iOS LLVM Pass代码Review（二）

一、环境配置

目前本人使用环境：

> 🌟 macos 12.6
>
> xcode 14.0.1
>
> clang&&llvm 13.0.1

那就找xcode 对应版本的llvm，本人安装的是llvm 13.0.1_2

```
sudo brew install llvm@13
```

二、编译

1、拷贝llvm 导师项目

```
git clone https://github.com/banach-space/llvm-tutor
```

2、配置llvm和项目路径

```
export LLVM_DIR=/opt/homebrew/Cellar/llvm@13/13.0.1_2
export LLVM_TUTOR_DIR=/Users/xiakejie/ME/break/llvm-tutor
```

3、编译dylib

```
mkdir build
cd $build
cmake -DLT_LLVM_INSTALL_DIR=$LLVM_DIR  ../
make
```

4、练练手

拿libHelloWorld.dylib

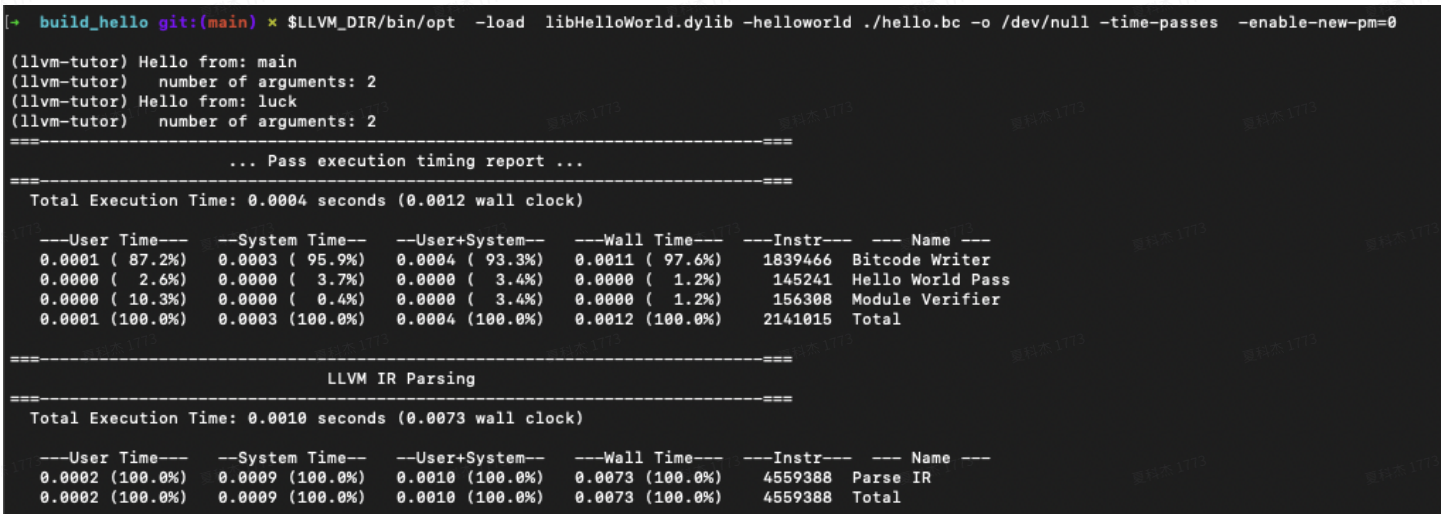
.c 转.bc 文件上一篇文章有讲到

```
$LLVM_DIR/bin/clang -O1 -S -emit-llvm  $LLVM_TUTOR_DIR/inputs/input_for_hello.c

$LLVM_DIR/bin/opt  -load  libHelloWorld.dylib -helloworld ./hello.bc -o /dev/nul

$LLVM_DIR/bin/opt -load  ./libHelloWorld.dylib -help | grep hello

