

Xử lý ký tự với stringr : : CHEAT SHEET



Thư viện stringr cung cấp các công cụ hữu ích để làm việc với kiểu dữ liệu định dạng ký tự.

Tìm kiếm ký tự phù hợp

str_detect(string, **pattern**) Kiểm tra chuỗi có chứa ký tự cho trước.
`str_detect(fruit, "a")`

str_which(string, **pattern**) Tìm vị trí của chuỗi trong vector chứa ký tự cho trước.
`str_which(fruit, "a")`

str_count(string, **pattern**) Đếm ký tự cho trước trong chuỗi.
`str_count(fruit, "a")`

str_locate(string, **pattern**) Tìm vị trí chính xác của chuỗi có chứa ký tự cho trước. Xem **str_locate_all**.
`str_locate(fruit, "a")`

Trích xuất ký tự

str_sub(string, start = 1L, end = -1L) Trích xuất ký tự theo vị trí.
`str_sub(fruit, 1, 3); str_sub(fruit, -2)`

str_subset(string, **pattern**) Trả ra chuỗi trong vector có chứa ký tự cho trước.
`str_subset(fruit, "b")`

str_extract(string, **pattern**) Trả ra ký tự đầu tiên phù hợp với điều kiện lọc dưới dạng vector. Xem thêm **str_extract_all**.
`str_extract(fruit, "[aeiou]")`

str_match(string, **pattern**) Trả ra chuỗi ký tự đầu tiên phù hợp với điều kiện lọc. Xem thêm **str_match_all**.
`str_match(sentences, "(a|the) ([^]+)")`

Quản lý độ dài của chuỗi

str_length(string) Số lượng ký tự có trong 1 chuỗi.
`str_length(fruit)`

str_pad(string, width, side = c("left", "right", "both"), pad = " ") Chèn thêm ký tự để chuỗi có độ dài cố định.
`str_pad(fruit, 17)`

str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...") Tính lược ký tự trong chuỗi.
`str_trunc(fruit, 3)`

str_trim(string, side = c("both", "left", "right")) Cắt ký tự trắng bắt đầu hoặc kết thúc của chuỗi.
`str_trim(fruit)`

Biến đổi chuỗi

str_sub() <- value. Thay chuỗi con bằng giá trị mới.
`str_sub(fruit, 1, 3) <- "str"`

str_replace(string, **pattern**, replacement) Thay thế ký tự đầu tiên trong chuỗi thỏa mãn điều kiện.
`str_replace(fruit, "a", "-")`

str_replace_all(string, **pattern**, replacement) Thay thế tất cả ký tự trong chuỗi thỏa mãn điều kiện.
`str_replace_all(fruit, "a", "-")`

str_to_lower(string, locale = "en")¹ Chuyển sang chữ thường.
`str_to_lower(sentences)`

str_to_upper(string, locale = "en")¹ Chuyển sang chữ hoa.
`str_to_upper(sentences)`

str_to_title(string, locale = "en")¹ Chuyển sang định dạng tên.
`str_to_title(sentences)`

Gộp và tách chuỗi

str_c(..., sep = "", collapse = NULL) Gộp nhiều ký tự thành một.
`str_c(letters, LETTERS)`

str_c(..., sep = "", collapse = NULL) Gộp các ký tự/chuỗi trong một véc-tơ thành một chuỗi duy nhất.
`str_c(letters, collapse = "")`

str_dup(string, times) Lặp lại chuỗi.
`str_dup(fruit, times = 2)`

str_split_fixed(string, **pattern**, n) Tách một véc-tơ chuỗi với điều kiện và số lượng chuỗi con xác định trước.
`str_split_fixed(fruit, " ", n=2)`

glue::glue(..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}") Tạo chuỗi từ chuỗi và sử dụng {} để thực hiện câu lệnh (nếu có).
`glue::glue("Pi is {pi}")`

glue::glue_data(.x, ..., .sep = "", .envir = parent.frame(), .open = "{", .close = "}") Sử dụng dataframe để tạo chuỗi mới và dùng {} để thực hiện câu lệnh.
`glue::glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")`

