

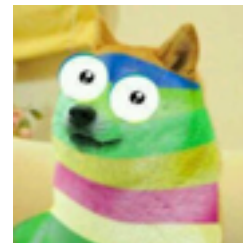


中国移动开发者大会
Mobile Developer Conference China 2016

把玩编译器，Clang 有意思 ^ ^


孙源 sunnyxx

- 姓名：孙源
- ID：sunnyxx
- 公司：滴滴出行
- 博客：<http://blog.sunnyxx.com>
- 微博：@我就叫Sunny怎么了
- GitHub：<http://github.com/forkingdog>



- 问：编译器可以编译程序，但编译器本身也是个程序，那它一定是由更早的编译器编译而成的，那... 最早的一个编译器是哪儿来的？



-  Apple 编译器 Clang-LLVM 架构初识
- 你的源码是如何一步步成为可执行文件的？
- 我们能用 Clang 做什么有意思的事情？

- LLVM - Low Level Virtual Machine
- Clang /'klæŋ/ - C Language Family Frontend for LLVM

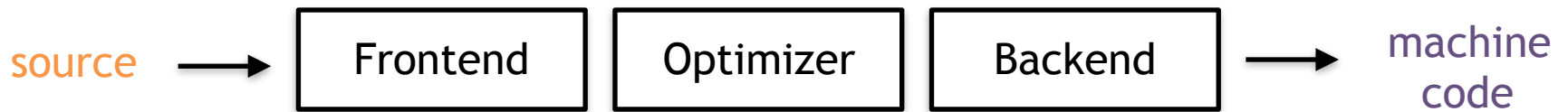


GCC 用的好好的，Apple 为啥要自己搞一套？

- GCC 的 Objective-C Frontend 不给力
- GCC 插件、工具、IDE 的支持薄弱
- GCC 编译效率和性能
- Apple 收回对工具链的控制 (lldb, lld, swift...)