```



5、编写代码规范检查的pass

1、检查类名(Interface)，不能带有下划线

2、检查变量(Interface)，不能带有下划线

```
#include <iostream>
#include "clang/AST/AST.h"
#include "clang/AST/ASTConsumer.h"
#include "clang/ASTMatchers/ASTMatchers.h"
#include "clang/ASTMatchers/ASTMatchFinder.h"
#include "clang/Frontend/CompilerInstance.h"
#include "clang/Frontend/FrontendPluginRegistry.h"

using namespace clang;
using namespace std;
using namespace llvm;
using namespace clang::ast_matchers;
```

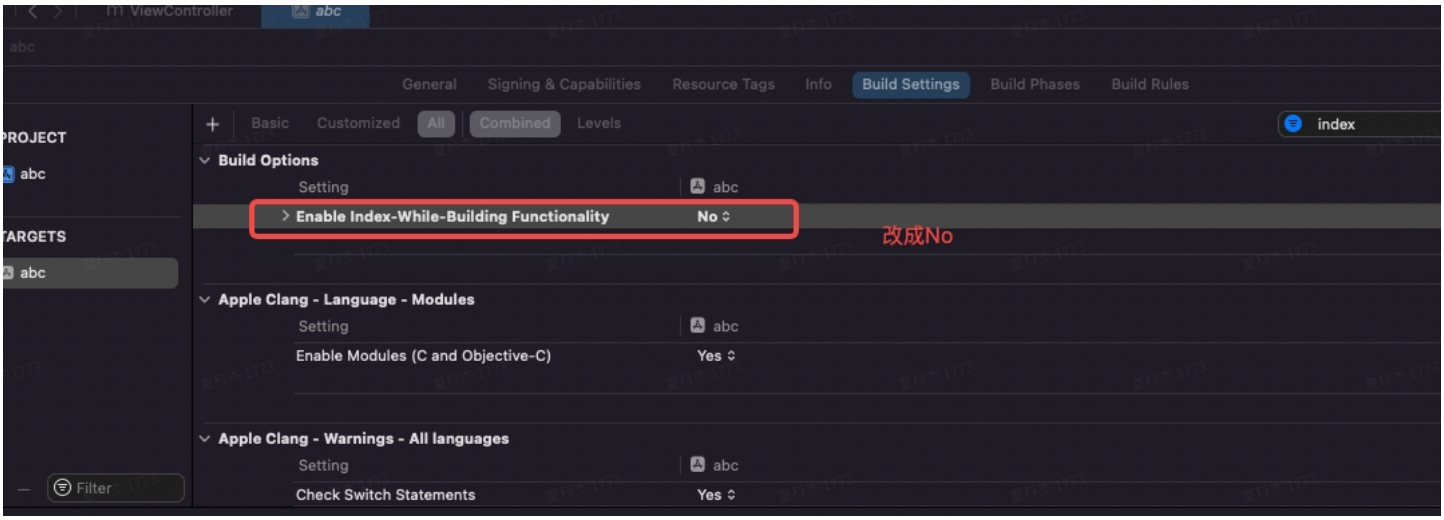
```
13
14 namespace CodeReview {
15     class TestHandler : public MatchFinder::MatchCallback{
16     private:
17         CompilerInstance &ci;
18
19     public:
20         TestHandler(CompilerInstance &ci) :ci(ci) {}
21
22         //判断是否是用户源文件
23         bool isUserSourceCode(const string filename) {
24             //文件名不为空
25             if (filename.empty()) return  false;
26             //非xcode中的源码都认为是用户的
27             if (filename.find("/Applications/Xcode.app/") == 0) return false;
28             if (filename.find("/Applications/Xcode13.4.app/") == 0) return false
29             return  true;
30         }
31
32         // 代码检查的回调方法
33         void run(const MatchFinder::MatchResult &Result) {
34
35             // 检查类名(Interface)，不能带有下划线
36             if (const ObjCInterfaceDecl *decl = Result.Nodes.getNodeAs<ObjCInter
37                 string filename = ci.getSourceManager().getFilename(decl->getSou
38                 if ( !isUserSourceCode(filename) ) return;
39                 size_t pos = decl->getName().find('_');
40                 if (pos != StringRef::npos) {
41                     DiagnosticsEngine &D = ci.getDiagnostics();
42                     // 获取位置
43                     SourceLocation loc = decl->getLocation().getLocWithOffset(po
44                     D.Report(loc, D.getCustomDiagID(DiagnosticsEngine::Warning,
45                 }
46             }
47             // 检查变量(Interface)，不能带有下划线
48             if (const VarDecl *decl = Result.Nodes.getNodeAs<VarDecl>("VarDecl")
49                 string filename = ci.getSourceManager().getFilename(decl->getSou
50                 if ( !isUserSourceCode(filename) ) return;
