clang-tidy 编译安装及开发指南

19 级 马坚

1.LLVM 安装

1.1 版本信息： Ubuntu 22.04 LTS　　LLVM 14.0.0.0

1.2 编译安装（cmake 安装）

(1) 依赖工具安装：

① 更新系统，终端执行： sudo apt update　sudo apt upgrade

② 安装依赖工具，终端执行：

sudo apt install gcc g++ cmake make build-essential

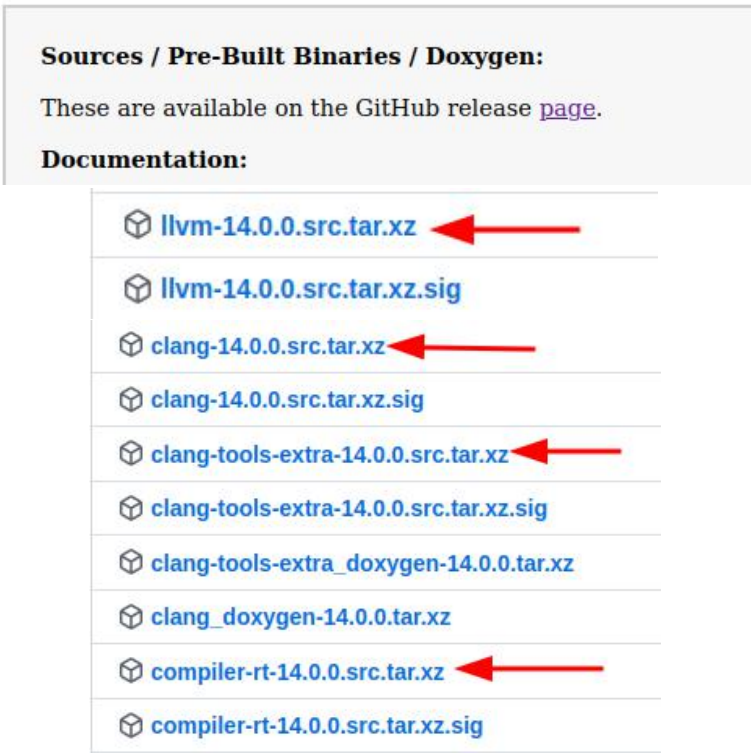
③ 检查对应版本信息

1) gcc (Ubuntu 11.2.0-19ubuntu1) 11.2.0
2) g++ (Ubuntu 11.2.0-19ubuntu1) 11.2.0
3) cmake version 3.22.1
4) GNU Make 4.3

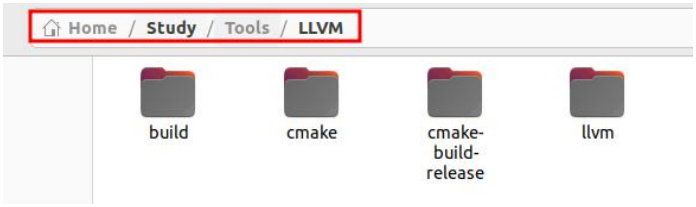
(2) LLVM 源码下载编译：

① 从官网【https://releases.llvm.org/download.html】下载如下四个源码文件：

1) LLVM source code
2) Clang source code
3) clang-tools-extra code
4) compiler-rt source code

