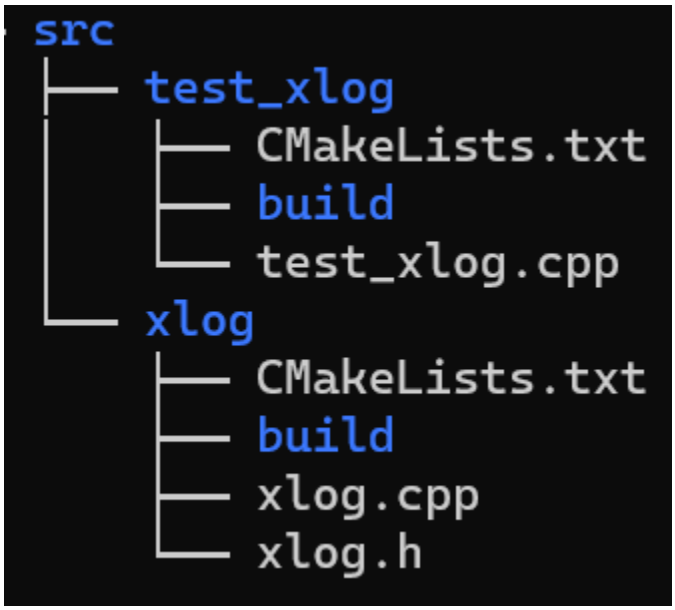
3.7.5. cmake make

3.7.6. visual studio cmake

4. 1.4 cmake构建静态库与动态库

4.1. 前置准备

4.1.1.



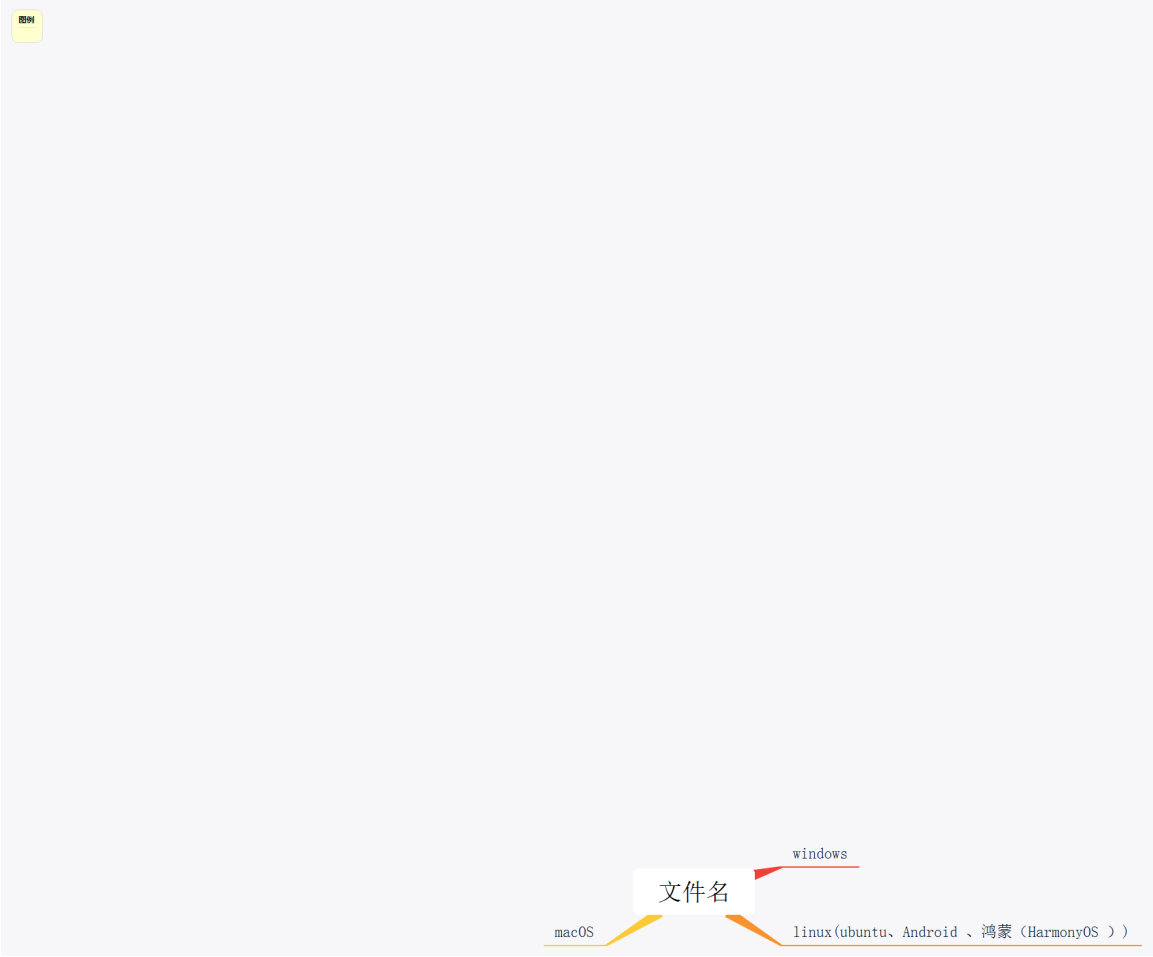
4.1.2. 本节课尽量精简使用cmake特性，后面可以会一步步加入自动操作

4.1.3. 本节课尽量精简使用cmake特性，后面可以会一步步加入自动操作

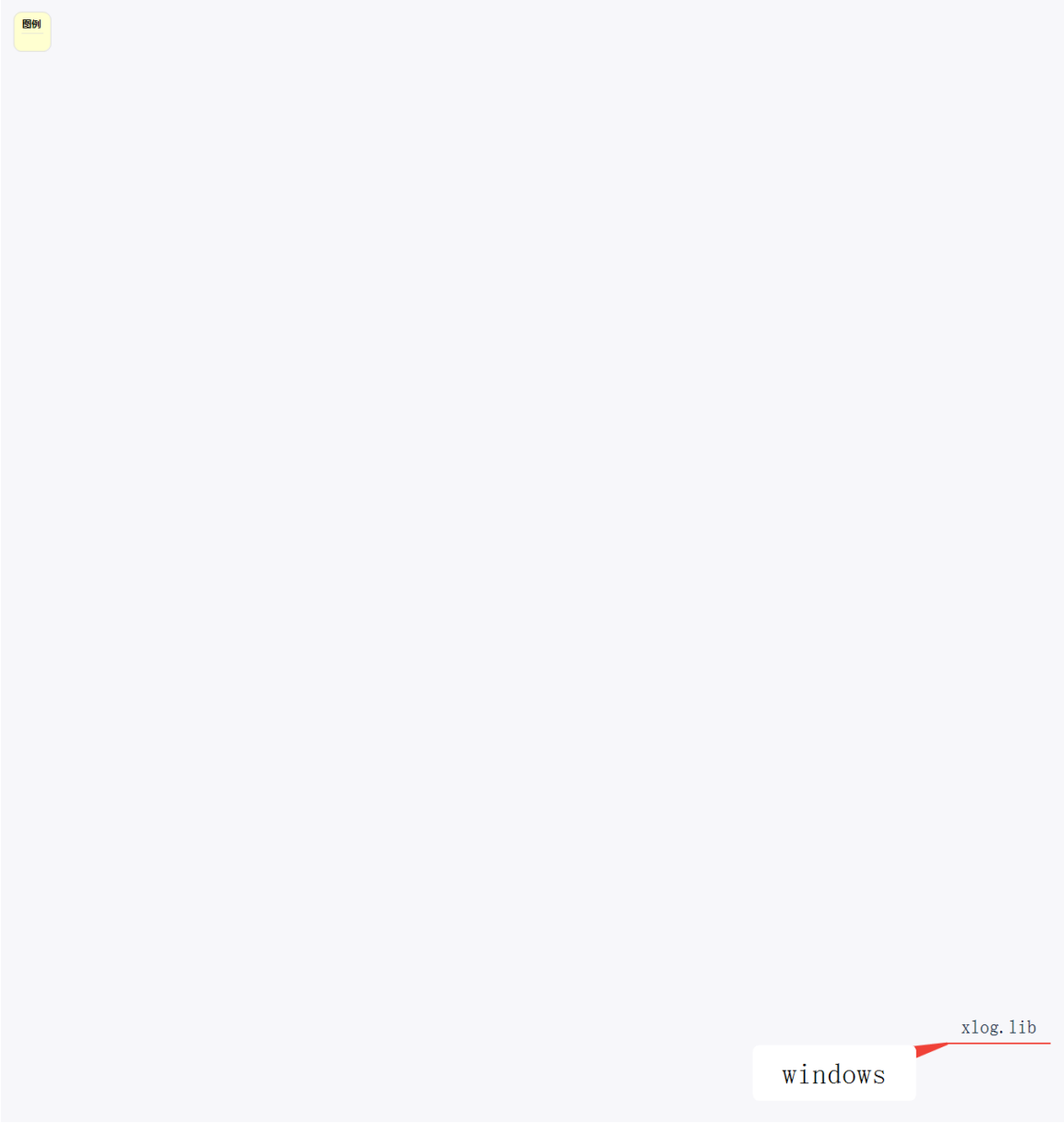
4.2. 动态库和静态库概念(xlog)

