AWK 学习笔记

1．AWK 简介

AWK 是 Alfred V.Aho, Peter J.Weinberger，Brian W.Kerninghan 三人在 1977 设计和实施的，最初是为了试验 Unix 中的 grep 和 sed 工具怎样可以一般化地在文本之外还能处理数据（grep 和 sed 主要是文本处理工具，但 AWK 同时善于处理文本与数据）。AWK 名字的来源是创造此语言的三个人的名字首字母缩写。它的主要来源是 grep, sed 和 C。后继者主要有 Perl。

AWK 非常适合于处理格式化的文本和数据，比如改变数据的格式、验证其合法性、寻找某些属性的项、数字求和、输出数据报表等。数据的结构化越强，使用 AWK 会越方便。相比于 sed，它有字段（sed 只有行而没有内置的字段模型）和数字处理功能。相比于 C 和 Perl，由于它自动化了某些处理流程（比如读取文件和分割字段），可以使得某些任务以比 C 和 Perl 少得多的代码来完成。

AWK（相对于 Perl）的不足：一是各种复杂的数据结构难以实现；二是对 Unicode 的支持不好。

2．AWK 编程模型

AWK 程序的基本使用语法如下：

awk [*-v var=value*] *-Fre 'pattern { action }' var=value datafile(s)*

awk [*-v var=value*] *-Fre -f scriptfile var=value datafile(s)*

用命令行与用脚本文件(scriptfie)是等价的。一般来说，若 pattern、action 的代码量比较大，倾向于使用脚本文件。

对于以上语法说明如下：

- -v 选项定义的变量在脚本运行之前即存在，可以在脚本的 BEGIN 流程中被调用；
- 命令行参量（不用-v 定义的）只有到输入的第一行读入时才有效，即其在 BEGIN 流程中无效；
- -F 选项将字段分割符(FS)设为一个正则表达式 re；
- datafile(s)可以是单个文件，也可以是多个文件，可使用正则表达式）。如果用“ - ”（不包括引号）表示从命令行输入。
- 命令行输入‘ pattern{action} ’时单引号是必不可少的 !不要忘记。pattern 和 action 都是可选的，但是不能都省略。省略 pattern 时默认匹配所有行；省略 action 时默认为输出整行。为了分别 pattern 和 action，action 需要用{}括起来。

脚本文件的一般格式为：

```
BEGIN{
…


}
pattern1 { action1 }
pattern2 { action2 }
…
END{
…
}
```

这也说明了 AWK 的编程模型：程序开始处理 BEGIN 流程（一般是设置分隔符、定义变量以及输出信息行等），然后进入主循环，读入数据每一行，本行数据设为$0，行数为 NR(某一文件的行数为 FNR),同时根据 FS 变量将$0 切分为 NF 个字段，分别用$1,$2,…,$NF 标识。程序对每一行用 pattern 进行匹配，若匹配上则运行 action。在所有行处理完后， 程序将处理 END 流程（一般是进行后期处理，输出综合结果等）。

说明：

- 程序默认的字段分割符(FS)是“ [\t]+ ”（Tab 或空格），输出分割符(OFS)是“ ”，（空格），默认记录分隔符(RS)是“ \n ”,可根据需要在命令行或 BEGIN 中修改；
- pattern 可以是以下四种格式：
  (1) 表达式（expression）. 在表达式为真时执行 action.
  (2) 正则表达式(/regular expression/)，正则表达式匹配上时执行 action.

(3) 复合模式(compound pattern),用 &&(AND), ||(OR),!(NOT)和括号组合出的模式，组合模式为真时执行 action.

