R for bioinformatics, Strings and regular expression

HUST Bioinformatics course series

Wei-Hua Chen (CC BY-NC 2.0)

08 August, 2019

section 1: TOC

前情提要

Talks so far:

1. introduction to R
2. R language basics, part 1
3. R language basics, part 2
4. R language basics, part 3, factors
5. data wrangler, part 1
6. data wrangler, part 2

packages we have touched so far

1. tidyverse
   - dplyr
   - tidyr
   - ggplot2
2. readr
3. tibble
4. forcats ...

本次提要

`stringr`

1. basics
   - length
   - uppercase, lowercase
   - unite, separate
   - string comparisons, sub string
2. regular expression

section 2: contents

get ready for the class

```
library(tidyverse);
```

```
## -- Attaching packages ---------------------------------------------------------
```

```
## v ggplot2 3.2.0     v purrr   0.3.2
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
```

```
## -- Conflicts ------------------------------------------------------------------
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(stringr);
```

其它著名的 packages

- stringi

section 2: simple manipulations ...

string

```
string1 <- "This is a string";
string2 <- 'If I want to include a "quote" inside a string, I use single quotes';


( string3 <- "a multiline
string" );
```

```
## [1] "a multiline \nstring"
```

```
## 注意与上面的区别
writeLines( string3 );
```

```
## a multiline
## string
```

quotes & other special characters

```
( double_quote <- "\"" );
```

```
## [1] "\""
```

```
( single_quote <- '\'' );
```

```
## [1] "'"
```

```
( x <- "\u00b5" )
```

```
## [1] "µ"
```

```
## 注意不同!!!!
( y <- "\\" )
```

```
## [1] "\\"
```

```
writeLines( y );
```

```
## \
```

string length

```
## 系统自带
nchar( c("a", "R for data science", NA) );
```

```
## [1]  1 18 NA
```

```
## stringr
str_length(c("a", "R for data science", NA));
```

```
## [1]  1 18 NA
```

string combine

```
## 系统自带
paste( "a", "b", "c", sep = "" );
```

```
## [1] "abc"
```

```
## stringr
str_c( "a", "b", "c" );
```

```
## [1] "abc"
```

```
paste( c( "a", "b", "c" ), 2, sep = "" );
```

```
## [1] "a2" "b2" "c2"
```

```
str_c( c( "a", "b", "c" ), 2 );
```

```
## [1] "a2" "b2" "c2"
```

string comparison

```
## direct comparison ; 可用于排序 ...
"A" > "abc";
```

```
## [1] FALSE
```

```
##
library(pracma);
```

```
##
## Attaching package: 'pracma'

## The following object is masked from 'package:purrr':
##
##     cross
```

```
strcmp( "chen", "chenweihua" );
```

```
## [1] FALSE
```

```
strcmpi( "chen", "CHEN" );
```

```
## [1] TRUE
```

other simple functions

```
toupper( letters[1:10] );
```

```
##  [1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J"
```

```
tolower( LETTERS[1:5] );
```

```
## [1] "a" "b" "c" "d" "e"
```

```
library(stringi);
stri_reverse( "ABC" );
```

```
## [1] "CBA"
```

tricks

- stringi package 里的 function 都以 stri_ 开头
- strinr 则以 str_ 开头

section 3: regular expression basics

what is regular expression (正则表达式) ?

在介绍更多 string manipulation 函数之前，先介绍正则表达式:

a sequence of characters that define a search pattern.

```
## 比如: [ab] 表示寻找 a 或 b
c( "abc", "chen", "liu", "blah" ) %>% str_subset( "[ab]" );
```

```
## [1] "abc"  "blah"
```

```
## 匹配并取出字符中间的数字
c( "a1334bc", "ch13e_45n", "liu", "bl00ah" ) %>% str_extract( "\\d+" );
```

```
## [1] "1334" "13"   NA     "00"
```

