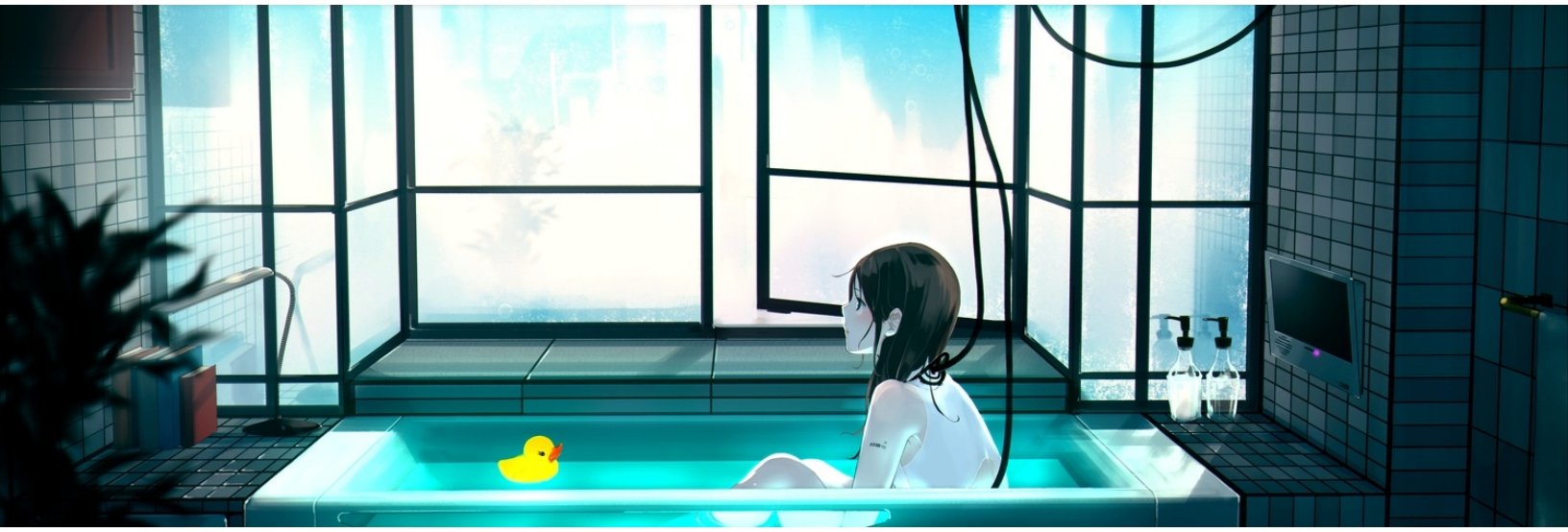

Haskell 编译期的字面量注入

Felis sap...
函数式编程、编程语言、编程 话题的优秀回答者
已关注

需求很简单：在 Haskell 编译期进行特定的计算，将计算结果注入到某个 Haskell 字面量，在其他模块中可以直接使用。这个计算过程可能相当耗时，一般来讲 ghc inliner 不会尝试编译期展开，使用注入可以强行将运行时开销转移到编译期；也有可能带有副作用，比如读写文件系统、环境变量等，这样适合用于在编译期一次性注入某些配置（比如当前项目的 git revision），而且注入不同配置无需修改 Haskell 代码。

进行编译期计算，最简单的机制就是 Template Haskell 了。ghc 自带的 template-haskell 库提供了一套 Haskell AST 定义，可以用 Haskell 实现生成 AST 的代码，这些代码在编译期由 ghc 执行。具体到类型上（参考文档 Language.Haskell.TH.Syntax）：

- 所有的宏都在 Q monad 上定义。可以认为 Q monad 是 Template Haskell 编译期运行的 IO monad，里面的确可以 lift 任意的 IO action，所以可以写一个从某个 http server 下载一段代码然后编译的宏；跟普通 IO 相比 Q 额外维护了一些编译器状态，可以用于诸如查询类型信息、生成新标识符等功能；
- 编译期可以运行的宏类型有 Q Exp、Q [Dec]、Q Type、Q Pat，分别代表生成表达式、声明、类型和模式。具体到字面量注入这个应用，我们只需要 Q Exp；
- 定义宏和使用宏的模块要求分开。在使用宏的模块中，启用 TemplateHaskell 扩展，然后可以使用 $(q_exp) 语法的表达式，q_exp 类型为 Q Exp

看到这里，最简单的字面量注入方法呼之欲出：

```haskell
naive_fib :: Integer -> Integer
naive_fib 0 = 0
naive_fib 1 = 1
```

```
fib_runner :: Integer -> Q Exp
fib_runner n = pure (LitE (IntegerL (naive_fib n)))
-- in another module, with TemplateHaskell extension on
fib_1024 :: Integer
fib_1024 = $(fib_runner 1024)
```