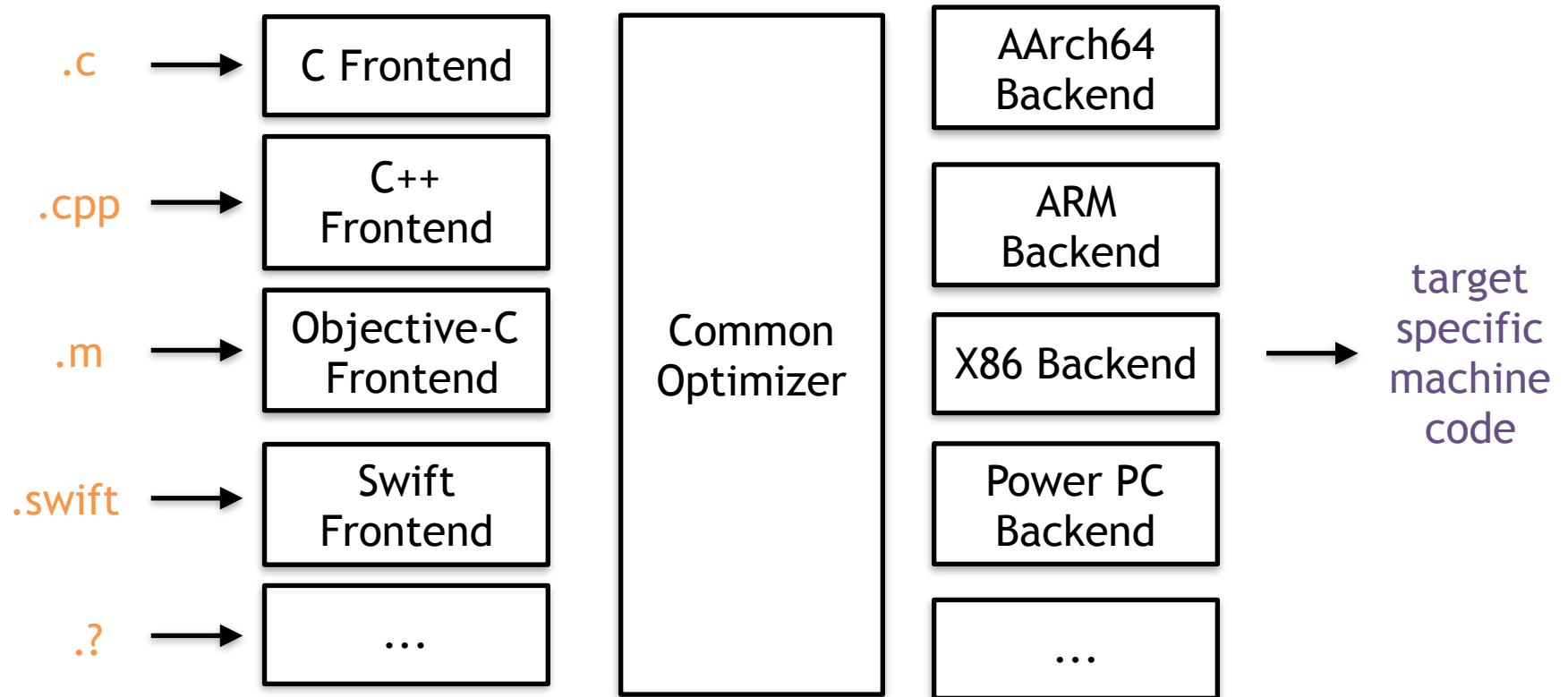


Three-Phase 编译器架构

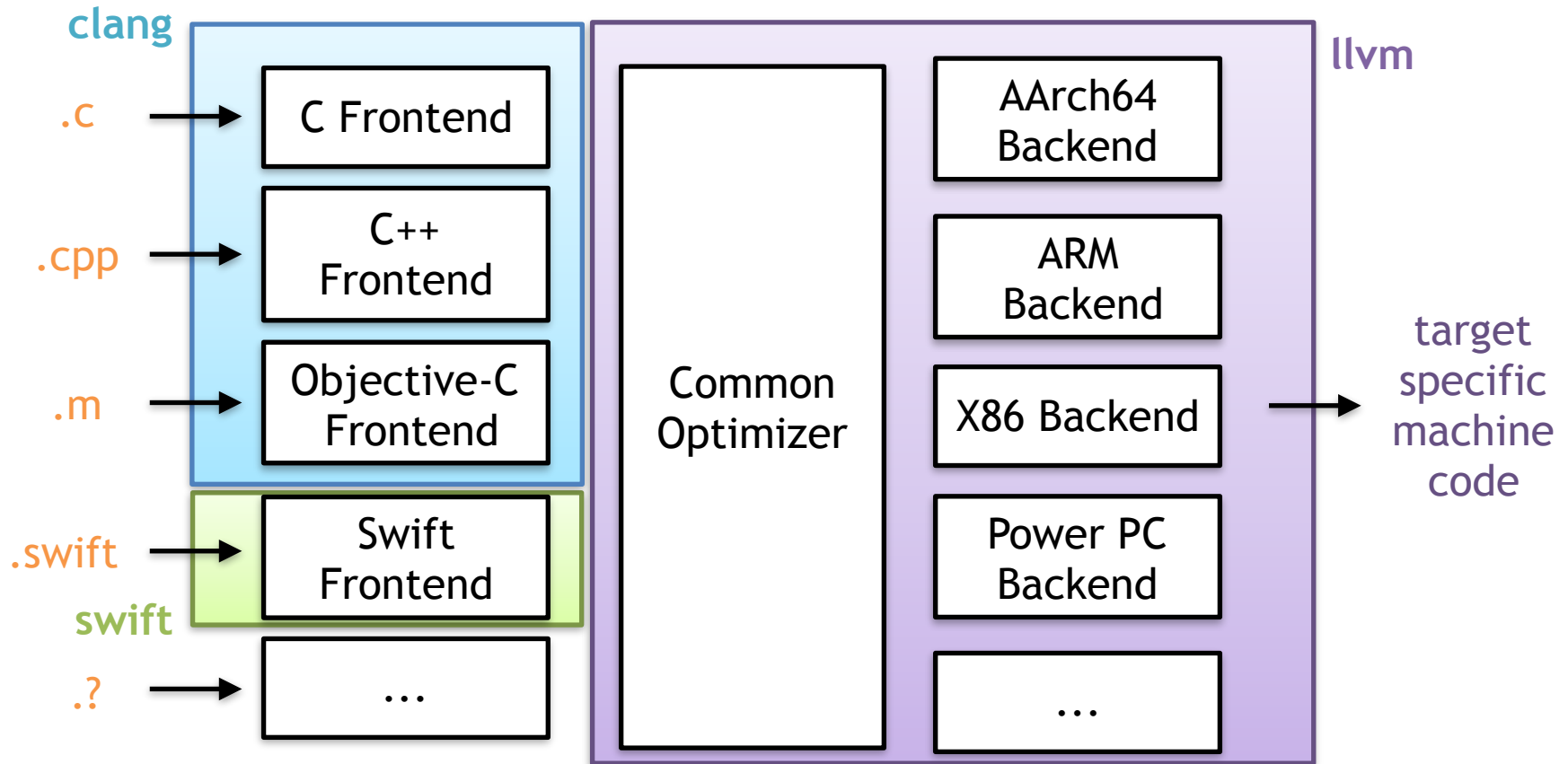


⚠ M (Language) * N (Target) = $M * N$ Compilers

Clang/Swift - LLVM 编译器架构



Clang/Swift - LLVM 编译器架构



PS: Swift Frontend 中还包含一层 SIL 及 Optimizer

Clang + LLVM 代码规模

- Total 400W
- C++ 235W

github.com/AlDanial/cloc v 1.70 T=472.28 s (36.4 files/s, 8470.9 lines/s)

Language	files	blank	comment	code
C++	6507	302077	420778	1627278
C/C++ Header	3198	99007	169324	406639
C	2770	36050	148231	121953
Assembly	1716	56701	173940	111549
Objective C	1346	14895	70242	50831
HTML	35	3160	280	26471
Python	210	5986	7640	23301
Windows Module Definition	68	1521	0	14635
Objective C++	355	4105	22984	14121
CMake	437	2019	1462	13144
OCaml	74	1774	2804	5722
YAML	74	152	1319	3365
Bourne Shell	38	349	605	3226
Perl	14	689	482	3154
Go	21	401	598	2980
OpenCL	135	920	1423	2869
Pascal	11	902	3645	1761
CUDA	71	657	1782	1358
DOS Batch	17	139	24	890
Lisp	9	181	206	810
XML	42	23	4	686
CSS	9	144	50	660
JavaScript	4	79	150	518
C#	6	46	93	359
JSON	11	52	0	357
vim script	8	38	46	283
Bourne Again Shell	4	34	96	227
MSBuild script	1	0	7	224
make	7	44	10	135
C Shell	1	13	14	118
Markdown	3	45	0	98
Windows Resource File	1	10	11	60
Fortran 95	1	3	0	18
Windows Message File	1	3	0	13
Rust	3	6	11	13
INI	1	1	0	6
NAnt script	1	0	0	5
Fortran 90	1	0	260	0
SUM:	17211	532250	1028545	2439861

Swift Frontend 代码规模



- C++ 43W

```
github.com/AlDanial/cloc v 1.70 T=62.38 s (159.6 files/s, 16696.3 lines/s)
```

Language	files	blank	comment	code
C++	533	62726	62671	312494
Swift	8238	54068	121746	218181
C/C++ Header	708	25005	36851	85162
Windows Module Definition	49	1420	0	10771
Python	110	2370	3254	9832
CMake	168	1050	1215	6547
Markdown	15	1971	0	6479
Objective C++	21	843	818	3794
Bourne Again Shell	12	373	432	2709
HTML	3	639	141	2409
Objective C	19	241	136	992
JSON	35	0	0	743
Lisp	5	109	226	732
INI	1	224	0	647
C	7	106	58	552
CSS	2	10	8	407
vim script	8	50	13	271
make	4	36	5	165
JavaScript	1	28	19	106
D	3	17	12	94
Ruby	1	7	2	87
Bourne Shell	10	19	16	74
Perl	1	7	3	69
Assembly	1	14	39	30
YAML	1	0	0	26
MUMPS	1	1	0	2
SUM:	9957	151334	227665	662575

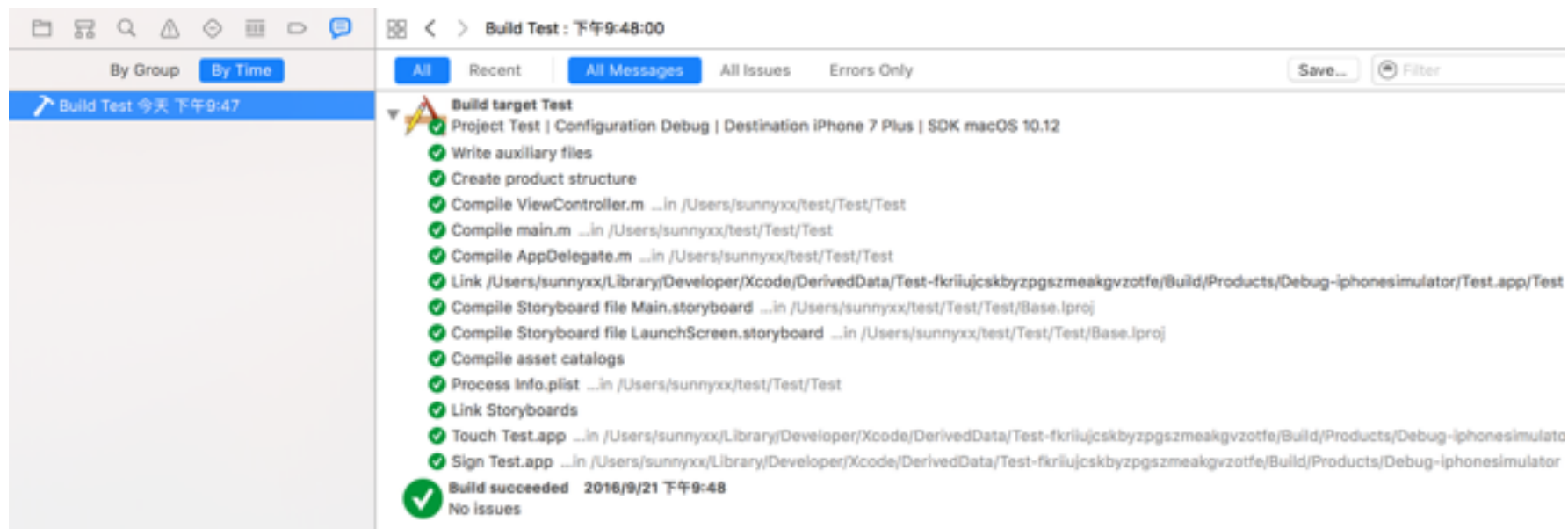
看 Clang-LLVM 源码的感受

- 代码巨多、需要一定 C++ 基础
- 远离安逸的 Xcode Build System, CMake Ninja 都比较陌生
- 目录明了、分层清晰、风格规范、注释覆盖度高 (~40%)
- 代码结构朴素但合理, 均以 library 的形式整合, 便于组合与复用

-  Apple 编译器 Clang-LLVM 架构初识
-  你的源码是如何一步步成为可执行文件的?
- 我们能用 Clang 做什么有意思的事情?



当我们按下 Run 之后...