useful tools

https://regexr.com/ https://regex101.com/

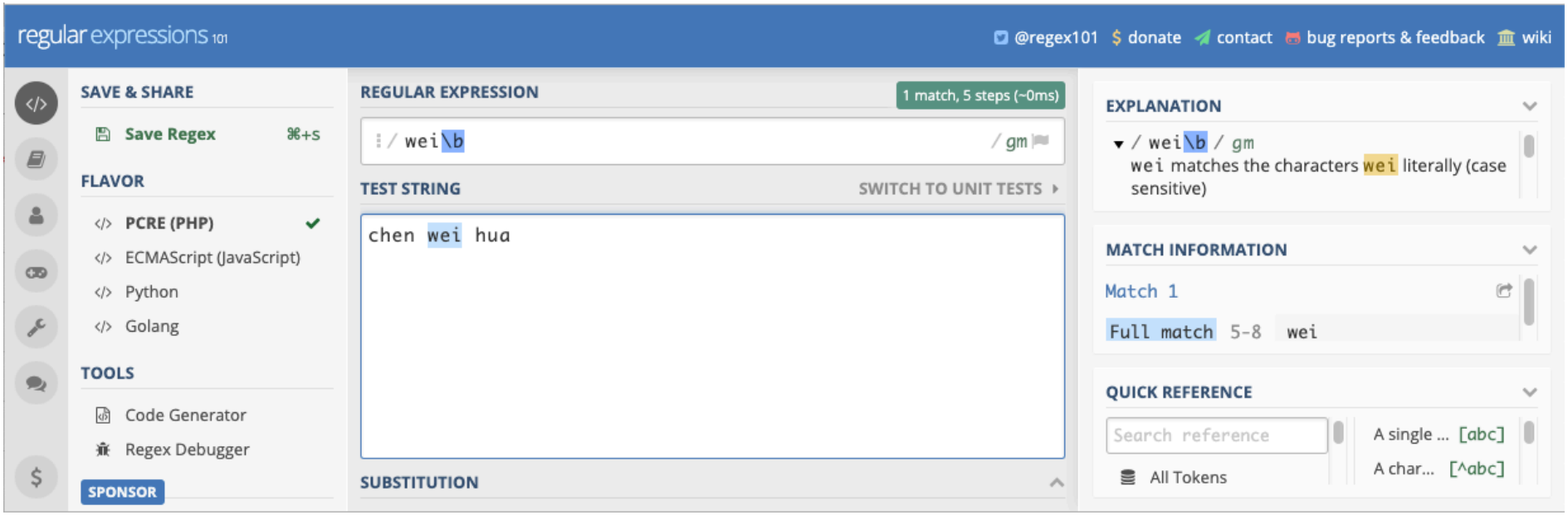


Figure 1: regex101

正则表达式的组成部分

1. 字符规则（Character classes）：（不）匹配什么样的字符

Character Classes	
[[:digit:]] or \\d	Digits; `[0-9]`
\\D	Non-digits; `[^0-9]`
[[:lower:]]	Lower-case letters; `[a-z]`
[[:upper:]]	Upper-case letters; `[A-Z]`
[[:alpha:]]	Alphabetic characters; `[A-z]`
[[:alnum:]]	Alphanumeric characters `[A-z0-9]`
\\w	Word characters; `[A-z0-9_]`
\\W	Non-word characters
[[:xdigit:]] or \\x	Hexadec. digits; `[0-9A-Fa-f]`
[[:blank:]]	Space and tab
[[:space:]] or \\s	Space, tab, vertical tab, newline, form feed, carriage return
\\S	Not space; `[^[:space:]]`
[[:punct:]]	Punctuation characters; `!"#$%&'()*+,-./:;<=>?@[]^_`{|}~`
[[:graph:]]	Graphical char.; `[[:alnum:][:punct:]]`
[[:print:]]	Printable characters; `[[:alnum:][:punct:]\\s]`
[[:cntrl:]] or \\c	Control characters; `\n`, `\r` etc.

