# 通过手写LLVM IR过程入门LLVM

程序语言与编译技术实验室 李威威

2019/12/18

CONTENT

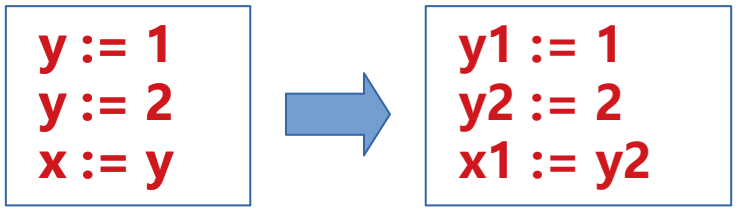
# 目录

# LLVM IR 总体描述

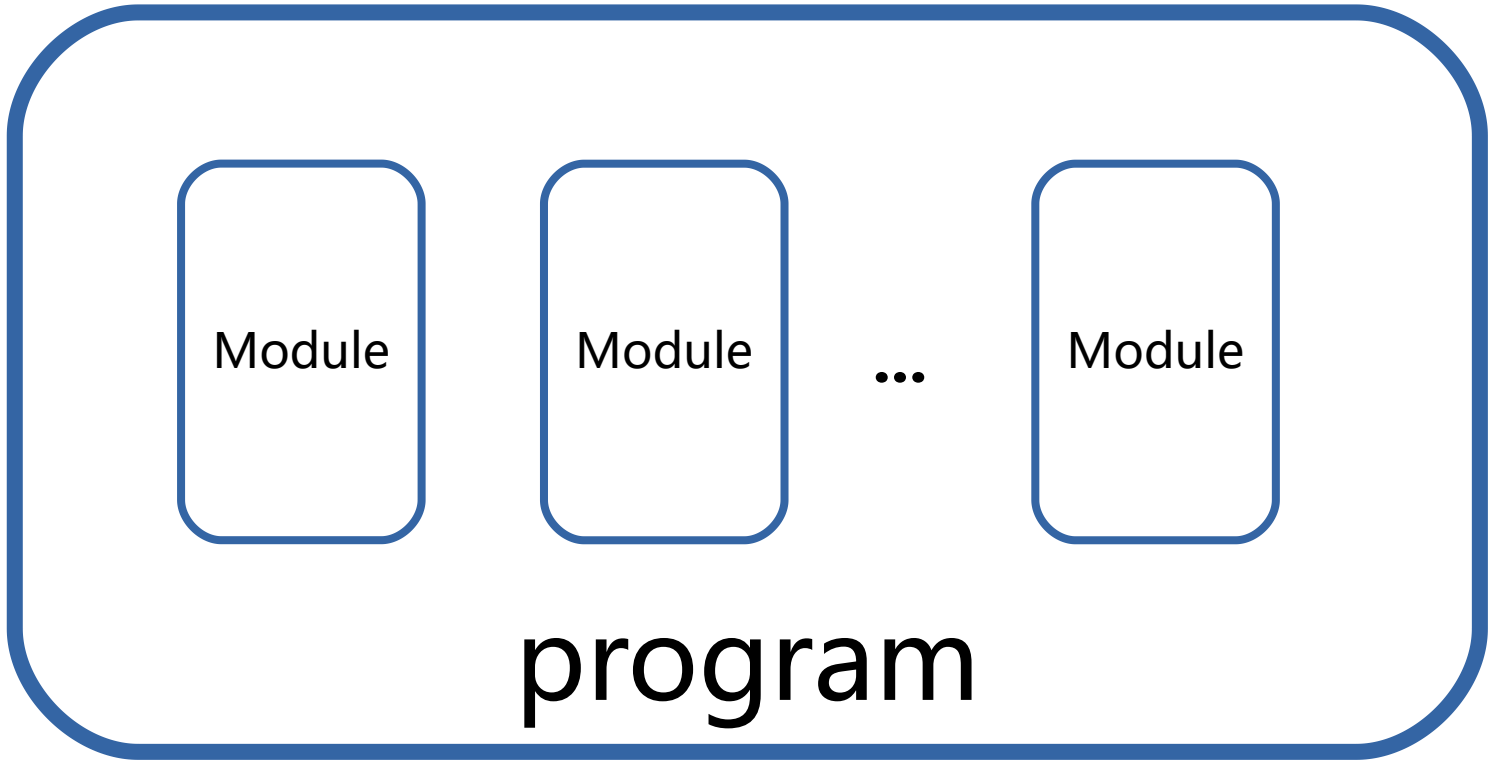
- LLVM IR（Intermediate Representation）

# LLVM IR 总体描述

- 特征
  - SSA形式 + 无限寄存器
    - 每个变量都只被赋值一次
    - 容易确定操作间的依赖关系，便于优化分析
  - 采用3地址的方式
    - ADD EAX, EBX
    - %2 = add i32 %0, %1
  - 强类型系统
    - 每个Value都具备自身的类型



