R for bioinformatics, Strings and regular expression

HUST Bioinformatics course series

Wei-Hua Chen (CC BY-NC 4.0)

09 October, 2024

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: simple string manipulations ...

get ready for the class

```
library(tidyverse);
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

```
library(stringr);
```

其它著名的 packages

- stringi

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/

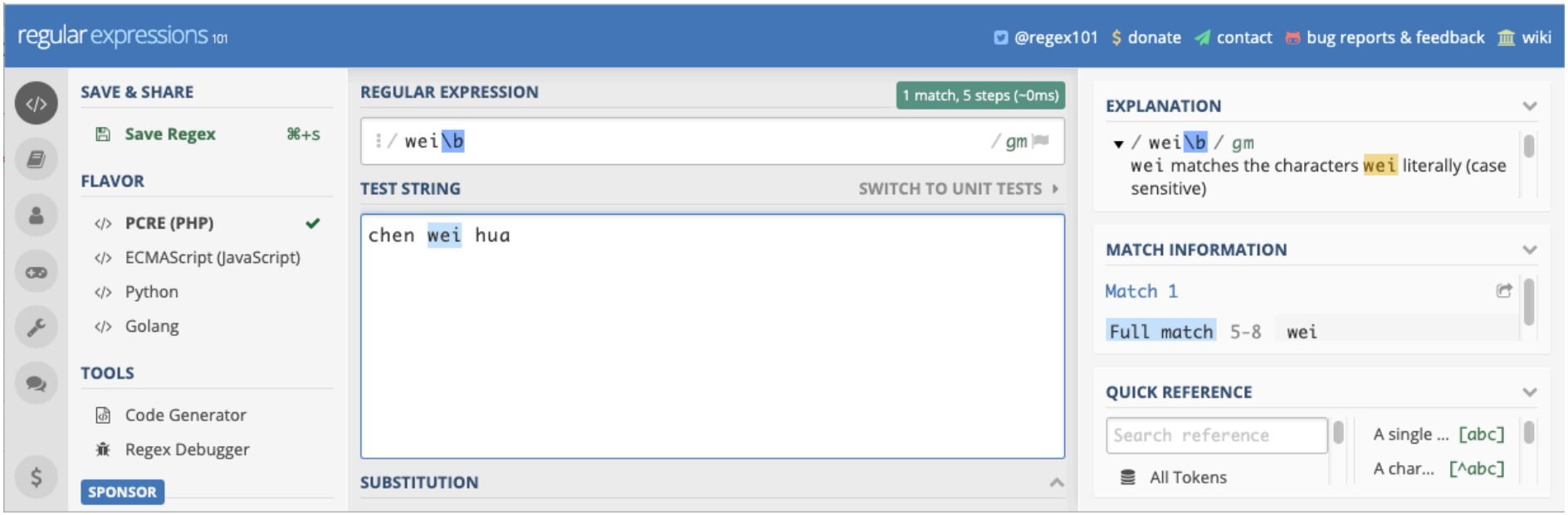


Figure 1: regex101

正则表达式的任务

1. 匹配模式
2. 匹配规则：匹配（或不匹配）什么样的字符
3. 位置规则：在何处匹配（或不匹配）
4. 数量规则：符合规则字符串的数量

1. 匹配模式

用于定义模糊匹配的模式

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.