4.2.1. 静态库

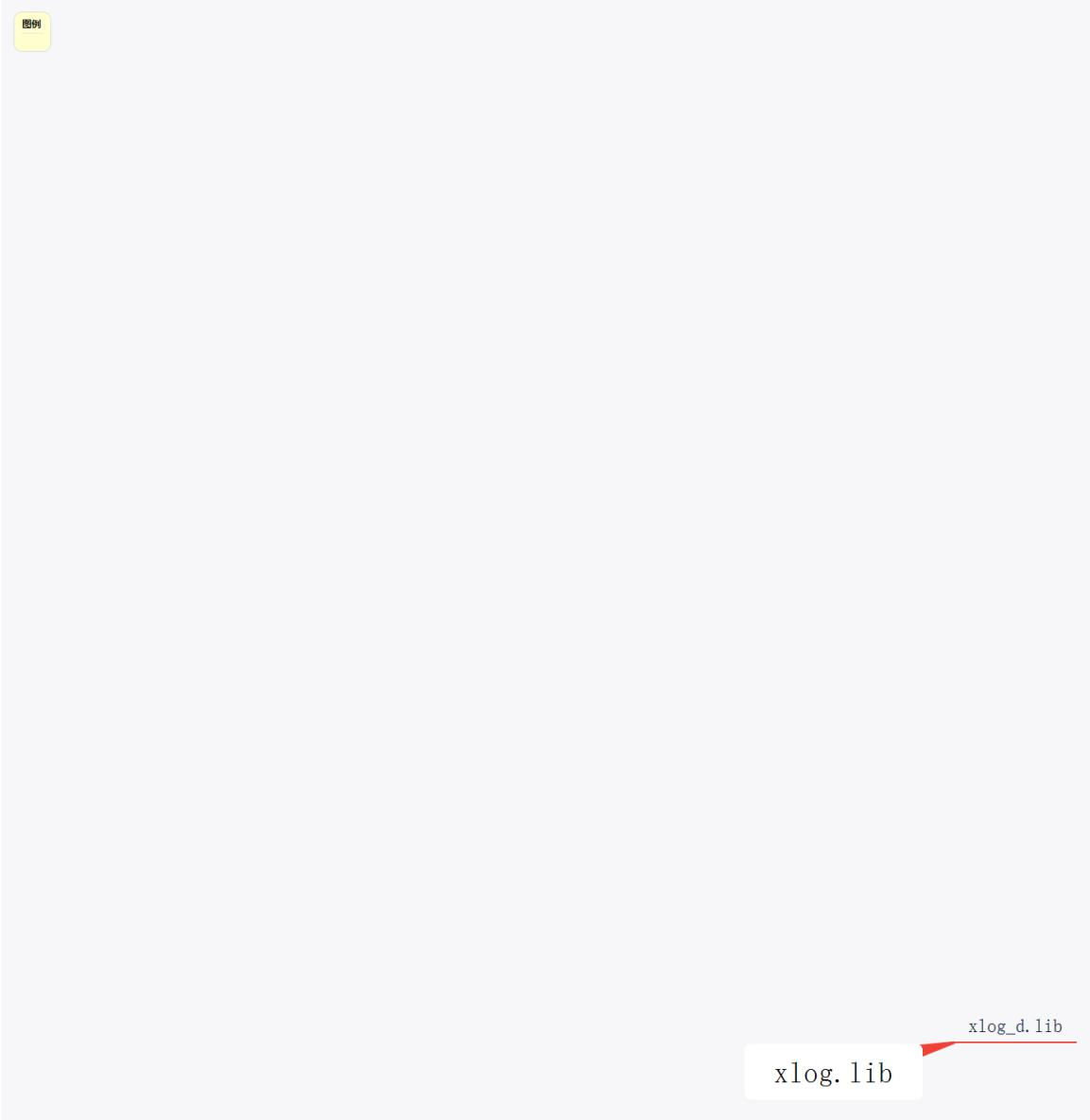
文件名



windows

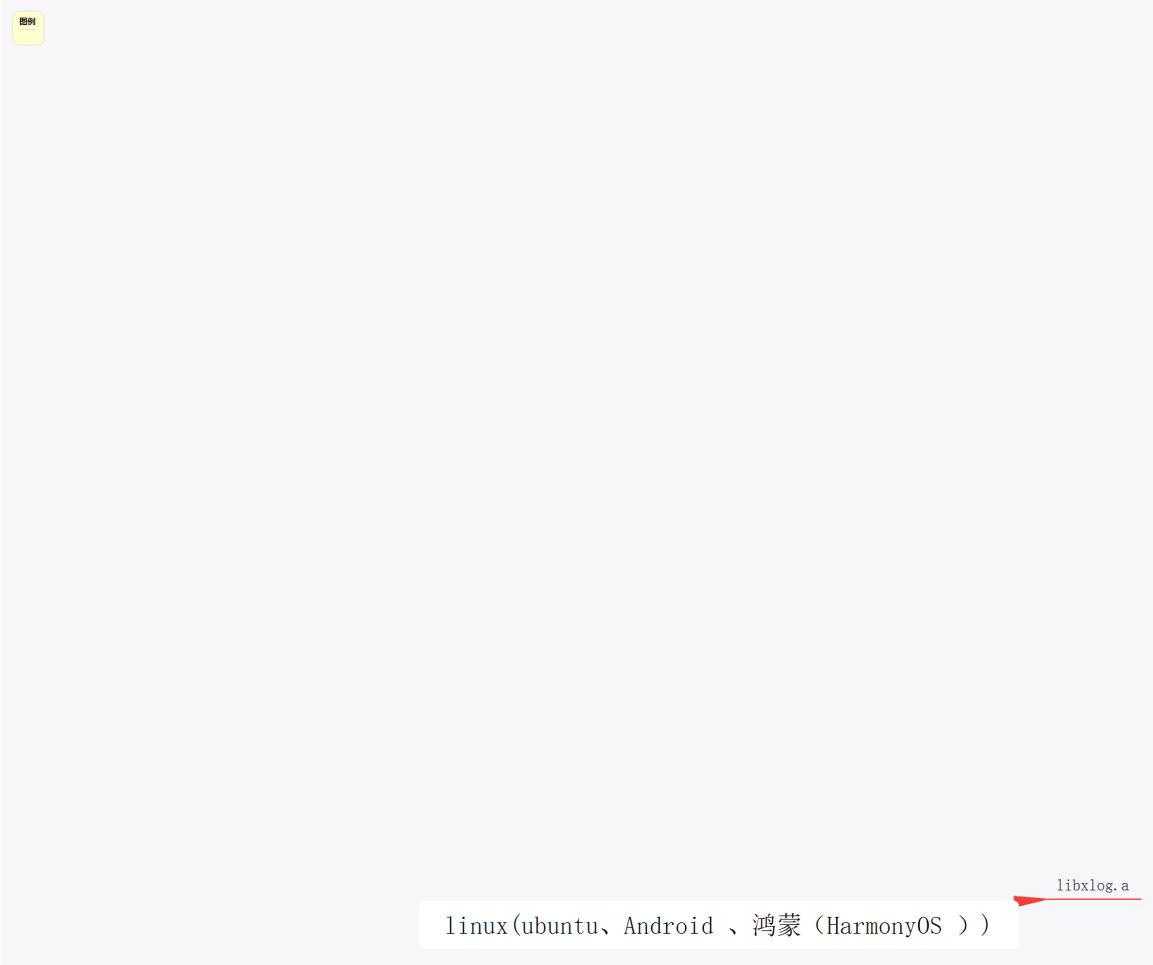


xlog.lib



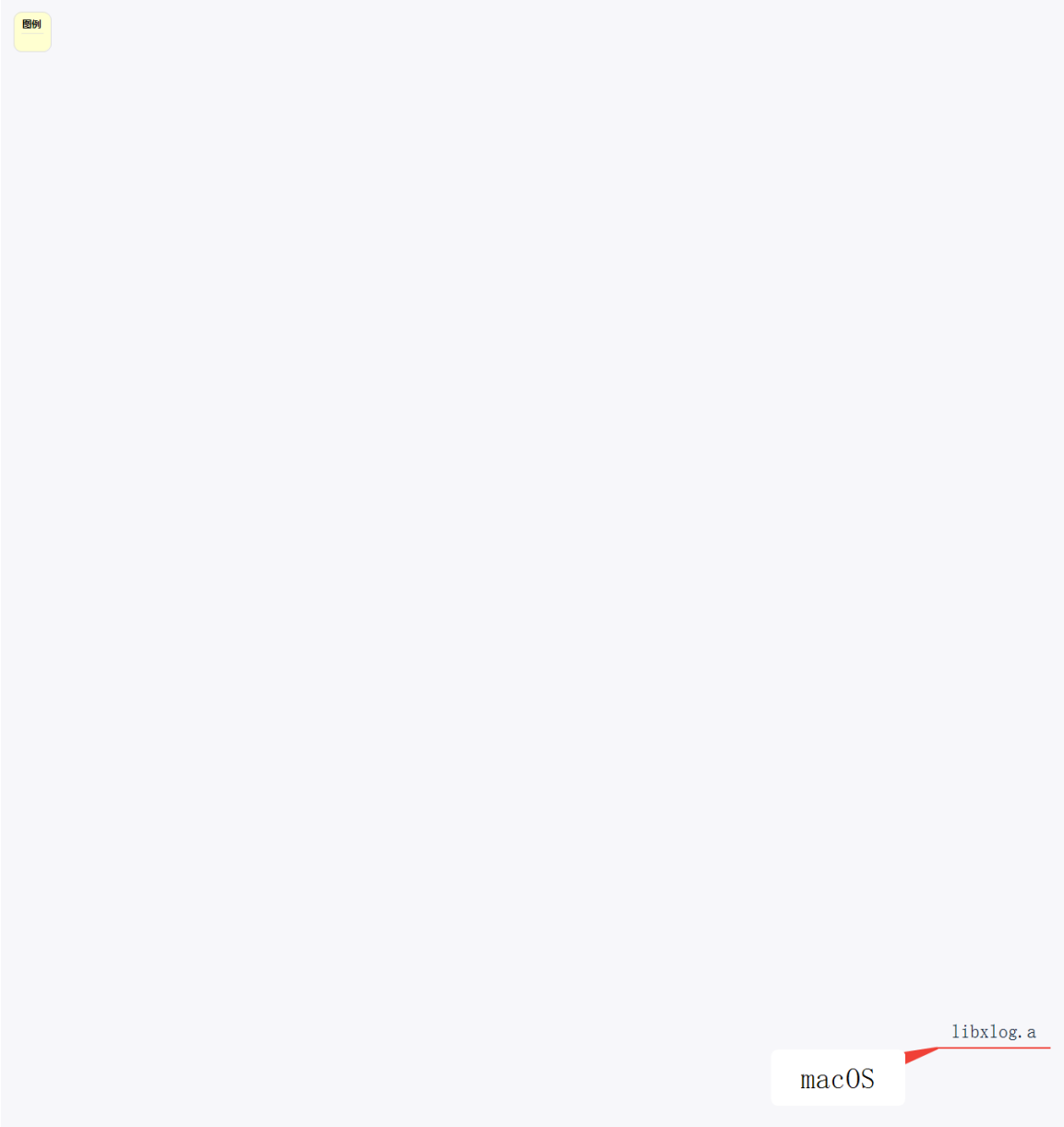
xlog_d.lib

linux(ubuntu、Android 、鸿蒙（HarmonyOS ）)