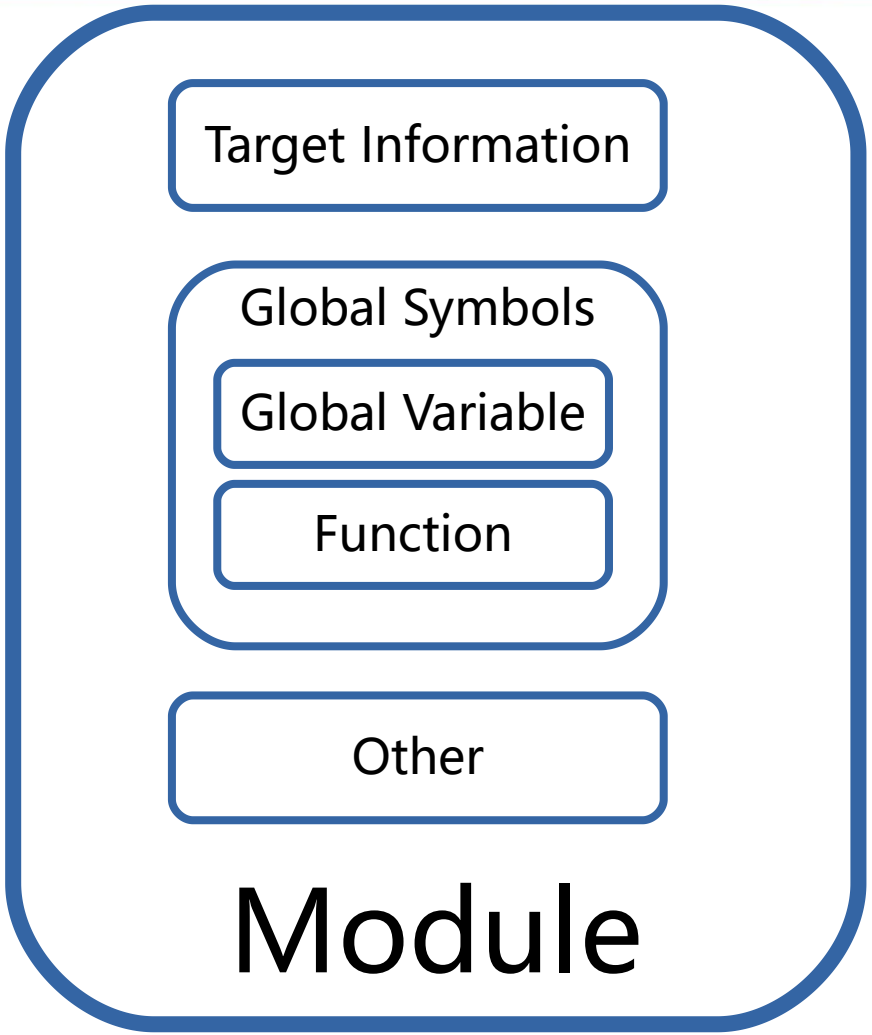
# LLVM IR 程序构成



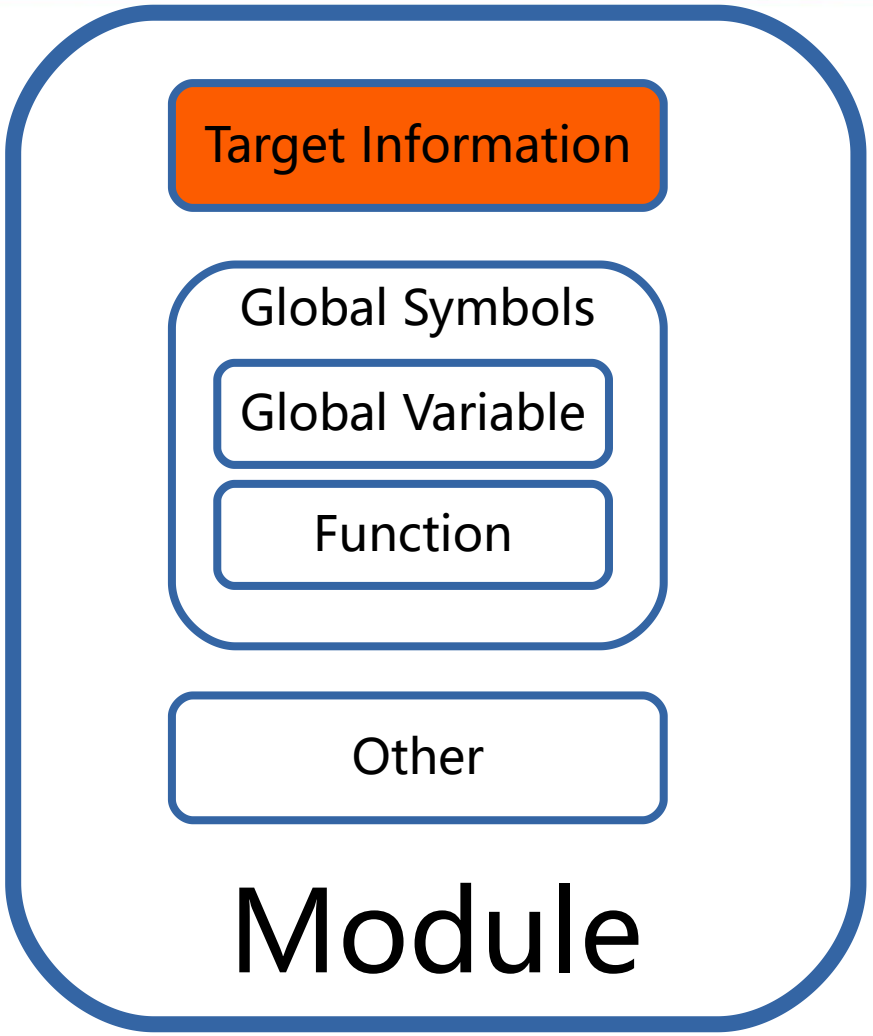

程序可由1或多个Module构成

LLVM linker会将各个Module合并起来

# LLVM IR 程序构成——Module



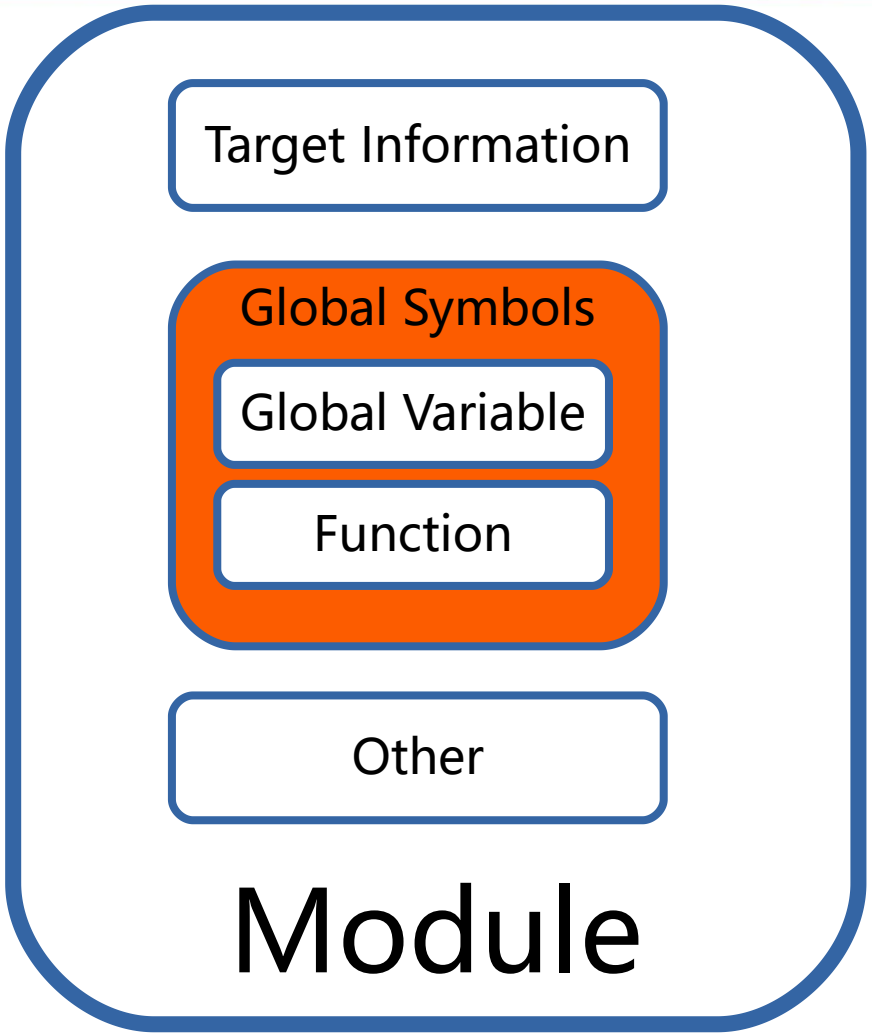
# LLVM IR 程序构成——目标信息



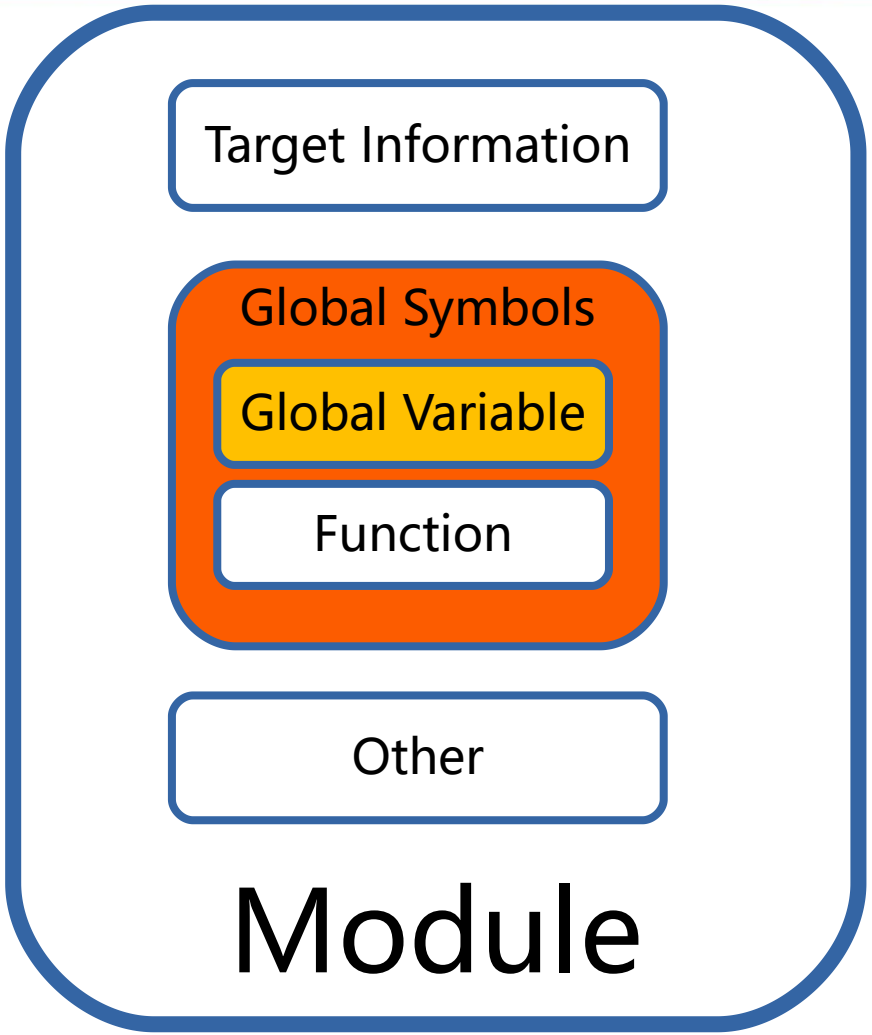

目标内存排布信息：
