FORCEFIELD语言

设计及实现

王远轩 肖梦华

2010.8

FORCEFIELD 语言说明

云计算脚本语言

王远轩 肖梦华

2010.08

语言简介

我们在云计算脚本语言的基本要求之上，增加了一些其它的语言特性，并命名这个语言为ForceField语言。它是一种应用于云计算平台的轻量级脚本语言，支持数字、字符串、字典及函数四种对象，支持基本的循环、赋值、函数调用等基本功能，也支持远程调用、分层作用域、函数递归等语义，同时支持函数作为参数、函数Curry化等函数式编程语法。此外，我们为ForceField语言实现了一套在线调试系统，方便开发者调试自己的程序。

基本语法

根据比赛要求，ForceField语言使用了一系列大写字符作为它的关键字。由于比赛页面中已经给定了SET、RETURN、IF等关键字的用法，这里不再赘述。接下来我们将分别介绍比赛页面中没有涉及到的内容，其中也包括了我们自己设计的一些语法。

字典类型

比赛页面中提到了HTTP这一内置字典，为了保持语法的一致性，我们引入了字典这一类型。ForceField的字典类型和常见语言的字典类型几乎一样。它的声明方式也很简单，我们并没有提供初始化某一个字典的语法，需要创建一个字典对象时只需直接赋值即可。例如，我们要声明一个名为foo的字典对象，同时将它的bar键置为hello：

```
SET foo[“bar”] = hello
```

这条命令会先在可访问的作用域中查找名为foo的变量，如果已存在foo字典则直接修改bar键，否则先创建foo字典，再修改，如果foo存在但它的类型不是字典则报错。

远程函数调用

当ForceField程序员需要调用一个远程脚本时，他需要修改两个地方：在server.conf中声明远程服务器脚本的URL，例如

```
script1, http://127.0.0.1:12345/add
```

然后在本地脚本中声明调用这个脚本需要的参数，以ADDNUM为例：

```
REMOTE @script1 ADDNUM(x, y)
```

这样就能调用远程的ADDNUM脚本了。

变量作用域

类似Python，在ForceField中只有函数调用才会创建新的作用域，同时能访问外层变量）。

但和Python不同的是，Python中函数对外层作用域的访问是只读的，例如这个程序是非法的：

```
def f():
    x += 1

x = 0
f()
```

在ForceField语言中，拥有类似语义的这个程序则是被允许的：

```
FUNC F()
     SET x = x + 1
     RETURN x
END

SET x = 0
CALL F()
RETURN x
```

最终这个脚本会返回1。

为了更好的说明作用域的机制，我们接下来将通过深度优先搜索解八皇后程序的运行(queen.input)为例，来说明ForceField的作用域的设定。（所有图片均截图自ForceField在线调试工具）

Level 0 binding(s)	
HTTP	{'n': 4}
i	4
n	4
found	0
SERVER_TIME	Function
WRITE_LOG	Function
col	{0: 0, 1: 0, 2: 0, 3: 0}

```
1  SET n = HTTP["n"]
2
3  FOR i = 0 TO n - 1 STEP 1 DO
4    SET col[i] = 0
5  END
6
7  SET found = 0
8
9  FUNC search(row)
10   IF row >= n THEN
11     SET found = 1
```

执行到第9行时，最外层作用域有HTTP、SERVER_TIME和WRITE_LOG三个默认变量，以及程序运行到现在产生的变量。当第9行程序执行完成后，最外层作用域会增加一个search函数。

Level 0 binding(s)	
search	Function
HTTP	{'n': 4}
i	4
n	4
found	0
SERVER_TIME	Function
WRITE_LOG	Function
col	{0: 0, 1: 0, 2: 0, 3: 0}
Level 1 binding(s)	
row	0

```
1  SET n = HTTP["n"]
2
3  FOR i = 0 TO n - 1 STEP 1 DO
4    SET col[i] = 0
5  END
6
7  SET found = 0
8
9  FUNC search(row)
10   IF row >= n THEN
11     SET found = 1
12     RETURN 1
13   END
14
15   IF found == 1 THEN
16     RETURN 1
17   END
18
19   FOR col[row] = 0 TO n - 1 STEP 1 DO
```

此处为调用search(0)后的结果，参数row被置为0，且属于第一层作用域。另外可以看到最内层作用域已经有一个search变量，且类型为Function。

Level 0 binding(s)	
search	Function
HTTP	{'n': 4}
i	2
n	4
found	0
SERVER_TIME	Function
WRITE_LOG	Function
col	{0: 0, 1: 2, 2: 0, 3: 0}
Level 1 binding(s)	
valid	0
row	0
Level 2 binding(s)	
row	1
Level 3 binding(s)	
row	2

```
1  SET n = HTTP["n"]
2
3  FOR i = 0 TO n - 1 STEP 1 DO
4    SET col[i] = 0
5  END
6
7  SET found = 0
8
9  FUNC search(row)
10   IF row >= n THEN
11     SET found = 1
12     RETURN 1
13   END
14
15   IF found == 1 THEN
16     RETURN 1
17   END
18
19   FOR col[row] = 0 TO n - 1 STEP 1 DO
20     SET valid = 1
21     FOR i = 0 TO row - 1 STEP 1 DO
```

