Если вы видите что-то необычное, просто сообщите мне.

Функциональщина

- Отчет по stack скриптованию: как и почему...

Отчет по stack скриптованию: как и почему...

Введение

Почему stack скрипт?

Если вы делитесь маленьким, одиночным модулчем, самостоятельным примером haskell, то stack script дает нам простой способ получить воспроизводимую сборку, просто зафиксировав зависимости с помощью Stackage внутри комментариев в начале кода на Haskell.

Есть как минимум две дополнительные причины, кроме воспроизведения сборки приложений, возможно вам захочится использовать Stack скриптовый функционал:

- Низкий конфигурационный уровень: написание независимого комплириуемого файла с Haskell кодом, с зависимостями без необходимости настроек нового stack или cabal проекта.
- Использование Hashell как я зыка скриптования, изи замена для Shell/Bash/Zsh. Этот способ использования объединяется с использованием Turtle библиотеки, так же у этого подхода есть недостатки.

О статье

Stack это инструмент построения, главным образом, разработанный для воспроизвдения сборки приложений, выполняемое с помощью специального `resolver`(разрешателя зависимостей) в конфигурацийонном файле, обычно ваш проект `stack.yaml` и `package.yaml`. С помощью возможностей скриптования Stack, мы можем воспроизвести сборку приложения указывая `resolver`, но перенося эту спецификацию в файле который мы собираем или как аргумент командной строки. Отсюда, с целью упростить, мы предположим что эти скрипты запускаются вне stack проекта, и stack вызывается в той же директории что и скрипт.

> Заметка: Когда мы запускаем stack скрипт внутри проекта stack, важно принять во внимание, что stack прочитает настройки из project.yaml и stack.yaml, что можно привести к проблемам.

Примеры кода

Содераxние

Эта статья содержить следующие примеры использования скриптов и stack:

- Базовый пример интепретатора скриптов
- Просто Servant сервер который раздает статические данные вашей текущей папки.
- Пример stack как замена bash
- Использование скрипта за для запуска ghci

Базовый пример stack скрипта

Для нашего первого примера, мы будем использовать stack для запуска одного файла написаного на Haskell в виде скрипта.

Вот исходный код, который мы хотим запустить, в файле под названием `simple.hs`:

```
main :: IO ()
main = putStrLn "compiled & run"
```

Для его запуска с stack интерпретатора, мы можем выполнить следующее:

```
$ stack script simple.hs --resolver lts-14.18
```

Аргументы для `resolver` обязательны, и stack скомпилирует и запустить простой `simple.hs` файл сразу после того как будет вызван `lts-14.18` снимок.

Как альтернатива, мы можем сложить всю конфигурационную информацию в сам script, как показано ниже:

```
{- stack script
 --resolver lts-14.18
-}
main :: IO ()
main = putStrLn "compiled & run"
```

что может быть скомпилено и запущенно с помощью:

```
stack simple.hs
```

Простой Servant сервер.

Можно использовать haskell, и скриптовые возможности Stack, вместе с Turtle блиблотекой как замену shell скриптов. Для этого нам нужно добавить следующие строки в начало haskell файла:

```
#!/usr/bin/env stack
{- stack script
 --compile
 --copy-bins
 --resolver lts-14.17
 --install-ghc
 --package "turtle text foldl async"
 --ghc-options=-Wall
```

```
-}
```

Это stack скрипт делает пару вещуй:

- --compile и --copy-bins создает бинарник основываясь на имени файла
- устанавливает ghc, если необходимо, с помощью `install-gch`
- собирает скрипт с набором пакетов из lts-14.17

С помощью turle, мы получаем переносимый способ для запуска shell команд, мне удалсоь создать отличную haskell программу для замены shell скрипта, который я использовал для автоматизации задачь для развертывания этого блога.

Основа моего скрипта развертывания - `turtle`, как видно дальше, ниже представлен полный пример:

```
import qualified Turtle as Tu
import qualified Control.Foldl as L
import qualified Data.Text as T
import Control.Concurrent.Async
import System.IO

argParser :: Tu.Parser Tu.FilePath
argParser = Tu.argPath "html" "html destination directory"

main :: IO ()
main = do
  -- 53 files copied over into destinationDir
  hSetBuffering stdout NoBuffering
  destinationDir <- Tu.options "Build blog and copy to directory" argParser
  Tu.with (Tu.mktempdir "/tmp" "deploy") (mainLoop destinationDir)
```

Одна отличная штука про `turtle` это `Tu.with` функция, которая позволяет запускать нашу `main` логику во временной директории, которая в дальнейшем ощичаещтся после заврешения `mainLoop`.

Использование stack скрипта для запуска ghci.

Мы уже видели примеры stack скриптов, но есть еще, что должно быть в наборе разработчика Haskell. Stack скрипты можно использовать для запуска ghci repl. Представим мы работаем над новой ADT, и мы хотим написать новый объект `QuickCheck`, как нам может помочь script?

Следующий заголовок загрузит список приведенный ниже в ghci repl:

```
{- stack
 --resolver nightly
 --install-ghc
 exec ghci
 --package "QuickCheck checkers"
-}
module XTest where
```

Отметим еще пару вещей о порядке аргументов:

- Файл скомпилируется, и затем откроет консоль с загруженным модулем XTest.
- Если `exec ghci` сразу же не стоит за `stack`, тогда `--packages` должен быть перед `exec ghci`

ghcid

Теперь можно запустить скрипт выше с помощью ghcid, для получения практически постоянной обратной связи компилятора используя следующую команду:

```
bash$ ghcid -c "stack XTest.hs"
```

Заключение

Я часто нахожу себя за написанием маленьких haskell обрывков, однако но обычно это связано с изучением новых типов данных, использованием библиотке, или воспроизвдения примеров из статей или книг. В этом случае, `Stack` скриптовая возможность позволяет мне указать зависимости с помощью снимка в заголовке фала, и не беспокоится о ломающих изменениях, или настройках проекта со всеми верными зависимостями. Я должен обратиться к вам товарищи хаскеллята, использовать возможность stack скриптов когда кто-то делится своим кодом в сети, чтобы помочь остальным запустить их код сегодня, и в любое другое время в будущем.