МОДУЛЬНАЯ СИСТЕМА В HASKELL

Михаил Беляев

4 ноября 2015

Понятие модуля

- Программа на Haskell — это набор модулей
- Модули ограничивают область действия имён
- Очень похоже на то, как это работает в других языках

Отличия модулей Haskell от модулей в некоторых других языках

- Модули «плоские» — иерархии нет
  - Имеющаяся у некоторых пакетов «иерархия» типа A.B -> A.B.C -> A.B.C.D это просто имена
- Модули не являются значениями, ими нельзя обмениваться, у них нет параметров
  - В языках семейства ML это зачастую не так
- Если ваш файл не имеет объявления модуля внутри, всё его содержимое неявно заворачивается в модуль с именем, совпадающим с именем файла

Объявление модуля

```
module <NAME> [<EXPORTS>] where
<BODY>
```

где EXPORTS это список экспортируемых имён в круглых скобках.
Если EXPORTS отсутствует, все имена из модуля экспортируются.

Что может экспортировать модуль

- Функции
- Типы данных и (**отдельно!**) их конструкторы
- Классы типов
- Экземпляры классов типов для отдельных классов и типов
    - Поскольку у них нет имён, они экспортируются *всегда*
- Другие модули

ПРИМЕР

```haskell
module Tree ( Tree(Leaf,Branch), toList ) where

data Tree a                    = Leaf a
                               | Branch (Tree a) (Tree

toList :: Tree a -> [a]
toList (Leaf x)                = [x]
toList (Branch left right) = toList left ++ toList
```

Синтаксический сахар для экспорта

- Не экспортировать ничего (кроме экземпляров классов)

```
module Tree() where ...
```

- Экспортировать всё

```
module Tree where ...
```

- Экспортировать всё внутри чего-то

```
module Tree(Tree(..), toList) where ...
```

Импорт имён из модуля

```haskell
module Whatever( toList ) where

toList :: (a, a) -> [a]
toList (x,y) = [x,y]

module Main where

import Tree
import Whatever

toList x ???
```

По умолчанию каждый модуль импортирует модуль стандартной библиотеки `Prelude`, в котором содержатся базовые типы, операции и классы.

Использование имён с явным указанием модуля

```
module Main where

import Tree
import Whatever

foo = Whatever.toList (1,1)
bar = Tree.toList (Leaf 2)
```

Важно: пробелы рядом с символом `.` ставить нельзя — это будет означать вызов оператора «.»

Модули как средство абстракции

- Модуль может экспортировать только объявления типов и функций

```
module Tree(Tree, toList) where
    data Tree a
    toList :: Tree a -> [a]
```

- Реализации можно поместить в другой модуль

```
module Tree.Private(Tree(..), toList) where
  import Tree(..)

  data Tree a                    = Leaf a
                                 | Branch (Tree a
  toList :: Tree a -> [a]
  toList (Leaf x)                = [x]
```

Расширенные импорты

- Импортирование только определённых имён из модуля

```haskell
module Main where

import Tree(Tree(..)) -- импорт только типа Tre
```

Расширенные импорты

- Можно импортировать модули *без* определённых имён

```haskell
module Main where

import Prelude hiding (map) -- скрыть стандартн

map :: (a -> b) -> [a] -> [b]
map f = ...
```

Расширенные импорты

- Импорт с обязательным указанием модуля

```haskell
module Main where

import qualified Tree

foo = Tree.toList (Tree.Leaf 2)
toList (Leaf 2) -- не скомпилируется
```

РАСШИРЕННЫЕ ИМПОРТЫ

- Импорт с переименованием

```
module Main where

import Tree as T

foo = T.toList (T.Leaf 2)
```

Расширенные импорты

- Все виды конструкции `import` можно комбинировать между собой

```
module Main where

import Tree hiding(toList)
import qualified Tree(toList) as T

foo = T.toList (Leaf 2)
```

РЕЭКСПОРТ МОДУЛЕЙ

Можно экспортировать содержимое других модулей

```
module A where
    ...
module B (module A) where
    import A(...)
    ...
```

Теперь при `import B` в его содержимое будет неявно добавлено всё, что из модуля A видно в модуле B

Если модуль B не импортирует ничего из модуля A, программа не скомпилируется. Но возможны неявные импорты.

Пакеты в Haskell

- Пакеты не специфицированы языком, но являются расширением компилятора GHC
- Пакет — это набор библиотек и программ
- Пакет — это набор модулей
- Пакеты формируются, управляются и передаются с помощью утилиты `cabal`

HACKAGE

- hackage.haskell.org — база данных всех открытых пакетов с документацией и описанием модулей
- Hoogle — поисковая система по Hackage

Установка пакетов из Hackage

```
cabal install <packet>
```

После этого вы можете свободно использовать модули из пакета в своих программах

Практика именования модулей для Hackage

- Все модули общего пользования (общие библиотеки) принято называть `Data.*`
  - `Data.List`
  - `Data.String`
  - `Data.Tree`
  - `Data.Map`
- Модули для работы с общими классами типов принято называть `Control.*`
  - `Control.Monad`
  - `Control.Applicative`
  - `Control.Monad.Trans.State`
- Поскольку иерархии на самом деле нет, общие функции можно помещать «внутрь» стандартных модулей
- Остальные модули именуются на усмотрение авторов

Основные сценарии использования утилиты cabal

`cabal install <packet>` — установить пакет последней версии
`cabal install <packet>-<version>` — установить пакет последней версии
`cabal update` — обновить базу данных из Hackage

Обновление версий установленных пакетов делается путём переустановки

`cabal` как система сборки

- Пакет описывается с помощью `cabal`-файла (`packet.cabal`)
- Заменяет собой `make`
- Управляет сборкой, зависимостями, документацией
- Производит набор файлов, готовых к отправке в Hackage
- Выходит за рамки данного курса

Некоторые пакеты: библиотеки

- `base` — стандартная библиотека, поставляется с компилятором
- `array` — массивы
- `containers` — контейнеры (деревья и графы)
- `directory` — работа с файловой системой
- `parallel` — простое параллельное программирование
- `transformers`/`mtl` — монады-трансформеры
- `bytestring` — эффективный ввод-вывод
- `parsec` — стандартная библиотека парсер-комбинаторов
- `cabal` — утилита `cabal` как библиотека
- `haskell-src` — парсер Haskell на Haskell

Некоторые пакеты: программы

Программы устанавливаются через cabal так же, как и библиотеки

- `cabal-install` — сама утилита `cabal`
- `happy` — вспомогательный парсер
- `ghc-mod` — программа, реализующая функциональность IDE (но не саму IDE)
- `pandoc` — универсальный конвертер документов