

Seaman.h.zhang

博客园 :: 首页 :: 新随笔 :: 联系 :: 订阅  :: 管理 34 Posts :: 0 Stories :: 2 Comments :: 0 Trackbacks

公告

昵称: seaman.kingfall

园龄: 4年3个月

粉丝: 4

关注: 1

+加关注

搜索

找找看

谷歌搜索

常用链接

我的随笔

我的评论

我的参与

最新评论

我的标签

我的标签

练习题(6)

合一(3)

递归(3)

中断(2)

类型变量(2)

数字(2)

列表(2)

Haskell(2)

recursive(2)

比较(2)

更多

随笔分类

Haskell(2)

Prolog(32)

随笔档案

2015年8月 (7)

2015年7月 (22)

2015年6月 (5)

最新评论

1. Re:Learn Prolog Now 翻译 - 第一章 - 事实, 规则和查询 - 第一节, 一些简单的例子学习!

--深蓝医生

2. Re:Learn Prolog Now 翻译 - 第一章 - 事实, 规则和查询 - 第一节, 一些简单的例子翻译了这么多了, 而且每天一篇, 不能望其项背啊。

Learn Prolog Now 翻译 - 第十二章 - 文件相关操作 - 第一节, 使用不同文件组织程序

学习到这个阶段, 你可能已经使用append/3和member/2写了很多程序。你可能每次都需要将它们的实现代码拷贝到使用它们的程序文件中。而且, 经过几次这样做之后, 你就会感觉每次不停的拷贝是非常重复和麻烦的事情。如果你可以在一个文件中定义它们, 然后在需要的地方使用, 这将会是令人愉快的, 而且也是更加合理的做法。当然, Prolog提供了这样的方式去组织程序。

程序中的读操作

事实上, 你已经知道有一种方式, 可以告诉Prolog读取文件中的谓词定义, 如下:

```
[FileName1]
```

并且这个命令可以告诉Prolog对文件中的内容进行编译。但是有两件有用的事情你也应该知道: 第一, 你可以一次编译多个文件:

```
[FileName1, FileName2, ..., FileNameN]
```

第二, 更为重要的是, 文件编译不仅仅可以用于交互环境, 如果你将下面代码:

```
:- [FileName1, FileName2, ..., FileNameN].
```

放在你的程序代码文件头部 (比如命名为main.pl的文件), 那么事实上你是告诉Prolog首先编译列出的文件, 然后再读取程序的其他部分。

这个特性给予了我们重用谓词的简单方式。比如, 假设你将所有基础列表操作相关的谓词放在一个文件中 (比如append/3, member/2, reverse/2等等), 名字是listPredicates.pl, 如果你想要使用它们, 可以这么输入:

```
:- [listPredicates].
```

到需要使用它们的程序文件的开头。Prolog将会编译listPredicates当读取程序文件时, 所以listPredicates中定义的所有谓词能可用了。

这里有一点需要注意的, 当Prolog装载读取文件时, 它不会检查这些文件是否已经编译过。如果文件中的谓词已经存在知识库中, 因为文件在之前已经编译过, Prolog还是会重新编译它们, 虽然这完全没有必要。而且在编译非常大的文件时, 会导致性能的损耗。

内置的谓词 ensure_loaded/1 可以更加智能的加载读取文件, 它的效果和之前的一致, 使用如下:

```
:- ensure_loaded([listPredicates]).
```

Prolog将会检查listPredicates.pl是否已经加载过, 并且当文件有修改时, 才会再次加载。

--Benjamin Yan

阅读排行榜

1. Learn Prolog Now 翻译 - 第三章 - 递归 - 第一节, 递归的定义(1168)
2. Learn Prolog Now 翻译 - 第一章 - 事实, 规则和查询 - 第一节, 一些简单的例子(1087)
3. Learn Prolog Now 翻译 - 第一章 - 事实, 规则和查询 - 第二节, Prolog语法介绍(781)
4. Haskell学习笔记二: 自定义类型(767)
5. Learn Prolog Now 翻译 - 第六章 - 列表补遗 - 第一节, 列表合并(753)

