llvm 上手第一个pass检查驼峰

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

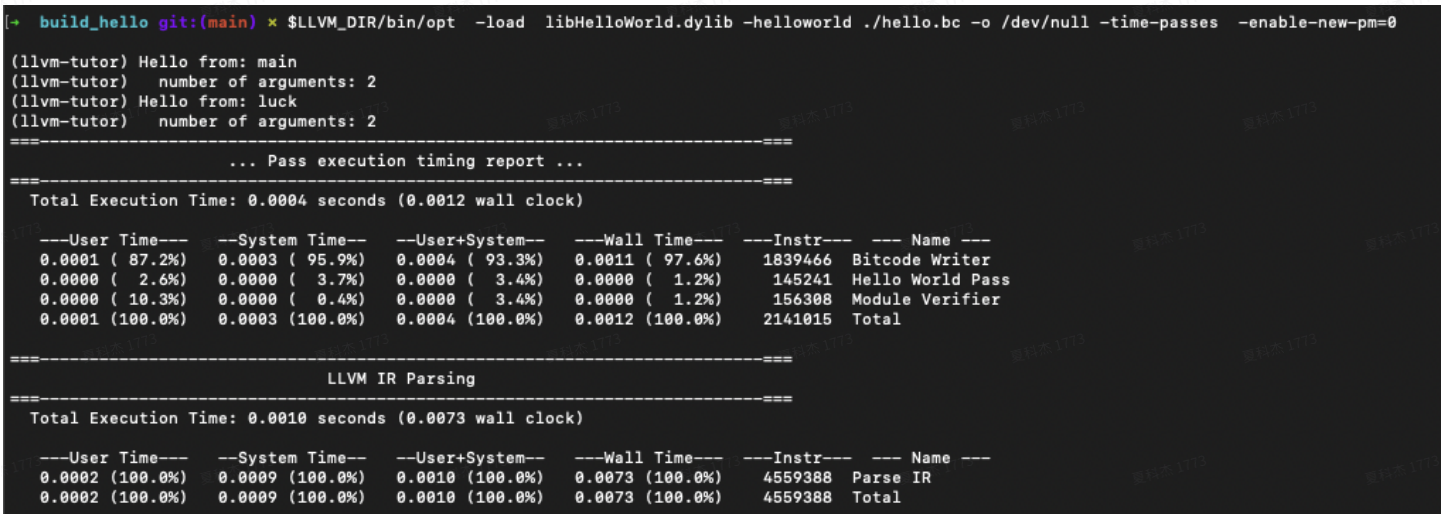
```
mkdir build
cd $build
cmake -DLT_LLVM_INSTALL_DIR=$LLVM_DIR  ../
make
```

