# Kaleidoscope

## 代码解释(7/8)

万花筒语言 - LLVM 新手入门教程

https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/LangImpl08.html

PLCT - SSC

# 编译为目标代码

```
ready> def mul(x y) x*y;
Read function definition:define double @mul(double %x, double %y) {
entry:
  %y2 = alloca double, align 8
  %x1 = alloca double, align 8
  store double %x, double* %x1, align 8
  store double %y, double* %y2, align 8
  %x3 = load double, double* %x1, align 8
  %y4 = load double, double* %y2, align 8
  %multmp = fmul double %x3, %y4
  ret double %multmp
}

^D

Wrote output.o
```

test.cpp

```
#include <iostream>

extern "C" {
double mul(double, double);
}

int main() {
  double x, y;
  std::cout << "Please input two nums:";
  std::cin >> x >> y;
  std::cout << x << "*" << y << "=" << mul(x, y) << std::endl;
  return 0;
}
```

```
clang++ -c test.cpp -o test.o
clang++ test.o output.o -o test
./test
Please input two nums:12 13
12*13=156
```

# 去除之前的Jit和pass相关代码

```
#include "../include/KaleidoscopeJIT.h“
#include "llvm/Transforms/InstCombine/InstCombine.h"
#include "llvm/Transforms/Scalar.h"
#include "llvm/Transforms/Scalar/GVN.h"
#include "llvm/Transforms/Utils.h"
#include <cstdint>

using namespace llvm::orc;
```

```
#include "llvm/ADT/Optional.h“
#include "llvm/Support/FileSystem.h"
#include "llvm/Support/Host.h"
#include "llvm/Support/raw_ostream.h"
#include "llvm/Support/TargetRegistry.h"
#include <system_error>

using namespace llvm::sys;
```

```
static std::unique_ptr<legacy::FunctionPassManager> TheFPM;
static std::unique_ptr<KaleidoscopeJIT> TheJIT;
```

```
Function *FunctionAST::codegen() {
  if (Value *RetVal = Body->codegen()) {
    // Run the optimizer on the function.
    TheFPM->run(*TheFunction);
  }
}
```

```
static void InitializeModuleAndPassManager() {
  TheModule->setDataLayout(TheJIT->getTargetMachine().createDataLayout());

  TheFPM = std::make_unique<legacy::FunctionPassManager>(TheModule.get());

  TheFPM->add(createPromoteMemoryToRegisterPass());
  TheFPM->add(createInstructionCombiningPass());
  TheFPM->add(createReassociatePass());
  TheFPM->add(createGVNPass());
  TheFPM->add(createCFGSimplificationPass());

  TheFPM->doInitialization();
}
```

```
static void HandleDefinition() {
  if (auto FnAST = ParseDefinition()) {
    if (auto *FnIR = FnAST->codegen()) {
      TheJIT->addModule(std::move(TheModule));
      InitializeModuleAndPassManager();
    }
}
```

```
static void HandleTopLevelExpression() {
  if (auto FnAST = ParseTopLevelExpr()) {
    if (FnAST->codegen()) {
      auto H = TheJIT->addModule(std::move(TheModule));
      InitializeModuleAndPassManager();

      auto ExprSymbol = TheJIT->findSymbol("__anon_expr");
      assert(ExprSymbol && "Function not found");

      double (*FP)() = (double (*)())(intptr_t)cantFail(ExprSymbol.getAddress());
      fprintf(stderr, "Evaluated to %f\n", FP());
      TheJIT->removeModule(H);
    }
  }}
```

```
static void MainLoop() {
  while (true) {
    fprintf(stderr, "ready> ");
  }
}
```

```
int main() {
  InitializeNativeTarget();
  InitializeNativeTargetAsmPrinter();
  InitializeNativeTargetAsmParser();

  BinopPrecedence['='] = 2;
  BinopPrecedence['<'] = 10;
  BinopPrecedence['+'] = 20;
  BinopPrecedence['-'] = 20;
  BinopPrecedence['*'] = 40;

  fprintf(stderr, "ready> ");
  getNextToken();

  TheJIT = std::make_unique<KaleidoscopeJIT>();

  InitializeModuleAndPassManager();

  MainLoop();




  return 0;
}
```

# main函数

```
int main() {




  BinopPrecedence['<'] = 10;
  BinopPrecedence['+'] = 20;
  BinopPrecedence['-'] = 20;
  BinopPrecedence['*'] = 40; // highest.

  fprintf(stderr, "ready> ");
  getNextToken();



  InitializeModuleAndPassManager();

  MainLoop();

  // 初始化所有的体系结构
  InitializeAllTargetInfos();
  InitializeAllTargets();
  InitializeAllTargetMCs();
  InitializeAllAsmParsers();
  InitializeAllAsmPrinters();

  // 获取本机的体系结构，并设置
  auto TargetTriple = sys::getDefaultTargetTriple();
  TheModule->setTargetTriple(TargetTriple);

  std::string Error;
  auto Target = TargetRegistry::lookupTarget(TargetTriple, Error);

  // 判断是否获得表示本机体系结构的Target
  if (!Target) {
    errs() << Error;
```

# main函数

```
// 获取本机的体系结构，并设置
auto TargetTriple = sys::getDefaultTargetTriple();
TheModule->setTargetTriple(TargetTriple);

std::string Error;
auto Target = TargetRegistry::lookupTarget(TargetTriple, Error);

// 判断是否获得表示本机体系结构的Target
if (!Target) {
  errs() << Error;
  return 1;
}

// 构建通用的CPU版本
auto CPU = "generic";
auto Features = "";

TargetOptions opt;
auto RM = Optional<Reloc::Model>();
auto TheTargetMachine =
    Target->createTargetMachine(TargetTriple, CPU, Features, opt, RM);

// 配置模块，设置数据布局
TheModule->setDataLayout(TheTargetMachine->createDataLayout());

// 指定目标文件
auto Filename = "output.o";
std::error_code EC;
raw_fd_ostream dest(Filename, EC, sys::fs::OF_None);

if (EC) {
  errs() << "Could not open file: " << EC.message();
  return 1;
}

// 生成代码，并运行pass
legacy::PassManager pass;
```

# main函数

```
  TargetOptions opt;
  auto RM = Optional<Reloc::Model>();
  auto TheTargetMachine =
      Target->createTargetMachine(TargetTriple, CPU, Features, opt, RM);

  // 配置模块，设置数据布局
  TheModule->setDataLayout(TheTargetMachine->createDataLayout());

  // 指定目标文件
  auto Filename = "output.o";
  std::error_code EC;
  raw_fd_ostream dest(Filename, EC, sys::fs::OF_None);

  if (EC) {
    errs() << "Could not open file: " << EC.message();
    return 1;
  }

  // 生成代码，并运行pass
  legacy::PassManager pass;
  auto FileType = CGFT_ObjectFile;

  if (TheTargetMachine->addPassesToEmitFile(pass, dest, nullptr, FileType)) {
    errs() << "TheTargetMachine can't emit a file of this type";
    return 1;
  }

  pass.run(*TheModule);
  dest.flush();

  // 结束
  outs() << "Wrote " << Filename << "\n";

  return 0;
}
```