评论排行榜

1. Learn Prolog Now 翻译 - 第一章 - 事实, 规则和查询 - 第一节, 一些简单的例子(2)

推荐排行榜

1. Haskell学习笔记二: 自定义类型(1)
2. Learn Prolog Now 翻译 - 第三章 - 递归 - 第四节, 更多的实践和练习(1)

模块

现在设想你正在写一个程序进行电影知识库的管理。你已经设计了一个谓词 `printActors` 用于打印一部电影的主要演员, 以及一个谓词 `printMovies` 用于打印某个导演的所有电影。两个不同的定义储存在不同的文件中, 名为 `printActors.pl` 和 `printMovies.pl`, 同时两个文件都有名为 `displayList/1` 的辅助谓词。下面是第一个文件内容:

```
% This is the file: printActors.pl
printActors(File) :-
    setof(Actor, starring(Actor, File), List),
    displayList(List).

displayList([]) :- nl.
displayList([X | L]) :-
    write(X), tab(1),
    displayList(L).
```

下面是第二个文件内容:

```
% This is the file: printMovies.pl

printMovies(Director) :-
    setof(File, directed(Director, File), List),
    displayList(List).

displayList([]) :- nl.
displayList([X | L]) :-
    write(X), nl,
    displayList(L).
```

注意 `displayList/1` 在两个文件中有不同的定义: 打印演员是用按行的方式 (使用 `tab/1`), 而打印电影是用按列的方式 (使用 `nl/0`)。这会让 Prolog 困惑吗? 让我们看看, 我们使用下面的代码将两个文件同时加载:

```
% This is the file: main.pl

:- [printActors].
:- [printMovies].
```

上面的代码写在 `main.pl` 文件的开头, 编译主程序文件会报出如下的提示信息:

```
?- [main].
{consulting main.pl...}
{consulting printActors.pl...}
{printActors.pl consulted, 10 msec 296 bytes}
{consulting printMovies.pl...}
The procedure displayList/1 is being redefined.
    Old file: printActors.pl
    New file: printMovies.pl
Do you really want to redefine it? (y, n, p, or ?)
```

发生了什么? 由于 `printActors.pl` 和 `printMovies.pl` 两个文件都定义了名为 `displayList/1` 的谓词, 所以 Prolog 需要选择其中的一个定义 (同一谓词在知识库

中不能有不同的定义)。

如何解决这个问题呢? 在有些情况下, 你可能真的需要重新定义谓词。但是现在你不能——因为打印演员和打印电影希望使用不同的方式进行。有一种解决方法是: 给其中一个谓词另外一个名字。但是这个方法是很笨拙的。你希望每个文件都是逻辑自包含的程序实体, 不希望浪费时间和精力为了其他文件而进行谓词的重新命名。所以获取概念独立性最自然的方式是使用Prolog的模块系统。

模块从根本上允许隐藏谓词定义。你可以决定哪些谓词应该是对外公开的(即, 可以由其他文件的程序来调用), 以及哪些谓词应该是私有的(即, 只能在模块内部调用)。这样的话, 就不允许在模块外部调用私有的谓词, 这样两个模块内部名字一样的私有谓词就不会发生冲突了。在我们的例子中, `displayList/1` 这个谓词是私有谓词的最好候选: 它都是在各自的文件中起辅助作用的, 并且两个实现之间没有任何关联。

可以将文件转换为模块, 只需要在文件开头进行模块声明。模块声明的形式如下:

```
:- module(ModuleName, List_of_Predicates_to_be_Exported).
```

上面的声明指定了模块的名字, 及其公共谓词的列表, 即你希望导出的谓词列表。这些谓词是能够在模块外部被访问的。

让我们将电影知识库程序的文件模块化。我们只需要在第一个文件的开头包含下面一行:

```
% This is ths file: printActors.pl

:- module(printActors, [printActors/1]).

printMovies(Director) :-
    setof(File, directed(Director, File), List),
    displayList(List).

displayList([]) :- nl.
displayList([X | L]) :-
    write(X), nl,
    displayList(L).
```

这里我们定义了一个称为`printActors`的模块, 有一个公共谓词`printActors/1`。谓词`displayList/1`仅仅在模块`printActors`内部可见, 所以它的定义不会对其他模块有任何影响。

类似地, 我们可以将第二个文件也模块化:

```
% This is the file: printMovies.pl

:- module(printMovies, [printMovies/1]).

printMovies(Director) :-
    setof(File, directed(Director, File), List),
    displayList(List).

displayList([]) :- nl.
displayList([X | L]) :-
    write(X), nl,
    displayList(L).
```

