**第四章-Optimizer&Jit调用设计文档**

一．Optimizer函数优化

static void InitializeModuleAndPassManager() {

// Open a new module.

TheModule = llvm::make\_unique<Module>("my cool jit", TheContext);

// Create a new pass manager attached to it.

TheFPM = llvm::make\_unique<legacy::FunctionPassManager>(TheModule.get());

// Do simple "peephole" optimizations and bit-twiddling optzns.

TheFPM->add(createInstructionCombiningPass());

// Reassociate expressions.

TheFPM->add(createReassociatePass());

// Eliminate Common SubExpressions.

TheFPM->add(createGVNPass());

// Simplify the control flow graph (deleting unreachable blocks, etc).

TheFPM->add(createCFGSimplificationPass());

TheFPM->doInitialization();

}

二.Jit调用

1．在主函数中调用一些InitializeNativeTarget等函数并添加一个全局变量TheJITbing初始化

InitializeNativeTarget();

InitializeNativeTargetAsmPrinter();

InitializeNativeTargetAsmParser();

..

TheJIT = llvm::make\_unique<KaleidoscopeJIT>();

2.在优化函数中设置布局

TheModule->setDataLayout(TheJIT->getTargetMachine().createDataLayout());

// Create a new pass manager attached to it.

TheFPM = llvm::make\_unique<legacy::FunctionPassManager>(TheModule.get());

3.重新解析顶层表达式

if (auto FnAST = ParseTopLevelExpr()) {

if (auto \*FnIR = FnAST->codegen()) {

fprintf(stderr, "Read function call:");

FnIR->print(errs());

fprintf(stderr, "\n");

// JIT the module containing the anonymous expression, keeping a handle so

// we can free it later.

auto H = TheJIT->addModule(std::move(TheModule));

InitializeModuleAndPassManager();

// Search the JIT for the \_\_anon\_expr symbol.

auto ExprSymbol = TheJIT->findSymbol("\_\_anon\_expr");

assert(ExprSymbol && "Function not found");

// Get the symbol's address and cast it to the right type (takes no

// arguments, returns a double) so we can call it as a native function.

/\*int(\*FP)() = (int(\*)())cantFail(ExprSymbol.getAddress());

fprintf(stderr, "Evaluated to %d\n", FP());\*/

// Delete the anonymous expression module from the JIT.

TheJIT->removeModule(H);

}

}