target datalayout = "e-m:e-i64:64-f80:128-n8:16:32:64-S128"

目标宿主信息：
target triple = "x86_64-pc-linux-gnu"

# LLVM IR 程序构成——全局符号表



# LLVM IR 程序构成——全局变量

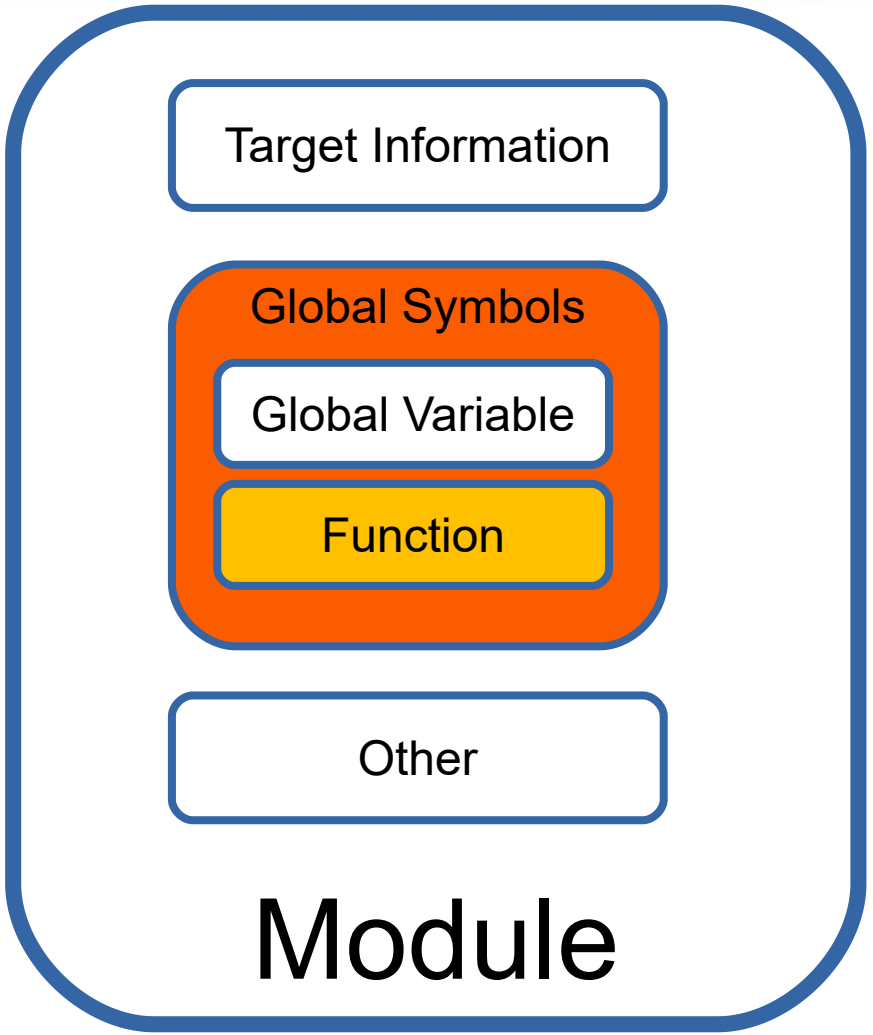



全局变量：

@a = global i32 1

@0 = private unnamed_addr constant [12 x i8]
c"result: %d\0A\00"