libxlog.a

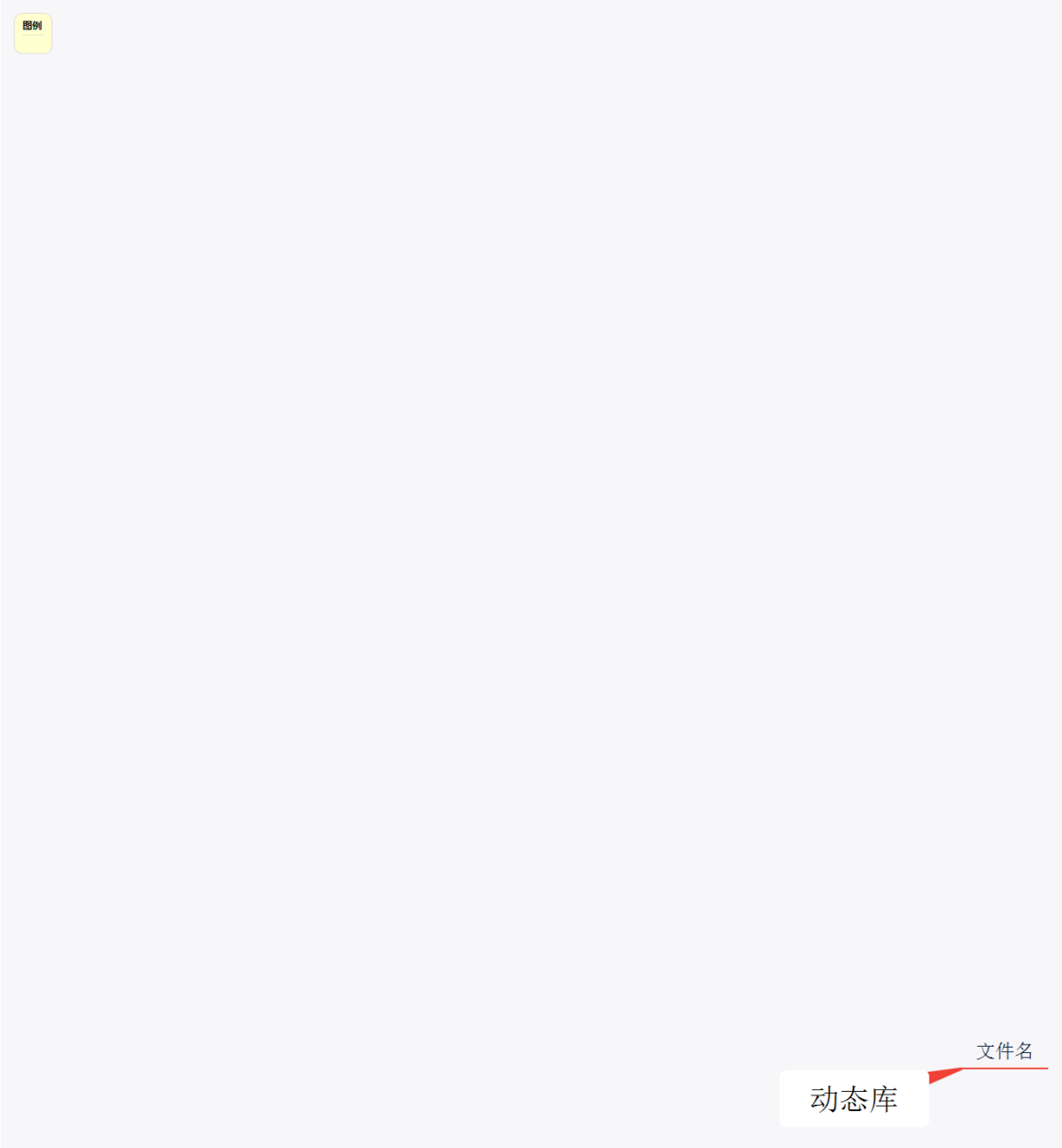
macOS



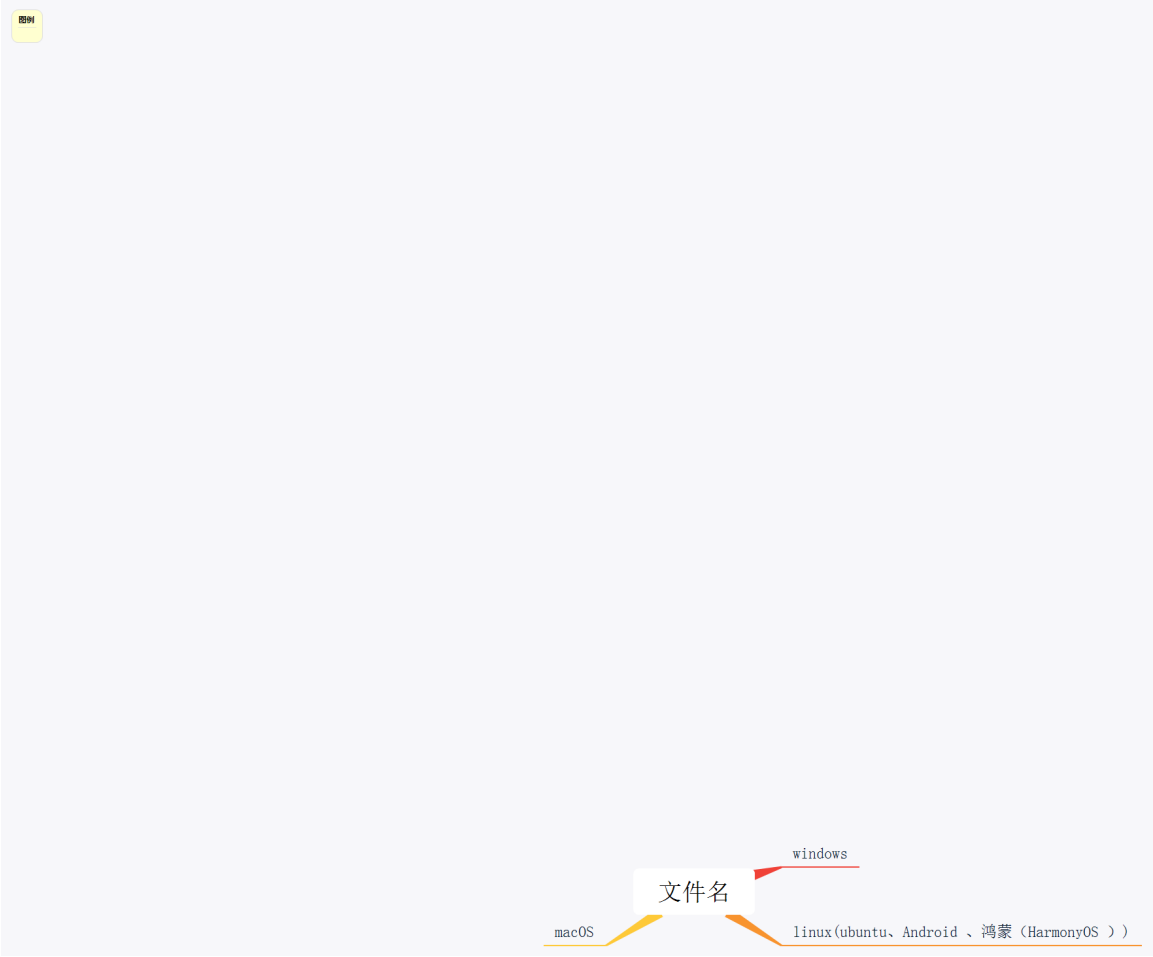
libxlog.a

基本可以理解为编译后的二进制代码，类似.o

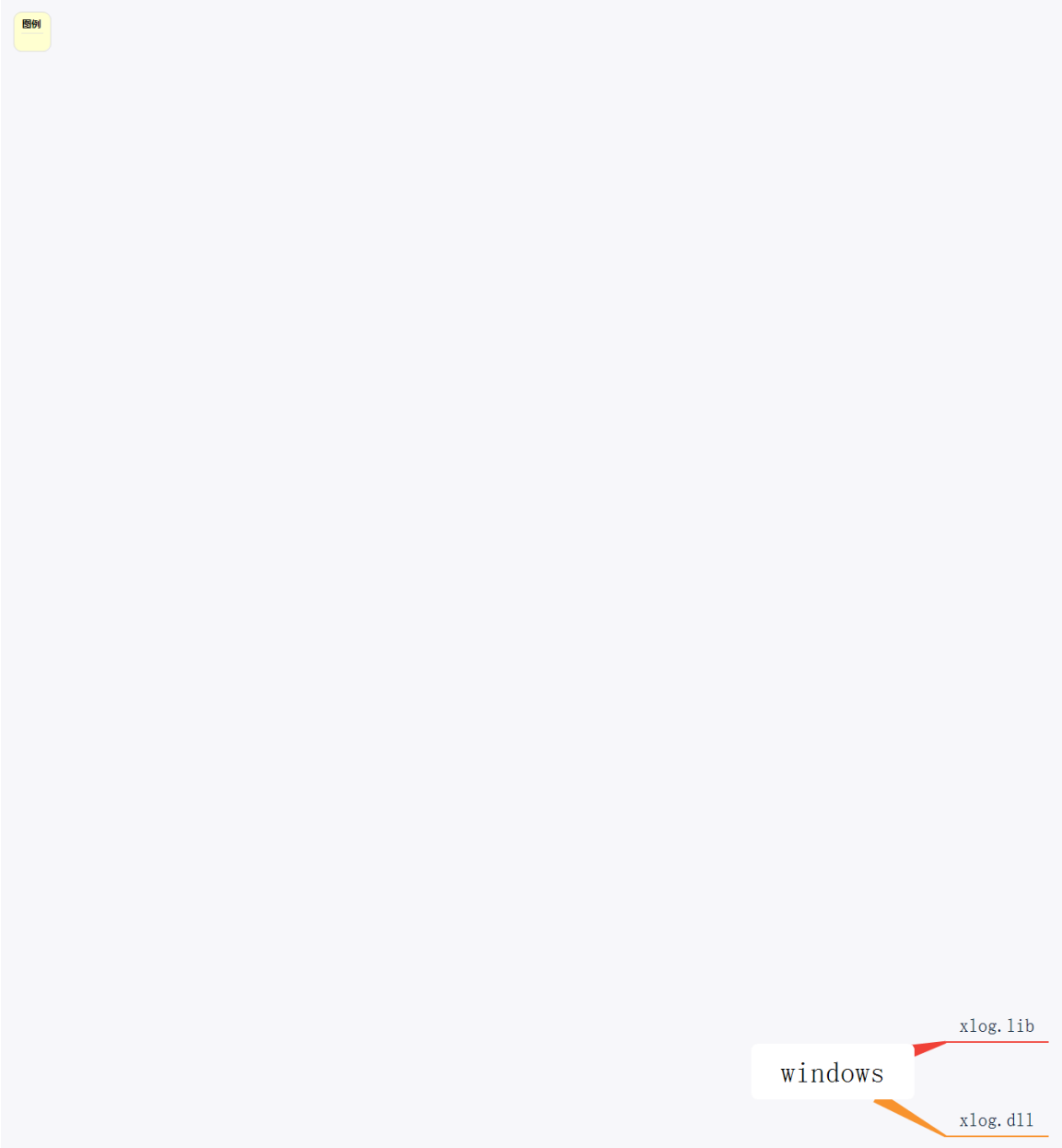
4.2.2. 动态库



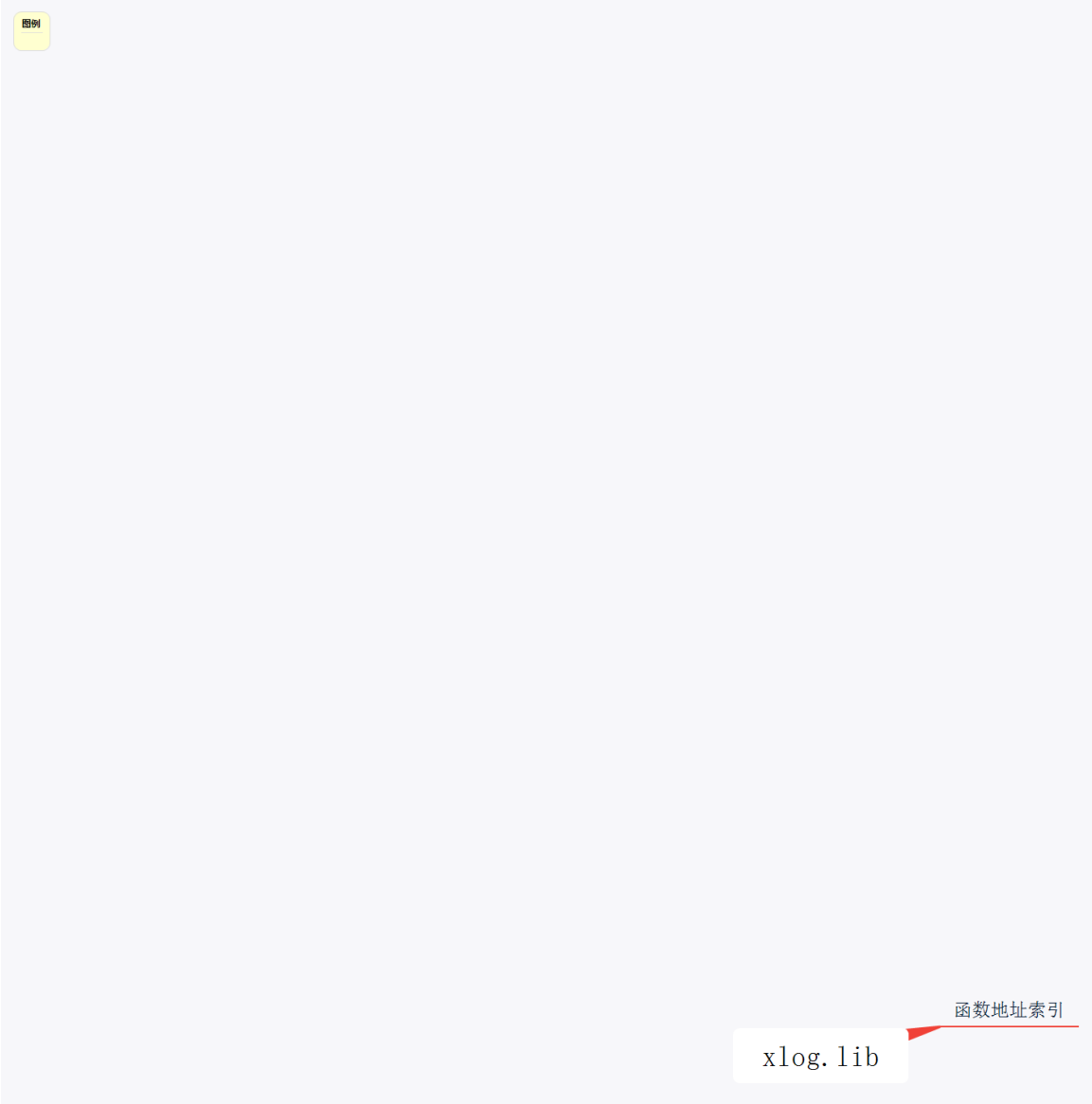