(4) 范围模式（pattern1,pattern2），从 pattern1 匹配上的行，直到 pattern2 匹配上的行，包括这两行。**如果 pattern2 一直没有匹配上，则到文件的末尾。**

- action 默认一行一条指令，可包括多行。如果需要在一行中包括多个语句，需要用分号(；)分割开。但是一行一个语句则不需要加分号。这里有与 C 语言相同又有不同的地方，请注意区别。

3. 常量与转义符

AWK 的常量有字符由字符串常量与数值常量组成，字符串常量含引号，而数值常量没有。AWK 的转义符如下：

符号	描述
\a	警告字符，通常是 ASCII BEL 字符
\b	退行
\f	Formfeed
\n	换行
\r	回车
\t	TAB
ddd	八进制
c	任何字符 *c* 比如\"代表"

4. 变量

AWK 的变量包括用户定义变量（User-defined Variables）、系统变量(System Variables)和字段变量（Field Variables）。分别说明如下：

4.1 用户定义变量

AWK 的变量不需要声明，也不需要初始化，直接使用。**每一个变量同时有一个字符值和数值，AWK 根据上下文环境决定作为数值或字符串处理，这是 AWK 极其独特之处。**AWK 自动将变量初始化为空值，如果用作数字将作为 0。如果需要强制使用字符串，可使用 number “”（空格在 AWK 中是字符串连接符），若需要强制用作数值，可以用 string + 0。

另外应注意，**在 AWK 中，$表示字段，用户变量不需要加$，这是 AWK 与 shell 或者 Perl 不同之处！**在 shell 中，变量定义时不加$，再次引用时则需要用$，而在 Perl 中，无论定义和引用时都需要加$ (Perl 中$表示标量，另有@和%符号表示数组和 Hash 变量)。

4.2 系统变量

AWK 中包括两种系统变量:默认值可被改变的变量和处理过程中变量。AWK 的系统变量列表如下：

变量	意　义	默认值
FS	字段分割符	**“ [\t]+ ”**
OFS	输出字段分隔符	“　”
RS	记录分割符	**”\n”**
ORS	输出记录分隔符	**”\n”**
OFMT	输出数字格式	**”%.6g”**
CONVFMT	控制数值向字符串转换	**”%.6g”**
NF	字段数	-
NR	当前输入记录数（总数），**只有到 END 时，NR 才等于总记录数**	-
FNR	当前文件的相对记录数	
FILENAME	当前输入文件名	
ARGC	命令行参数数目	
ARGV	命令行参数数组	
ENVIRON	环境变量数组	
RSTART	match()匹配到的初始位置	
RLENGTH	match()匹配到的字符串长度	
SUBSEP	数组分隔符。将(I,J) 转换为 I SUBSEP J，模拟多维数组用	

注意：

- **ARGV 数组由 ARGV[0] ,…,ARGV[ARGC-1]组成，第一个元素指标是 0 而不是 1。这与 AWK 中的一般数组不同，而与 C 一致。**
- **ENVIRON 数组在 shell 与 AWK 的交互中非常有用。使用 ENVIRON[“PARA_NAME”] 来获取环境变量$PARA_NAME 的值，其中的引号””不可少！**

4.3 字段变量

从$1,$2 一直到$NF，整行用$0 标识。**注意，如果$0 被赋予新值，所有的$1，$2,..和 NF 都会被重新计算。同样，若$i,被改变，$0 将用 OFS 重新计算。**

5 数组

AWK 提供一维数组来存储字符串和数值。数组和数组元素不需要声明，也不需要指定元素个数。A**WK 特别之处是数组下标总是字符串型的，所以 AWK 数组总是关联数组（Associative Arrays），相当于 Perl 的 Hash Array。这一点是 AWK 区别于 C 及 Perl 之处。C 的数组下标是整数，Perl 分别普通数组和 Hash 数组。**

遍历数组的命令是：

```
for (variable in array)
        statement
```

注意数组下标的输出次序是依赖于 AWK 的实现的！

测试数组中是否存在某元素的命令是：

```
if ( subscript in A )
```

注意如果用 if (A[subscript] == “”) 命令，将可能创建一个新元素！

删除数组元素的命是：delete array[subscript]

用 for (i in array) delete array[i]；可以删除所的有元素，但是新版的 gawk 可以用简单的 delete array；命令删除整个数组。