# LLVM IR 程序构成——函数声明与定义



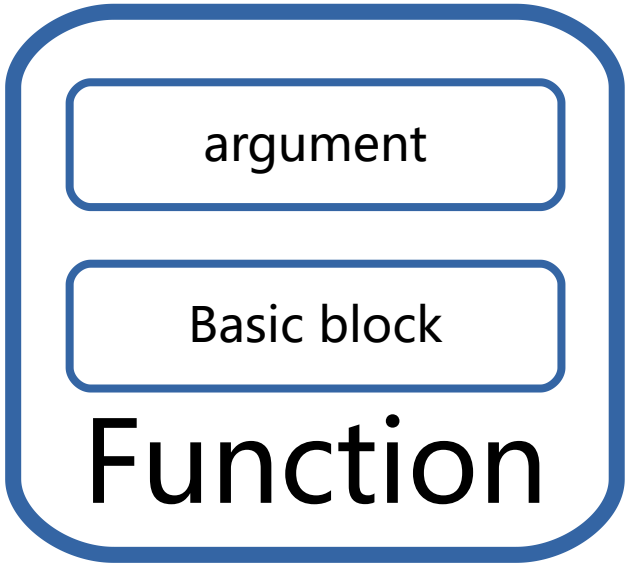

函数声明

```
declare i32 @printf(i8*, ...)
```

函数定义：

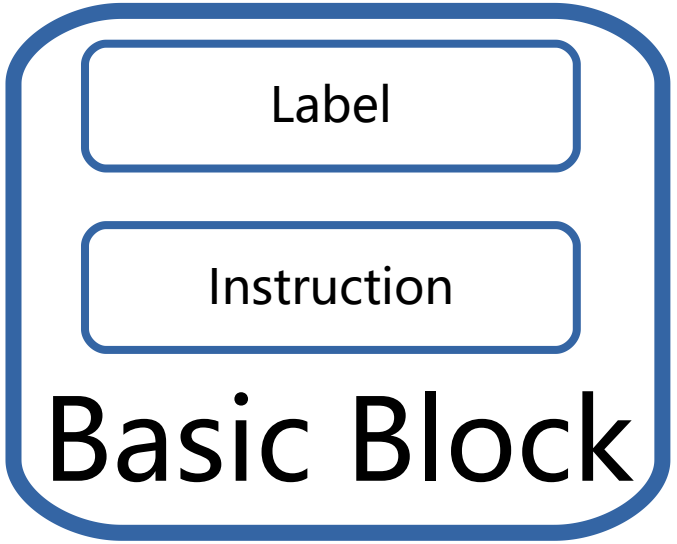
```
define i32 @main() {
    ...
}
```

# LLVM IR 程序构成——Function



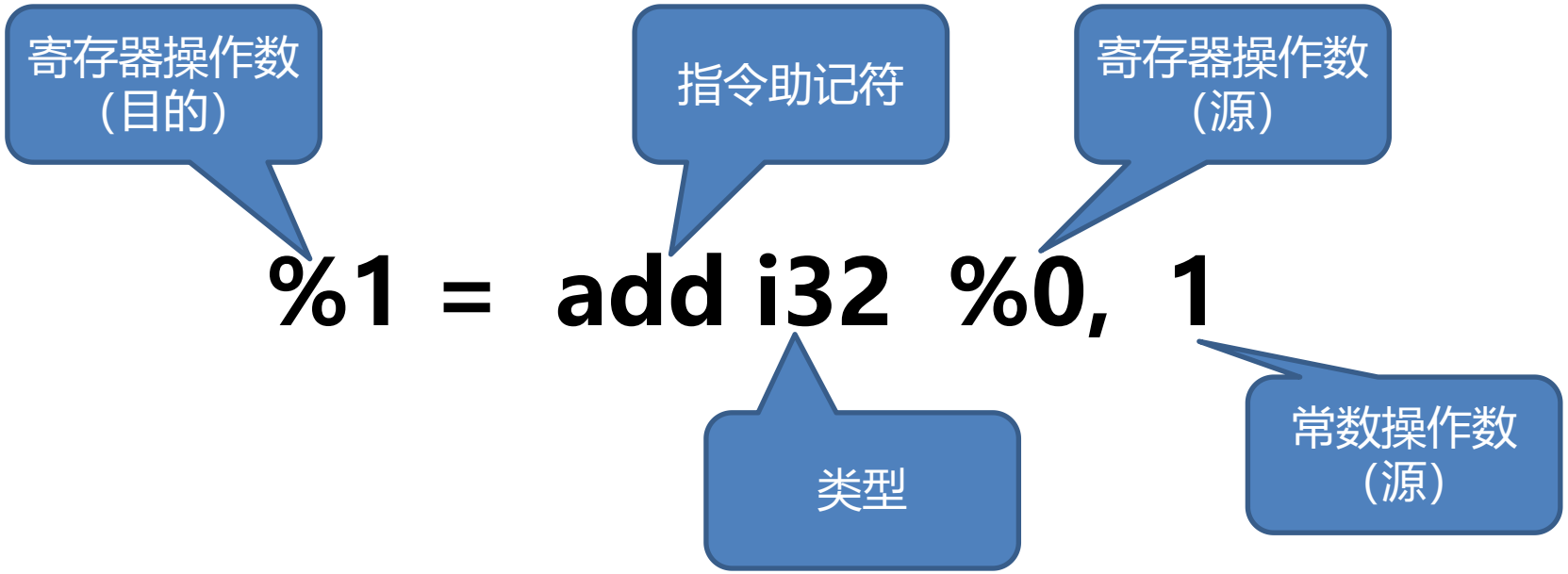

```
define i32 @test(i32, i32) {
bb0:
  ...
bb1:
  ...
}
```

# LLVM IR 程序构成——BasicBlock

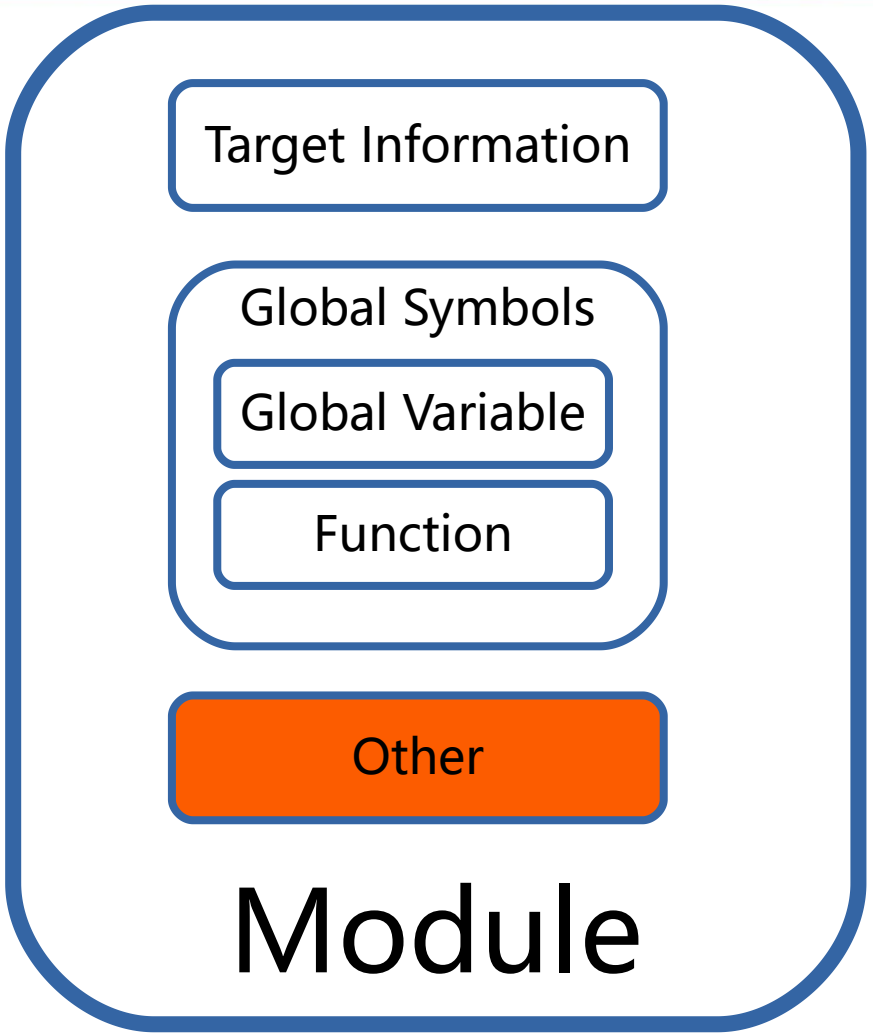



```
bb0 :
  %0 = load i32, i32* @a
  %1 = add i32 %0, 1
  %2 = call i32 (i8*, ...) @printf(i8*
getelementptr inbounds ([12 x i8], [12 x i8]*
@0, i32 0, i32 0), i32 %1)
  ret i32 0
```