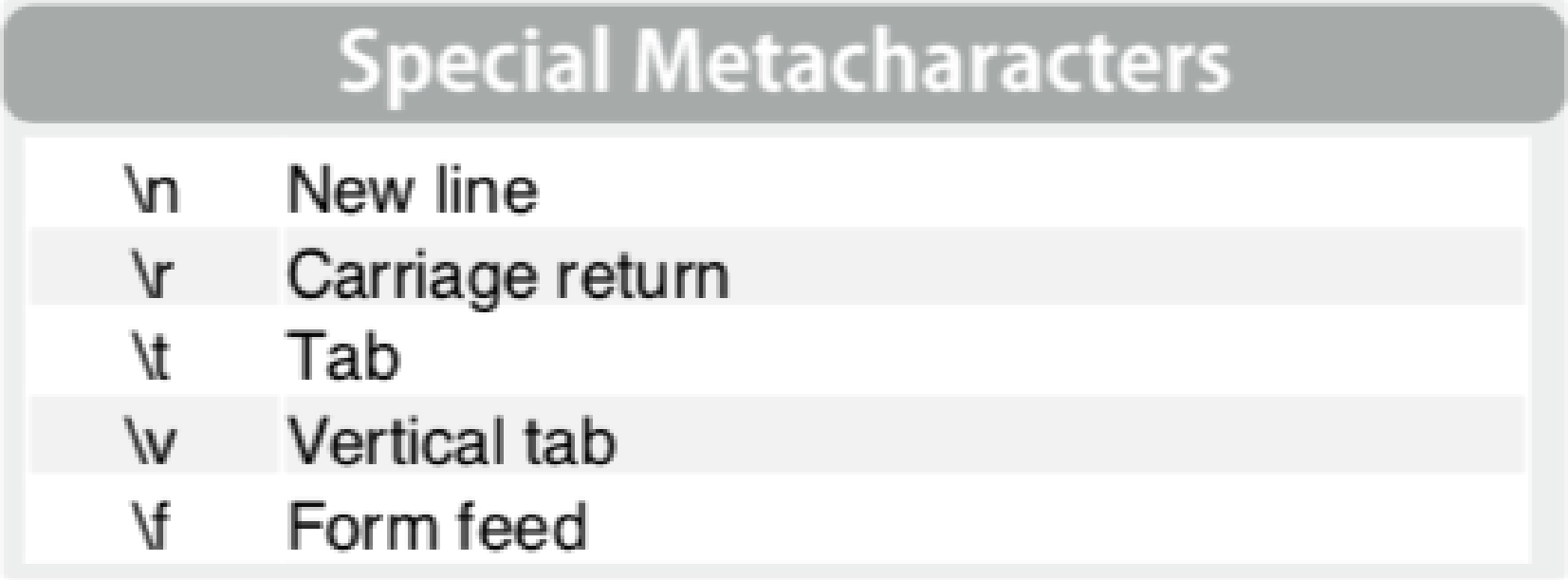
Figure 2: 匹配模式 1

1. 匹配模式 2: 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 3: classes and groups

1. 匹配模式 3：特别字符



Special Metacharacters

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

Figure 4: 特别字符

2. 匹配规则：匹配什么样的字符

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

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

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

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

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

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