Figure 2: 字符规则

示例

```
"abc_123_??$$^" %>% str_extract( "\\s+" ); ## 此字串包括 空格 吗？
```

```
## [1] NA
```

```
"abc_123_??$$^" %>% str_extract( "\\d+" ); ## 数字 ??
```

```
## [1] "123"
```

```
"abc_123_??$$^" %>% str_extract( "\\w+" ); ## [A-z0-9_]
```

```
## [1] "abc_123_"
```

str_extract：取出第一个匹配的部分

2. 匹配位置

Anchors	
^	Start of the string
$	End of the string
\\b	Empty string at either edge of a word
\\B	NOT the edge of a word
\\<	Beginning of a word
\\>	End of a word

Figure 3: 匹配位置

示例

```
## 以 wei 结束的字符串
c("chen wei hua", "chen wei", "chen") %>% str_subset( "wei$" );
```

```
## [1] "chen wei"
```

```
## 以 wei 结束的字
c("chen wei hua", "chen wei", "chen") %>% str_subset( "wei\\b" );
```

```
## [1] "chen wei hua" "chen wei"
```

3. 匹配数量

Quantifiers	
*	Matches at least 0 times
+	Matches at least 1 time
?	Matches at most 1 time; optional string
{n}	Matches exactly n times
{n,}	Matches at least n times
{,n}	Matches at most n times
{n,m}	Matches between n and m times

Figure 4: 匹配数量

示例

```
##
"1234abc" %>% str_extract( "\\d+" );
```

```
## [1] "1234"
```

```
"1234abc" %>% str_extract( "\\d{3}" );
```

```
## [1] "123"
```

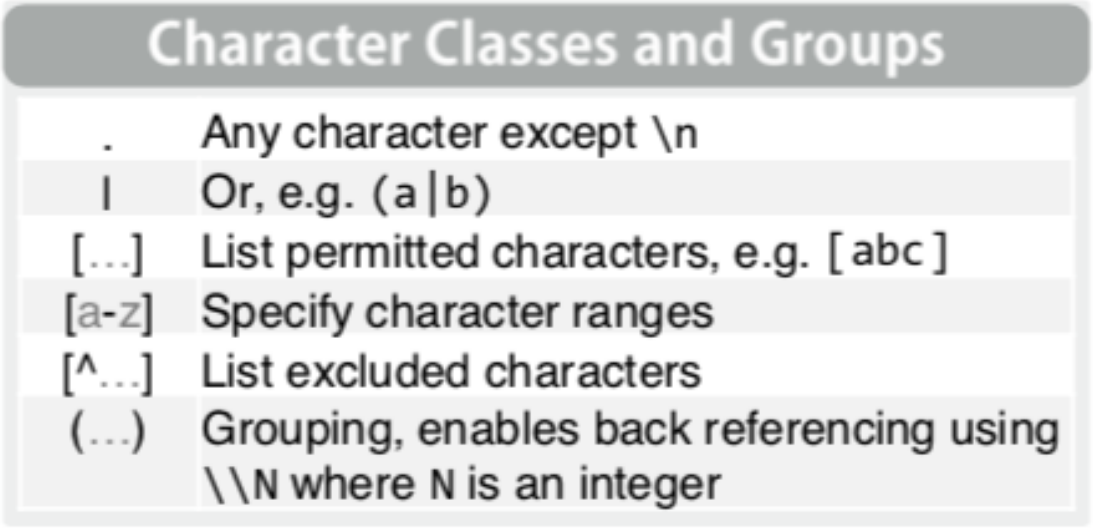
```
"1234abc" %>% str_extract( "\\d{5,6}" );
```

```
## [1] NA
```

```
"1234abc" %>% str_extract( "\\d{2,6}" );
```

```
## [1] "1234"
```