# LLVM IR 程序构成——Instruction



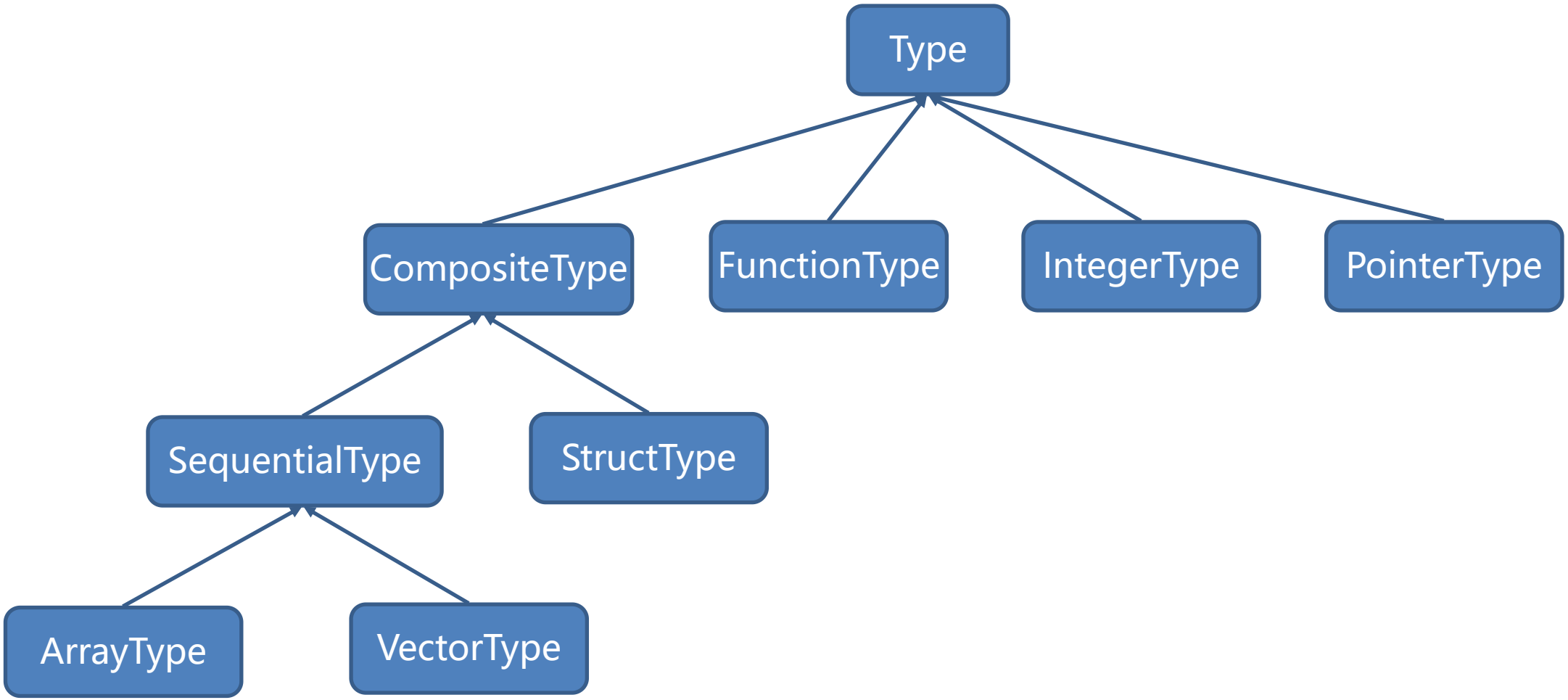
# LLVM IR 程序构成——其它信息



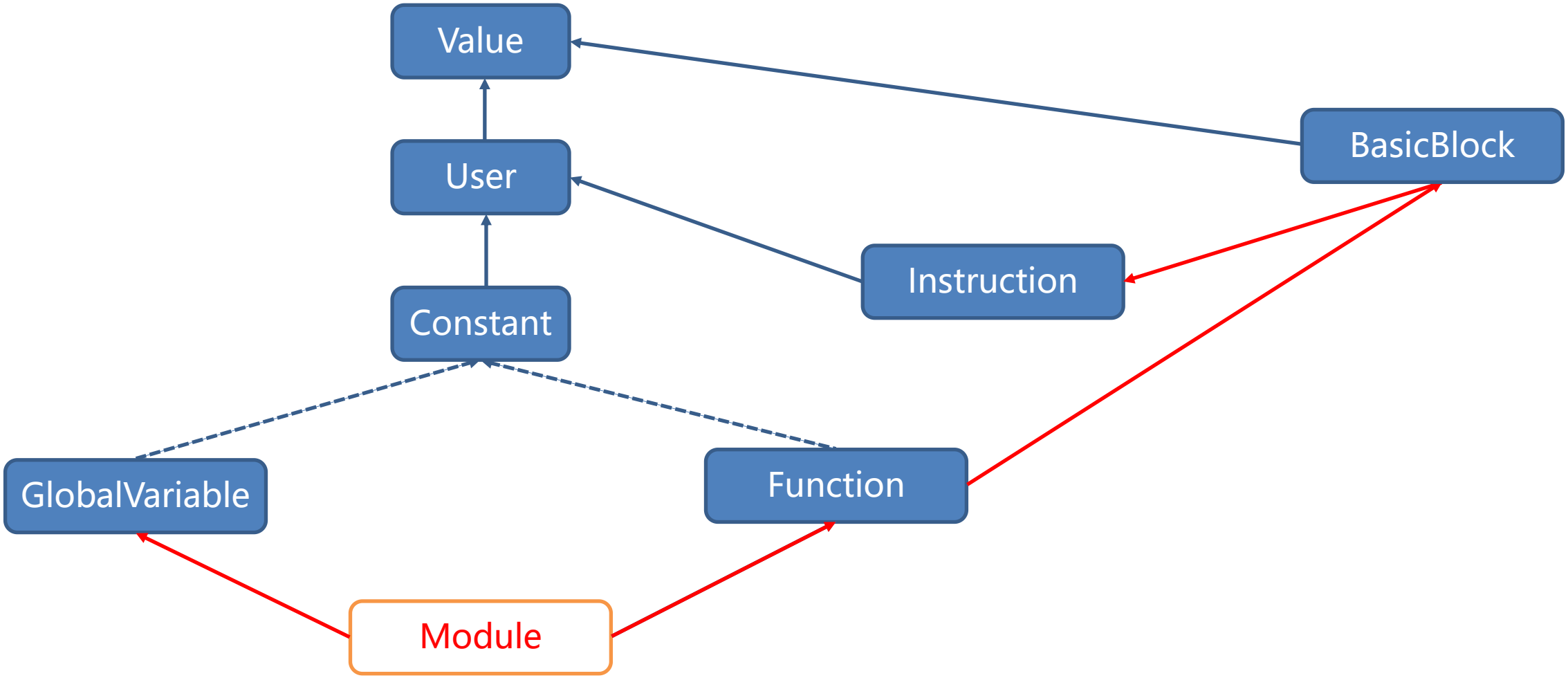

元数据信息：

!1 = !{!"clang version 6.0.0-1ubuntu2 (tags/RELEASE_600/final)"}

# LLVM IR代码生成——类型



# LLVM IR 代码生成——数值



# LLVM IR代码生成——Value类

- Value Class
  - 该类型代表一个可能用于指令操作数的带类型数据
  - Value类会维护一个该数据使用者的列表，该表可以描述程序中的def-use信息
    - 特定的Value可能在程序可能在程序中用到多次
    - 提供了一些方法用于访问该链表，如use迭代器，或者进行use替换

# LLVM IR代码生成——User类

- User Class
  - 该类型提供一个操作数表，用于描述用户所需要索引的所有Values，该表中每个操作数都直接指向一个Value,提供了 use-def信息
    - 提供了操作数迭代器来访问操作数

# LLVM IR代码生成——IRBuilder类

- IRBuilder Class
  - 该类提供了独立的接口创建各种IR指令，并将它们插入基本块中
    - 创建指令：Create<XXX>，xxx为IR指令名
      - BranchInst * CreateBr (BasicBlock *Dest)
      - Value * CreateAnd (Value *LHS, Value *RHS, const Twine &Name="")
    - 继承自IRBuilderBase的接口
      - 设置插入点：void SetInsertPoint (BasicBlock *TheBB)
      - 获取数据：ConstantInt * getInt8 /16/32/64(uint8/16/32/64_t C)
      - 获取类型接口：IntegerType * getInt1/8/16/32/64/128Ty ()