AWK 的多维数组是用一维数组来模拟的。比如，可以用

```
for (i = 1; i <=10; i++)
        for (j = 1; j <=10; j++)
                arr[i, j ] = 0
```

if ((i, j) in arr) 方式来使用多维数组。AWK 实际用 arr[i SUBSEP j]来代替 arr[i, j]。

如果要实现多维数组的循环，可以采取以下方式：

```
for (k in arr) {
    split(k, x, SUBSEP)
    i=x[1]
    j=x[2]
    …(i,j)
}
```

6 操作符

AWK 的操作符基本与 C 语言相同，但也有些例外。按优先级从低至高列表如下：

操 作	符 号	说 明
赋值	= += -= *= /= %= ^=	^=是特别的操作符，C 没有

操 作	符 号	说 明
条件	?:	
逻辑或	||	
逻辑与	&&	
数组元素	**in**	**if (i in a)**
匹配	**~ !~**	**C 没有**
关系	< <= == != >= >	
字符串连接	**(blank)**	**“a” “ab” (=”abc”)**
加减	+ -	
乘除求模	* / %	
单目加减	+ -	-x 正负
逻辑非	!	!$1
开方	**^ (**)**	**有些现代版本的 awk 可以用****
自加、自减	++ --	++x x++
字段运算符	**$**	**$i 第 i 字段**
括号	**()**	

7 流程控制

AWK 的流程控制基本沿用 C 的流程控制语句，但是没有 do until, switch 语句。AWK 的流程控制语句如下：

{ statement }

if (expression) statement

if (expression) statement1 else statment2

while (expression) statement

for(expression1; expression2;expression3) statement

do statement while (expression)

for (variable in array) statement （这是 AWK 特别的对于数组中每个变量循环）

break (退出 while for do 循环)

continue (进入到下一 while for do 循环)

next (AWK 特别的指令，开始进入下一主输入循环，处理下一行记录，非常有用)

exit

exit expression （马上进入 END 流程；如果在 END 流程内，结束程序，以 expression 值为返回值）

8 函数

AWK 的函数包括系统函数和自定义函数。系统函数又可分为算术函数和字符串函数

8．1 算术函数

函 数	说 明
cos(x)	
sin(x)	
int(x)	求整，截去而不是四舍五入
log(x)	
exp(x)	
sqrt(x)	
atan2(y,x)	Argtan(y/x)
rand()	
srand()	

8.2 字符串函数

函 数	说 明
index(s,t)	返回 t 在 s 中的第一个位置，如果没找到返回 0
length(s)	返回 s 的字符个数
substr(s,p)	返回 s 中从 p 开始的所有字符串
substr(s,p,n)	返回 s 中从 p 开始的 n 个字符 比如 substr(“abcd”，1，3)将得到“abc”
gsub(r,s)	替换$0 中所有的正则表达式 r 为字符串 s,返回替换数
gsub(r,s,t)	替换字符串 t 中所有的正则表达式 r 为字符串 s
sub(r,s)	替换$0 中左边最长的匹配 r 的子串
sub(r,s，t)	替换 t 中左边最长的匹配 r 的子串
split(s,a)	用 FS 切割字符串 s 为数组 a,返回字段数
split(s,a,fs)	用 fs 切割字符串 s 为数组 a,返回字段数 如使用 split(“7/4/76”，arr，“/”)后， arr[“1”]为 7，arr[“2”]为 4， arr[“3”]为 76

spintf(fmt,expr-list)	格式化输出
strtonum(str)	GAWK 扩展，返回 str 的数组，可认八进制(0 开头)和十六进制(0x 开头)
asort(source|,desc)	GAWK 扩展，对数组内容排序，若指定 desc,拷贝排序内容到 desc,source,返回元素个数
asorti(source|,desc)	GAWK 扩展。按指标进行排序
tolower(string)	转换为小写
toupper(string)	转换为大写
dcgettext(string[,domain], [,category])	GAWK 国际化函数
dcngettext(string1,string2, number[,domain][,category] [,category])	GAWK 国际化函数
bindtextdomain(directory [,domain]	指定信息转换信息的目录