文件名



windows

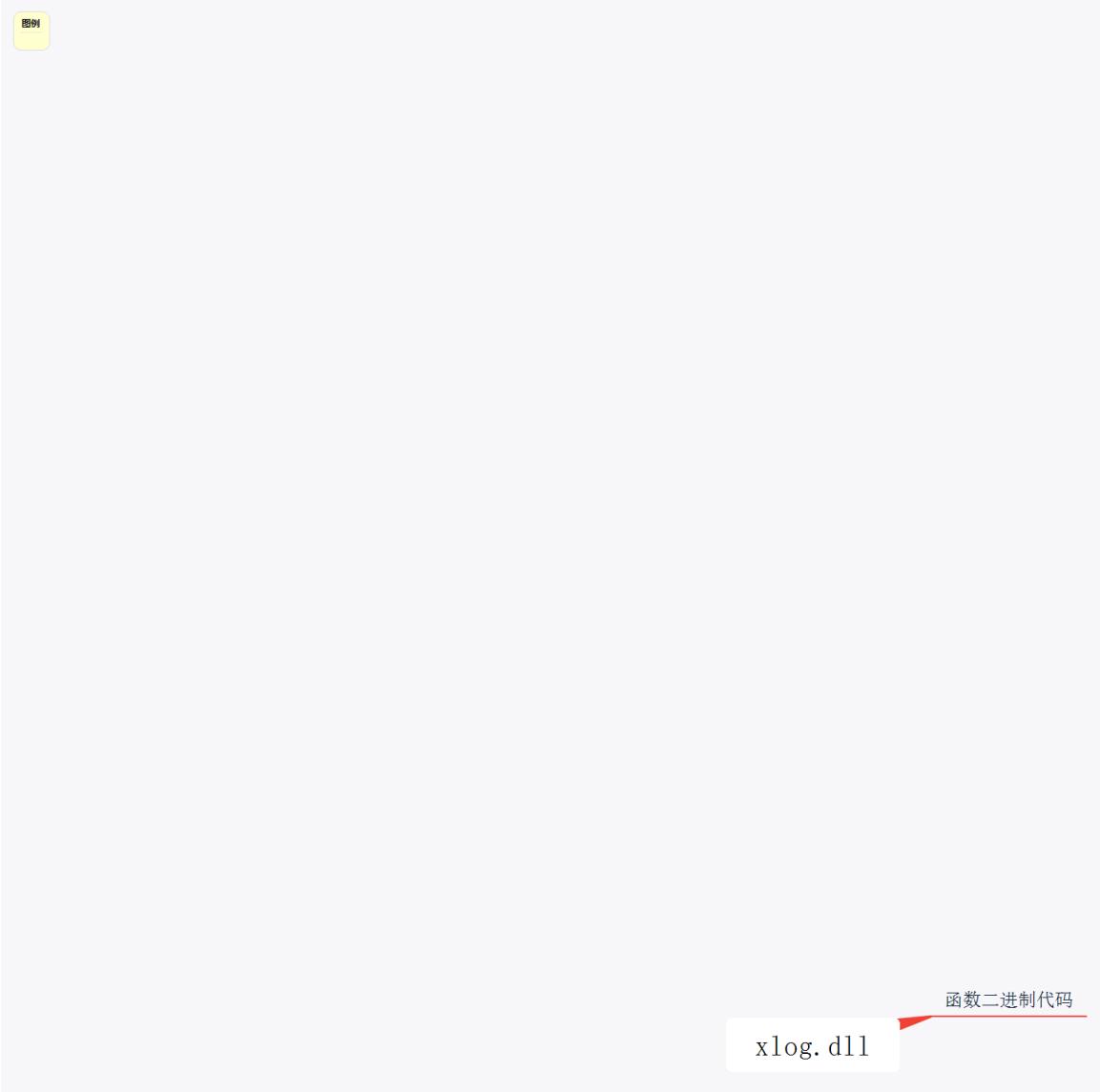


xlog.lib



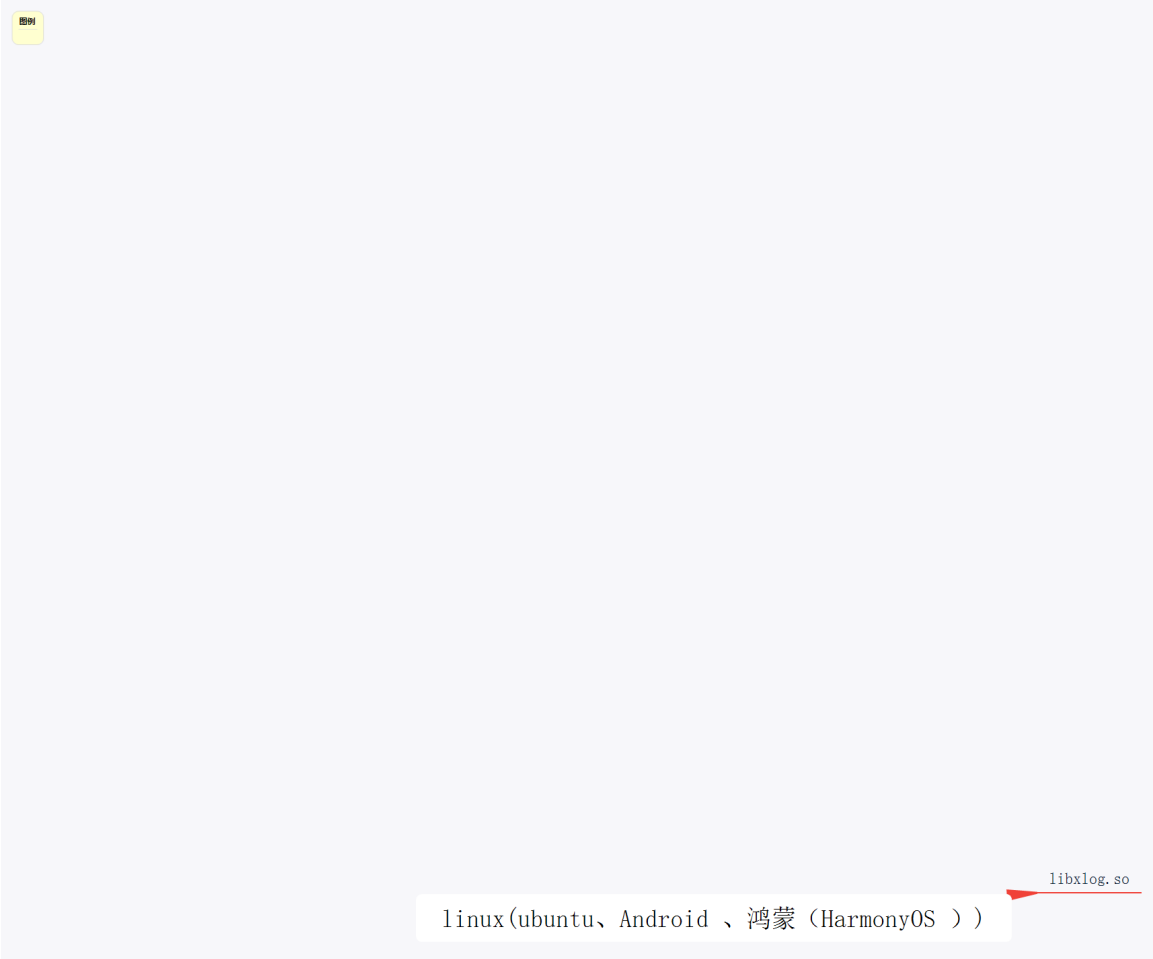
函数地址索引

xlog.dll



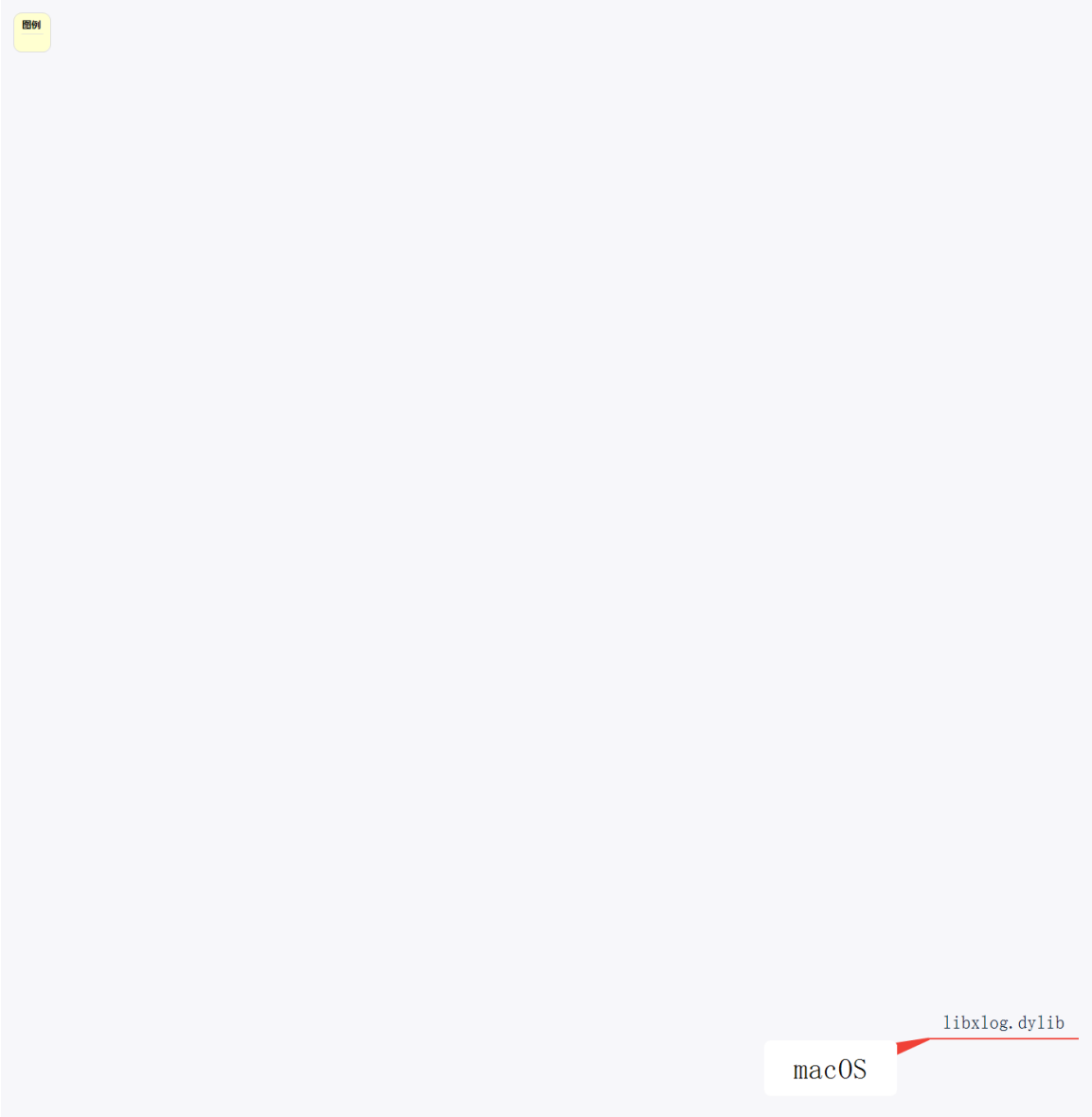
函数二进制代码

linux(ubuntu、Android 、鸿蒙（HarmonyOS ）)



libxlog.so

macOS



libxlog.dylib

4.2.3. 头文件作用

函数名称和参数类型（用于索引查找函数地址）

不引用，可以自己直接声明函数

知道名字可以调用系统api查找函数

4.3. cmake编译静态库

4.3.1. # src/xlog/CMakeLists.txt

cmake_minimum_required (VERSION 3.0)

project (xlog)

add_library(xlog STATIC xlog.cpp)