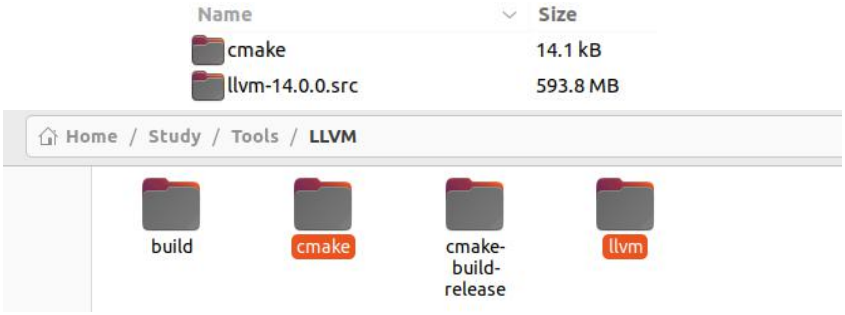


② 新建目录 LLVM，我的结构如下：/home/jackyma/Study/Tools/LLVM

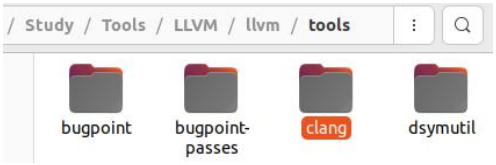


③ 源码文件路径放置

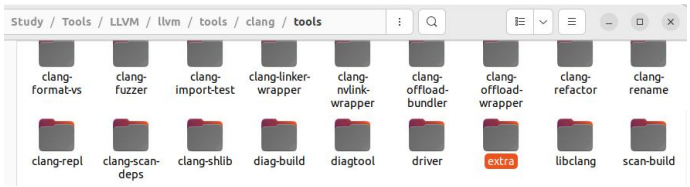
1）将 LLVM source code 源码（内含 cmake 和 llvm-14.0.0.src）解压到先前创建的 LLVM 目录下，更名为 llvm。



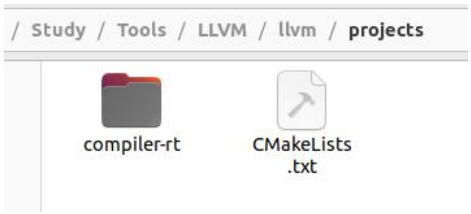
2）将 clang source code 源码（含 cmake）解压到 llvm/tools，更名为 clang。



3）将 clang-tools-extra 源码（含 cmake）解压到 llvm/tools/clang/tools，更名为 extra。



4）将 compiler-rt source 源码（含 cmake）解压到 llvm/projects，更名为 compiler-rt。



(3) 编译安装

① 在 LLVM 目录下创建 build 文件夹（与 llvm 同级）并在 build 目录下执行以下操作；

② 终端执行（将默认的 Debug 模式换成 Release 模式）：

```
cmake -G "Unix Makefiles" -DCMAKE_BUILD_TYPE=Release ../llvm
```

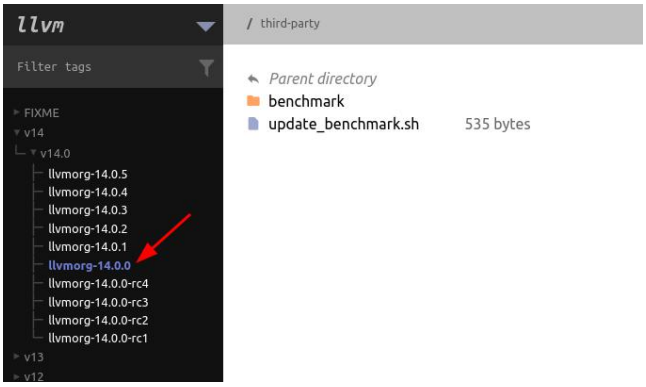
（注：此处 Release 版本与 Debug 版本的区别在于 Debug 版本大小约 50G，Release 版本为 7G，其他区别可自行百度）

这里会报错：add_subdirectory given source "" which is not an existing directory

```
"
-- check-shadowcallstack does nothing.
-- Clang version: 14.0.0
-- Not building amdgpu-arch: hsa-runtime64 not found
-- Registering Bye as a pass plugin (static build: OFF)
CMake Error at CMakeLists.txt:1256 (add_subdirectory):
  add_subdirectory given source
  "/home/jackyma/Study/Tools/LLVM/llvm/../third-party/benchmark" which is not
  an existing directory.


-- Configuring incomplete, errors occurred!
See also "/home/jackyma/Study/Tools/LLVM/build/CMakeFiles/CMakeOutput.log".
See also "/home/jackyma/Study/Tools/LLVM/build/CMakeFiles/CMakeError.log".
```

解决方案：1.手动添加目录 2.注释 CMakeLists.txt:1256 行内容
方案 1：访问网址
【https://elixir.bootlin.com/llvm/llvmorg-14.0.0/source/third-party】下载缺少内容，并按照提示手动添加至目标位置。（因为用不到，我本人选择方案 2）



方案 2：打开 LLVM/llvm 目录下的 CMakeLists.txt，三个#注释掉这令人不开心的玩意。

```
1255   set(HAVE_STD_REGEX ON CACHE BOOL "OK" FORCE)
1256   # add_subdirectory(${LLVM_THIRD_PARTY_DIR}/benchmark
1257   #  ${CMAKE_CURRENT_BINARY_DIR}/third-party/benchmark)
1258   # add_subdirectory(benchmarks)
1259 endif()
1260
```

