GHC 源码剖析：语法分析

陈逸凡，吴昊泽

2017 年 5 月 28 日

1 概要

GHC 在前端使用 Alex[**alex**] 和 Happy[**happy**] 这两个在 Haskell 开发中常用的工具来生成词法分析器和语法分析器。在这一模块的源文件[**ghcparser**] 中，除了词法分析器（ Lexer.x ）和语法分析器（ Parser.y ）的生成文件之外，还包括了辅助的语法定义文件（ RdrHsSyn.hs ），以及处理 C-FFI（ C Foreign function interface ）和 Haddock 文档生成所需的辅助函数等。接下来主要介绍 alex 与 happy 这两个工具和 GHC 在语法解析过程中一些比较重要的额外处理。

2 Alex

2.1 功能和结构

Alex[**alex**] 是一个主要用 Haskell 开发的词法解析器生成器。功能与 C 语言开发常用的 Lex 或 Flex 类似，将用户定义的描述文件转换为包含生成的扫描器函数的 Haskell 源码。一个描述文件主要包括前缀代码，wrapper 声明，宏定义，规则和后缀代码这些部分。前缀代码和后缀代码为 Haskell 代码，主要功能有声明导入模块，定义导出模块，定义全局类型等。需要注意的是在 Alex 中，用户需要自己声明和定义 token 的类型和值，一般放置在后缀代码这一板块中。

2.2 规则

由于 Haskell 是静态类型的纯函数式语言，所以相比 Lex 中的规则，Alex 描述文件的规则中模式所对应的代码有更严格的要求，每条规则对应的动作代码，应当是符合约定类型的函数。

对于要求动作函数满足约定类型这一较强的限制，Alex 提供了 wrapper 声明的语法，wrapper 声明的作用在于，约定了每个动作的类型，从而使得用户可以使用预定义的高阶 API，免去自定义的麻烦。不同的 wrapper 对应不同样式的函数类型，如"basic" wrapper 约定每条规则的输入类型为 String 输出类型为自定义的 token 类型；"posn" wrapper 则在输入中提供了匹配串的位置信息；"monad" wrapper 和"monadUserState" wrapper 则提供了传递全局状态的 monad 结构，其中后者允许用户自定义全局信息，举个例子，如果需要统计被匹配到的所有 identifier 数目，就需要定义 AlexUserState 包含一个累计 identifier 数目的条目。

而不声明 wrapper 时，用户只能使用最底层的 API。有趣的一点是，使用底层 API 时，用户需要声明更底层的函数输入类型和获取单个字符的函数。Alex 开放输入类型约定的主要原因在于，Haskell 标准库的 String 实现是非常低效的，在强调性能的场合，往往使用 Text 或 ByteString 这样的高性能字符串实现。在定义好最基本的函数之后，每个模式对应的 action 就应该有统一的类型 $AlexInput \to Int \to AlexReturnaction$，或 $user \to AlexInput \to Int \to Maybe\ (AlexInput, Int, action)$，后者相比前者，增加了对于谓词的支持。

```
main = putStrLn $ show (1 :: Int)
```