4.4. cmake 链接静态库

4.4.1. #src/test_xlog/CMakeLists.txt

cmake_minimum_required (VERSION 3.0)

project (test_xlog)

 # 指定头文件路径

include_directories ("../xlog")

 # 指定库文件加载路径

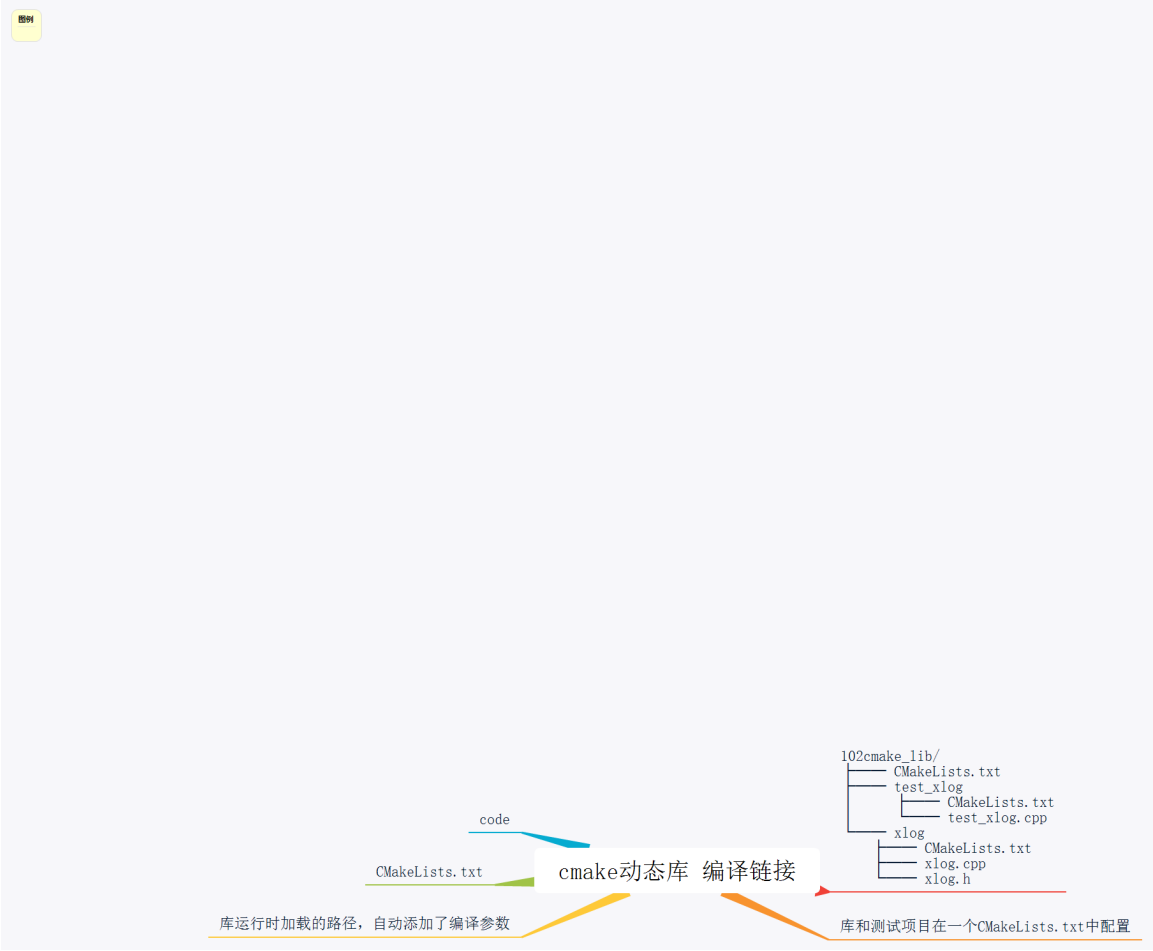
link_directories ("../xlog/build")

add_executable(test_xlog test_xlog.cpp)

#指定加载的库

target_link_libraries (test_xlog xlog)

4.5. cmake动态库 编译链接



4.5.1. 102cmake_lib/

```
├── CMakeLists.txt
├── test_xlog
│   ├── CMakeLists.txt
│   └── test_xlog.cpp
└── xlog
    ├── CMakeLists.txt
    ├── xlog.cpp
    └── xlog.h
```

4.5.2. 库和测试项目在一个CMakeLists.txt中配置

4.5.3. 库运行时加载的路径，自动添加了编译参数

-Wl,-rpath,/opt/mker/poco/lib

4.5.4. CMakeLists.txt

cmake_minimum_required (VERSION 3.20)

project (xlog)

include_directories ("xlog")

add_executable(test_xlog ../test_xlog/test_xlog.cpp)

add_library(xlog SHARED ../xlog/xlog.cpp)

target_link_libraries (test_xlog xlog)

-Wl,-rpath,/opt/mker/poco/lib

xlog_EXPORTS

__declspec(dllexport)

__declspec(dllimport)

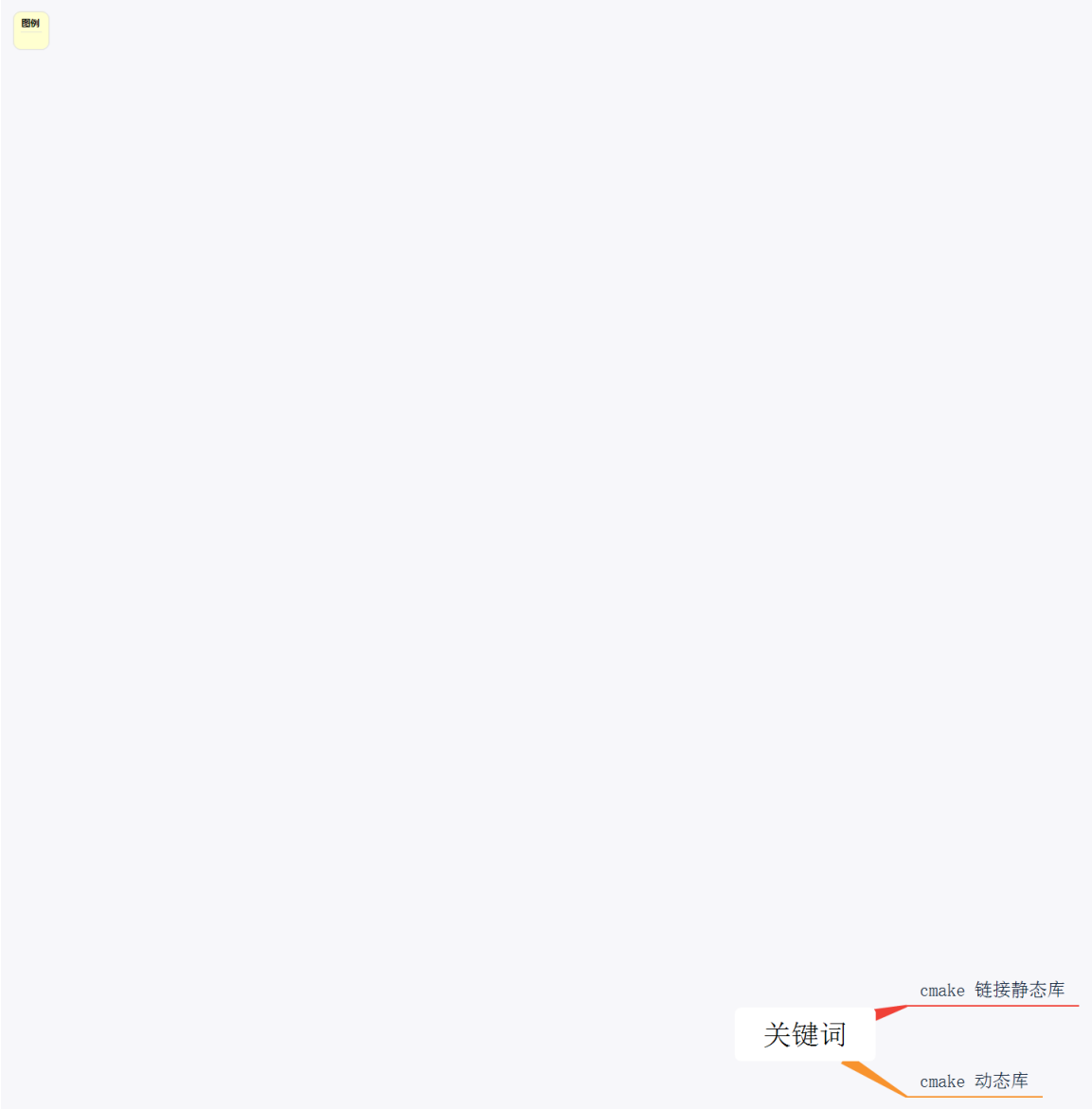
cmake自动给库项目添加 库名称_EXPORTS 预处理变量

4.5.5. code


```
#ifndef XLOG_H
#define XLOG_H
//__declspec(dllexport)
//__declspec(dllexport) 导出XLog类的函数到lib文件中
// xlog库文件调用 dllexport
// test_xlog 调用 dllimport
#ifndef _WIN32
  #define XCPP_API
#else
  #ifdef xlog_EXPORTS
    #define XCPP_API __declspec(dllexport) //库项目调用
  #else
    #define XCPP_API __declspec(dllimport) //调用库项目调用
```

```
    #endif
#endif
class XCPP_API XLog
{
public:
    XLog();
};
#endif
```