这回就可成功 cmake 了。

③ 终端执行（编译过程相当慢）：sudo make install（可以添加 -j 参数，指定多线程编译），我是 sudo make install -j 4 （虚拟机开多线程可能会崩溃，因为编译时内存占用极高，如果多线程失败，可以单线程编译，速度会有所下降）。

④ 检测是否安装成功，终端执行： clang --version，查看是否正确打印版本信息。

```
jackyma@cyber:~$ clang --version
clang version 14.0.0
Target: x86_64-unknown-linux-gnu
Thread model: posix
InstalledDir: /usr/local/bin
jackyma@cyber:~$
```

2.Clion 编译开发 clang-tidy

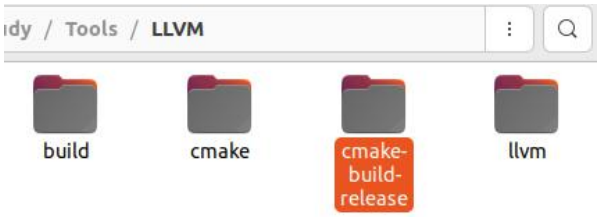
2.1 开发工具： Clion-2022.1.2

下载安装：官网

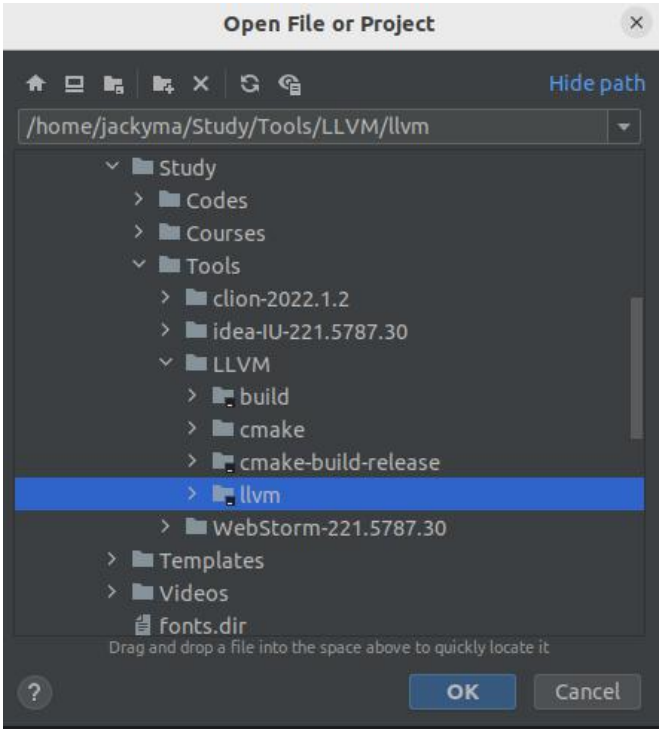
【https://www.jetbrains.com/clion/download/#section=linux】，选择 linux 版本下载安装即可；

2.2 在 Clion 中编译 clang-tidy

(1) 在 LLVM 目录下新建目录 cmake-build-release；



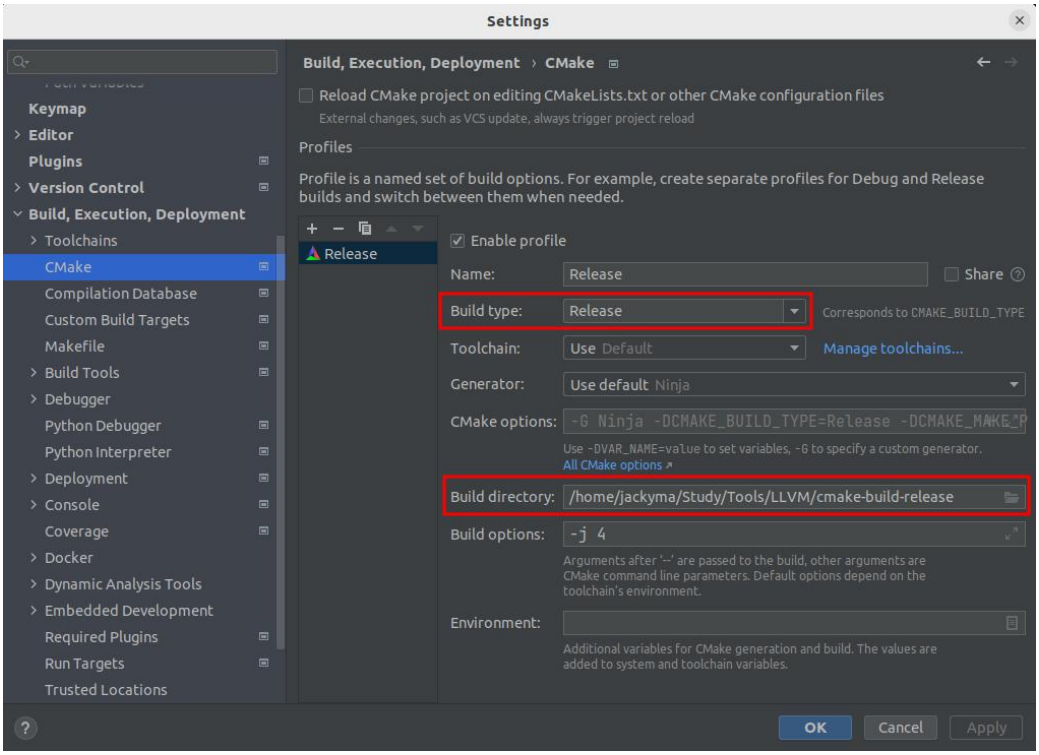
(2) Clion 中导入项目，选择 file-->open ,选择 llvm 项目导入；



