Speak my true name, summon my constructor

Felis sap...

函数式编程、编程语言、编程 话题的优秀回答者

已关注

众所周知，Haskell 有选择性 export 的机制，可以用来隐藏一个数据类型的定义。大多数时候这个特性是好的，它是 modularity 的关键——库作者只提供一个数据类型的名字和 kind signatures，以及一些能够操作该数据类型的函数（比如 smart constructors），则其他用户只用关心这些函数如何使用，不用（也不能）操心数据类型的实现方式、对应的构造器如何去模式匹配等细节。

这个机制偶尔也会成为绊脚石——要是库作者没有考虑到我的 use case，该给的函数或者 instance 没给，我自己也没法实现，那就很尴尬了……最近刚碰到这样的一个例子：GHC 的 codebase 里，所有的 IR 类型都支持 pretty-print，生成的 dump 确实很好看，然而我想要的是 ugly-print，也就是 GHC 默认 derive 机制生成的 Show instance。以下是一段 Cmm dump 的 pretty 版本：

```
cCVB: // global
    I64[Sp - 16] = stg_upd_frame_info;
    P64[Sp - 8] = _sCUu::P64;
    _sCUp::P64 = P64[_sCUu::P64 + 16];
    I64[Hp - 16] = sat_sCUt_info;
    P64[Hp] = _sCUp::P64;
    _cCVr::P64 = Hp - 16;
    R2 = _cCVr::P64;
    Sp = Sp - 16;
    call fact_info(R2) args: 24, res: 0, upd: 24;
```

和它的 ugly 版本：

为了使用 GHC API 写编译器，需要完全掌握各种 IR 的数据类型的含义，一个很重要的方法就是编译一些小的例子，然后导出 dump 来学习。与 pretty-print 的 dump 相比，显然 ugly-print 的 dump 能够直接对应到数据类型的定义上，两者结合阅读就不用在看到 pretty-print 版本里不认识的语法特性时手动去查 GHC codebase 里相应的 print 逻辑如何实现了。

通过 GHC 的 standalone deriving 扩展，我们可以在自己的 module 里去为别人的 datatype 去 derive 一些 Show instance。当然，这需要别人把该 datatype （及其依赖的所有 datatype）的完整定义给 export 出来，才能够 derive。在为这些 IR types 实现 Show instance 时，屡屡碰到一些 GHC team 故意藏起来定义的 datatype，如果该 datatype 起码实现了 pretty-print 的话，我可以用 pretty-print 的结果转为字符串糊弄一下，但少数的 datatype 既没有 Show instance，也不支持 pretty-print，还把定义给藏起来了，连手动去模式匹配一下都办不到……

好了，背景故事就扯到这为止。Haskeller 在需要对被隐藏定义的 datatype 进行模式匹配时，有 2 种方法：

- 给那个库发 pull request 骚扰作者
- 使用黑魔法召唤这个 datatype 的 constructor

所以本文宗旨就是介绍如何施法召唤一个 datatype 的 constructor，并将其用于模式匹配：Speak my true name, summon my constructor.

首先给项目加上 template-haskell 依赖。以下是召唤一个 constructor，并将其用于 pattern matching 的术式：

```
summon :: Name -> String -> (Name -> Pat) -> Q Pat
summon t c mp = do
    info <- reify t
    case info of
        TyConI dec -> do
            let dec2cons dec' =
                    case dec' of
                        DataD _ _ _ _ cons _ -> cons
                        NewtypeD _ _ _ _ con _ -> [con]
                        _ ->
                            error $
                            show dec' ++
                            " is not a data or newtype declaration."
                con2name con =
                    case con of
                        NormalC name _ -> name
                        RecC name _ -> name
                        InfixC _ name _ -> name
                        ForallC _ _ con' -> con2name con'
```

```
        case find ((== c) . nameBase) cnames of
            Just n -> pure $ mp n
            _ ->
                fail $
                c ++ " is not among the constructors of " ++ nameBase t
    _ -> fail $ nameBase t ++ " is not a plain type constructor."
```

介绍一下 Template Haskell 相关的几个类型。首先是 Q Monad —— 每一个黑魔法术式，不对，Template Haskell splice 的类型可以分为 Q Exp、Q Pat、Q Type、Q [Dec]，分别代表生成表达式、模式、类型和声明。Q Monad 用于管理代码生成相关的一些状态（比如生成 fresh identifier），同时可以 lift IO action。这里，我们需要用召唤出的 constructor 实现模式匹配，所以 splice 的类型为 Q Pat。