当我们按下 Run 之后...

```
CompileC /Users/sunnyxx/Library/Developer/Xcode/DerivedData/Test-fkriiujcskbyzpgszmeakgvzotfe/Build/Intermediates/
Test.build/Debug-iphonesimulator/Test.build/Objects-normal/x86_64/main.o Test/main.m normal x86_64 objective-c
com.apple.compilers.llvm.clang.1_0.compiler
cd /Users/sunnyxx/test/Test
export LANG=en_US.US-ASCII
export PATH="/Applications/Xcode.app/Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/usr/bin:/
Applications/Xcode.app/Contents/Developer/usr/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin"
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/clang -x objective-c -arch
x86_64 -fmessage-length=0 -fdiagnostics-show-note-include-stack -fmacro-backtrace-limit=0 -std=gnu99 -fobjc-arc -fmodules
-gmodules -fmodules-cache-path=/Users/sunnyxx/Library/Developer/Xcode/DerivedData/ModuleCache -fmodules-prune-
interval=86400 -fmodules-prune-after=345600 -fbuild-session-file=/Users/sunnyxx/Library/Developer/Xcode/DerivedData/
ModuleCache/Session.modulevalidation -fmodules-validate-once-per-build-session -Wnon-modular-include-in-framework-module
-Werror=non-modular-include-in-framework-module -Wno-trigraphs -fpascal-strings -O0 -fno-common -Wno-missing-field-
initializers -Wno-missing-prototypes -Werror=return-type -Wdocumentation -Wunreachable-code -Wno-implicit-atomic-
properties -Werror=deprecated-objc-isa-usage -Werror=objc-root-class -Wno-arc-repeated-use-of-weak -Wduplicate-method-
match -Wno-missing-braces -Wparentheses -Wswitch -Wunused-function -Wno-unused-label -Wno-unused-parameter -Wunused-
variable -Wunused-value -Wempty-body -Wconditional-uninitialized -Wno-unknown-pragmas -Wno-shadow -Wno-four-char-
constants -Wno-conversion -Wconstant-conversion -Wint-conversion -Wbool-conversion -Wenum-conversion -Wshorten-64-to-32 -
Wpointer-sign -Wno-newline-eof -Wno-selector -Wno-strict-selector-match -Wundeclared-selector -Wno-deprecated-
implementations -DDEBUG=1 -DOBJC_OLD_DISPATCH_PROTOTYPES=0 -isysroot /Applications/Xcode.app/Contents/Developer/
Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator10.0.sdk -fasm-blocks -fstrict-aliasing -Wprotocol -
Wdeprecated-declarations -mios-simulator-version-min=10.0 -g -Wno-sign-conversion -Winfinite-recursion -fobjc-abi-
version=2 -fobjc-legacy-dispatch -quote /Users/sunnyxx/Library/Developer/Xcode/DerivedData/Test-
fkriiujcskbyzpgszmeakgvzotfe/Build/Intermediates/Test.build/Debug-iphonesimulator/Test.build/Test-generated-files.hmap -
I/Users/sunnyxx/Library/Developer/Xcode/DerivedData/Test-fkriiujcskbyzpgszmeakgvzotfe/Build/Intermediates/Test.build/
Debug-iphonesimulator/Test.build/Test-own-target-headers.hmap -I/Users/sunnyxx/Library/Developer/Xcode/DerivedData/Test-
fkriiujcskbyzpgszmeakgvzotfe/Build/Intermediates/Test.build/Debug-iphonesimulator/Test.build/Test-all-target-headers.hmap
-iquote /Users/sunnyxx/Library/Developer/Xcode/DerivedData/Test-fkriiujcskbyzpgszmeakgvzotfe/Build/Intermediates/
Test.build/Debug-iphonesimulator/Test.build/Test-project-headers.hmap -I/Users/sunnyxx/Library/Developer/Xcode/
DerivedData/Test-fkriiujcskbyzpgszmeakgvzotfe/Build/Products/Debug-iphonesimulator/include -I/Users/sunnyxx/Library/
Developer/Xcode/DerivedData/Test-fkriiujcskbyzpgszmeakgvzotfe/Build/Intermediates/Test.build/Debug-iphonesimulator/
Test.build/DerivedSources/x86_64 -I/Users/sunnyxx/Library/Developer/Xcode/DerivedData/Test-fkriiujcskbyzpgszmeakgvzotfe/
Build/Intermediates/Test.build/Debug-iphonesimulator/Test.build/DerivedSources -F/Users/sunnyxx/Library/Developer/Xcode/
DerivedData/Test-fkriiujcskbyzpgszmeakgvzotfe/Build/Products/Debug-iphonesimulator -MMD -MT dependencies -MF /Users/
sunnyxx/Library/Developer/Xcode/DerivedData/Test-fkriiujcskbyzpgszmeakgvzotfe/Build/Intermediates/Test.build/Debug-
iphonesimulator/Test.build/Objects-normal/x86_64/main.d --serialize-diagnostics /Users/sunnyxx/Library/Developer/Xcode/
DerivedData/Test-fkriiujcskbyzpgszmeakgvzotfe/Build/Intermediates/Test.build/Debug-iphonesimulator/Test.build/Objects-
normal/x86_64/main.dia -c /Users/sunnyxx/test/Test/Test/main.m -o /Users/sunnyxx/Library/Developer/Xcode/DerivedData/
Test-fkriiujcskbyzpgszmeakgvzotfe/Build/Intermediates/Test.build/Debug-iphonesimulator/Test.build/Objects-normal/x86_64/
main.o
```

当我们按下 Run 之后...

```
/Applications/Xcode.app/Contents/Developer/  
Toolchains/XcodeDefault.xctoolchain/usr/bin/  
clang -x objective-c -fobjc-arc ..... main.m  
-o main.o
```


Clang 命令

- Clang 在概念上是编译器前端，同时，在命令行中也作为一个“黑盒”的 Driver
- 封装了编译管线、前端命令、LLVM 命令、Toolchain 命令等，一个 Clang 走天下
- 方便从 gcc 迁移过来



gcc



clang

拆解编译过程



main.m

```
#import <Foundation/Foundation.h>

int main() {
    @autoreleasepool {
        id obj = [NSObject new];
        NSLog(@"Hello world: %@", obj);
    }
    return 0;
}
```

1. Preprocess - 预处理

- import 头文件
- macro 展开
- 处理 ‘#’ 打头的预处理指令，如 #if

1. Preprocess - 预处理

```
$clang -E main.m
```

```
...
...
...
# 181 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/
SDKs/MacOSX10.12.sdk/System/Library/Frameworks/Foundation.framework/Headers/
Foundation.h" 2 3
# 1 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/
SDKs/MacOSX10.12.sdk/System/Library/Frameworks/Foundation.framework/Headers/
FoundationLegacySwiftCompatibility.h" 1 3
# 185 "/Applications/Xcode.app/Contents/Developer/Platforms/MacOSX.platform/Developer/
SDKs/MacOSX10.12.sdk/System/Library/Frameworks/Foundation.framework/Headers/
Foundation.h" 2 3
# 6 "main.m" 2

int main() {
    @autoreleasepool {
        id obj = [NSObject new];
        NSLog(@"Hello world: %@", obj);
    }
    return 0;
}
```

1. Preprocess - 预处理

```
$clang -E -fmodules main.m
```

```
@import Foundation;
int main() {
    @autoreleasepool {
        id obj = [NSObject new];
        NSLog(@"Hello world: %@", obj);
    }
    return 0;
}
```

2. Lexical Analysis - 词法分析

- 词法分析，也作 Lex 或者 Tokenization
- 将预处理过的代码文本转化成 Token 流
- 不校验语义

2. Lexical Analysis - 词法分析

```
$clang -fmodules -fsyntax-only -Xclang -dump-tokens main.m
```

```
int 'int' [StartOfLine] Loc=<main.m:7:1>
identifier 'main' [LeadingSpace] Loc=<main.m:7:5>
l_paren '(' [LeadingSpace] Loc=<main.m:7:9>
r_paren ')' [LeadingSpace] Loc=<main.m:7:10>
l_brace '{' [LeadingSpace] Loc=<main.m:7:12>
at '@' [StartOfLine] [LeadingSpace] Loc=<main.m:8:5>
identifier 'autoreleasepool' [LeadingSpace] Loc=<main.m:8:6>
l_brace '{' [LeadingSpace] Loc=<main.m:8:22>
identifier 'id' [StartOfLine] [LeadingSpace] Loc=<main.m:9:9>
identifier 'obj' [LeadingSpace] Loc=<main.m:9:12>
equal '=' [LeadingSpace] Loc=<main.m:9:16>
l_square '[' [LeadingSpace] Loc=<main.m:9:18>
identifier 'NSObject' [LeadingSpace] Loc=<main.m:9:19>
identifier 'new' [LeadingSpace] Loc=<main.m:9:28>
r_square ']' [LeadingSpace] Loc=<main.m:9:31>
semi ';' [LeadingSpace] Loc=<main.m:9:32>
...
```