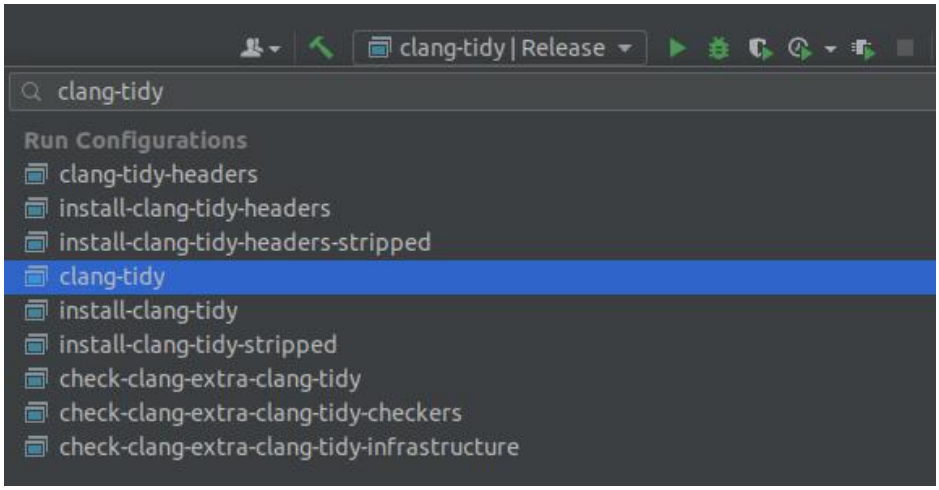
配置项目编译选项，选择

file--->settings--->Build,Exception,Deployment--->cmake，

Build type 选择 Release，Build directory 选择刚刚新建的目录，Build options 添加编译时线程数，点击 Apply 即可；



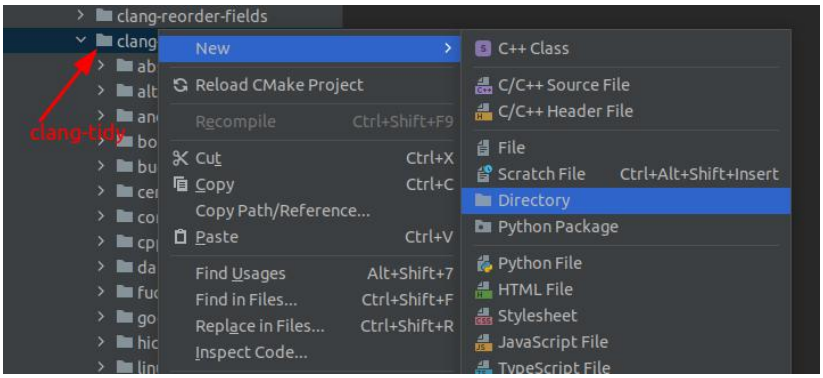
在 Clion 右上角编译 target 中选择 clang-tidy ，然后 run 即可完成 clang-tidy 项目编译。



