



从 Haskell 到 WebAssembly (2)



Felis sap... 

函数式编程、编程语言、编程 话题的优秀回答者

已关注

开源哥、圆角骑士魔理沙、霜月琉璃、阅千人而惜知己、祖与占等 59 人赞了该文章

GHC 的编译管线和插件机制

上期讲了 WebAssembly 跑起来和写起来是什么感觉。这期讲讲怎么把 Haskell 代码变成 WebAssembly 代码。首先要设法从 GHC 的编译管线里面把某个 IR 捞出来（而且要 in-memory 的形式，要是 dump 到文本再解析那就实在太。。）

这里我们先只关心单个 Haskell module 的 pipeline。在 GHC 进程启动以后，首先会解析 major mode，如果 major mode 对应 "ghc --make" 或者 "ghc -c" 等编译 Haskell 代码的模式，经过依赖分析以后，会对每一个 Haskell module 启动一个 pipeline，相关代码在 GHC API 的 DriverPipeline 模块中。

在 pipeline 中，Haskell 源代码经过预处理器展开 CPP 宏以后，经历的中间表示有：

1. Parsed module。保留了所有的源代码语法特性的 AST。
2. Renamed module。解析所有 identifier 的作用域，诸如 lambda 参数之类的 identifier 重命名到全局唯一，从模块导出的 identifier 带上模块前缀。从这一步开始，GHC 要求跑一个模块的 pipeline 之前，其依赖模块的 pipeline 必须已经跑过，并且成功通过类型检查，将相关信息存入其 interface file 以供查询。
3. Typechecked module。类型检查过的模块。类型检查一通过，可以生成 interface file 了，其他依赖该模块的模块在编译时需要查询 interface file 里面的各种信息。值得一提的是，前面这几个中间表示共用一套 AST，至于如何区分诸如 identifier 等细节的类型差异？依靠 Trees that Grow 设计模式。
4. Core。这是一个极小的带类型 lambda calculus，基于某个 System F 变种。大多数优化都是在 Core 上跑的带类型优化。Core 是有形式化 spec，并且保证 Core 类型不出错，运行时就不



所有的 thunk allocation 和 evaluation 的时机直接由 term 的形状决定。大多数 Haskell to JavaScript 编译器 (haste 和 ghcjs) 选择用 STG 作为编译到 JavaScript 的中间表示。

6. Cmm。一个长得勉强有点像 C (毕竟叫 C-minus-minus) 的玩意，作为各种不同平台代码生成器共用的中间表示，设计目的是“平台无关汇编”。当然，GHC 的年代比 LLVM 年代早多了所以用这个，要是 LLVM 火了以后再開坑，估计就直接用 LLVM IR 了。
7. 默认的汇编后端，生成 gcc 能识别的 .asm 文本，然后 gcc 编译成 .o 文件。LLVM 后端会生成 LLVM IR 文本，调用 llc/opt 编译。C 后端直接生成 C 代码编译。

我们要从 Haskell 编译到 WebAssembly，自然要挑一个接近底层的 IR，这里我们选择 Cmm，从 Cmm 开始做代码生成，大致相当于开发一个新架构的原生后端。（选其他的也不是不行，我跟同事讨论时开玩笑说能不能做个新后端把 x64 assembly 编译到 WebAssembly，他说你还是直接用 Emscripten 编译个 qemu 算了。。）

接下来的任务是：设法劫持正常的 GHC pipeline，加载我们自定义的逻辑，获得 in-memory IR 并进行我们自己的代码生成。如果你是个想要写函数式编译器来玩玩的萌新，觉得拿 GHC 搞有戏的话，听我一句劝，转去弄 Idris 后端还来得及。还不听劝？好吧，你大致会经历几个阶段：

1. 看了一些 GHC API 的文档和博客啥的，觉得有搞头。写了一个能编译单个 .hs 的 demo。
2. 能编译自己的 .hs 模块了，这很 OK，然后你如果想成为新世界的卡密，不对，写面向新架构的编译器的话，你需要把整个 Haskell 标准库，连带整个运行时都能编译过去。这个过程叫“booting”。但是你的小儿科 demo 没有办法编译标准库，因为标准库的编译流程相当复杂，一堆涉及 autoconf, sed 之类的乱七八糟的脚本。。。
3. 经过 n 个小时以后，你发现 ezyang 的某篇 [blog](#) 提到，GHC 有个叫 [frontend plugin](#) 的机制，可以用来制造一个 ghc wrapper，这个 ghc wrapper 用于替代正常的 ghc，执行自定义逻辑。很好，你写了一个 ghc frontend plugin，然后额外花了 n 个小时，处理诸如 cabal/stack 支持、package database 相关参数等琐碎问题，总算能把 ghc wrapper 跑起来了。
4. ghc wrapper 里面可以实现自定义逻辑，但是手头还是没有结构化的 IR，咋整？如果需要的是 Typechecked module 或者 Core 那都比较好办，不过 STG 和 Cmm 是没有暴露出相关表示的，在 pipeline 里面一个 doCodegen 一把梭，直接从 desugared Core 搞到汇编了。你又花了 n 小时，找到了 GHC 的 Hooks 机制，可以用来对 GHC 运行中的一些函数下钩子，其中有个 runPhaseHook 可以把整个 pipeline 换掉！你有了一个大胆的想法。
5. 又过了 n 小时，你的自定义 pipeline 上线了，可喜可贺。