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、写一个驼峰命名规范检查的pass

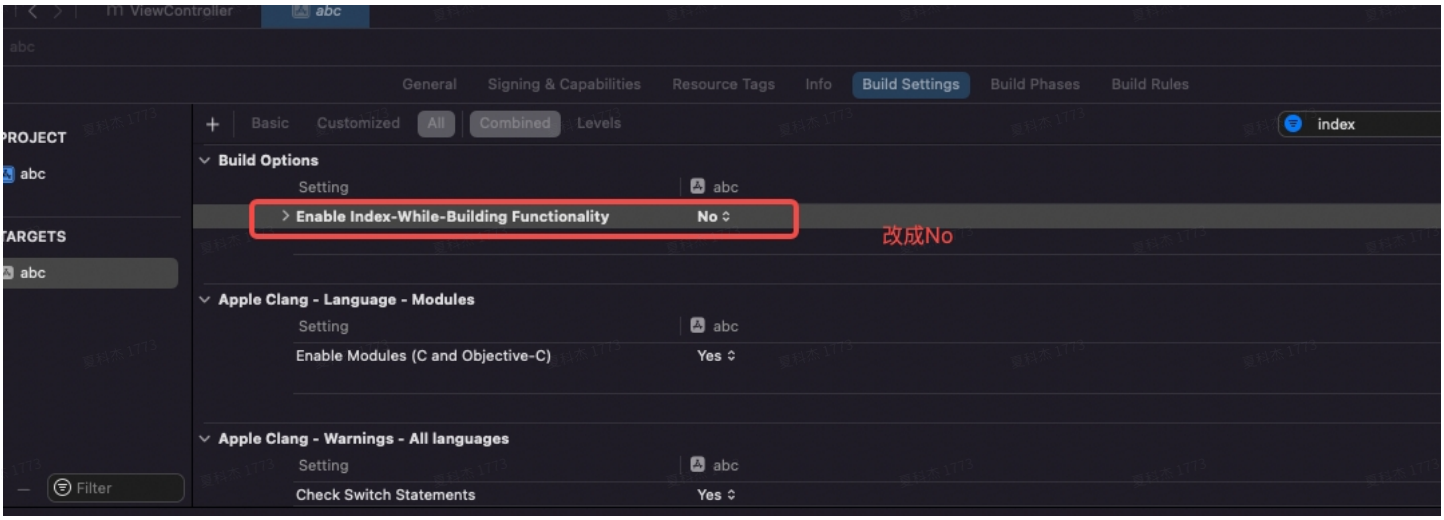
```
#include <iostream>
#include "clang/AST/AST.h"
#include "clang/AST/ASTConsumer.h"
#include "clang/ASTMatchers/ASTMatchers.h"
#include "clang/ASTMatchers/ASTMatchFinder.h"
#include "clang/Frontend/CompilerInstance.h"
#include "clang/Frontend/FrontendPluginRegistry.h"

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

namespace TestPlugin {
    class TestHandler : public MatchFinder::MatchCallback{
```

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

```
63          MatchFinder matcher;
64          TestHandler handler;
65
66      public:
67          TestASTConsumer(CompilerInstance &ci) :handler(ci) {
68              matcher.addMatcher(objcInterfaceDecl().bind("ObjCInterfaceDecl"), &h
69              matcher.addMatcher(varDecl().bind("VarDecl"), &handler);
70              matcher.addMatcher(objcMethodDecl().bind("ObjCMethodDecl"), &handler
71          }
72          void HandleTranslationUnit(ASTContext &Ctx) {
73              printf("TestPlugin1: All ASTs has parsed.");
74              DiagnosticsEngine &D = Ctx.getDiagnostics();
75              // 在编译log中可以看到
76              //D.Report(D.getCustomDiagID(DiagnosticsEngine::Warning, "TestPlugin
77             // D.Report(D.getCustomDiagID(DiagnosticsEngine::Error, "TestPlugin错
78              matcher.matchAST(Ctx);
79          }
80      };
81
82
83      // 定义触发插件的动作
84      class TestAction : public PluginASTAction{
85      public:
86          unique_ptr<ASTConsumer> CreateASTConsumer(CompilerInstance &CI,
87                                                    StringRef InFile){
88              return unique_ptr<TestASTConsumer> (new TestASTConsumer(CI));
89          }
90
91          bool ParseArgs(const CompilerInstance &CI,
92                         const std::vector<std::string> &arg){
93              return true;
94          }
95      };
96  }
97  // 告知clang,注册一个新的plugin
98  static FrontendPluginRegistry::Add<TestPlugin::TestAction>
99  X("TestPlugin", "Test a new Plugin");
100 // X 变量名，可随便写，也可以写自己有意思的名称
101 // TestPlugin  插件名称，很重要，这个是对外的名称
102 // Test a new Plugin  插件备注
```