3. Semantic Analysis - 语法分析

- 语法分析，在 Clang 中由 Parser 和 Sema 两个模块配合完成

- 验证语法是否正确

```
main.m:8:32: error: expected ';' at end of declaration  
id obj = [NSObject new]
```

- 根据当前语言的语法，生成语意节点，并将所有节点组合成抽象语法树 (AST)

3. Semantic Analysis - 语法分析

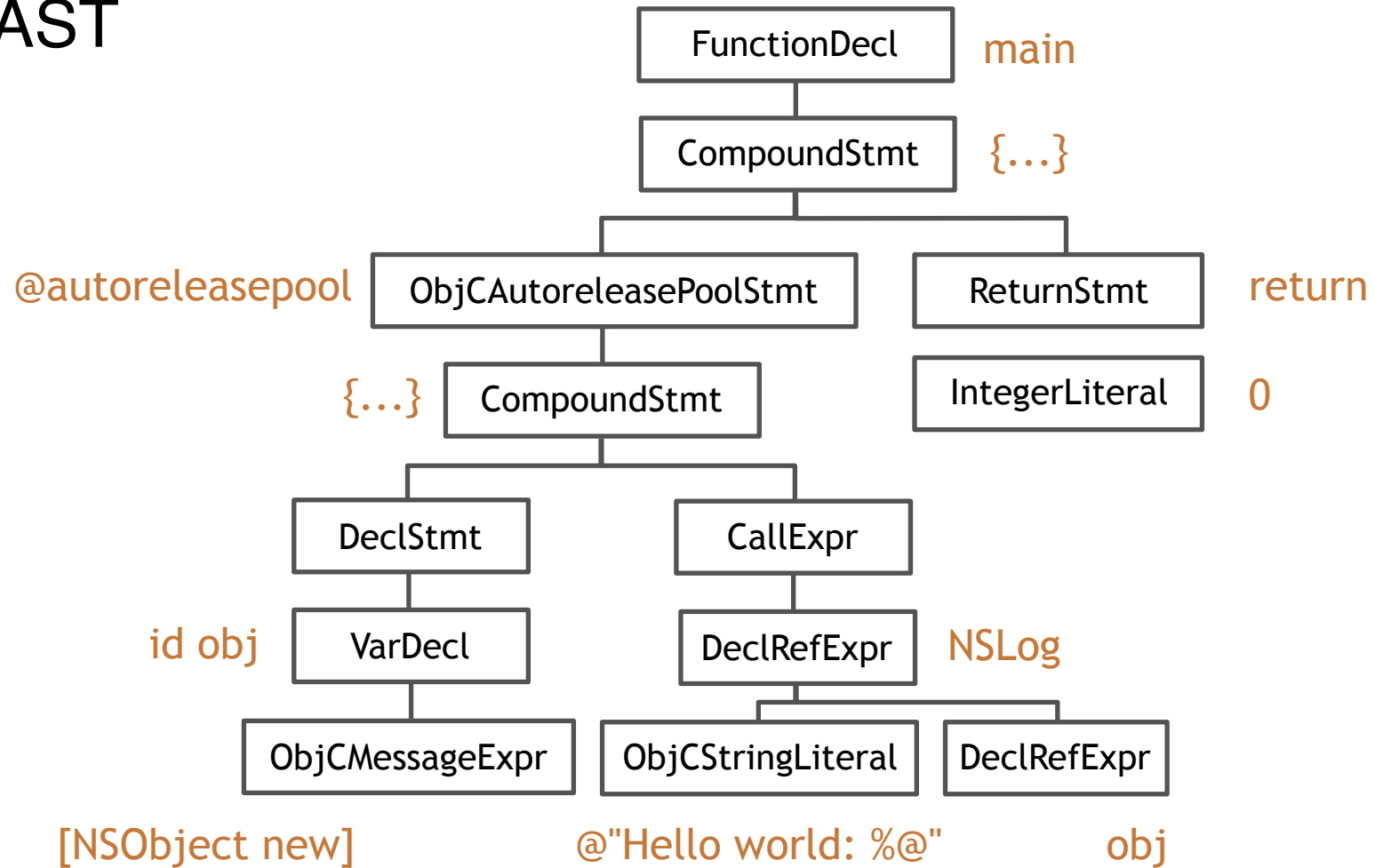
```
$clang -fmodules -fsyntax-only -Xclang -ast-dump main.m
```

```

|-FunctionDecl 0x7fe881035b38 <line:6:1, line:12:1> line:6:5 main 'int ()'
|  |-CompoundStmt 0x7fe88133ac28 <col:12, line:12:1>
|  |  |-ObjCAutoreleasePoolStmt 0x7fe88133abe0 <line:7:5, line:10:5>
|  |  |  |-CompoundStmt 0x7fe88133abb8 <line:7:22, line:10:5>
|  |  |  |  |-DeclStmt 0x7fe88133a9e0 <line:8:9, col:32>
|  |  |  |  |  |-VarDecl 0x7fe88132b728 <col:9, col:31> col:12 used obj 'id':'id' cinit
|  |  |  |  |  |  |-ImplicitCastExpr 0x7fe881327778 <col:18, col:31> 'id':'id' <BitCast>
|  |  |  |  |  |  |  |-ObjCMessageExpr 0x7fe881327748 <col:18, col:31> 'NSObject *'
|  |  |  |  |  selector=new class='NSObject'
|  |  |  |  |  |  |-CallExpr 0x7fe88133ab50 <line:9:9, col:38> 'void'
|  |  |  |  |  |  |  |-ImplicitCastExpr 0x7fe88133ab38 <col:9> 'void (*)(id, ...)'
|  |  |  |  |  <FunctionToPointerDecay>
|  |  |  |  |  |  |-DeclRefExpr 0x7fe88133a9f8 <col:9> 'void (id, ...)' Function
|  |  |  |  |  0x7fe881327798 'NSLog' 'void (id, ...)'
|  |  |  |  |  |  |-ImplicitCastExpr 0x7fe88133ab88 <col:15, col:16> 'id':'id' <BitCast>
|  |  |  |  |  |  |  |-ObjCStringLiteral 0x7fe88133aa90 <col:15, col:16> 'NSString *'
|  |  |  |  |  |  |  |  |-StringLiteral 0x7fe88133aa58 <col:16> 'char [16]' lvalue "Hello world:
|  |  |  |  |  %@"
|  |  |  |  |  |  |-ImplicitCastExpr 0x7fe88133aba0 <col:35> 'id':'id' <LValueToRValue>
|  |  |  |  |  |  |  |-DeclRefExpr 0x7fe88133aab0 <col:35> 'id':'id' lvalue Var 0x7fe88132b728
|  |  |  |  |  'obj' 'id':'id'
|  |  |  |  |  |  |-ReturnStmt 0x7fe88133ac10 <line:11:5, col:12>
|  |  |  |  |  |  |  |-IntegerLiteral 0x7fe88133abf0 <col:12> 'int' 0

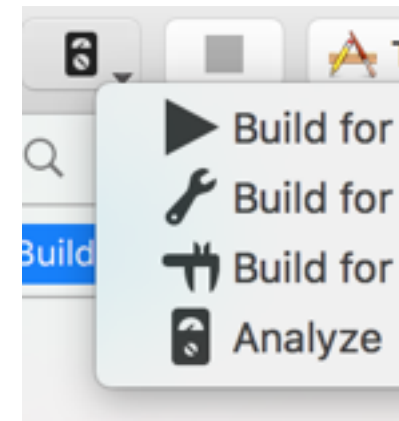
```

AST



Static Analysis - 静态分析

- 通过语法树进行代码静态分析，找出**非语法性错误**
- 模拟代码执行路径，分析出 control-flow graph (CFG)
- 预置了常用 Checker



4. CodeGen - IR 代码生成

- CodeGen 负责将语法树从顶至下遍历，翻译成 LLVM IR
- LLVM IR 是 Frontend 的输出，也是 LLVM Backend 的输入，前后端的桥接语言
- **与 Objective-C Runtime 桥接**

4. CodeGen - IR 代码生成 与 Objective-C Runtime 桥接

- Class / Meta Class / Protocol / Category 内存结构生成, 并存放在指定 section 中 (如 Class: `_DATA`, `_objc_classrefs`)
- Method / Ivar / Property 内存结构生成
- 组成 `method_list` / `ivar_list` / `property_list` 并填入 Class