总的来说，为了最大化与 Cabal 的兼容性，我们需要假装自己是 ghc，也完成 ghc 本身的编译任务生成 native code，但是可以夹带私货干点别的（配置和输入输出不能污染 process args/stdout 什么的，但是可以用环境变量）。

跟 GHC API 打交道相当不愉快，you've been warned。不过我靠对 Haskell 的爱扛下来了。

ghc-toolkit: Haskell to X 编译器的可重用框架



ghc-toolkit, 实现了这个封装。

使用 ghc-toolkit 大致只需要关心以下几个东西：

```
data HaskellIR = HaskellIR
  { parsed :: HsParsedModule
  , typeChecked :: TcGblEnv
  , core :: CgGuts
  , stg :: [StgTopBinding]
  , cmm :: [CmmDecl]
  , cmmRaw :: [RawCmmDecl]
  }

data CmmIR = CmmIR
  { cmm :: [CmmDecl]
  , cmmRaw :: [RawCmmDecl]
  }

data Compiler = Compiler
  { patch :: ModSummary -> HsParsedModule -> Hsc HsParsedModule
  , withHaskellIR :: ModSummary -> HaskellIR -> CompPipeline ()
  , withCmmIR :: CmmIR -> CompPipeline ()
  , finalize :: Ghc ()
  }

makeFrontendPlugin :: Ghc Compiler -> FrontendPlugin

data FakeGHCOptions = FakeGHCOptions
  { ghc, ghcLibDir :: FilePath
  , frontendPlugin :: GHC.FrontendPlugin
  }

fakeGHCMain :: FakeGHCOptions -> IO ()
```

解释一下。前面两个数据类型是 GHC 在编译 .hs/.cmm 时分别会获得的 IR 的集合。假如想要写一个基于 GHC 的 Haskell to X 编译器，那么把自定义的逻辑放在 Compiler 类型里面就行了——Hsc、CompPipeline、Ghc 等这几个不同 GHC phase 对应的 monad 都是 MonadIO instance，所以也支持插入任意副作用（比如读写你自己的 object file 之类的玩意）。每当 GHC 对一个模块启动 pipeline 时，会触发 withHaskellIR/withCmmIR 回调函数，执行你自己自定义的逻辑，其中你可以拿到当前模块的模块信息以及各种 IR。

为了把 Compiler 类型用起来，首先用 makeFrontendPlugin，可以把 Compiler 转换成一个 GHC 的 frontend plugin（Compiler 是在 Ghc monad 里面初始化的，按前面说的，初始化的逻辑



wrapper executable，这个 ghc wrapper executable 的行为大多数时候和 ghc 一致，不过用 --make 编译 Haskell 模块时，会自动加载 Compiler 中的各种回调函数。

简而言之，使用 ghc-toolkit，用上面的机制搞出一个假的 ghc，然后编译标准库或者自己的模块时，用这个假的 ghc 直接代替 ghc，即可把自己的 Haskell 编译器给跑起来了。可喜可贺。

最后值得一提的是，Compiler 里面额外提供了一个 patch 回调函数，这个玩意可以用来重写 parsed IR（实现重写其他 IR 的回调函数很容易，但是其他 IR 很容易改坏了，自己改要么把 GHC 给 crash 掉，要么编译出什么不可名状的玩意）。这个重写机制可以拿来干很多很有意思的事，比如给所有函数打个 INLINEABLE 标记，让 GHC 自动变成全局优化编译器（逃。GHC 8.6 已经加了 source plugin 功能，可以用 GHC Plugin 重写 parsed/renamed/typechecked IR，原理和这个基本上一样。

这期先讲这么多。感觉都是 GHC 相关的 hack，下一期就有 WebAssembly 了。

发布于 2018-08-10

「真诚赞赏，手留余香」

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

WebAssembly

Haskell

函数式编程

▲ 赞同 59



● 3 条评论

➤ 分享

★ 收藏



文章被以下专栏收录



不动点高校现充部

一切与编程语言理论、函数式编程相关的杂谈。

已关注



雾雨魔法店

<http://zhuanlan.zhihu.com/marisa/20419321>

已关注



从 Haskell 到
WebAssembly (1)

felis... 发表于不动点高校...



WebAssembly 系列 (六)
WebAssembly 的现在与未来

胡子大哈 发表于前端大哈

今天一大早 发现Google V8的core developer来BuckleScript 提 issue: Int64 compilation duplicates effects · Issue #1154 · bloomberg/bucklescript 很好奇 Google再用BuckleScript干什么...

张宏波

Web
什么
胡子

3 条评论

切换为时间排序

写下你的评论...



赵康

1 个月前

原谅我是看封面小图进来的.....

赞



neo lin

1 个月前

智猫姐姐不卖萌了?你的人设呢?

赞



Felis sapiens (作者) 回复 neo lin

1 个月前

卖不动了, 被榨干了

2 查看对话