# LLVM IR代码生成举例

hello.c:

```
void main() {
    puts("hello world!\n");
    return;
}
```

# LLVM IR代码生成举例

hello.c:

```
void main() {
    puts("hello world!\n");
    return;
}
```

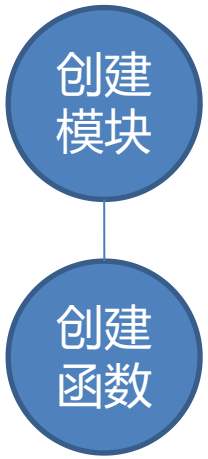
创建模块

创建模块： hello.c

# LLVM IR代码生成举例

hello.c:

```
void main() {
    puts("hello world!\n");
    return;
}
```




创建模块： hello.c

创建函数： void main()

# LLVM IR代码生成举例

hello.c:

```
void main() {
    puts("hello world!\n");
    return;
}
```


创建模块： hello.c

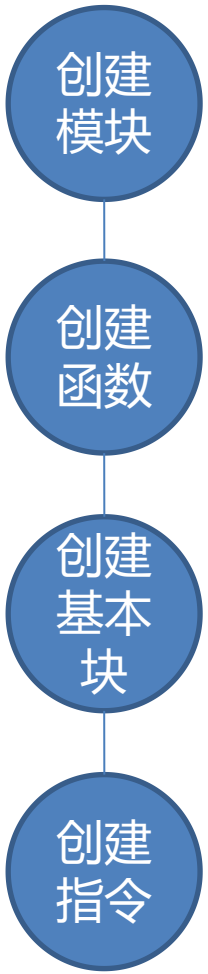
创建函数： void main()

创建基本块：entrypoint:

# LLVM IR代码生成举例

hello.c:

```
void main() {
    puts("hello world!\n");
    return;
}
```



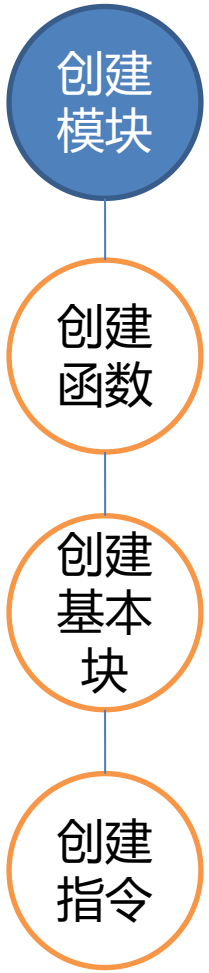

创建模块： hello.c

创建函数： void main()

创建基本块：entrypoint:

创建指令：
puts("hello world!\n");
创建函数声明：puts()
创建全局常量："hello world!\n"
return;

# LLVM IR代码生成举例




```
llvm::LLVMContext  context;
llvm::Module *module = new llvm::Module("hello.c", context);

llvm::IRBuilder<> builder(context);
```

对应LLVM IR代码：

```
; ModuleID = 'hello.c'
source_filename = "hello.c"
```

# LLVM IR代码生成举例

创建模块

创建函数

创建基本块

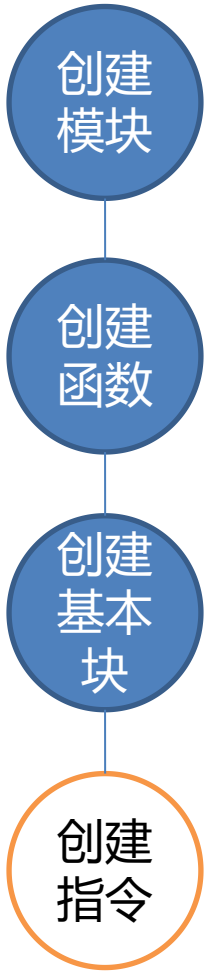
创建指令

```
llvm::FunctionType *funcType =
    llvm::FunctionType::get(builder.getVoidTy(), false);

llvm::Function *mainFunc =
    llvm::Function::Create(funcType,
        llvm::Function::ExternalLinkage, "main", module);
```

对应LLVM IR代码：

```
define void @main()
{

}
```

# LLVM IR代码生成举例

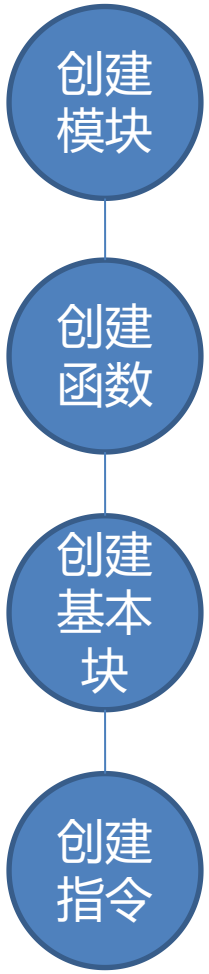



```
llvm::BasicBlock *entry =
    llvm::BasicBlock::Create(context, "entrypoint", mainFunc);

builder.SetInsertPoint(entry);
```

对应LLVM IR代码：

```
define void @main() {
entrypoint:

}
```

# LLVM IR代码生成举例



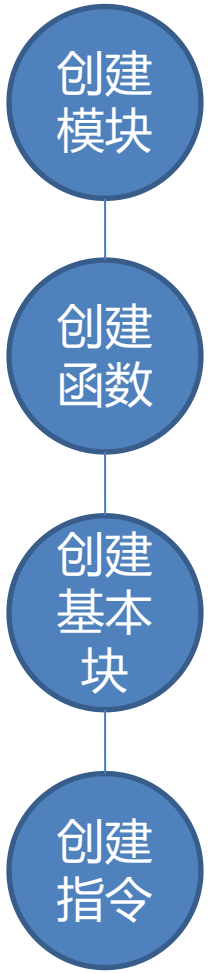

添加全局变量：

```
llvm::Value *helloWorld =
    builder.CreateGlobalStringPtr("hello world!\n");
```

对应LLVM IR代码：

```
@0 = private unnamed_addr constant [14 x i8]
                c"hello world!\0A\00"
```

# LLVM IR代码生成举例



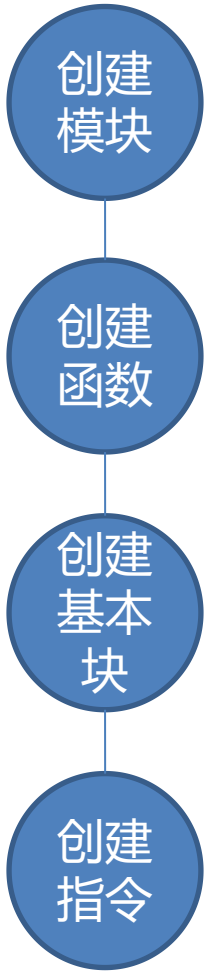

添加函数声明：

```
std::vector<llvm::Type *> putsArgs;
putsArgs.push_back(builder.getInt8Ty()->getPointerTo());
llvm::ArrayRef<llvm::Type*>  argsRef(putsArgs);
llvm::FunctionType *putsType =
    llvm::FunctionType::get(builder.getInt32Ty(), argsRef, false);

llvm::Constant *putsFunc =
    module->getOrInsertFunction("puts", putsType);
```

对应LLVM IR代码：

```
declare i32 @puts(i8*)
```

# LLVM IR代码生成举例




添加指令：

builder.CreateCall(putsFunc, helloWorld);

builder.CreateRetVoid();

对应LLVM IR代码：

```
define void @main() {
entrypoint:
  %0 = call i32 @puts(i8* getelementptr inbounds ([14 x i8], [14 x i8]* @0, i32  0, i32 0))
  ret void
}
```

