GHC 源码剖析：重命名

陈逸凡，吴昊泽

2017 年 6 月 1 日

1 重命名阶段介绍

语法分析 [1] 得到的解析树与真正表达语义的抽象语法树之间还存在差别，主要体现在解析树中的名字（或者说，标识符）都是未绑定的。个人认为，不绑定的主要原因是在 Haskell 语法中，同一层绑定是没有先后顺序的（也就是说，不需要先声明再使用），所以解析时获取定义信息是非常困难的，所以需要在生成解析树后再进行绑定。在需要先声明再调用的语言中，则可以边解析边生成符号表，相对来说要容易处理很多。

未绑定的名字带来了以下一些问题：

- 前文提到，Haskell 语法中用户可以自定义中缀操作符并约定优先级和结合方向。但未绑定的代码导致在语法分析时，获取不到这些运算性质的信息。所以在语法分析时，会假定所有中缀操作符都是左结合且有同一优先级的。进而，当一个表达式包含多个中缀操作符时，往往需要在获取操作符运算性质后重构其解析树才能符合其语义。
- 在接下来的类型检查阶段中，需要确认绑定关系才能获取每一个名字的类型，来验证类型检查的结果。而在优化和代码生成阶段，绑定信息显然也是必不可少的。
- 此外，在函数定义、let 绑定表达式以及 lambda 表达式（实际上都可以认为是 lambda 表达式的变形）中产生的绑定会掩盖（shadowing）当前作用域中已经定义的相同名字的绑定，这也是需要处理的问题之一。

所以说在做类型检查等后续操作时，需要一个步骤来确认绑定关系并根据定义重构部分语法树。在 GHC 中，有一个 renamer 阶段来完成这一工作。

限于时间有限，我们并未深究这一步骤的具体代码，只是简单介绍说一下这一阶段的功能和实现思路。

2 功能

从宏观来看，这一阶段的功能是将语法树中的 RdrName 翻译到 Name，也就是将未绑定的名字与定义它的代码位置做绑定。

重命名前后的 Name 结构见下，可以看出差别：

```
data RdrName
  = Unqual OccName
      -- Used for ordinary, unqualified occurrences
  | Qual ModuleName OccName
      -- A qualified name written by the user in
      --  *source* code.  The module isn't necessarily
      -- the module where the thing is defined;
      -- just the one from which it is imported
  | Orig Module OccName
      -- An original name; the module is the *defining* module.
      -- This is used when GHC generates code that will be fed
      -- into the renamer (e.g. from deriving clauses), but where
      -- we want to say "Use Prelude.map dammit".
  | Exact Name
      -- We know exactly the Name. This is used
      --  (a) when the parser parses built-in syntax like "[]"
      --      and "(,)", but wants a RdrName from it
      --  (b) by Template Haskell, when TH has generated a unique name

data Name = Name {
              n_sort :: NameSort,
```

```
        -- What sort of name it is
    n_occ  :: !OccName,
        -- Its occurrence name
    n_uniq :: Int#,
        -- Its identity
    n_loc  :: !SrcLoc
        -- Definition site
}
```

具体的但在这一过程中，renamer 还“顺带”完成了一些工作：

- 首先将所有名字扩展成包含完整模块层级的形式。
- 在绑定过程中，将局部作用域中掩盖了外部作用域的绑定的名字通过加后缀的方式重命名。
- 在获得关于操作符运算律的声明后，将包含这些操作符的语法树重构。
- 分析函数间的依赖关系，重点是分析出间接调用关系。
- 检查词法错误，包括变量不在作用域内、未使用的绑定和调用、模式匹配中重复的模式等

经过这些变换后，可以认为当前的抽象语法树已经完整地体现了源代码的语义。

3 大致实现方式

renamer 的核心是维护一个保存全局环境信息的 map，map 将每个名字映射到一个包含其定义信息的栈中，越靠近栈顶的定义，其作用域越接近当前调用位置。

```
type GlobalRdrEnv = OccEnv [GlobalRdrElt]
    -- An (OccEnv a) is a mapping from OccName to a

data GlobalRdrElt = GRE { gre_name :: Name
                        , gre_prov :: Provenance
```

```
                          , gre_par :: Parent }

data Provenance = LocalDef | Imported [ImportSpec]

data ImportSpec = ImpSpec { is_decl :: ImpDeclSpec
                          , is_item :: ImpItemSpec }

data Parent = NoParent | ParentIs Name
```

4 在 GHC 流水线中的调用方式

由于 Template Haskell[1] 的存在，生成的代码需要重复调用重命名和类型检查，所以在 GHC 流水线中，重命名和类型检查这两个阶段耦合度很高，合并为一轮。在 ghc/compiler/main/HscMain.hs 文件中可以发现，GHC 编译流水线中，对于解析完成的解析树，以类型检查函数为入口，在做类型推导的同时推进重命名过程。类型检查模块将在下一份报告中介绍。

参考文献

[1] The GHC team. ghc/compiler/rename/.

[1] 一种在编译期依据模板生成代码的元编程技术