8.3 字节处理函数

函 数	说 明
and(v1,v2)	GAWK 扩展，与
or(v1,v2)	GAWK 扩展，或
xor(v1,v2)	GAWK 扩展，异或
compl(val)	GAWK 扩展,val 的位补函数
lshift(val,count)	GAWK 扩展,左移
rshift(val,count)	GAWK 扩展,右移

8.4 时间函数

函 数	说 明
systime()	**GAWK 扩展，timestamp,1970 年开始的时间戳**
mktime(datespec)	**将“ YYYY MM DD HH MM SS [DST] ”格式的 string 转化为时间戳**
strftime(format [,timestamp])	**转化 timestamp 为字符串，默认 format 是“ %a %b %d %H:%M:%S %Z %Y ”,格式与 ISO C 1999 的定义一致。**

8.5 用户自定义函数

用户定义函数的格式是：

```
function name(parameter-list){
   statement
}
```

注意： AWK 自定义函数的变量是带值传递，数组是以引用方式传递。另一特别的地方是没在参数列表的变量是全局的， 因此定义函数私有变量的方式是将其加入到变量列表中。

比如将字符型 IP 转换为整型 IP 的函数如下：

```
function aton(char_ip,    int_ip,    arr){
      split(char_ip, arr,    ".")
      int_ip = (((arr[1] * 256 + arr[2]) * 256) + arr[3]) * 256 + arr[4]
      return int_ip
}
```

9 输入输出

9.1 输入

AWK 的输入可以通过命令行文件输入，也有通过 getline 函数输入。

getline 的使用方法如下：

函 数	说 明
getline	$0, NF, NR, FNR
getline var	var, NR, FNR
getline <file	$0, NF
getline var <file	Var
cmd | getline	$0, NF
cmd | getline var	var

注意：

- **while (getline <"file") 是危险的使用循环的方式，可能进入死循环（如果"file"不存在的话。安全的使用方式是 while(getline <"file" > 0)。**
- **处理文件时应注意 AWK 的 RS 默认为"\n"，它不处理 Windows 格式文件的 " \r\n"中的"\r"，所以最后一个字段会带一个 " \r " 。这在输出时为产生非常怪异的结果。因此在处理 Windows 传过来的文件时最好先用 dos2unix 命令转一下文件格式。**

9.2 输出

AWK 的输出语句如下：

函 数	说 明
print	在标准输出输出 $0
print expression, expression, …	以 OFS 分隔输出 expressions
print expression, expression, … >filename	输出到文件 filename 中
print expression, expression, … >>filename	追加到文件 filename 中
print expression, expression, … |command	输出到 command 的标准输入中
printf(format,expression, expression, …)	带格式输出，格式定义与 C 相同
printf(format,expression, expression, …) >filename	带格式输出
printf(format,expression, expression, …) >>filename	带格式输出
printf(format,expression, expression, …) |command	带格式输出
close(filename), close(command)	中断 print 与 filename 或 command 的联系
system(command)	执行任务 command

10 总结

AWK 适用于管理小型的个人的数据库、生成报表、验证数据的合法性、生成索引和进行其它文本准备工作以及试验算法。GAWK 可以很容易地获取位和字段数据、数据排序以及进行简单的网络通信[2]。尽管 AWK 是一种小众语言，但是其在格式化数据的处理方面至今起着不可替代的作用。

11. 参考文献

【1】 Alfred V.Aho, Brian W.Kerninghan and Peter J.Weinberger. The AWK Programming Language. Addison-Wesley Publishing Company. 1988

【2】 Dale Dougherty and Arnold Robbins. Sed & Awk, 2nd edition. O'Reilly. 1997

【3】 Arnold Robbins. Effective AWK Programming, 3rd edition. O'Reilly. 2001

廖海仁(liaohairen@gmail.com)

2008.12 初稿 2010.12 修改