```
( string3 <- "a multiline
string" ); ## 含有回车的字符串
```

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

```
string3 %>% str_detect( "\n" ); ##
```

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

2. 匹配规则：不匹配什么样的字符

```
"abc_123_??$$^" %>% str_detect( "\\s+", negate = T ); ## 此字串 不包括 空格 吗？
```

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

```
"abc_123_??$$^" %>% str_detect( "\\d+", negate = T ); ## 数字 ??
```

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

```
"abc_123_??$$^" %>% str_detect( "\\w+" , negate = T); ## [A-z0-9_]
```

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

```
( string3 <- "a multiline
string" ); ## 含有回车的字符串
```

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

```
string3 %>% str_detect( "\n", negate = T); ##
```

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

2. 匹配规则：不匹配的其它表达方法

```
!str_detect( "abc_123_??$$^", "\\s+" ); ## 此字串 不包括 空格 吗？
```

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

```
!str_detect( "abc_123_??$$^", "\\d+" ); ## 数字 ??
```

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

```
!str_detect( "abc_123_??$$^", "\\w+" ); ## [A-z0-9_]
```

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

```
( string3 <- "a multiline
string" ); ## 含有回车的字符串
```

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

```
!str_detect( string3, "\n"); ##
```

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

2. 位置规则：在何处匹配（或不匹配）

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 5: 匹配位置

位置匹配：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"
```

位置不匹配：示例

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

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

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

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

3. 数量规则：规定符合规则字符串的数量

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 6: 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"
```

section 4: tasks of regular expression

tasks of regular expression

1. detect patterns (检测)：检查目标 string 里有无 pattern
2. locate patterns (定位)
3. extract patterns (抽取匹配的字串)
4. replace patterns (替换)
5. split by patterns (分割)

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
```

1. detection patterns, cont.

count patterns：统计匹配的数量

```
x <- c("why", "video", "cross", "extra", "deal", "authority");
str_detect(x, "[aeiou]");
```

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

```
str_count(x, "[aeiou]");
```

```
## [1] 0 3 1 2 2 4
```

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 ??

str_extract vs. str_match

```
x;
```

```
## [1] "why"       "video"     "cross"     "extra"     "deal"      "authority"
```

```
str_extract(x, "[aeiou]");
```

```
## [1] NA  "i" "o" "e" "e" "a"
```

```
str_match(x, "(.)[aeiou](.)"); ## extract the characters on either side of the vowel ????
```

```
##      [,1]  [,2] [,3]
## [1,] NA    NA   NA
## [2,] "vid" "v"  "d"
## [3,] "ros" "r"  "s"
## [4,] NA    NA   NA
## [5,] "dea" "d"  "a"
## [6,] "aut" "a"  "t"
```

?? 为什么有 3 个输出 ???

str_extract_all 和 str_match_all

```
x;
```

```
## [1] "why"       "video"     "cross"     "extra"     "deal"      "authority"
```

```
str_extract_all( x, "[aeiou]+" );
```

```
## [[1]]
## character(0)
##
## [[2]]
## [1] "i"  "eo"
##
## [[3]]
## [1] "o"
##
## [[4]]
## [1] "e" "a"
##
## [[5]]
## [1] "ea"
##
## [[6]]
## [1] "au" "o"  "i"
```

```
str_match_all( x, "[aeiou]+" );
```

4. replace patterns (匹配并替换)

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

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

```
str_replace_all( "123_abc_456_789" , "\\d+", "###");
```

```
## [1] "###_abc_###_###"
```

5. split by patterns (匹配并分割)

```
str_split(x, "");
```

```
## [[1]]
## [1] "w" "h" "y"
##
## [[2]]
## [1] "v" "i" "d" "e" "o"
##
## [[3]]
## [1] "c" "r" "o" "s" "s"
##
## [[4]]
## [1] "e" "x" "t" "r" "a"
##
## [[5]]
## [1] "d" "e" "a" "l"
##
## [[6]]
## [1] "a" "u" "t" "h" "o" "r" "i" "t" "y"
```

其它字符串函数

- sub(pattern, replacement, string)
- gsub(pattern, replacement, string)
- stringr::str_replace_all(string, pattern, replacement)
- fixed(): match exact bytes
- coll(): match human letters
- boundary(): match boundaries

更多请见 `cheatsheets/` 目录下的:

- strings.pdf
- regular_expression.pdf

更多：stringr 包内的函数

更多内容见这里：https://stringr.tidyverse.org

高级应用示例

```
( 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
```

高级应用示例，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"
```

高级应用示例，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
```

```
## 注：直接 copy & paste 代码可能造成特殊字符有问题，比如 ^
```

section 5: Exercise and home work

小结

今次提要

1. string basics
   - length
   - uppercase, lowercase
   - unite, separate
   - string comparisons, sub string
2. regular expression
   - detect patterns
   - locate patterns
   - extract patterns
   - replace patterns
   - split patterns ...

下次预告

data iteration & parallel computing

练习 & 作业

- `Exercises and homework` 目录下 `talk07-homework.Rmd` 文件;
- 完成时间：见钉群的要求