4. CodeGen - IR 代码生成 与 Objective-C Runtime 桥接

- Non-Fragile ABI: 为每个 Ivar 合成 `OBJC_IVAR_$_` 偏移值常量
- 存取 Ivar 的语句 (`_ivar = 123; int a = _ivar;`) 转写成 `base + OBJC_IVAR_$_` 的形式

4. CodeGen - IR 代码生成 与 Objective-C Runtime 桥接

- 将语法树中的 `ObjCMessageExpr` 翻译成相应版本的 `objc_msgSend`, 对 `super` 关键字的调用翻译成 `objc_msgSendSuper`

4. CodeGen - IR 代码生成 与 Objective-C Runtime 桥接

- 根据修饰符 strong / weak / copy / atomic 合成 @property 自动实现的 setter / getter
- 处理 @synthesize

4. CodeGen - IR 代码生成 与 Objective-C Runtime 桥接

- 生成 `block_layout` 的数据结构
- 变量的 capture (`__block` / `__weak`)
- 生成 `_block_invoke` 函数

4. CodeGen - IR 代码生成 与 Objective-C Runtime 桥接

- ARC: 分析对象引用关系, 将 `objc_storeStrong/objc_storeWeak` 等 ARC 代码插入
- 将 `ObjCAutoreleasePoolStmt` 转译成 `objc_autoreleasePoolPush/Pop`
- 实现自动调用 `[super dealloc]`
- 为每个拥有 ivar 的 Class 合成 `.cxx_destructor` 方法来自动释放类的成员变量, 代替 MRC 时代的 `"self.xxx = nil"`

4. CodeGen - IR 代码生成 与 Objective-C Runtime 桥接

```

namespace {
struct FinishARCDealloc final : EHScopeStack::Cleanup {
    void Emit(CodeGenFunction &CGF, Flags flags) override {
        const ObjCMethodDecl *method = cast<ObjCMethodDecl>(CGF.CurCodeDecl);

        const ObjCImplDecl *impl = cast<ObjCImplDecl>(method->getDeclContext());
        const ObjCInterfaceDecl *iface = impl->getClassInterface();
        if (!iface->getSuperClass()) return;

        bool isCategory = isa<ObjCCategoryImplDecl>(impl);

        // Call [super dealloc] if we have a superclass.
        llvm::Value *self = CGF.LoadObjCSelf();

        CallArgList args;
        CGF.CGM.getObjCRuntime().GenerateMessageSendSuper(CGF, ReturnValueSlot(),
                                                         CGF.getContext().VoidTy,
                                                         method->getSelector(),
                                                         iface,
                                                         isCategory,
                                                         self,
                                                         /*is class msg*/ false,
                                                         args,
                                                         method);
    }
};
}

```

合成 [super dealloc]

CodeGen - IR 代码生成

```
$clang -S -fobjc-arc -emit-llvm main.m -o main.ll
```

```
define i32 @main() #0 {
    %1 = alloca i32, align 4
    %2 = alloca i8*, align 8
    store i32 0, i32* %1, align 4
    %3 = call i8* @objc_autoreleasePoolPush() #3
    %4 = load %struct._class_t*, %struct._class_t**
@"OBJC_CLASSLIST_REFERENCES_$_", align 8
    %5 = load i8*, i8** @OBJC_SELECTOR_REFERENCES_, align 8, !invariant.load !7
    %6 = bitcast %struct._class_t* %4 to i8*
    %7 = call i8* bitcast (i8* (i8*, i8*, ...)* @objc_msgSend to i8* (i8*,
i8*)*)(i8* %6, i8* %5)
    %8 = bitcast i8* %7 to %0*
    %9 = bitcast %0* %8 to i8*
    store i8* %9, i8** %2, align 8
    %10 = load i8*, i8** %2, align 8
    notail call void (i8*, ...) @NSLog(i8* bitcast
(%struct.__NSConstantString_tag* @unnamed_cfstring_ to i8*), i8* %10)
    call void @objc_storeStrong(i8** %2, i8* null) #3
    call void @objc_autoreleasePoolPop(i8* %3)
    ret i32 0
}
```

Optimize - 优化 IR

```
$clang -O3 -S -fobjc-arc -emit-llvm main.m -o main.ll
```

```
define i32 @main() #0 {
    %1 = tail call i8* @objc_autoreleasePoolPush() #3
    %2 = load i8*, i8** bitcast (%struct._class_t**
@"OBJC_CLASSLIST_REFERENCES_$_" to i8**), align 8
    %3 = load i8*, i8** @OBJC_SELECTOR_REFERENCES_, align 8, !invariant.load
!7
    %4 = tail call i8* bitcast (i8* (i8*, i8*, ...)* @objc_msgSend to i8*
(i8*, i8*)*)(i8* %2, i8* %3), !clang.arc.no_objc_arc_exceptions !7
    %5 = bitcast i8* %4 to %0*
    %6 = bitcast %0* %5 to i8*
    notail call void (i8*, ...) @NSLog(i8* bitcast
(%struct.__NSConstantString_tag* @unnamed_cfstring_ to i8*), i8* %4), !
clang.arc.no_objc_arc_exceptions !7
    tail call void @objc_release(i8* %6) #3, !clang.imprecise_release !7
    tail call void @objc_autoreleasePoolPop(i8* %1) #3, !
clang.arc.no_objc_arc_exceptions !7
    ret i32 0
}
```

LLVM Bitcode - 生成字节码

```
$clang -emit-llvm -c main.m -o main.bc
```

```

0  DEC0170B 00000000 14000000 00120000 07000001 4243C0DE 35140000 06000000
32  620C3024 9296A6A5 F7D73F4F D33EEDDF FC4F0B51 804C0100 210C0000 75040000
64  0B822000 02000000 13000000 07812391 41C80449 06103239 9201840C 25050819
96  1E048B62 801C4502 42920B42 E4103214 3808184B 0A327288 48901420 434688A5
128 00193242 E4480E90 9123C450 4151818C E183E58A 04394606 51180000 F3000000
160 1B4825F8 FFFFFFFF 01D08030 20C8211D E6211CC4 811DCAA1 00E8211C D2811DDA
192 601CC281 1DD8611E 00730807 76988772 00087628 87799887 36800779 28877148
224 87792887 36300778 68877020 07C01CC2 811DE6A1 1C00C21D DEA10DCC 411EC2A1
256 1DCAA10D E0E11DD2 C11DE8A1 1CE4A10D CA811DD2 A11D007A 90877A28 07607087
288 77680373 90877068 87726803 78788774 70077A28 07796883 72608774 68873670
320 87777087 36608772 08077300 08777887 36480777 30877968 03738087 36688770
352 A0077400 CC211CD8 611ECA01 200CE11D DAC01DC2 C11DE6A1 00CC011E DAA01DC2
384 811ED001 30877060 87792807 8098077A 08877158 87368007 7978077A 288771A0
416 87779087 3610877A 30077328 07796883 7948077D 2807000F 00821EC2 411ECEA1
448 1CE8A10D C6011EEA 01808772 700779C8 07770008 7A080779 388772A0 87363087
480 7208077A A8077928 877900D6 601CF021 0EEC800D 06C01CF0 611EE481 0D06A01D
512 DA011FD8 600DE661 1ECA810D D6601EE6 A11CE480 0DD6601E E6A11CE6 800DD660
544 1EE6A11C E8C00DE2 800DD660 1EE6611E CA610E00 A21EDC61 1EC2C11C CAA10DCC
576 011EDAA0 1DC2811E D0013087 70608779 280780A8 87792887 36988777 30077A68
608 03736087 7708077A 00CC211C D8611ECA 01D84008 FFFFFFFF 3F00CF06 62088065
640 834108C0 02541B8C E2FFFFFF FF07A00D 006903A8 18FFFFFF FF3F10E4 900EF310
672 0EE2C00E E5D006F4 100EE9C0 0E6D300E E1C00EEC 300F8039 84033BCC 43390004
704 3B94C33C CC431BC0 833C94C3 38A4C33C 94431B98 033CB443 38900360 0EE1C00E
736 F3500E00 E10EEFD0 06E6200F E1D00EE5 D006F0F0 0EE9E00E F4500EF2 D006E5C0
768 0EE9D00E 003DC843 3D940330 B8C33BB4 8139C843 38B44339 B4013CBC 433AB803
800 3D94833C B441390D 433AB443 1BB8C33B B8431BB0 43398483 3900843B BC431BA4
832 833B98C3 3CB48139 C0431BB4 4338D003 3A00E610 0EEC300F E50010EE F00E60E0