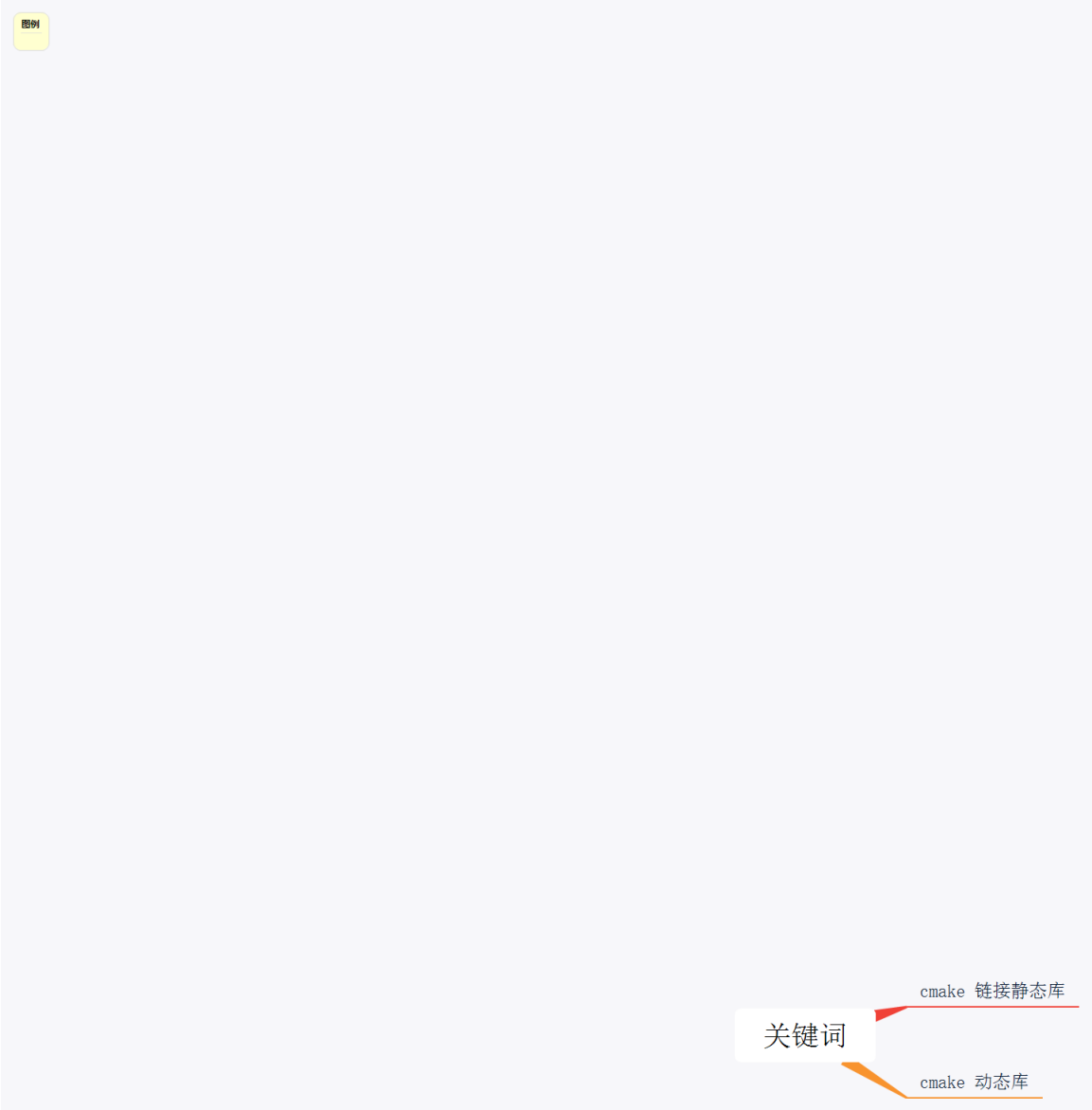
4.6. 关键词



4.6.1. cmake 链接静态库

4.6.2. cmake 动态库

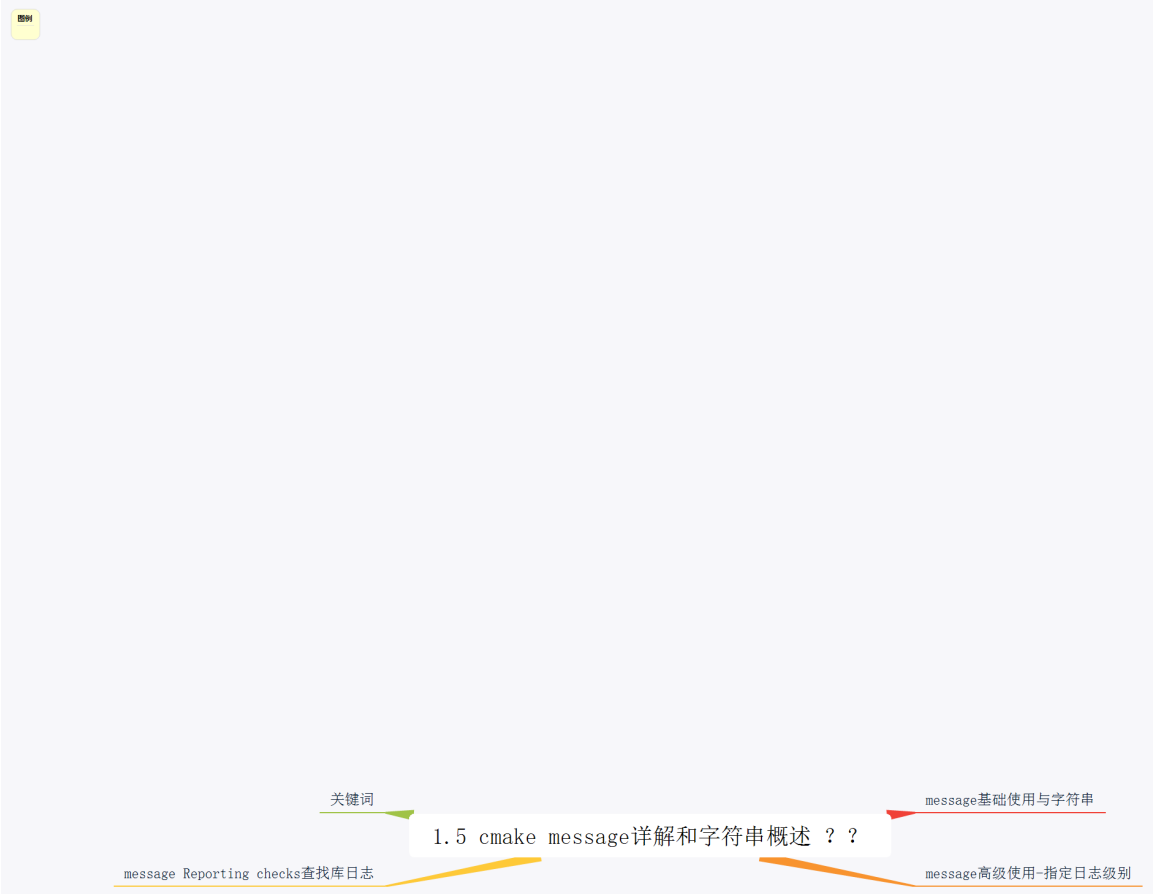
5. 关键词



5.1. cmake 链接静态库

5.2. cmake 动态库

6. 1.5 cmake message详解和字符串概述？？



6.1. message基础使用与字符串

6.1.1. 字符串

引用“” 转义\" 换行拼接\ 特殊格式[=] [=]

message(字符串)

message("字符串拼接方法

字符串第二行

")

**message("字符串拼接方法 **

字符串不换行

")

message("字符串拼接方法1" "拼接2" "拼接3")

message("字符串转义 \n \r \t \\ \$ \" ")

message([=[所有转义变量无效 \n \r]=]

6.1.2. message（arg1 arg2 arg3 ）

6.2. message高级使用-指定日志级别

6.2.1. #生成到此终止 cmake -S . -B b --log-level ERROR

#message(FATAL_ERROR "运行终止 生成终止FATAL_ERROR")

message(SEND_ERROR "继续运行生成终止 SEND_ERROR")

message(WARNING "WARNING显示行号")

message(STATUS "STATUS显示--")

message(VERBOSE "VERBOSE 默认不显--")

message(DEBUG "DEBUG 默认不显--")

message(TRACE "TRACE 默认不显--")

#ERROR(FATAL_ERROR SEND_ERROR) > WARNING > STATUS > VERBOSE > DEBUG >TRACE

6.3. message Reporting checks查找库日志

6.3.1. Reporting checks

message(<checkState> "message text" ...)