接下来是 reify 函数。reify 函数能够在 Q Monad 中查询一个 Name 对应的信息。我们在调用 summon 时，当前 scope 里能用的 Name 只有某个 datatype 自身的名字，所以先将这个名字进行 reify，然后查看查询结果，我们关心的是 TyConI，也就是普通的 type constructor。这里我们匹配到了一个 Dec 类型的值，前面说过 Dec 对应 declaration，也就是说我们通过一个 datatype 的名字获取了它的完整定义。

直接将这个 Dec 重新导出相当于重新定义一个相同的 datatype，会与当前 scope 中已有的版本发生冲突，所以我们退而求其次，遍历这个 Dec，获取所有的 constructor 的 Name，在其中查找是否有 Name 匹配 summon 函数中 String 参数指定的 constructor 名字，有则将其提取出来——到这一步，召唤术成功了一大半，我们打破了 module barrier，获取了 constructor 的 Name，这个 Name 可以用来生成各种代码。没有经过 reify 的过程，直接将 String 转换成 Name 是不能实施召唤的。我们需要用这个 Name 来模式匹配某一个 constructor。所以 summon 的另一个参数是类型为 Name -> Pat 的回调函数，这个函数用于组装出我们需要的 pattern 代码。

Template Haskell splices 的 definition site 和 use site 需要分开在不同 module 中。在这个 splice 定义好以后，在需要用到它的 module 中，import 该 splice 的定义 module，打开 TemplateHaskell 扩展，然后可以这样用：

```
showStgBinderInfo :: GHC.StgBinderInfo -> String
showStgBinderInfo x = case x of
    $(summon ''GHC.StgBinderInfo "NoStgBinderInfo" (\n -> ConP n [])) -> "NoStgBinderI
    $(summon ''GHC.StgBinderInfo "SatCallsOnly" (\n -> ConP n [])) -> "SatCallsOnly"
```

在 case expression 中，需要提供一个 pattern 的地方，我们用 $(....) 代替，括号内是一个类型为 Q Pat 的表达式，GHC 在编译该 module 时会运行 Q Monad 中的计算，生成一个 pattern。给 summon 传递参数时，需要传递特定 datatype 的 Name，我们可以有 2 种方法获取它：

- 传递一个 String，然后手动通过 lookupTypeName / lookupValueName 在 type/value 的

''T 在当前 scope 的 type 中查找 T

现在，使用 summon，我们即可在知道一个 datatype 的名字与它的 constructor 名字的前提下，召唤它的 constructor，用于（库作者不希望你做的）模式匹配。

以上是 Template Haskell 的一个简单的 use case。这个扩展还可以做许多其他有趣的事，展开来讲的话也许要开个 Deep Dark Haskell 系列坑了，专门讲各种不优雅、不安全的 Haskell 编程技巧（逃

参考资料：

9.26. Template Haskell

template-haskell-2.12.0.0: Support library for Template Haskell

发布于 2017-04-20

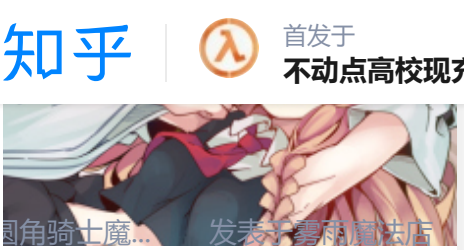
Haskell GHC（编程套件） 函数式编程

文章被以下专栏收录

推荐阅读

上海2017年函数式编程分享会□视频

上海2017年函数式编程分享会 - 播



圆角骑士魔... 发表于雾雨魔法店

寸光寸阴 发表于Haske...

周末有时间我再整理下文字记录吧[二哈]

祖与占

祖与

7 条评论

切换为时间排序

写下你的评论...

人工智能企鹅 1 年前

我就是想知道你放个C妈的意思是...

6

扶蓝 1 年前

我也想知道你放个C妈的意思...

2

徐阿明 1 年前

C妈： “？？？”

1

L lian wein 1 年前

标题有意思，js好像也通用

赞

Oling Cat 1 年前

嗷嗷嗷C妈！！！

另外真的没有人关心文章内容么= =?

1

Felis sapiens (作者) 回复 Oling Cat 1 年前

反正文章只是图的附赠品（逃

1 查看对话

wkGCaSS 1 年前

看见c妈就点进来了。。。