51                 size_t pos = decl->getName().find('_');
52                 if (pos != StringRef::npos && pos != 0) {
53                     DiagnosticsEngine &D = ci.getDiagnostics();
54                     SourceLocation loc = decl->getLocation().getLocWithOffset(po
55                     D.Report(loc, D.getCustomDiagID(DiagnosticsEngine::Warning,
56                 }
57             }
58         }
59     };
```

```
60
61
62      // 定义语法树的接受事件
63      class TestASTConsumer: public ASTConsumer{
64      private:
65          MatchFinder matcher;
66          TestHandler handler;
67
68      public:
69          TestASTConsumer(CompilerInstance &ci) :handler(ci) {
70              matcher.addMatcher(objcInterfaceDecl().bind("ObjCInterfaceDecl"), &h
71              matcher.addMatcher(varDecl().bind("VarDecl"), &handler);
72              matcher.addMatcher(objcMethodDecl().bind("ObjCMethodDecl"), &handler
73          }
74          void HandleTranslationUnit(ASTContext &Ctx) {
75              printf("CodeReview1: All ASTs has parsed.");
76              DiagnosticsEngine &D = Ctx.getDiagnostics();
77              // 在编译log中可以看到
78              //D.Report(D.getCustomDiagID(DiagnosticsEngine::Warning, "CodeReview
79             // D.Report(D.getCustomDiagID(DiagnosticsEngine::Error, "CodeReview错
80              matcher.matchAST(Ctx);
81          }
82      };
83
84
85      // 定义触发插件的动作
86      class TestAction : public PluginASTAction{
87      public:
88          unique_ptr<ASTConsumer> CreateASTConsumer(CompilerInstance &CI,
89                                                    StringRef InFile){
90              return unique_ptr<TestASTConsumer> (new TestASTConsumer(CI));
91
92          }
93
94          bool ParseArgs(const CompilerInstance &CI,
95                         const std::vector<std::string> &arg){
96              return true;
97          }
98      };
99  }
100
101
102 // 告知clang,注册一个新的plugin
103 static FrontendPluginRegistry::Add<CodeReview::TestAction>
104 X("CodeReview", "Test a new Plugin");
105 // X 变量名，可随便写，也可以写自己有意思的名称
106 // CodeReview  插件名称，很重要，这个是对外的名称
```

```
107 // Test a new Plugin  插件备注
```