进入到 cmake-build-release 下 bin 目录，可以看到编译完成的 clang-tidy 可执行文件；

2.3 在 Clion 中新建 clang-tidy 规则集

(1) clang-tidy 目录下新建子目录“gjb”



(2) 在子目录 gjb 下创建 GjbTidyModule.cpp，其内容可从其他 check 目录下直接复制过来，但要进行以下修改。

① 更改类名为 GjbModule；

② 删除 addCheckFactories()函数下的内容；

③ 将 ClangTidyModuleRegistry::Add<>尖括号中内容更改为 GjbModule；

④ 将***ModuleAnchorSource 修改为 GjbModuleAnchorSource；

⑤ 修改相关注释内容，结果如下（文末附上代码）：

```cpp
//
// Created by jackyma on 6/16/22.
//

#include "../ClangTidy.h"
#include "../ClangTidyModule.h"
#include "../ClangTidyModuleRegistry.h"

namespace clang {
namespace tidy {
namespace gjb {

class GjbModule : public ClangTidyModule {
public:
  void addCheckFactories(ClangTidyCheckFactories &CheckFactories) override {
  }
};

} // namespace gjb

// Register the GjbTidyModule using this statically initialized variable.
static ClangTidyModuleRegistry::Add<gjb::GjbModule>
    X( Name: "gjb-module",  Desc: "Adds Gjb lint checks.");

// This anchor is used to force the linker to link in the generated object file
// and thus register the GjbModule.
volatile int GjbModuleAnchorSource = 0;

} // namespace tidy
} // namespace clang
```

(3) 在子目录 test 下创建 Cmakelist.txt（也可从其他 check 中直接复制过来，详见文末附录 2）

① add_clang_library 后的第一项改为“clangTidyGjbModule”

② add_clang_library 后加入 GjbTidyModule.cpp

(4) 在 clang-tidy 目录下修改 Cmakelist.txt

① 增加 add_subdirectory(gjb)

```
61  add_subdirectory(darwin)
62  add_subdirectory(fuchsia)
63  add_subdirectory(gjb)
64  add_subdirectory(google)
65  add_subdirectory(hicpp)
```

② 在“set(ALL_CLANG_TIDY_CHECKS” 后加入 clangTidyGjbModule

```
80  set(ALL_CLANG_TIDY_CHECKS
81    clangTidyAndroidModule
82    clangTidyAbseilModule
83    clangTidyAlteraModule
84    clangTidyBoostModule
85    clangTidyBugproneModule
86    clangTidyCERTModule
87    clangTidyConcurrencyModule
88    clangTidyCppCoreGuidelinesModule
89    clangTidyDarwinModule
90    clangTidyFuchsiaModule
91    clangTidyGjbModule
92    clangTidyGoogleModule
93    clangTidyHICPPModule
94    clangTidyLinuxKernelModule
95    clangTidyLLVMModule
```

(5) 在 clang-tidy 目录下修改 ClangTidyForcelinker.h

在命名空间 tidy 下增加：

```
// This anchor is used to force the linker to link the GjbModule.
extern volatile int GjbModuleAnchorSource;
static int LLVM_ATTRIBUTE_UNUSED GjbModuleAnchorDestination =
    GjbModuleAnchorSource;
```

```
67
68  // This anchor is used to force the linker to link the GjbModule.
69  extern volatile int GjbModuleAnchorSource;
70  static int LLVM_ATTRIBUTE_UNUSED GjbModuleAnchorDestination =
71      GjbModuleAnchorSource;
72
```

(6) 在 clang-tidy 目录终端下

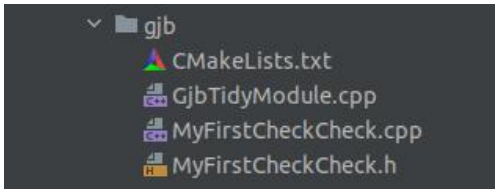
目录：~/Study/Tools/LLVM/llvm/tools/clang/tools/extra/clang-tidy