Sắp xếp ký tự

str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...) ¹ Trả ra kết quả là một véc-tơ thứ tự ký tự sau khi sắp xếp. `x[str_order(x)]`

str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...) ¹ Sắp xếp lại vị trí ký tự.
`str_sort(x)`

Các hàm hữu ích khác

str_conv(string, encoding) Thay đổi encoding của chuỗi. `str_conv(fruit, "ISO-8859-1")`

str_view(string, **pattern**, match = NA) Xem kết quả HTML với ký tự đầu tiên thỏa mãn điều kiện lọc.
`str_view(fruit, "[aeiou]")`

str_view_all(string, **pattern**, match = NA) Xem kết quả HTML với tất cả ký tự thỏa mãn điều kiện lọc.
`str_view_all(fruit, "[aeiou]")`

str_wrap(string, width = 80, indent = 0, exdent = 0) Co ngắn độ dài của chuỗi.
`str_wrap(sentences, 20)`

¹ Xem thêm bit.ly/ISO639-1.



Những điều cần biết

Điều kiện lọc trong stringr sẽ được diễn dịch thành biểu thức chính quy (regular expressions – viết tắt là regex).

Trong R, ta viết biểu thức chính quy (regex) dưới dạng chuỗi ký tự trong dấu ngoặc kép hoặc ngoặc đơn.

Không phải tất cả các ký tự đều có thể thể hiện trực tiếp trong R dưới dạng chuỗi. Một số trường hợp phải thể hiện dưới dạng ký tự đặc biệt với ý nghĩa riêng.

Ký tự	Ý nghĩa
\\	\
\"	"
\\n	Dòng mới

Thực hiện câu lệnh `?"` để xem thêm.

Do đó, bất cứ khi nào ký tự `\` xuất hiện trong regex, ta phải viết dưới dạng hai dấu gạch sổ `\\` trong chuỗi thể hiện của regex.

Sử dụng hàm `writeLines()` để xem R thực hiện chuỗi với các ký tự đặc biệt như thế nào

```
writeLines("\\ ")
# \
```

```
writeLines("\\ is a backslash")
# \ is a backslash
```

Giải thích

Điều kiện lọc trong stringr được diễn dịch thành biểu thức chính quy (regex). Để thay đổi chế độ mặc định, ta có thể sử dụng các hàm sau:

`regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...)` Điều chỉnh biểu thức chính quy, cho phép bỏ qua các điều kiện đặc biệt, cho phép R ghi chú bên trong một biểu thức chính quy.
`str_detect("I", regex("i", TRUE))`

`fixed()` Khớp các ký tự dưới dạng bytes, tuy nhiên có thể sẽ bị sót các ký tự có khả năng biểu diễn dưới nhiều hình thức khác nhau.
`str_detect("u0130", fixed("i"))`

`coll()` Khớp các ký tự dưới dạng bytes nhưng sử dụng locale để xác định chính xác ký tự.
`str_detect("u0130", coll("i", TRUE, locale = "tr"))`

`boundary()` Khớp với các ký tự phân biệt giữa câu, từ, dòng mới...
`str_split(sentences, boundary("word"))`

Biểu thức chính quy (regex) - Biểu thức chính quy hay regex là phương pháp chính xác để mô tả điều kiện lọc trong chuỗi

```
see <- function(rx) str_view_all("abc ABC 123\t!?\()\}n", rx)
```