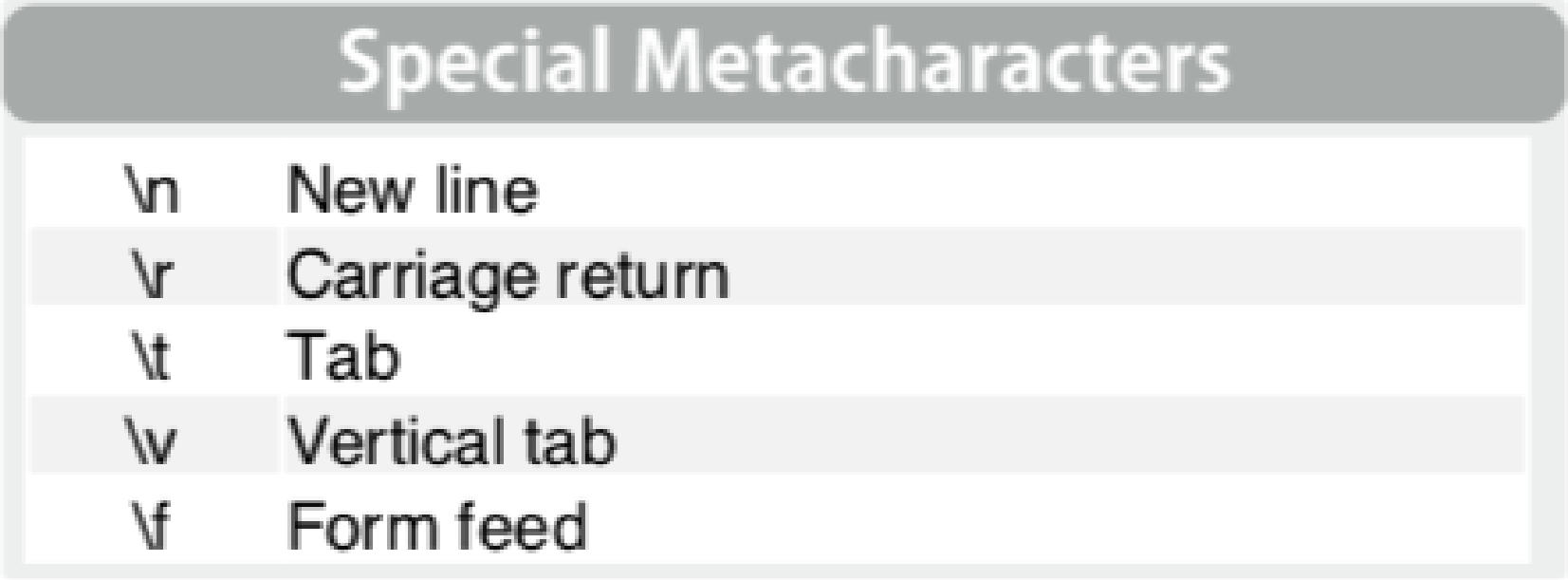
4. classes and groups



Character Classes and Groups	
.	Any character except \n
|	Or, e.g. (a|b)
[...]	List permitted characters, e.g. [abc]
[a-z]	Specify character ranges
[^...]	List excluded characters
(...)	Grouping, enables back referencing using \\N where N is an integer

Figure 5: classes and groups

5. 特别字符



Special Metacharacters	
\n	New line
\r	Carriage return
\t	Tab
\v	Vertical tab
\f	Form feed

Figure 6: 特别字符

more to read and excercise

1. regular expression basic
2. Regular Expressions Fundamentals: excercise 1
3. Data wrangling : Cleansing: excercise 1
4. Data wrangling : Cleansing: excercise 2
5. Data wrangling : Cleansing: excercise 3

section 4: tasks of regular expression

tasks of regular expression

1. detect patterns： 检查目标 string 里有无 pattern

```
grep( "\\d+", c( "123", "abc", "wei555hua" ) ); ##
```

```
## [1] 1 3
```

```
grepl( "\\d+", c( "123", "abc", "wei555hua" ) ); ##
```

```
## [1]  TRUE FALSE  TRUE
```

```
c( "123", "abc", "wei555hua" ) %>% str_detect( "\\d+" );
```

```
## [1]  TRUE FALSE  TRUE
```