执行 python add_new_check.py gjb my-first-check

这里的 python 一定要是 2.x 版本，不然会报错。成功后结果如下：

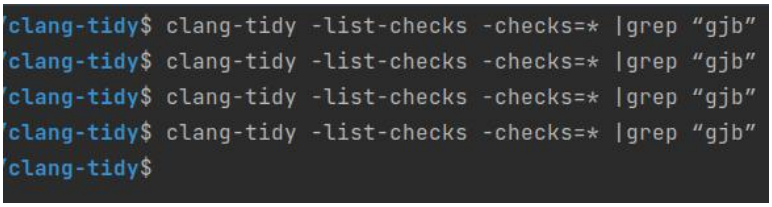
```
jackyma@cyber:~/Study/Tools/LLVM/llvm/tools/clang/tools/extra/cl
Updating gjb/CMakeLists.txt...
Creating gjb/MyFirstCheckCheck.h...
Creating gjb/MyFirstCheckCheck.cpp...
Updating gjb/GjbTidyModule.cpp...
Updating ../docs/ReleaseNotes.rst...
Creating ../test/clang-tidy/checkers/gjb-my-first-check.cpp...
Creating ../docs/clang-tidy/checks/gjb-my-first-check.rst...
Updating ../docs/clang-tidy/checks/list.rst...
Done. Now it's your turn!
```

目录 gjb 结构如下：

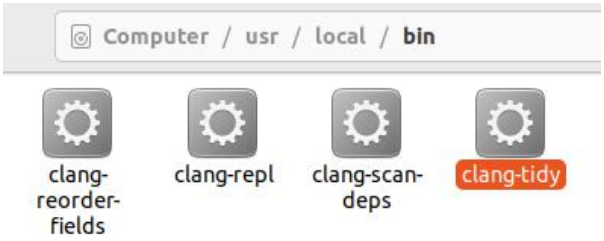


(7) 构建 clang-tidy

执行 clang-tidy -list-checks -checks=* |grep "gjb" 检验是否成功创建规则集 gjb 。发现什么也没有？？？



这是因为我们先前用的 sudo make install，这条命令将可执行文件 clang-tidy 写入到了 /user/local/bin 系统目录下，当我们在命令行输入 clang-tidy 时，执行的是这个位置的 clang-tidy，而非我们刚刚构建生产的 clang-tidy 。



Ctrl+Alt+T 新开终端，输入以下命令

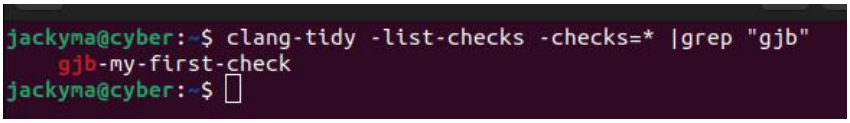
cd /usr/local/bin/ sudo rm clang-tidy

接着输入 sudo gedit ~/.bashrc

在文件最下方增加一行，用于添加环境变量：

export PATH=/home/jackyma/Study/Tools/LLVM/cmake-build-release/bin:${PATH}

再次执行 clang-tidy -list-checks -checks=* |grep "gjb" 可以查看到结果 。



[备注]
如有任何疑问，可联系本人咨询，Git 仓库地址：
【https://bdgit.educoder.net/pvqf6mep3/SQA-Homework.git】
联系方式：15366403320（微信同号）

[附录 1] GjbTidyModule.cpp

```
//
// Created by jackyma on 6/16/22.
//

#include "../ClangTidy.h"
#include "../ClangTidyModule.h"
#include "../ClangTidyModuleRegistry.h"
#include "MyFirstCheckCheck.h"

namespace clang {
namespace tidy {
namespace gjb {

class GjbModule : public ClangTidyModule {
public:
  void addCheckFactories(ClangTidyCheckFactories &CheckFactories) override {
    CheckFactories.registerCheck<MyFirstCheckCheck>(
        "gjb-my-first-check");
  }
};

} // namespace gjb

// Register the GjbTidyModule using this statically initialized variable.
static ClangTidyModuleRegistry::Add<gjb::GjbModule>
    X("gjb-module", "Adds Gjb lint checks.");

// This anchor is used to force the linker to link in the generated object file
// and thus register the GjbModule.
volatile int GjbModuleAnchorSource = 0;

} // namespace tidy
} // namespace clang
```

[附录 2] CMakeLists.txt

```
set(LLVM_LINK_COMPONENTS
        FrontendOpenMP
        Support
        )

add_clang_library(clangTidyGjbModule
        GjbTidyModule.cpp

        LINK_LIBS
        clangTidy
        clangTidyUtils

        DEPENDS
        omp_gen
        )

clang_target_link_libraries(clangTidyGjbModule
        PRIVATE
        clangAnalysis
        clangAST
        clangASTMatchers
        clangBasic
        clangLex
        clangSerialization
        clangTooling
        )
```