现在，编译带 fib_1024 的模块，ghc 会忠实地计算 naive_fib 1024，将计算结果作为 Integer literal 注入到 fib_1024。假设这个调用真的在宇宙热寂之前跑完了，届时我们获得的编译产物——不论是 ghci 的 bytecode 还是 ghc 的 native code，里面的 fib_1024 都一定绑定到了计算好的结果。

S-表达式爱好者们大概不会喜欢这种写法。最好让构造 AST 的代码和真的 AST 看上去没什么两样。那么可以用 Template Haskell 自带的 QuasiQuotes：

```
fib_runner :: Integer -> Q Exp
fib_runner n = [| naive_fib n |]
```

这个 [| |] 围住的代码可以用普通的 Haskell 表达式语法来写。整个 Oxford bracket 返回的类型是 Q Exp，不过 Template Haskell 提供了其他几个 quoter 来生成其他东西，比如 [t| |] 类型是 Q Type，里面也可以用普通的 Haskell 类型语法。这样一来，Template Haskell 的这几个内置 quoter 相当于 Haskell 的 parser 接口，可以用于将一段代码解析成 AST，在 Q monad 中可以对 AST 进行进一步分析和处理，应用很广泛。

前面的使用 QuasiQuote 版本的 fib_runner 实际上有点小问题——它跟第一个版本的效果并不等价。quote 展开以后的效果，实际上相当于

```
fib_runner :: Integer -> Q Exp
fib_runner n = pure (AppE (VarE 'naive_fib) (LitE (IntegerL n)))
```

这不就相当于 fib_1024 = naive_fib 1024 吗？完全不会起编译期缓存的作用。所以 quoter 的写法需要稍加小心：

```
fib_runner :: Integer -> Q Exp
fib_runner n = let result = naive_fib in [| result |]
```

以保证 quote 展开之后是将编译期计算结果注入到目标位置，而非直接插入表示计算过程的表达式。

这里注入的计算结果类型相当简单。如果是一个复杂的类型，比如用户自定义的 ADT 呢？

```
data BinTree a = Tip | Branch a (BinTree a) (BinTree a)
liftTree :: BinTree a -> Q Exp
```

针对上面的例子：怎样将一个 BinTree a 类型的值注入到字面量中？liftTree 的实现思路大致是：对 bt 进行模式匹配，自顶向下地将其转化为一系列 AppE 调用。如果我们直接使用前面的 QuasiQuote 语法实现 liftTree，ghc 会提示我们未实现 BinTree a 的 Lift 实例。所以 Lift 这个 type class 的作用体现出来了：对某个类型的值进行分析，转化成能够生成该值的 Template Haskell AST。Lift 类型类可以自动 derive：

```
{-# LANGUAGE DeriveLift, StandaloneDeriving #-}
deriving instance Lift a => Lift (BinTree a)
liftTree :: Lift a => BinTree a -> Q Exp
liftTree = lift
```

很多时候，我们希望 Lift 的一个类型，并没有提供 Lift 实例，我们可以用类似的 Standalone Deriving 语法生成 orphan instance 来做 lift，当然，前提是当前 scope 里 ghc 能够访问到该类型的全部定义。如果想避免 orphan instance，可以手动用 newtype 包装一下，只生成 newtype 的 Lift 实例。

假如希望注入的类型，作者既没有提供 Lift 实例，也没有 export 完整定义，难道万事皆休？

首先，我们看看 Template Haskell 从理论上来说，能够拿到哪些信息，拿不到哪些信息：每个 module 的编译产物包含一个 interface file（也就是 .hi 文件，对应 ghc 内部的 ModIface 类型）和 object file（就是包含 native code 的那种），这两者肯定是能读的。object file 恢复不

Template Haskell 理论上可以：

- 读取 inlineable 函数的 RHS（right-hand side）
- 读取任意类型的完整定义

实际上前者如果使用 ghc api 强行 hack，确实可以办到，篇幅所限不在这里展示；后者可以使用 Template Haskell 中的 reify 接口，首先构造一个类型的 Name，接下来可以使用 Name 召唤这个类型的全部定义（哪怕被隐藏），包括每一个 Constructor，然后这些 Constructor 就可以用于模式匹配，实现 Lift 之类的 type class 的实例了。这个召唤术我在之前的 Speak my true name, summon my constructor 一文中简单介绍过。

不过，召唤出 Constructor 以后，手写 instance 还是一件很无聊的事，能不能避免呢？使用 th-lift: Derive Template Haskell's Lift class for datatypes.，即可使用 Template Haskell，自动为 datatype 生成 Lift 实例。