```

```

..                               BC..5
b 0$. . . . . ? 0 . > . . . 0 Q . L | u
  0x00000000 0x14000000 0x00120000 0x07000001 0x4243C0DE 0x35140000 0x06000000
  0x620C3024 0x9296A6A5 0xF7D73F4F 0xD33EEDDF 0xFC4F0B51 0x804C0100 0x210C0000 0x75040000
  0x0B822000 0x02000000 0x13000000 0x07812391 0x41C80449 0x06103239 0x9201840C 0x25050819
  0x1E048B62 0x801C4502 0x42920B42 0xE4103214 0x3808184B 0x0A327288 0x48901420 0x434688A5
  0x00193242 0xE4480E90 0x9123C450 0x4151818C 0xE183E58A 0x04394606 0x51180000 0xF3000000
  0x1B4825F8 0xFFFFFFFF 0x01D08030 0x20C8211D 0xE6211CC4 0x811DCAA1 0x00E8211C 0xD2811DDA
  0x601CC281 0x1DD8611E 0x00730807 0x76988772 0x00087628 0x87799887 0x36800779 0x28877148
  0x87792887 0x36300778 0x68877020 0x07C01CC2 0x811DE6A1 0x1C00C21D 0xDEA10DCC 0x411EC2A1
  0x1DCAA10D 0xE0E11DD2 0xC11DE8A1 0x1CE4A10D 0xCA811DD2 0xA11D007A 0x90877A28 0x07607087
  0x77680373 0x90877068 0x87726803 0x78788774 0x70077A28 0x07796883 0x72608774 0x68873670
  0x87777087 0x36608772 0x08077300 0x08777887 0x36480777 0x30877968 0x03738087 0x36688770
  0xA0077400 0xCC211CD8 0x611ECA01 0x200CE11D 0xDAC01DC2 0xC11DE6A1 0x00CC011E 0xDAA01DC2
  0x811ED001 0x30877060 0x87792807 0x8098077A 0x08877158 0x87368007 0x7978077A 0x288771A0
  0x87779087 0x3610877A 0x30077328 0x07796883 0x7948077D 0x2807000F 0x00821EC2 0x411ECEA1
  0x1CE8A10D 0xC6011EEA 0x01808772 0x700779C8 0x07770008 0x7A080779 0x388772A0 0x87363087
  0x7208077A 0xA8077928 0x877900D6 0x601CF021 0x0EEC800D 0x06C01CF0 0x611EE481 0x0D06A01D
  0xDA011FD8 0x600DE661 0x1ECA810D 0xD6601EE6 0xA11CE480 0x0DD6601E 0xE6A11CE6 0x800DD660
  0x1EE6A11C 0xE8C00DE2 0x800DD660 0x1EE6611E 0xCA610E00 0xA21EDC61 0x1EC2C11C 0xCAA10DCC
  0x011EDAA0 0x1DC2811E 0xD0013087 0x70608779 0x280780A8 0x87792887 0x36988777 0x30077A68
  0x03736087 0x7708077A 0x00CC211C 0xD8611ECA 0x01D84008 0xFFFFFFFF 0x3F00CF06 0x62088065
  0x834108C0 0x02541B8C 0xE2FFFFFF 0xFF07A00D 0x006903A8 0x18FFFFFF 0xFF3F10E4 0x900EF310
  0x0EE2C00E 0xE5D006F4 0x100EE9C0 0x0E6D300E 0xE1C00EEC 0x300F8039 0x84033BCC 0x43390004
  0x3B94C33C 0xCC431BC0 0x833C94C3 0x38A4C33C 0x94431B98 0x033CB443 0x38900360 0x0EE1C00E
  0xF3500E00 0xE10EEFD0 0x06E6200F 0xE1D00EE5 0xD006F0F0 0x0EE9E00E 0xF4500EF2 0xD006E5C0
  0x0EE9D00E 0x003DC843 0x3D940330 0xB8C33BB4 0x8139C843 0x38B44339 0xB4013CBC 0x433AB803
  0x3D94833C 0xB441390D 0x433AB443 0x1BB8C33B 0xB8431BB0 0x43398483 0x3900843B 0xBC431BA4
  0x833B98C3 0x3CB48139 0xC0431BB4 0x4338D003 0x3A00E610 0x0EEC300F 0xE50010EE 0xF00E60E0

```

Signed Int big (select some data)

0 out of 4640 bytes

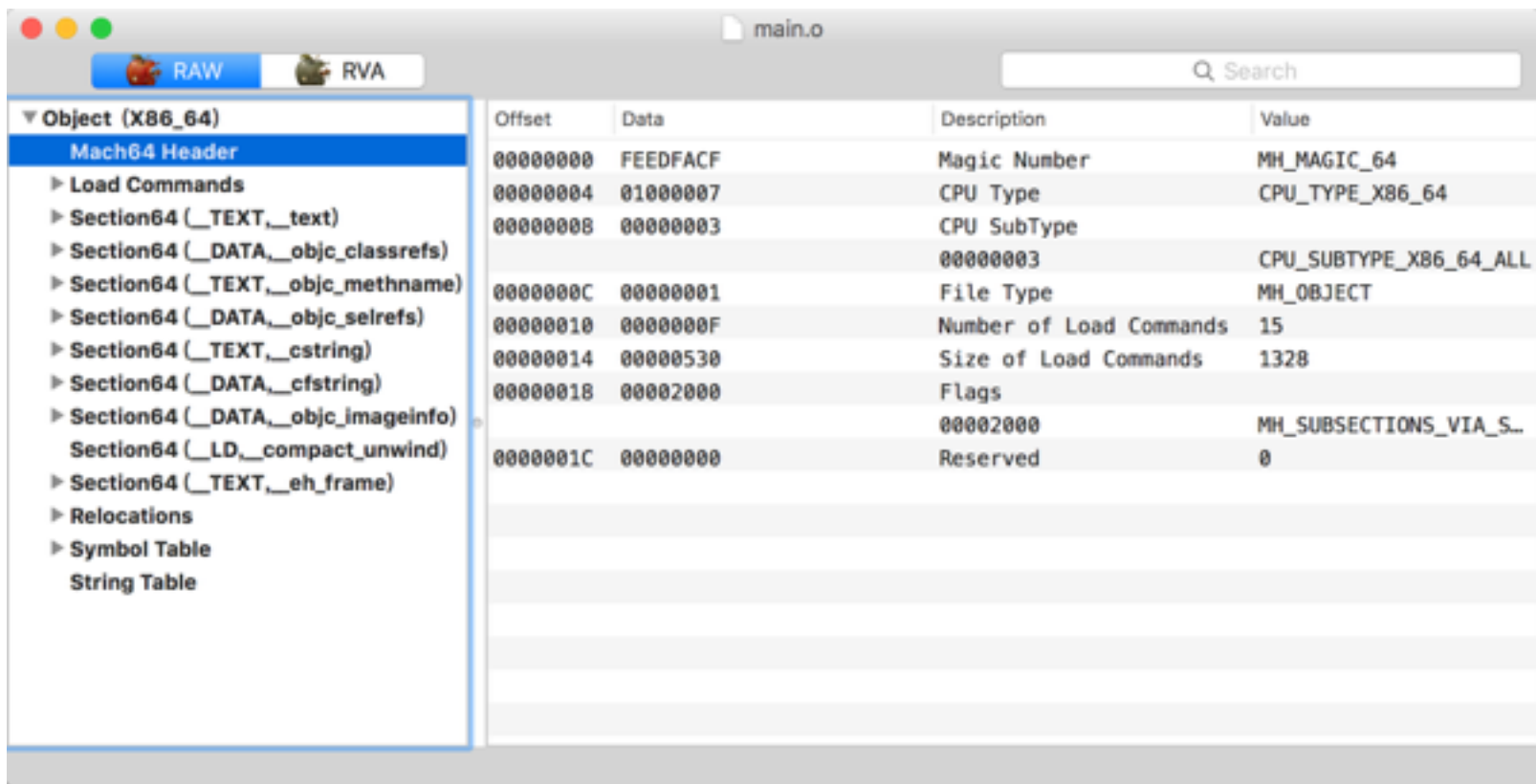
Assemble - 生成 Target 相关汇编

```
$clang -S -fobjc-arc main.m -o main.s
```

```
_main:                                     ## @main
    .cfi_startproc
## BB#0:
    pushq   %rbp
Ltmp0:
    .cfi_def_cfa_offset 16
Ltmp1:
    .cfi_offset %rbp, -16
    movq    %rsp, %rbp
Ltmp2:
    .cfi_def_cfa_register %rbp
    subq   $32, %rsp
    movl   $0, -4(%rbp)
    callq  _objc_autoreleasePoolPush
    movq   L_OBJC_CLASSLIST_REFERENCES_$(%rip), %rcx
    movq   L_OBJC_SELECTOR_REFERENCES_(%rip), %rsi
    movq   %rcx, %rdi
    movq   %rax, -24(%rbp)          ## 8-byte Spill
    callq  _objc_msgSend
    leaq   L__unnamed_cfstring_(%rip), %rcx
    movq   %rax, -16(%rbp)
    movq   -16(%rbp), %rsi
    movq   %rcx, %rdi
    movb   $0, %al
    callq  _NSLog
    leaq   -16(%rbp), %rdi
    xorl   %edx, %edx
    movl   %edx, %esi
    callq  _objc_storeStrong
    movq   -24(%rbp), %rdi          ## 8-byte Reload
    callq  _objc_autoreleasePoolPop
    xorl   %eax, %eax
    addq   $32, %rsp
    popq   %rbp
    retq
```