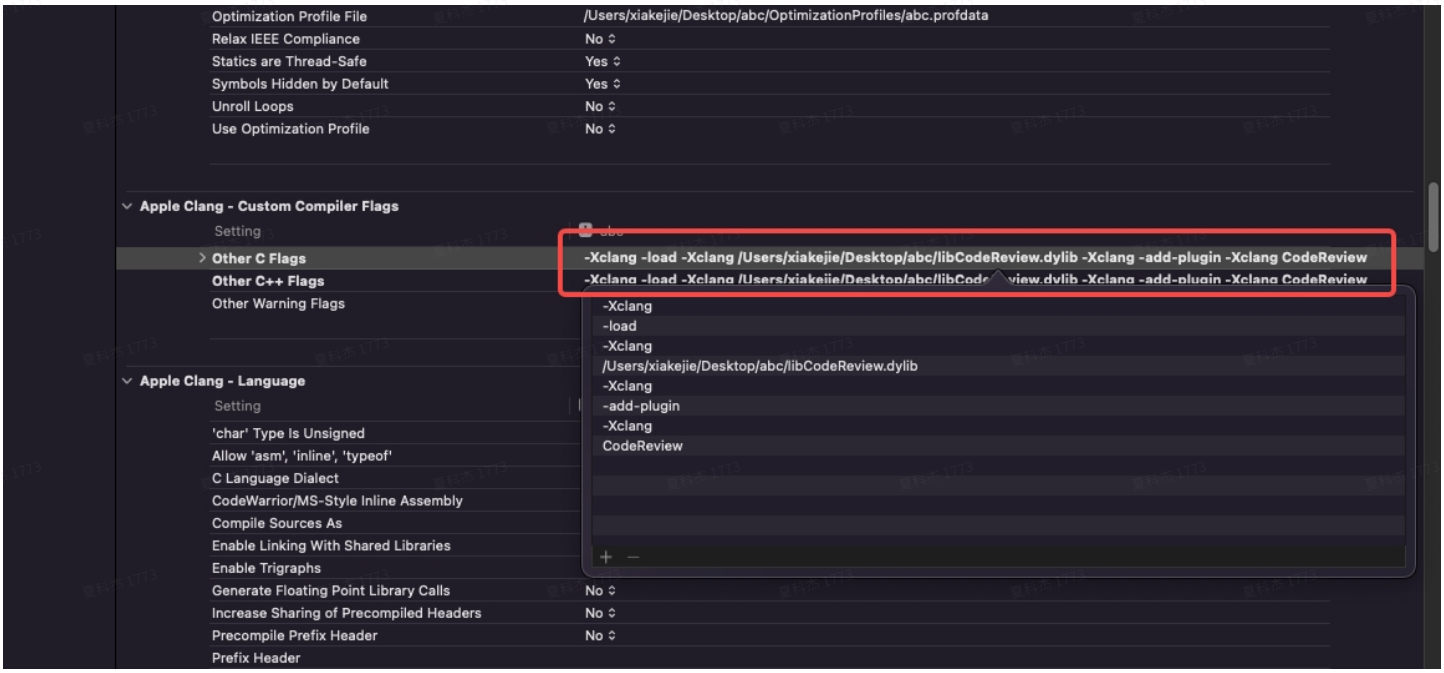
5、xcode 配置

5.1、配置Build Options

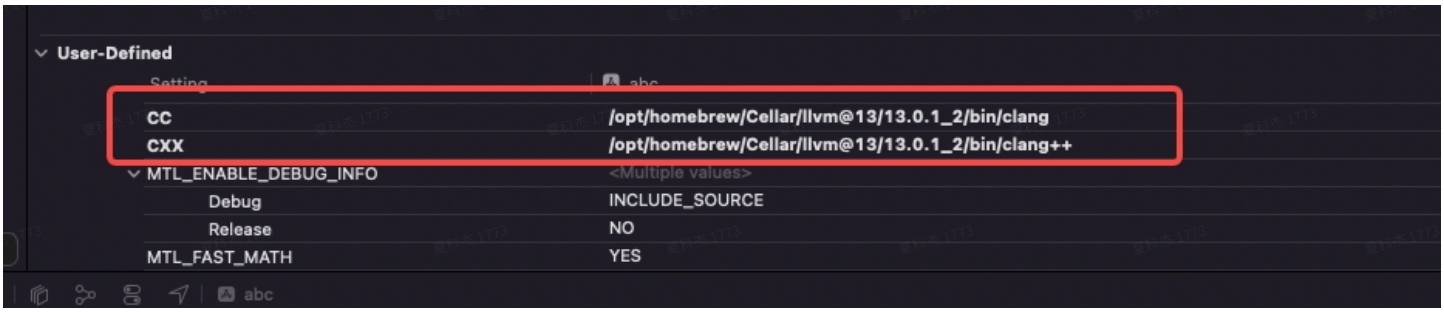


5.2、配置Other C Flags && Other C++ Flags

```
1 -Xclang -load -Xclang /Users/xiakejie/Desktop/abc/libCodeReview.dylib -Xclang -a
```