以上，是打破规矩的第一种方法。但是为什么注入一个编译期字面量，就一定要实现 Lift 实例呢？第二种方法简单粗暴得多——

```
import Data.Binary
import qualified Data.ByteString.Lazy as LBS
import qualified Data.ByteString.Unsafe as BS
import Language.Haskell.TH.Syntax
import System.IO.Unsafe

liftLazyByteString :: LBS.ByteString -> Exp
liftLazyByteString lbs =
 AppE
   (VarE 'LBS.fromStrict)
   (AppE
      (VarE 'unsafePerformIO)
      (AppE
         (AppE
            (VarE 'BS.unsafePackAddressLen)
            (LitE (IntegerL (fromIntegral (LBS.length lbs)))))
         (LitE (StringPrimL (LBS.unpack lbs)))))

liftTyped :: Binary a => a -> Q Type -> Q (TExp a)
liftTyped x qt = do
 t <- qt
 unsafeTExpCoerce $
   pure $
   SigE (AppE (VarE 'Data.Binary.decode) (liftLazyByteString $ encode x)) t
```

common data types. 库提供了 Strict/Lazy ByteString 的 Lift 实例，但是那个实例实现是转换成 [Word8] 然后直接用 list 的 Lift 实例，也就是翻译成一系列 Cons 的 apply，相当低效。这里 liftLazyByteString 实现的注入可以保证整个 ByteString 的内容成为 object file 中 data 字段的一整块。实现原理是先构造出一个 Addr# 代表编译后 data 字段中 raw string 的地址，然后使用 bytestring 库的 unsafePackAddressLen 将其包装为一个 ByteString，这个包装过程是 O(1) 的，最后套一层 unsafePerformIO 使最后生成的 Exp 类型对应一个纯的 ByteString，这里不涉及任何可观测的副作用，所以可以使用。

既然能够注入任意 ByteString，那么注入任意可序列化的类型就是顺理成章的事，而支持序列化的类型远多于支持 Lift 的类型。liftTyped 会将待注入的值序列化，然后将“从某个（编译期已确定的）ByteString 反序列化”的逻辑注入到目标表达式。注意这个宏生成的类型是 TExp a 而非 Exp，TExp 是 typed Template Haskell expression，比原本的 Exp 类型增加了一些类型安全性，在 quote 时使用语法是 $$(typed_splice)，区分原来的 $(untyped_splice)。

这里的序列化使用的是 ghc 自带库 binary 提供的 Binary class，使用其他的序列化库，如 cereal、store 等亦可。如果某类型作者没有导出定义也没有提供任何序列化器，那么在该类型满足一定限制（不含可变引用、不含函数闭包）的情况下，可以使用 ghc 8.2 提供的 Compact Region 功能，召唤一个万能序列化器（比如这里的实现：Data.Compact.Serialize）

这个简单的技巧，个人认为优于强行打破 module 壁障、实现 Lift 实例的。从空间占用而言，需要注入的数据规模较大时，序列化后注入的表示明显占用空间可以更小（如果使用 zlib 然后注入压缩版的 ByteString，或者直接用 UPX 压缩可执行文件，还可以进一步缩小）；另外可以避免 orphan instance，以及强行实现 Lift instance 的不变；从依赖的角度而言，也不需要引入 ghc 自带库以外的依赖。

之所以想到做编译期注入字面量的功能，最近的一个 use case 就是需要在 Haskell 代码里查询自己是怎样被 build 的：包括 ghc 路径和版本、Cabal 配置参数、本 package 的名字，等等，不一而足，而且这个功能需要单独做成一个库，别的库只要依赖它就可以查询自己的 build 信息。思路是先在 Cabal 的 Setup.hs 脚本里用 Cabal 提供的 Hook 机制（之前提过，见 使用 Cabal hook 构建复杂 Haskell 项目）把 configure 之后确定下来的配置序列化存下来，然后在编译期读取存下来的配置并注入到字面量中。Cabal 的很多类型没法实现 Lift instance 这一点很烦，不过等人来吃火锅的途中总算灵光一闪，想到了那个简单的技巧并做出来了。（自动生成的 haddock 文档参考 Language.Haskell.GHC.Alter.BuildInfo.TypedSplices，这些都是 Haskell 编译期可以查询到的 build 信息）

后记：

9月到了。

编辑于 2017-09-02

赞赏

还没有人赞赏，快来当第一个赞赏的人吧！

Haskell GHC（编程套件） 函数式编程

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

推荐阅读

Notes on Haskell Debugging

elis... 发表于不动点高校...

Haskell开发环境配置

寸光寸阴 发表于Haske...

幻想中的Haskell - Compiling Combinator

圆角骑士魔... 发表于雾雨魔法店

在H typ

miro

3 条评论

切换为时间排序

写下你的评论...

dram 1 年前

序列化后注入的是不是可以 link 一个进去然后用 ffi 会舒服点……

赞

Felis sapiens (作者) 回复 dram 1 年前

杨博 1 年前

类型推断好复杂

赞