# LLVM IR代码生成举例

```
llvm::LLVMContext  context;
llvm::Module *module =
    new llvm::Module("hello.c", context);
llvm::IRBuilder<> builder(context);

llvm::FunctionType *funcType =
    llvm::FunctionType::get(builder.getVoidTy(),
        false);
llvm::Function *mainFunc =
    llvm::Function::Create(funcType, llvm::
        Function::ExternalLinkage, "main",
        module);
llvm::BasicBlock *entry =
    llvm::BasicBlock::Create(context,
        "entrypoint", mainFunc);
```

```
builder.SetInsertPoint(entry);
llvm::Value *helloWorld =
    builder.CreateGlobalStringPtr("hello world!\n");
std::vector<llvm::Type *> putsArgs;
    putsArgs.push_back(builder.getInt8Ty()->
        getPointerTo());
llvm::ArrayRef<llvm::Type*>  argsRef(putsArgs);
llvm::FunctionType *putsType =
    llvm::FunctionType::get(builder.getInt32Ty(),
        argsRef, false);
llvm::Constant *putsFunc =
    module->getOrInsertFunction("puts",
        putsType);
builder.CreateCall(putsFunc, helloWorld);
builder.CreateRetVoid();
```

# LLVM IR代码生成举例

编译：clang++ `llvm-config --cxxflags --ldflags  --libs` toy.cpp -o toy

```
; ModuleID = 'hello.c'
source_filename = "hello.c"
@0 = private unnamed_addr constant [14 x i8] c"hello world!\0A\00"
define void @main() {
entrypoint:
  %0 = call i32 @puts(i8* getelementptr inbounds ([14 x i8], [14 x i8]* @0, i32 0, i32 0))
  ret void
}
declare i32 @puts(i8*)
```

注意：LLVM每个版本都在变动内部API，使用时候遇到错误先查版本对不对

# 谢谢

欢迎交流合作

2019/12/18

# 附录：示例程序

```
#include "llvm/ADT/ArrayRef.h"
#include "llvm/IR/LLVMContext.h"
#include "llvm/IR/Module.h"
#include "llvm/IR/Function.h"
#include "llvm/IR/BasicBlock.h"
#include "llvm/IR/IRBuilder.h"
#include "llvm/IR/Constants.h"
#include "llvm/Support/Debug.h"
#include <vector>
#include <string>
int main()
{
  llvm::LLVMContext context;
  llvm::Module *module = new llvm::Module("hello.c", context);
   llvm::IRBuilder<> builder(context);
  llvm::FunctionType *funcType = llvm::FunctionType::get(builder.getVoidTy(), false);
  llvm::Function *mainFunc =
    llvm::Function::Create(funcType, llvm::Function::ExternalLinkage, "main", module);
  llvm::BasicBlock *entry = llvm::BasicBlock::Create(context, "entrypoint", mainFunc);
  builder.SetInsertPoint(entry);
  llvm::Value *helloWorld = builder.CreateGlobalStringPtr("hello world!\n");
  std::vector<llvm::Type *> putsArgs;
  putsArgs.push_back(builder.getInt8Ty()->getPointerTo());
  llvm::ArrayRef<llvm::Type*>  argsRef(putsArgs);
  llvm::FunctionType *putsType =
    llvm::FunctionType::get(builder.getInt32Ty(), argsRef, false);
  llvm::Constant* putsFunc = module->getOrInsertFunction("puts", putsType);
  builder.CreateCall(putsFunc, helloWorld);
  builder.CreateRetVoid();
  module->print(llvm::dbgs(), nullptr);
}
```

编译：

clang++ `llvm-config --cxxflags --ldflags  --libs` toy.cpp -o toy

生成IR文件：

./toy 2>toy.ll

执行IR文件：

lli toy.ll

# 附录：示例程序(LLVM C接口)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include "llvm-c/Core.h"
int main() {
  LLVMTypeRef int_t = LLVMInt32Type();
  LLVMTypeRef void_t = LLVMVoidType();
  LLVMTypeRef char_t = LLVMInt8Type();
  LLVMTypeRef char_p = LLVMPointerType(char_t, 0);

  LLVMModuleRef module = LLVMModuleCreateWithName("hello.c");

  LLVMTypeRef puts_t = LLVMFunctionType(int_t, &char_p, 1, false);
  LLVMValueRef puts_f = LLVMAddFunction(module, "puts", puts_t);

  LLVMTypeRef func_t = LLVMFunctionType(void_t, NULL, 0, false);
  LLVMValueRef func = LLVMAddFunction(module, "main", func_t);
  LLVMBasicBlockRef bb = LLVMAppendBasicBlock(func, "entrypoint");

  LLVMBuilderRef builder = LLVMCreateBuilder();
  LLVMPositionBuilderAtEnd(builder, bb);
  LLVMValueRef args[1];
  args[0] = LLVMBuildGlobalStringPtr(builder, "hello world!\n", "");
  LLVMBuildCall(builder, puts_f, args, 1, "");
  LLVMBuildRetVoid(builder);

