Основы Template Haskell

Октябрь 2012 г.

Описание

- Compile-time meta-programming
- Разработан SPJ и Tim Sheard в 2002-ом году (MSR)
- Реализован в GHC с версии 6 (и только в нём)
- Ближайшие аналоги
  - LISP (runtime)
  - Python compiler/ast модули (runtime)
- Похожие инструменты
  - Препроцессоры (cpp, perl source filters, etc.)
  - eval

Преимущества

- Позволяет делать вещи, которые не реализовать другими средствами (DSL)
- Помогает избавиться от boilerplate-кода (auto derives)

Синтаксис

- Прагма -XTemplateHaskell
- Language.Haskell.TH, Language.Haskell.TH.Syntax
- Q Monad
- Name

```
Prelude Language.Haskell.TH> :t 'True
'True :: Name
Prelude Language.Haskell.TH> 'True
GHC.Types.True
Prelude Language.Haskell.TH> ''Maybe
Data.Maybe.Maybe
```

- $splice

```
$(deriveShow ''Test) :: Q [Dec]
$(tuple 10 "test") :: Q Exp
$(generateType 'Int 'Bool) :: Q Type
```

Quotation

- [| ... |], [e| ... |] :: Q Exp

```
[e| 1 + 2 + 3 |]
```

```
Prelude Language.Haskell.TH> runQ [| 1 + 2 + 3|]
InfixE (Just (InfixE (Just (LitE (IntegerL 1)))
(VarE GHC.Num.+) (Just (LitE (IntegerL 2)))))
(VarE GHC.Num.+) (Just (LitE (IntegerL 3)))
```

- [d| ... |] :: Q [Dec]

```
[d| const a b = a |]
```

- [t| ... |] :: Q Type

```
[t| Int -> String -> Bool |]
```

- [p| ... |] :: Q Pat

```
[p| (_, []) |]
```

Quasi-quotation

- GHC 6.10+
- Прагма -XQuasiQuotes
- `[quoter| string |]`
- ```
  data QuasiQuoter = QuasiQuoter
      { quoteExp  :: String -> Q Exp
      , quotePat  :: String -> Q Pat
      , quoteType :: String -> Q Type
      , quoteDec  :: String -> Q [Dec]
      }
  ```
- ```
  [persist|
      Person
          name String
          age Int
          deriving Show
      Car
          color String
          make String
          model String
          deriving Show
  |]
  ```

Пример 1

```
{-# LANGUAGE TemplateHaskell #-}

import TH1

a = $(compile [| 7 * 3 |])
b = $(compile [| 7 * 9 + 3 |])
c = $(compile [| 5 * 8 + 6 * 3 |])

main :: IO ()
main = return ()

--------------------------------------
% runghc -ddump-splices Main1.hs

InfixE (Just (LitE (IntegerL 7))) (VarE GHC.Num.*) (Just (LitE (IntegerL 3)))

Main.hs:5:7-25: Splicing expression
    compile [| 7 * 3 |] ======> 21

InfixE (Just (InfixE (Just (LitE (IntegerL 7))) (VarE GHC.Num.*) (Just (LitE (IntegerL 9)))))
(VarE GHC.Num.+) (Just (LitE (IntegerL 3)))

Main.hs:6:7-29: Splicing expression
    compile [| 7 * 9 + 3 |] ======> 66

InfixE (Just (InfixE (Just (LitE (IntegerL 5))) (VarE GHC.Num.*) (Just (LitE (IntegerL 8)))))
(VarE GHC.Num.+)
(Just (InfixE (Just (LitE (IntegerL 6))) (VarE GHC.Num.*) (Just (LitE (IntegerL 3)))))

Main.hs:7:7-33: Splicing expression
    compile [| 5 * 8 + 6 * 3 |] ======> 58
```