CHECK_START

Record a concise message about the check about to be performed.

CHECK_PASS

Record a successful result for a check.

CHECK_FAIL

Record an unsuccessful result for a check.

6.3.2. message("CMAKE_MESSAGE_INDENT = " ${CMAKE_MESSAGE_INDENT})

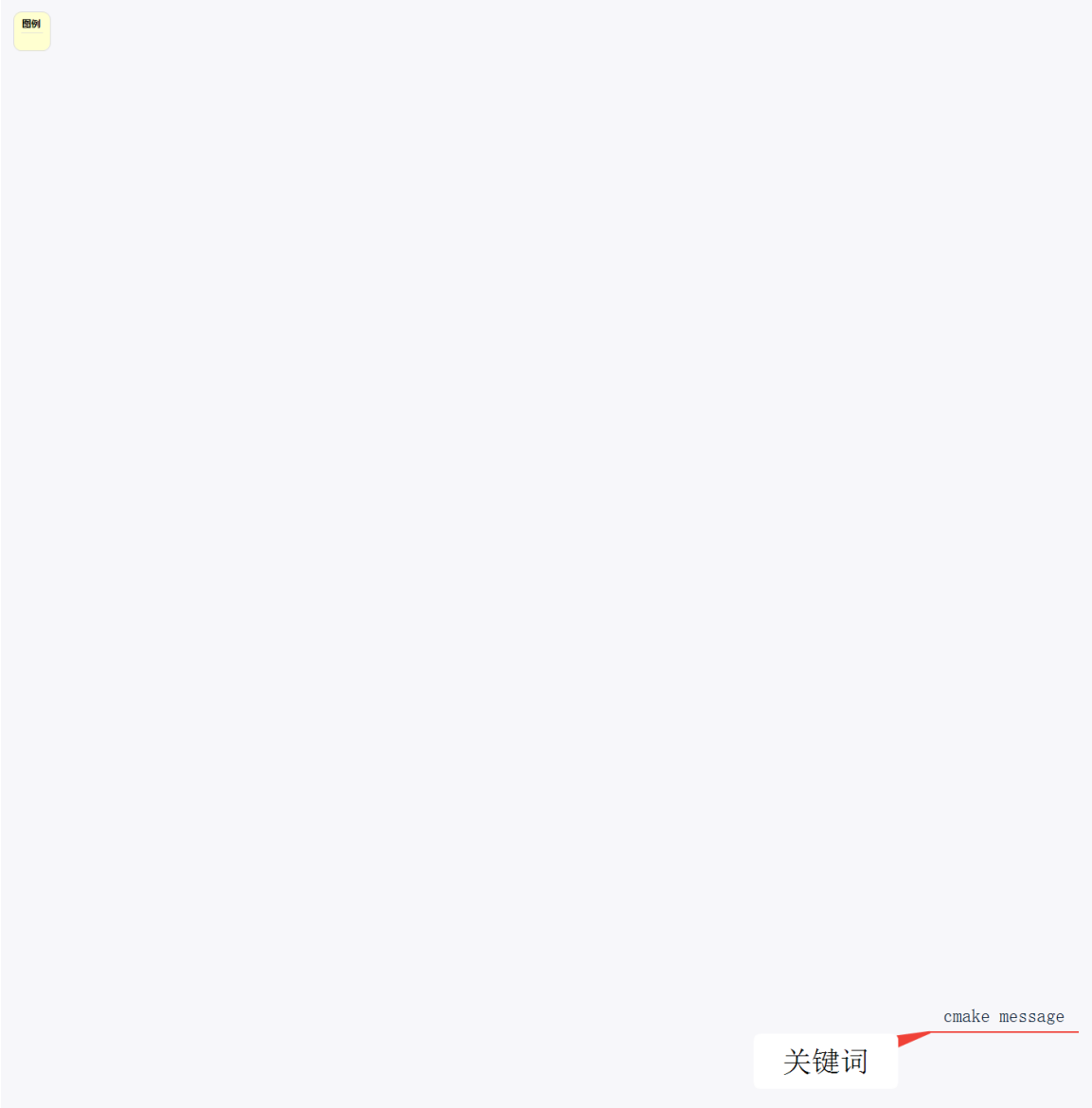
set(CMAKE_MESSAGE_INDENT " ## ") # 消息对齐

message(CHECK_START "Finding xcpp")

unset(miss)

```
message(CHECK_START "Finding xlog")
# ... do check, assume we find xlog
message(CHECK_PASS "found")

message(CHECK_START "Finding xthread")
# ... do check, assume we don't find xthread
set(miss ${miss}[xthread])
message(CHECK_FAIL "not found")

message(CHECK_START "Finding xsocket")
# ... do check, assume we don't find xsocket
set(miss ${miss}[xsocket])
message(CHECK_FAIL "not found")
set(CMAKE_MESSAGE_INDENT "")
if(miss)
  message(CHECK_FAIL "丢失组件: ${miss}")
else()
  message(CHECK_PASS "all components found")
endif()
```

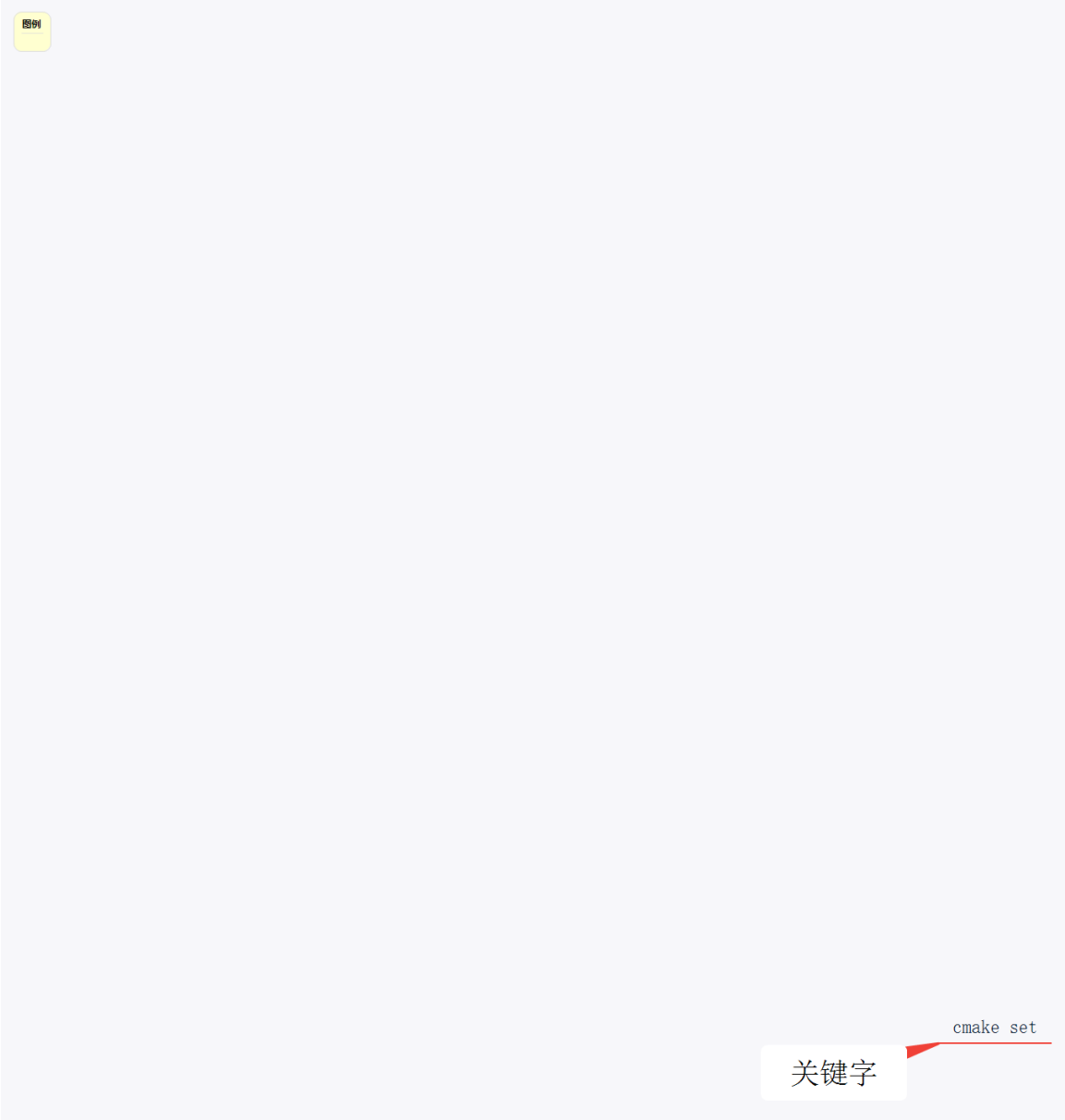
6.4. 关键词



6.4.1. cmake message

7. 1.6 cmake注释与普通变量入门？？

7.1. 关键字



7.1.1. cmake set

7.2. 变量让message输出不同的颜色

8. cmake常用特性，打印生成步骤