  LLVMDumpModule(module);
  LLVMDisposeModule(module);
  return 0;
}
```

编译：

clang `llvm-config --cflags --ldflags --system-libs --libs core` toy.c -o toy

生成IR文件：

./toy 2>toy.ll

执行IR文件：

lli toy.ll

# 参考资料

IR手册：http://www.llvm.org/docs/LangRef.html

核心类介绍：http://llvm.org/docs/ProgrammersManual.html

LLVM命令手册：http://llvm.org/docs/CommandGuide/index.html

前端实现tutorial：https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/index.html

# LLVM资源

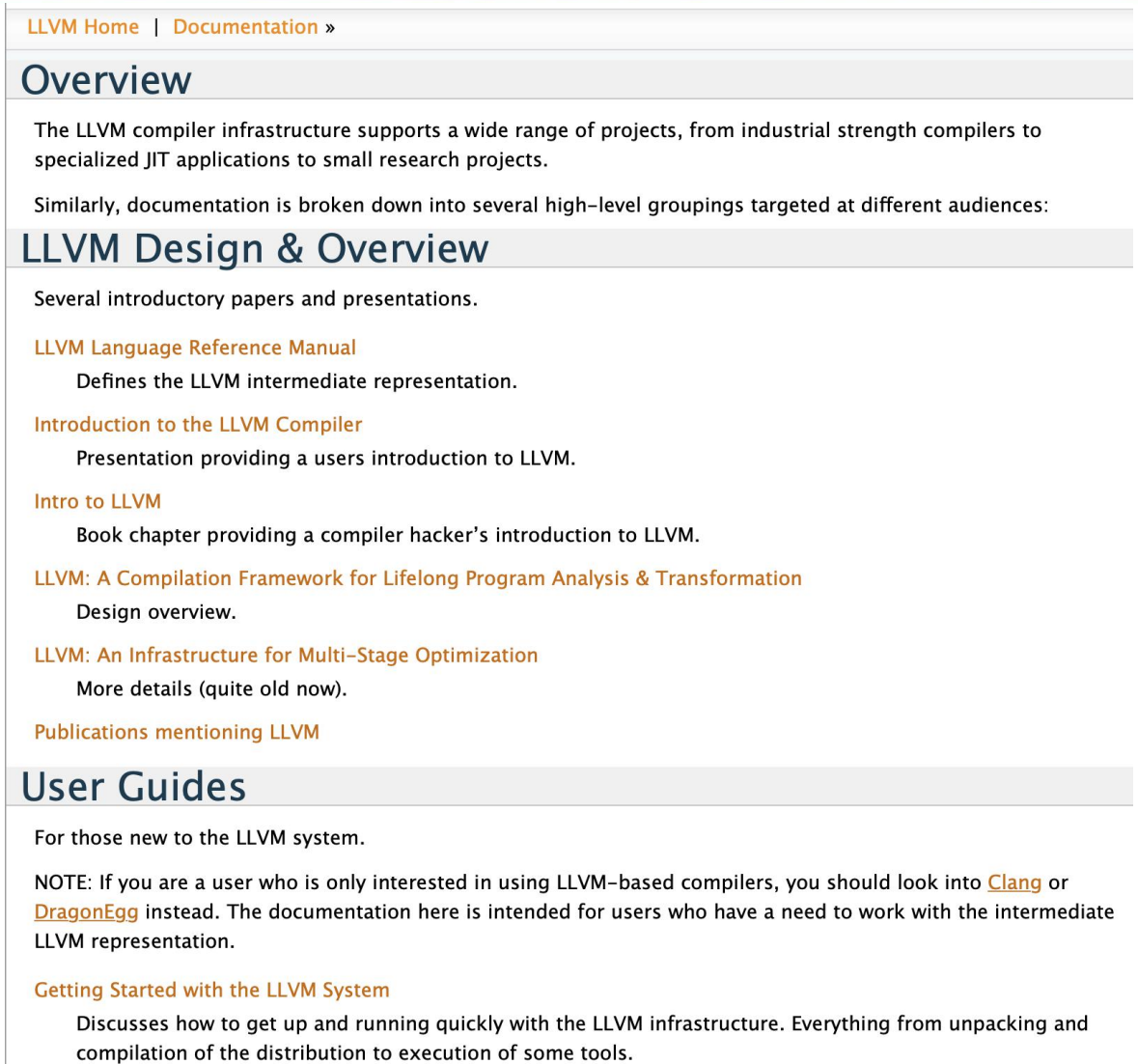
## 在线文档

- 用户参考文档
- 开发者参考文档
- 贡献者参考文档
- 等等

不足之处

- 缺少一个完整详细的命令行手册
- 文档显得有些杂乱
- 一些内容（特别是代码相关）过时了

http://www.llvm.org/docs/index.html



LLVM Home | Documentation »

## Overview

The LLVM compiler infrastructure supports a wide range of projects, from industrial strength compilers to specialized JIT applications to small research projects.

Similarly, documentation is broken down into several high-level groupings targeted at different audiences:

## LLVM Design & Overview

Several introductory papers and presentations.

LLVM Language Reference Manual
Defines the LLVM intermediate representation.

Introduction to the LLVM Compiler
Presentation providing a users introduction to LLVM.

Intro to LLVM
Book chapter providing a compiler hacker's introduction to LLVM.

LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation
Design overview.

LLVM: An Infrastructure for Multi-Stage Optimization
More details (quite old now).

Publications mentioning LLVM

## User Guides

For those new to the LLVM system.

NOTE: If you are a user who is only interested in using LLVM-based compilers, you should look into Clang or DragonEgg instead. The documentation here is intended for users who have a need to work with the intermediate LLVM representation.

Getting Started with the LLVM System
Discusses how to get up and running quickly with the LLVM infrastructure. Everything from unpacking and compilation of the distribution to execution of some tools.

# LLVM资源

## 开发者大会

- 技术报告
- 教程
- 闪电演讲
- 研究展示
- 等等

大部分可以获得

- 视频
- 幻灯片

http://llvm.org

## Developer Meetings

Upcoming:
October 22-23, 2019 (special events Oct 21) in San Jose, CA

Proceedings from past meetings:

- April 8-9, 2019
- October 17-18, 2018
- April 16-17, 2018
- October 18-19, 2017
- March 27-28, 2017
- November 3-4, 2016
- March 17-18, 2016
- October 29-30, 2015
- April 13-14, 2015
- October 28-29, 2014
- April 7-8, 2014
- Nov 6-7, 2013
- April 29-30, 2013
- November 7-8, 2012
- April 12, 2012
- November 18, 2011
- September 2011
- November 2010
- October 2009
- August 2008
- May 2007

## About

The LLVM developers' meeting is a bi-annual 2 day gathering of the entire LLVM Project community. The conference is organized by the LLVM Foundation and many volunteers within the LLVM community. Developers and users of LLVM, Clang, and related subprojects will enjoy attending interesting talks, impromptu discussions, and networking with the many members of our community. Whether you are a new to the LLVM project or a long time member, there is something for each attendee.

What can you can expect at an LLVM developers' meeting?

**Technical Talks**
: These 30 minute talks cover all topics from core infrastructure talks, to project's using LLVM's infrastructure. Attendees will take away technical information that could be pertinent to their project or general interest.

**Tutorials**
: Tutorials are 60 minute sessions that dive down deep into a technical topic. Expect in depth examples and explanations.

**Lightning Talks**
: These are fast 5 minute talks that give you a taste of a project or topic. Attendees will hear a wide range of topics and probably leave wanting to learn more.

**Birds of a Feather (BoF)**
: BoF sessions are more formal guided discussions about a specific topic. The presenter typically has slides to guide the discussion. The audience is given the opportunity to participate in the discussion.

**Student Research Competition**
: Students present their research using LLVM or related subproject. These are usually 25 minute technical presentations with Q&A. The audience will vote at the end for the winning presentation. Students also present their research during the poster session.

**Poster Session**
: An hour long poster session where selected posted are on display.

**Round Table Discussions**
: Informal and impromptu discussions on a specific topic. During the conference there are set time slots where groups can organize to discuss a problem or topic.

**Evening Reception**
: After a full day if technical talks and discussions, join your fellow attendees for an evening reception to continue the conversation and meet even more attendees.

# LLVM资源

## 邮件列表

- 社区用来技术讨论的主要方式之一
- 公开可见，方便查询
- 可以订阅、退订

http://lists.llvm.org

## Mailing Lists

If you can't find what you need in these docs, try consulting the mailing lists.

**Developer's List (llvm-dev)**

This list is for people who want to be included in technical discussions of LLVM. People post to this list when they have questions about writing code for or using the LLVM tools. It is relatively low volume.

**Commits Archive (llvm-commits)**

This list contains all commit messages that are made when LLVM developers commit code changes to the repository. It also serves as a forum for patch review (i.e. send patches here). It is useful for those who want to stay on the bleeding edge of LLVM development. This list is very high volume.

**Bugs & Patches Archive (llvm-bugs)**

This list gets emailed every time a bug is opened and closed. It is higher volume than the LLVM-dev list.

**Test Results Archive (llvm-testresults)**

A message is automatically sent to this list by every active nightly tester when it completes. As such, this list gets email several times each day, making it a high volume list.

**LLVM Announcements List (llvm-announce)**

This is a low volume list that provides important announcements regarding LLVM. It gets email about once a month.

# LLVM资源

## 在线交流

- 可以使用IRC（Internet Relay Chat，因特网中继聊天），在线技术交流
- 注意时差

IRC是一种历史悠久（1988年创造）、成熟稳定的网络即时通讯协议，被广泛地应用于在线通讯和网络聊天中。

### IRC

Users and developers of the LLVM project (including subprojects such as Clang) can be found in #llvm on irc.oftc.net.

This channel has several bots.

- Buildbot reporters
  - llvmbb – Bot for the main LLVM buildbot master. http://lab.llvm.org:8011/console
  - smooshlab – Apple's internal buildbot master.
- robot – Bugzilla linker. %bug <number>
- clang-bot – A geordi instance running near-trunk clang instead of gcc.