此处为search递归过程中的作用域情况，可以看到一、二、三三层作用域中均包含row变量，此时读取row时优先使用最外层（及第三层）的作用域，值为3。

函数式编程

ForceField对函数式风格有良好的支持，所有的函数都可以直接通过参数的形式传递。此外，ForceField还支持Curry化的函数，即分多次将参数传给函数。下面以回归测试程序中的curry_test为例介绍ForceField的函数式编程支持：

```
FUNC ADD_FOUR(a, b, c, foo)
  RETURN a + b * 2 + c * 3 + foo() * 4
END

FUNC GETONE()
  RETURN 1
END

SET f = ADD_FOUR(4)
SET g = f(3, 2)
SET h = g(GETONE)
```

这里ADD_FOUR是一个接收四个参数的函数，其中最后一个参数为一个函数。这个程序运行之后，f为一个接收三个参数(b,c,foo)的函数；g为一个接收单一参数的函数，且这个参数是一个函数；最后一行将GETONE这一函数传递给g，并将返回值20赋值给h。

语言实现

我们使用了ANTLR (http://www.antlr.org/)开源框架对ForceField脚本进行词法、语法分析。由于我们旨在实现一个脚本语言的原型，并不注重它的运行性能，因此略去了编译后端的代码生成、优化等阶段，而是直接使用Python解释执行由ANTLR生成的抽象语法树。ForceField脚本的词法、语法分析主要包含在Expr.g文件中，而解释执行则和Eval.g，environment.py等多个文件有关。

使用make命令编译后（在我们提交的版本中已经生成了语法解析的程序，不需要再执行），ANTLR会自动生成ExprLexer.py, Eval.py, ExprParser三个文件，用于解析脚本程序。

惰 性 求 值

在解释执行时，我们采用了惰性求值技术简化了解释器的实现。在environment.py中我们定义了Stmt、Expr、Function、RemoteCall四种基本语言元素，对应于抽象语法树中的不同结点类型。这些结点在语法树解析阶段被创建，创建之初并不会立即执行，直到解释器需要执行某一条语句或是需要某个表达式的值时，它们的eval/call方法才会被调用，从而真正的执行对应语句和表达式，这种求值方式和SICP 3.5章中提到的流编程模型中delay/force机制比较相似，大大简化了ForceField解释器中分支、循环和函数等语法的实现。

返 回 值 处 理

程序执行RETURN的时候，需要中断当前代码块的解释，退回到上一层求值环境中。一种实现方法是使用一个全局变量，表明当前代码块是否运行过RETURN语句，如果是则跳过所有Stmt对象的解释。这种方法增加了程序的复杂度，需要较多的额外检查。因此我们选择了使用Python异常返回值，这是一种end to end的解决方法。函数调用过程被包在一个try代码块中，返回时只需发起一个包含返回值的异常（ReturnValue类，继承了Exception类）即可。另外Python在捕捉异常时，只是创建了一个额外的frame，因此这里使用异常的开销并没有其它语言（如Java/C#/C++等）那样巨大。

作 用 域 维 护

在ForceField中不同层的作用域被保存在一个栈状的数据结构中。每一层的作用域都包含当前层变量值，保存在Python字典类型中。

CURRYING

基于惰性求值和保存在字典中的作用域，函数Currying可以比较容易的实现。当一个函数被调用时，解释器首先判断传入的参数是否足够，如果足够则直接调用；反之会创建一个CurryFunction类型并返回。CurryFunction被调用时会先将已经求得的结果保存到当前值域中，并调用之前定义的函数(CurryFunction.nextaction)。

在线调试

为了方便ForceField程序员，我们开发了一个在线调试的工具，包括重启程序、单步跟踪两个功能。当用户访问/debug/script_name时，服务器端会启动一个新的子进程解释该脚本，并截获这个进程的所有输出。用户点击单步跟踪后，浏览器就会发送一个Ajax请求给后台服务器，服务器发送一个SIGUSR1信号给解释器，使得解释器继续解释代码，到下一个语句结束后再暂停。

由于Python的subprocess模块不支持异步IO调用，直接读取子进程的stdout/stderr会导致服务器进程的阻塞。为了解决这个问题我们使用了开源的asyncproc模块，这个模块在Popen对象加了一层包装，使得他的stdout/stderr输出可以被异步读取。

后台服务器

我们使用了Tornado Web Server作为后台服务器。它的优点是非常小巧，一共不到6000行的Python代码，且使用了web.py这个简单好用的Web框架。同时由于使用了epoll系统调用，它的并发性能也很好。

测试脚本

为了保证新增的特性不影响之前脚本的正确运行，我们在scripts目录下维护了一组回归测试脚本用例，所有以_test.input结尾的脚本都会在回归测试中覆盖。同时为了方便自动化测试，我们加入了ASSERT关键字，这个关键字可以穿插在程序中，保证某一时刻程序的中间变量的正确性。

回归测试覆盖了基本运算、字典操作、递归函数、远程调用、函数式编程以及其他一些语言要素，在ForceField目录下运行python regression.py即可进行自动化的回归测试。

已知问题及改进

错误提示

目前ForceField的解释器假设程序员能写出语法、语义上正确的程序。当出现语法错误时解释器会报错退出，但是给出的信息很难帮助程序员检查语法错误。

惰性队列

因为ForceField采用了惰性的方式解释程序，因此可以较方便的扩充出惰性队列这一数据结构。由于时间有限（暑假大多数时间都在实验室），没来得及设计一个和ForceField风格接近的惰性队列的语法。

解 释 性 能

由于当前的ForceField程序是通过解释执行的，没有经过后端优化，因此性能比较低下。如果能结合PyPy或者LLVM等开源工具，可以大大提高ForceField的运行性能。