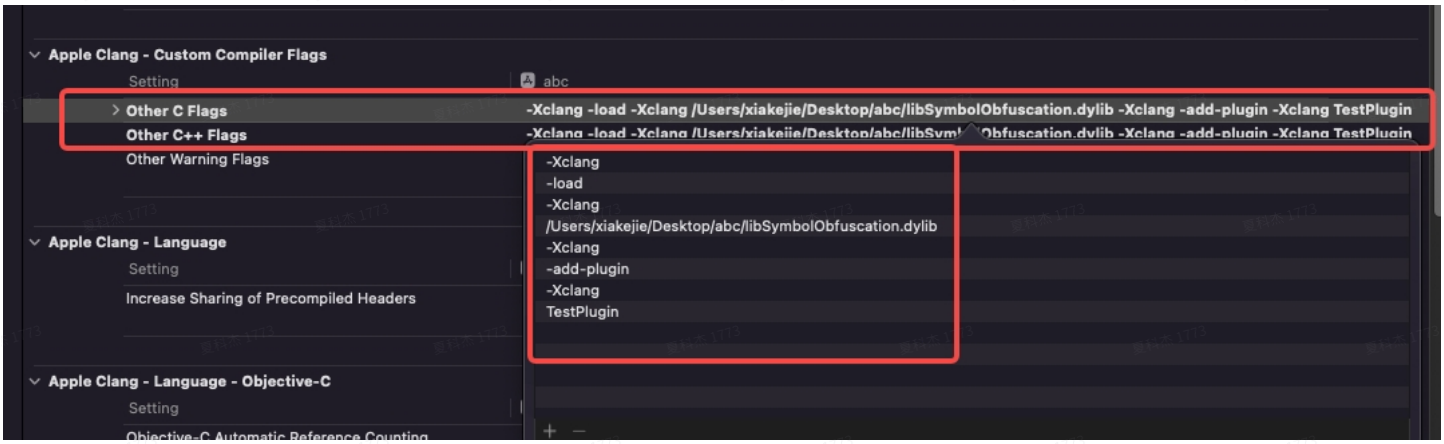
5、xcode 配置

5.1、配置Build Options

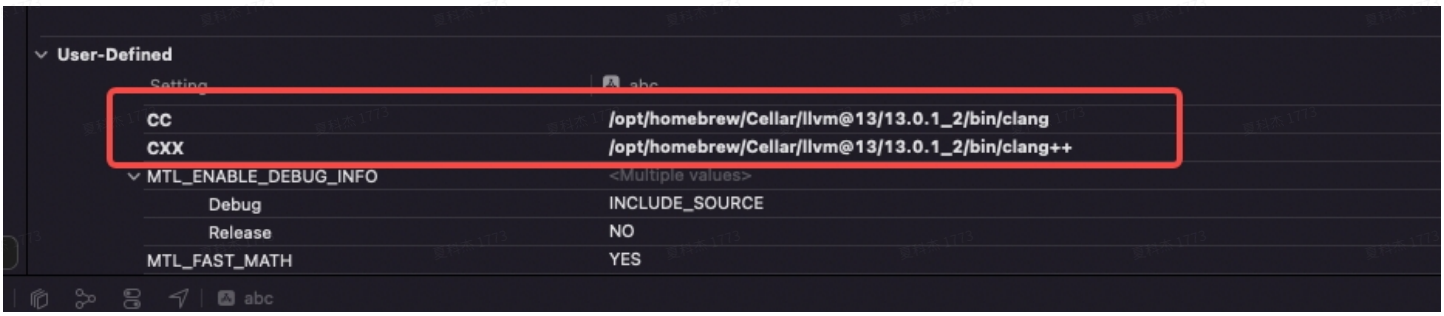


5.2、配置Other C Flags && Other C++ Flags

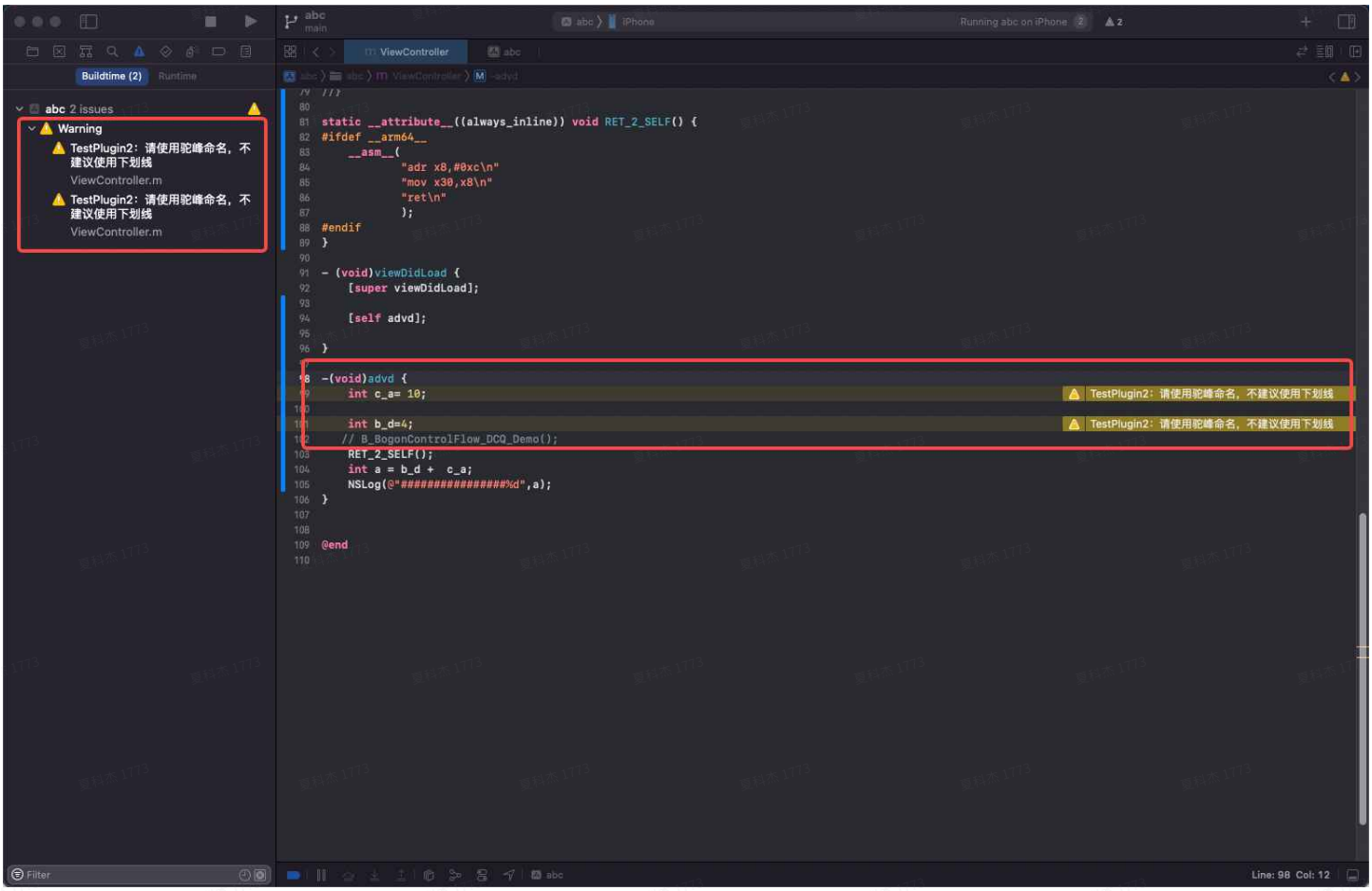
```
1 -Xclang -load -Xclang /Users/xiakejie/Desktop/abc/libSymbolObfuscation.dylib -Xc
```



5.3、User-Defined 配置cc 和cxx 路径



三、效果



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