Пример 1 (код)

```haskell
{-# LANGUAGE TemplateHaskell #-}

module TH1 where

import Debug.Trace (trace)
import Language.Haskell.TH

compile :: ExpQ -> ExpQ
compile exp = do
    exp' <- exp
    let input = trace ("\n"++show exp'++"\n") exp'
    litE $ IntegerL $ compile' input

compile' :: Exp -> Integer
compile' (InfixE (Just (LitE (IntegerL a))) op (Just (LitE (IntegerL b)))) =
    name2op op a b
compile' (InfixE (Just (LitE (IntegerL a))) op (Just b)) =
    name2op op a (compile' b)
compile' (InfixE (Just a) op (Just (LitE (IntegerL b)))) =
    name2op op (compile' a) b
compile' (InfixE (Just a) op (Just b)) =
    name2op op (compile' a) (compile' b)
compile' _ =
    error "Not implemented"

name2op :: (Num a, Integral a) => Exp -> (a -> a -> a)
name2op (VarE name)
    | name == '(+) = (+)
    | name == '(-) = (-)
    | name == '(*) = (*)
    | name == '(/) = div
name2op _ = error "Not implemented"
```

Пример 2

```
{-# LANGUAGE TemplateHaskell #-}

import TH2

data Test = Test
    { fieldA :: Int
    , fieldB :: Bool
    }

$(deriveShow ''Test)

main :: IO ()
main = print $ Test 1 False

---------------------------------------
% runghc -ddump-splices Main2.hs
Main2.hs:1:1: Splicing declarations
    deriveShow ''Test
  ======>
    Main2.hs:10:3-19
    instance Show Test where
      show rec_a1jH
        = ("Main.Test:\
           \\t"
           ++
             (Data.List.intercalate
                "\
                \\t"
                [("Main.fieldA: " ++ (show (fieldA rec))),
                 ("Main.fieldB: " ++ (show (fieldB rec)))]))
Main.Test:
    Main.fieldA: 1
    Main.fieldB: False
```

Пример 2 (код)

```haskell
{-# LANGUAGE TemplateHaskell, FlexibleInstances #-}

module TH2 where

import Data.List (intercalate)
import Language.Haskell.TH
import Language.Haskell.TH.Syntax (VarStrictType)

deriveShow :: Name -> DecsQ
deriveShow name = do
    info <- reify name
    case info of
        TyConI (DataD _ _ _ [RecC _ fields] _) ->
            [d| instance Show $(conT name) where
                    show rec = $(showRec fields)
            |]
        _ ->
            error "Not implemented"
  where
    showRec :: [VarStrictType] -> ExpQ
    showRec fields =
        let rec = show name ++ ":\n\t"
        in [| rec ++ intercalate "\n\t" $(listE $ map showField fields) |]

    showField :: VarStrictType -> ExpQ
    showField (fieldName, _, _) =
        let recName = mkName "rec"
            field = show fieldName ++ ": "
        in [| field ++ show $(appE (varE fieldName) (varE recName)) |]
```

Примеры проектов

- Yesod (models, templates, routes, etc.)
- Happstack (models)
- TwilightSparkle (models, auto derives)

Недостатки

- Не type-safe, не haskell-way
  - Что находится внутри Q [Dec], Q Exp ?
  - Можно генерировать нерабочий код и выяснить это только при непосредственном использовании
  - Очень сложно писать unit-тесты (в том числе из-за ограничений reify)
  - Имеет доступ к приватным определениям модулей
  - Лишь незначительно лучше препроцессора (не нужно разбирать AST)
  - QQ работает с обычной строкой, для которой надо писать свой парсер, что делает его поведении совершенно некотролируемым
- Сложно читать код
- Страшно выглядит
- Может выполнять при *компиляции* произвольный код
- TH-функции нельзя использовать в том же модуле, где они определены

Ссылки

- http://www.haskell.org/haskellwiki/Template_Haskell
- http://www.haskell.org/ghc/docs/latest/html/users_guide/template-haskell.html
- http://www.haskell.org/haskellwiki/Quasiquotation
- http://hackage.haskell.org/packages/archive/template-haskell/latest/doc/html/Language-Haskell-TH.html
- http://www.yesodweb.com/book/persistent
- http://www.yesodweb.com/blog/2012/10/yesod-pure
- http://stackoverflow.com/questions/10857030/whats-so-bad-about-template-haskell

Вопросы?