Assemble - 生成 Target 相关 Object (Mach-O)

```
$clang -fmodules -c main.m -o main.o
```



The screenshot shows a Mach-O object file viewer for 'main.o'. The interface has a 'RAW' button selected and a search bar. The left sidebar shows a tree view of the object's structure, with 'Mach64 Header' expanded. The main area displays a table of header fields.

Offset	Data	Description	Value
00000000	FEEDFACF	Magic Number	MH_MAGIC_64
00000004	01000007	CPU Type	CPU_TYPE_X86_64
00000008	00000003	CPU SubType	CPU_SUBTYPE_X86_64_ALL
0000000C	00000001	File Type	MH_OBJECT
00000010	0000000F	Number of Load Commands	15
00000014	00000530	Size of Load Commands	1328
00000018	00002000	Flags	MH_SUBSECTIONS_VIA_SYMBOLS
0000001C	00000000	Reserved	0

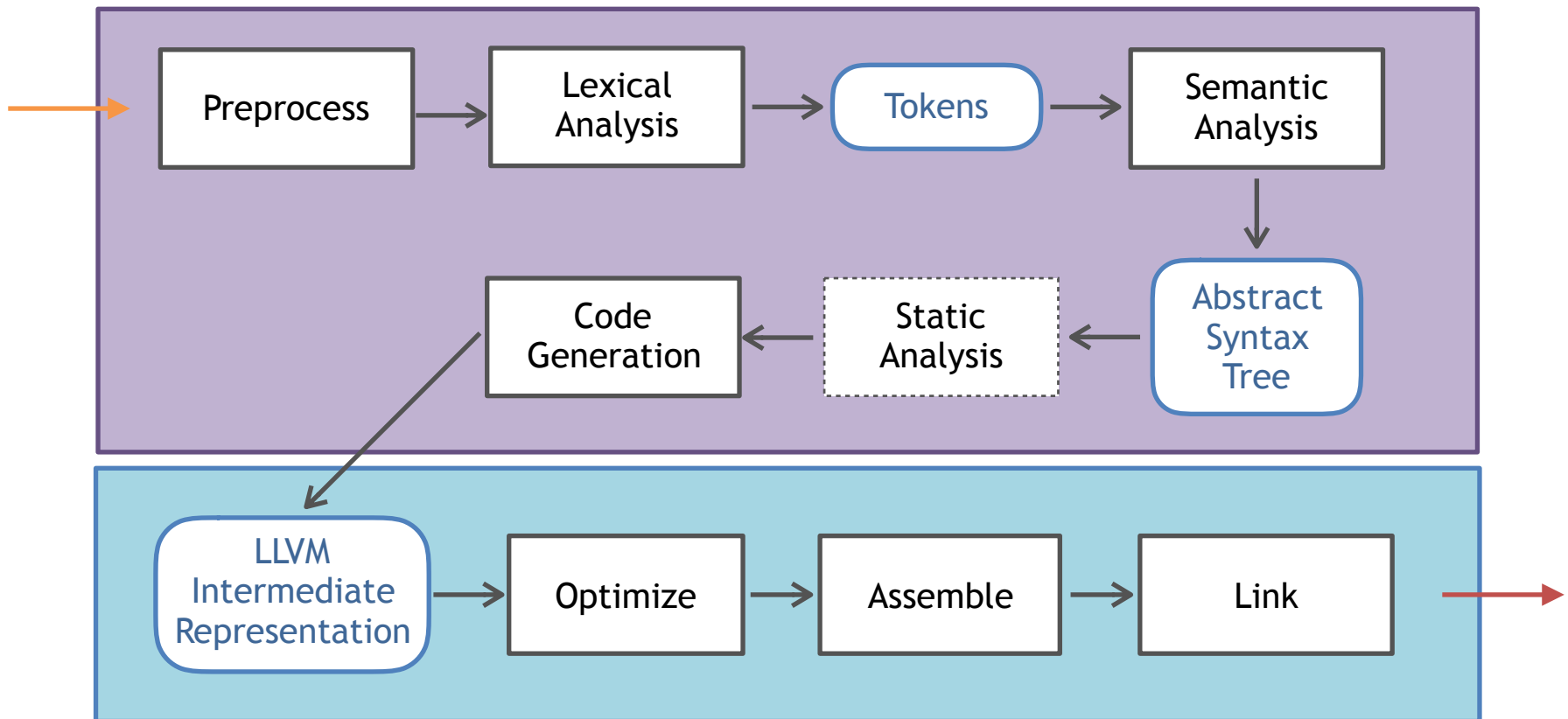
Link 生成 Executable




```
$clang main.m -o main
```

```
$/main
```

```
main[13595:2214602] Hello world: <NSObject: 0x7f9b01506700>
```

总结：Clang-LLVM 下，一个源文件的编译过程



-  Apple 编译器 Clang-LLVM 架构初识
-  你的源码是如何一步步成为可执行文件的？
-  我们能用 Clang 做什么有意思的事情？

我们能在 Clang 上做什么？

- LibClang
- LibTooling
- ClangPlugin

LibClang

- 😊 C API 来访问 Clang 的上层能力，如获取 Tokens、遍历语法树、代码补全、获取诊断信息
- 😊 API 稳定，不受 Clang 源码更新影响
- 😡 只有上层的语法树访问，不能获取到全部信息

LibClang - 如何使用

- 使用原始 C API
- 脚本语言：使用官方提供的 python binding 或开源的 node-js / ruby binding
- Objective-C：开源库 ClangKit

LibClang - Demo

```
@interface Sark : NSObject

@property (nonatomic, strong) id password;
@property (nonatomic, strong) id nickName;
@property (nonatomic, strong) id netWorking;
@property (nonatomic, strong) id suuny;
@property (nonatomic, strong) id backgrond;

@end
```