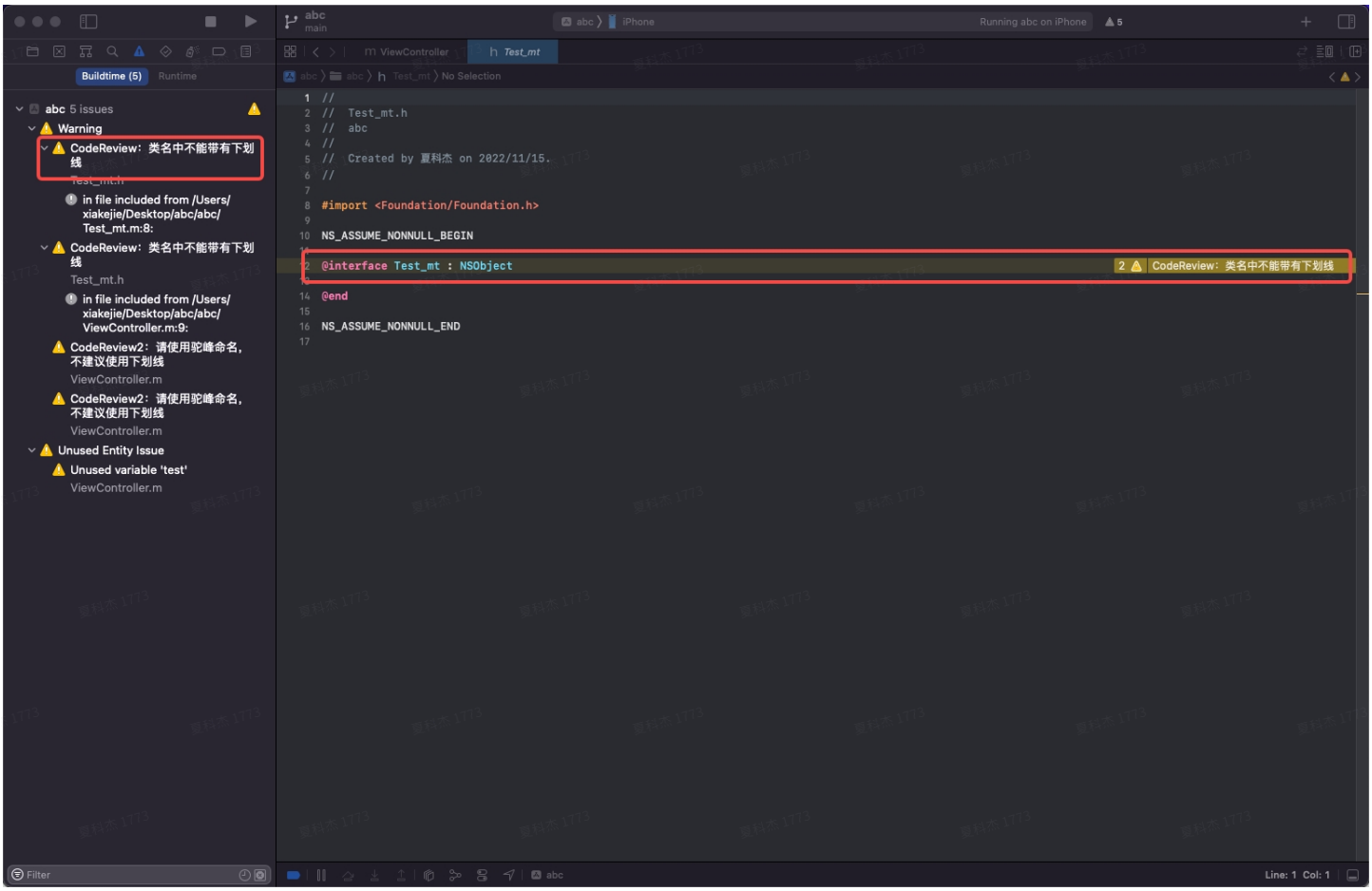


5.3、User-Defined 配置cc 和cxx 路径

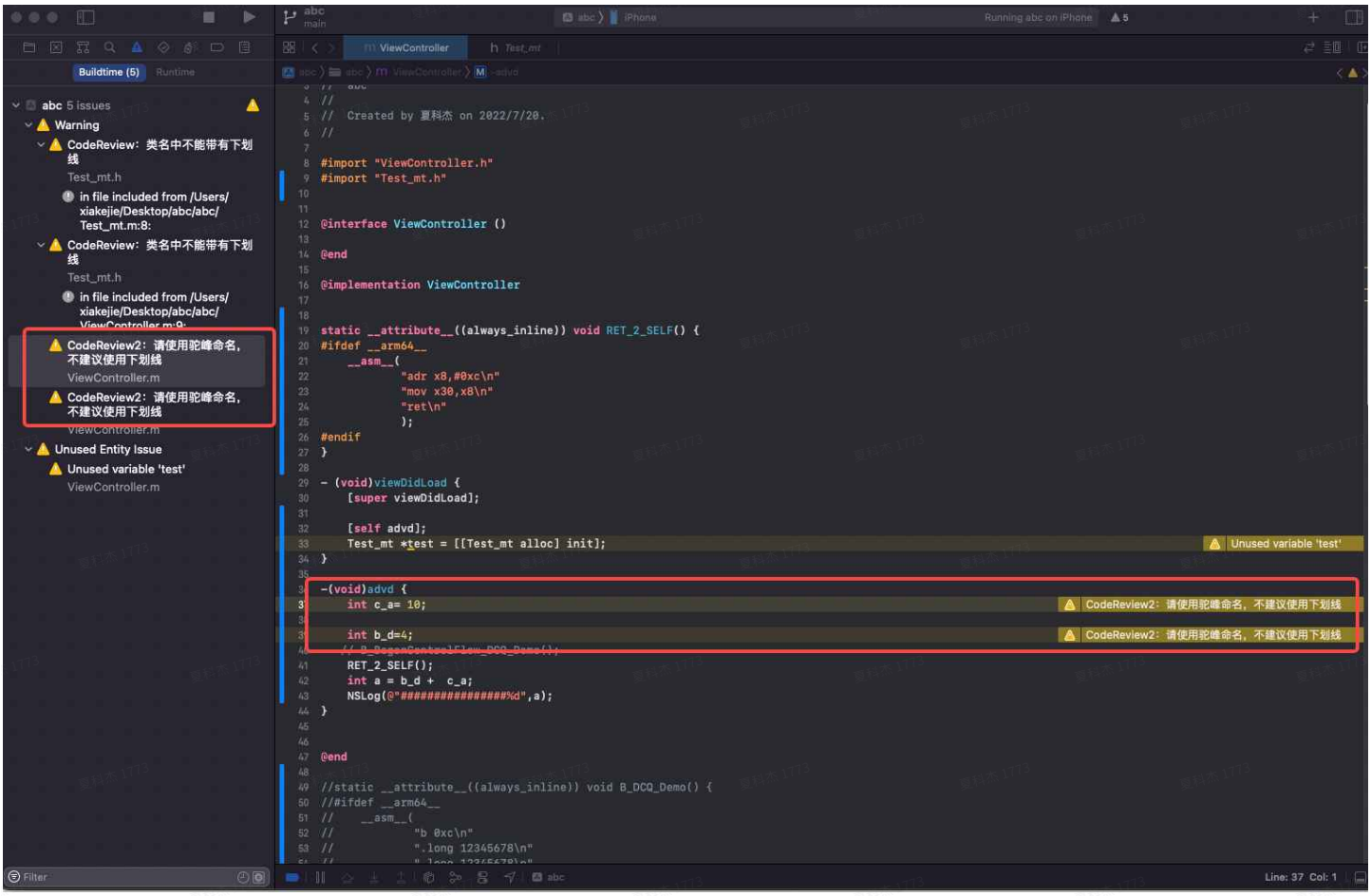


三、效果

1、检查类名(Interface)，不能带有下划线



2、检查变量(Interface)，不能带有下划线



四、后续

1、代码检查规范完善后，可以部署到jenkins 上做代码规范检查。

五、下一步

1、写代码混淆pass