同样地, `displayList/1`的定义只在`printMovies`这个模块中可见, 所以在加载两个模块文件中Prolog不会出现冲突和崩溃了。

可以使用内置谓词`use_module/1`加载模块, 模块中所有公共谓词都会被导入到当前知识库中。换种说法, 所有公共谓词都是能够访问的。修改`main.pl`文件:

```
:- use_module(printActors).  
:- use_module(printMovies).
```

如果你不希望使用模块中的所有公开谓词, 而只是其中的一部分, 你可以使用两个参数版本的`use_module`, 其中第二个参数是你真正希望使用的公开谓词列表, 如下:

```
% This is the file: main.pl  
  
:- use_module(printActors, [printActors/1]).  
:- use_module(printMovies, [printMovies/1]).
```

在`main`文件的头部, 我们显式地指出需要使用的谓词是: `printActors/1`和`printMovies/1`, 而且不需要其他的谓词(当然在这个例子中, 也没有其他公开的谓词可以使用了)。

库

很多通用的谓词在大多数Prolog实现中, 已经通过一种或者多种方式预先定义了。如果你在使用SWI Prolog, 可能已经注意到诸如`append/3`和`member/2`这些谓词已经是系统的组成部分。这是SWI Prolog特有的。其他一些Prolog的实现, 比如SICStus, 并没有内置这些谓词, 都是通过库的方式提供的。

库是定义了通用谓词的一些模块, 并且能够使用相同的命令进行加载使用。当你指定想要使用的特定库名字时, 你必须告诉Prolog将模块作为库的形式来加载, 这样Prolog才知道去什么地方找到这些模块(即, Prolog有特定的存放库的路径, 并不是程序代码所在的路径)。比如, 在程序文件的头部输入下面的命令:

```
:- use_module(library(lists)).
```

将会告诉Prolog加载名字为`lists`的库。在SICStus Prolog中, 这个库包含了一系列通用的列表处理谓词。

库十分有用并且能够提高开发效率。而且, 库中的代码都是优秀程序员写的, 几乎都是执行效率很高, 并且错误很少的。但是不同的Prolog实现中, 库的组织和实现的谓词并没有统一的标准。这意味着如果你想要你的程序运行在不同的Prolog实现下, 定义自己的库模块是更加容易和效率的, 这样不用尝试去理解和解决不同Prolog实现下的兼容性问题。

分类: Prolog

标签: 文件, 读取, 模块, 库

好文要顶

关注我

收藏该文





seaman.kingfall

关注 - 1

粉丝 - 4

[+加关注](#)

« 上一篇: [Learn Prolog Now 翻译 - 第十一章 - 知识库相关操作和解决方案的收集 - 第二节, 解决方案的收集](#)

» 下一篇: [Learn Prolog Now 翻译 - 第十二章 - 文件相关操作 - 第二节, 文件的读写](#)

posted on 2015-08-08 08:47 seaman.kingfall 阅读(366) 评论(0) 编辑 收藏

[刷新评论](#) [刷新页面](#) [返回顶部](#)

注册用户登录后才能发表评论, 请 [登录](#) 或 [注册](#), [访问网站首页](#)。

【推荐】超50万C++/C#源码: 大型实时仿真组态图形源码

【活动】看雪2019安全开发者峰会, 共话安全领域焦点

【培训】Java程序员年薪40W, 他1年走了别人5年的路

相关博文:

- [Learn Prolog Now 翻译 - 第三章 - 递归 - 第一节, 递归的定义](#)
- [Learn Prolog Now 翻译 - 第五章 - 数字运算 - 第一节, Prolog中的数字运算](#)
- [Learn Prolog Now 翻译 - 第六章 - 列表补遗 - 第一节, 列表合并](#)
- [Learn Prolog Now 翻译 - 第二章 - 合一和证明搜索 - 第一节, 合一](#)
- [Learn Prolog Now 翻译 - 第九章 - 语句深究 - 第一节, 语句的比较](#)

最新新闻:

- [知否 | 太空垃圾如何清理? 卫星测试用鱼叉击中太空垃圾碎片](#)
- [一线 | “美团配送”品牌发布: 对外开放配送平台 共享配送能力](#)
- [苍蝇落在食物上会发生什么? 让我们说的仔细一点](#)
- [科学家研究板块构造变化对海洋含氧量影响](#)
- [日本程序员节假日全员加班? 都是“令和”惹的祸](#)
- » [更多新闻...](#)

Copyright © seaman.kingfall
Powered by: .Text and ASP.NET
Theme by: .NET Monster