用 LibClang 的 Python Binding 实现一个 Property Name Linter

LibClang - Demo

```
import enchant, difflib
from clang.cindex import Index

if __name__ == '__main__':
    index = clang.cindex.Index.create()
    tu = index.parse(sys.argv[1])
    d = enchant.Dict("en_US")
    for c in tu.cursor.walk_preorder():
        if c and c.spelling:
            if (c.kind == clang.cindex.CursorKind.OBJC_PROPERTY_DECL):
                if (not d.check(c.spelling)):
                    best = None
                    best_ratio = 0
                    suggestions = set(d.suggest(c.spelling))
                    for sug in suggestions:
                        tmp = difflib.SequenceMatcher(None, c.spelling.lower(), sug).ratio()
                        if tmp > best_ratio:
                            best = sug
                            best_ratio = tmp
                    print "typo: " + c.spelling + ", do you mean: " + best + "?";
```

LibClang - Demo

```
$python property-linter.py main.m
```

```
typo: passWord, do you mean: password?  
typo: nickName, do you mean: nickname?  
typo: netWorking, do you mean: networking?  
typo: suuny, do you mean: sunny?  
typo: backgrond, do you mean: background?
```

LibTooling

- 😊 对语法树有完全的控制权
- 😊 可作为一个 **standalone** 命令单独的使用，如 **clang-format**
- 😡 需要使用 C++ 且对 Clang 源码熟悉

LibTooling - Demo

```
@interface Sark : NSObject
@property (nonatomic, copy) NSString *name;
- (void)becomeGay;
@end
```

实现一个简易 Objective-C -> Swift 源码转换器

LibTooling - Demo

```
| -ObjCInterfaceDecl 0x7ff94185dca0 <line:7:1, line:10:2> line:7:12 Sark
| | -super ObjCInterface 0x7ff9411a4608 'NSObject'
| | -ObjCPropertyDecl 0x7ff9411a24e0 <line:8:1, col:39> col:39 name 'NSString
*' readwrite copy nonatomic
| | -ObjCMethodDecl 0x7ff9411a2600 <line:9:1, col:18> col:1 - becomeGay
'void'
| | -ObjCMethodDecl 0x7ff9411a2688 <line:8:39> col:39 implicit - name
'NSString *'
| ` -ObjCMethodDecl 0x7ff9411a2710 <col:39> col:39 implicit - setName: 'void'
|   ` -ParmVarDecl 0x7ff9411a2798 <col:39> col:39 name 'NSString *'
```

创建 RecursiveASTVisitor, 在 AST 中重写感兴趣节点的 Visit 方法

LibTooling - Demo

```
$objc2swift test.m -- -fsyntax-only -fmodules
```

```
class Sark: NSObject {  
    var name: NSString?  
    func becomeGay() {  
    }  
}
```

ClangPlugin

- 😊 对语法树有完全的控制权
- 😊 作为插件注入到编译流程中，可以影响 **build** 和决定编译过程
- 😡 需要使用 C++ 且对 Clang 源码熟悉

ClangPlugin - Demo

```
14
15 @interface AppDelegate : UIResponder <UIApplicationDelegate>
16                                     ⚠️ 缺少 Objective-C 类名前缀
17 @property (strong, nonatomic) UIWindow *window;
18
19 @end
20
21
```

可以嵌入 Xcode 的 Linter, 提供可识别的诊断信息

ClangPlugin - Demo

```
bool VisitObjCInterfaceDecl(clang::ObjCInterfaceDecl *D) {
    const clang::SourceManager &SM = Context->getSourceManager();
    clang::FullSourceLoc loc = Context->getFullLoc(D->getLocStart());
    if (!SM.isInSystemHeader(loc)) {
        std::string name = D->getName();
        clang::DiagnosticsEngine &DE = *Diagnostics;
        if (std::islower(name[0]) || std::islower(name[1])) {
            unsigned int id =
DE.getCustomDiagID(clang::DiagnosticsEngine::Warning, "缺少 Objective-C
类名前缀");
                DE.Report(loc.getLocWithOffset(11), id);
            }
        }
    }
    return true;
}
```

ClangPlugin - Demo

```
24 @implementation ViewController
25
26 - (void)viewDidLoad {
27     [super viewDidLoad];
28     sunny:
29     goto sunny;
30 }
31
32
33 - (void)didReceiveMemoryWarning {
34     [super didReceiveMemoryWarning];
```

很遗憾，你的代码不符合公司价值观

ClangPlugin - Demo

```
24 @implementation ViewController
25
26 - (void)viewDidLoad {
27     [super viewDidLoad];
28     sunny:
29     goto HR 处办理离职手续 sunny;
30 }
```

很遗憾，你的代码不符合公司价值观




很遗憾，你的代码不符合公司价值观

Fix-it Replace "goto" with "请到 HR..."

```
33 - (void)didReceiveMemoryWarning {
34     [super didReceiveMemoryWarning];
35     // Dispose of any resources that can be recreated.
36 }
```

Like a Boss



-  Apple 编译器 Clang-LLVM 架构初识
-  你的源码是如何一步步成为可执行文件的？
-  我们能用 Clang 做什么有意思的事情？

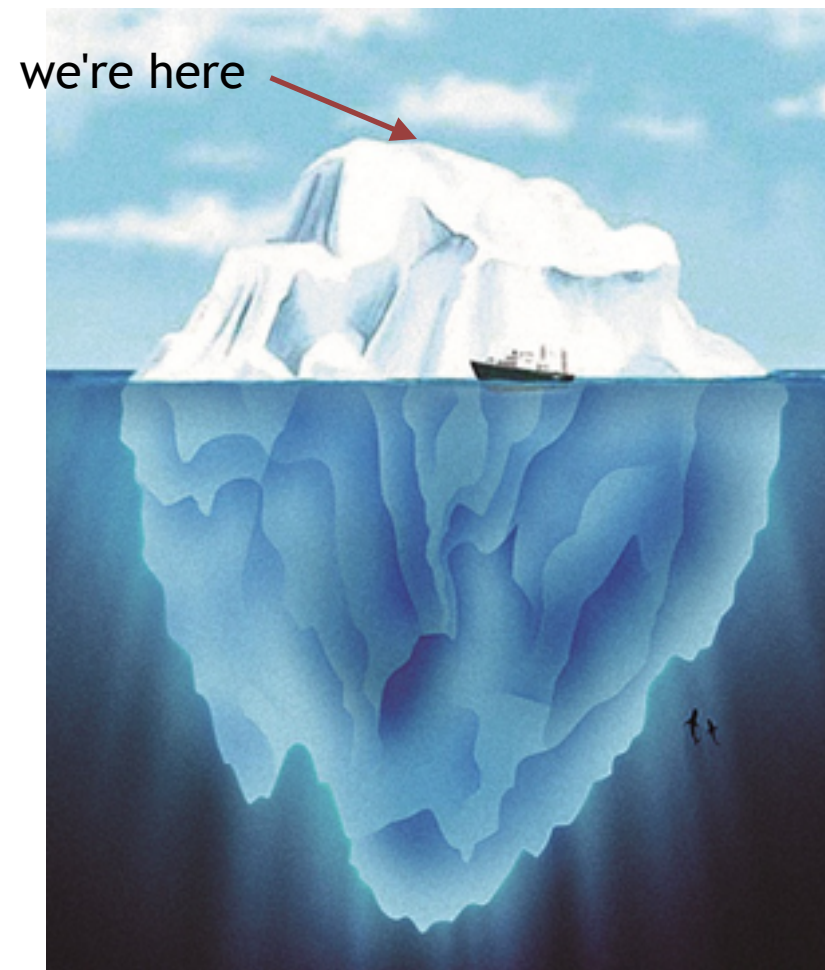
Clang-LLVM 相关资料

- <http://clang.llvm.org/docs/index.html>
- <http://blog.llvm.org/>
- <https://www.objc.io/issues/6-build-tools/compiler/>
- <http://llvm.org/docs/tutorial/index.html>
- <https://github.com/loarabia/Clang-tutorial>
- <http://lowlevelbits.org/getting-started-with-llvm/clang-on-os-x/>
- <https://kevinaboos.wordpress.com/2013/07/23/clang-tutorial-part-i-introduction/>
- <http://szelei.me/code-generator/>
- 《Getting Started with LLVM Core Libraries》
- 《LLVM Cookbook》

- 问：编译器可以编译程序，但编译器本身也是个程序，那它一定是由更早的编译器编译而成的，那... 最早的一个编译器是哪儿来的？

手写机器码？





Q



A



我就叫Sunny怎么了
扫一扫二维码图案，关注我吧