2. locate patterns (定位)

```
regexpr( "\\d+", c( "123", "abc", "wei555hua" ) ); ##
```

```
## [1]  1 -1  4
## attr(,"match.length")
## [1]  3 -1  3
## attr(,"index.type")
## [1] "chars"
## attr(,"useBytes")
## [1] TRUE
```

```
 c( "123", "abc", "wei555hua" ) %>% str_locate( "\\d+" );
```

```
##      start end
## [1,]     1   3
## [2,]    NA  NA
## [3,]     4   6
```

3. extract patterns (抽取匹配的字串)

```
c( "123", "abc", "wei555hua" ) %>% str_extract ( "\\d+" );
```

```
## [1] "123" NA    "555"
```

```
c( "123", "abc", "wei555hua" ) %>% str_match ( "\\d+" );
```

```
##      [,1]
## [1,] "123"
## [2,] NA
## [3,] "555"
```

?? str_extract`` 和 str_match "' 的区别在哪 ??

4. replace patterns (匹配并替换)

```
str_replace( c( "123", "abc", "wei555hua" ) , "\\d+", "###");
```

```
## [1] "###"       "abc"       "wei###hua"
```

其它函数

- sub(pattern, replacement, string)
- gsub(pattern, replacement, string)
- stringr::str_replace_all(string, pattern, replacement)

更多请见R regular expression cheatsheet.

stringr 函数

更多内容见这里：https://stringr.tidyverse.org

示例

```
( dat <-
tibble(chrom = readLines(textConnection("chr11:69464719-69502928
chr7:55075808-55093954
chr8:128739772-128762863
chr3:169389459-169490555
chr17:37848534-37877201
chr19:30306758-30316875
chr1:150496857-150678056
chr12:69183279-69260755
chr11:77610143-77641464
chr8:38191804-38260814
chr12:58135797-58156509")) ) );
```

```
## # A tibble: 11 x 1
##    chrom
##    <chr>
##  1 chr11:69464719-69502928
##  2 chr7:55075808-55093954
##  3 chr8:128739772-128762863
##  4 chr3:169389459-169490555
##  5 chr17:37848534-37877201
##  6 chr19:30306758-30316875
##  7 chr1:150496857-150678056
##  8 chr12:69183279-69260755
##  9 chr11:77610143-77641464
## 10 chr8:38191804-38260814
## 11 chr12:58135797-58156509
```

示例，cont.

任务：分为三列，chr, start, end

```
dat$chrom %>% str_split( '[:-]', simplify = T );
```

```
##       [,1]    [,2]        [,3]
##  [1,] "chr11" "69464719"  "69502928"
##  [2,] "chr7"  "55075808"  "55093954"
##  [3,] "chr8"  "128739772" "128762863"
##  [4,] "chr3"  "169389459" "169490555"
##  [5,] "chr17" "37848534"  "37877201"
##  [6,] "chr19" "30306758"  "30316875"
##  [7,] "chr1"  "150496857" "150678056"
##  [8,] "chr12" "69183279"  "69260755"
##  [9,] "chr11" "77610143"  "77641464"
## [10,] "chr8"  "38191804"  "38260814"
## [11,] "chr12" "58135797"  "58156509"
```

示例，cont.

另一种解决方案

```
library(tidyr)
extract(dat, chrom, into=c('chr', 'chrStart', 'chrEnd'),
        '([^:]+):([^-]+)-(.*)', convert=TRUE);
```

```
## # A tibble: 11 x 3
##    chr    chrStart    chrEnd
##    <chr>     <int>     <int>
##  1 chr11  69464719  69502928
##  2 chr7   55075808  55093954
##  3 chr8  128739772 128762863
##  4 chr3  169389459 169490555
##  5 chr17  37848534  37877201
##  6 chr19  30306758  30316875
##  7 chr1  150496857 150678056
##  8 chr12  69183279  69260755
##  9 chr11  77610143  77641464
## 10 chr8   38191804  38260814
## 11 chr12  58135797  58156509
```

section 5: Exercise and home work

more to read

https://r4ds.had.co.nz/strings.html

下次预告

data iteration & parallel computing