Ký tự	Regex	Ý nghĩa	Ví dụ
	a (etc.)	a (etc.)	<code>see("a")</code>
<code>\\.</code>	<code>\.</code>	.	<code>see("\\.")</code>
<code>\\!</code>	<code>\\!</code>	!	<code>see("\\!")</code>
<code>\\?</code>	<code>\\?</code>	?	<code>see("\\?")</code>
<code>\\ </code>	<code>\\ </code>		<code>see("\\ ")</code>
<code>\\(</code>	<code>\\(</code>	(<code>see("\\(")</code>
<code>\\)</code>	<code>\\)</code>)	<code>see("\\)")</code>
<code>\\{</code>	<code>\\{</code>	{	<code>see("\\{")</code>
<code>\\}</code>	<code>\\}</code>	}	<code>see("\\}")</code>
<code>\\n</code>	<code>\\n</code>	Xuống dòng	<code>see("\\n")</code>
<code>\\t</code>	<code>\\t</code>	Tab	<code>see("\\t")</code>
<code>\\s</code>	<code>\\s</code>	Khoảng trắng	<code>see("\\s")</code>
<code>\\d</code>	<code>\\d</code>	Số (\\D với điều kiện không phải số)	<code>see("\\d")</code>
<code>\\w</code>	<code>\\w</code>	Chữ (\\W với điều kiện không phải chữ)	<code>see("\\w")</code>
<code>\\b</code>	<code>\\b</code>	Đường biên của một từ	<code>see("\\b")</code>
	[[:digit:]]	Số	<code>see("[[:digit:]]")</code>
	[[:alpha:]]	Chữ	<code>see("[[:alpha:]]")</code>
	[[:lower:]]	Viết thường	<code>see("[[:lower:]]")</code>
	[[:upper:]]	Viết hoa	<code>see("[[:upper:]]")</code>
	[[:alnum:]]	Số hoặc chữ	<code>see("[[:alnum:]]")</code>
	[[:punct:]]	Dấu	<code>see("[[:punct:]]")</code>
	[[:graph:]]	Chữ, số hoặc dấu	<code>see("[[:graph:]]")</code>
	[[:space:]]	Dấu cách	<code>see("[[:space:]]")</code>
	[[:blank:]]	Khoảng trắng	<code>see("[[:blank:]]")</code>
	.	Tất cả ký tự trừ dòng mới	<code>see(".")</code>

[[:space:]]
new line
space
tab

[[:blank:]]
space
tab

[[:graph:]]

[[:punct:]]
.,:;?!\\|/`='+-^_~" '[] { } () < > @ # \$

[[:alnum:]]

[[:digit:]]
0 1 2 3 4 5 6 7 8 9

[[:alpha:]]

[[:lower:]]	[[:upper:]]
a	A
b	B
c	C
d	D
e	E
f	F
g	G
h	H
i	I
j	J
k	K
l	L
m	M
n	N
o	O
p	P
q	Q
r	R
s	S
t	T
u	U
v	V
w	W
x	X
z	Z

Điều kiện alt <- function(rx) str_view_all("abcde", rx)

regex	Ý nghĩa	Ví dụ
<code>ab d</code>	Hoặc	<code>alt("ab d")</code>
<code>[abe]</code>	Một trong	<code>alt("[abe]")</code>
<code>[^abe]</code>	Tất cả trừ	<code>alt("[^abe]")</code>
<code>[a-c]</code>	Trong khoảng	<code>alt("[a-c]")</code>

Điều kiện chặn anchor <- function(rx) str_view_all("aaa", rx)

regex	Ý nghĩa	Ví dụ
<code>^a</code>	Bắt đầu chuỗi	<code>anchor("^a")</code>
<code>a\$</code>	Kết thúc chuỗi	<code>anchor("a\$")</code>

Tim mở rộng look <- function(rx) str_view_all("bacad", rx)

regex	Ý nghĩa	Ví dụ
<code>a(=?c)</code>	Tim a theo sau là c	<code>look("a(=?c)")</code>
<code>a(?!c)</code>	Tim a với ký tự đằng sau không phải là c	<code>look("a(?!c)")</code>
<code>(?<=b)a</code>	Tim a với ký tự đằng trước là b	<code>look("(?<=b)a")</code>
<code>(?!b)a</code>	Tim a với ký tự đằng trước không phải b	<code>look("(?!b)a")</code>

Đếm số lượng quant <- function(rx) str_view_all(".a.aa.aaa", rx)

regex	matches	example
<code>a?</code>	Không hoặc một	<code>quant("a?")</code>
<code>a*</code>	Không hoặc nhiều	<code>quant("a*")</code>
<code>a+</code>	Một hoặc nhiều	<code>quant("a+")</code>
<code>a{n}</code>	Chính xác n	<code>quant("a{2}")</code>
<code>a{n,}</code>	n hoặc nhiều hơn	<code>quant("a{2,}")</code>
<code>a{n,m}</code>	Giữa n và m	<code>quant("a{2,4}")</code>

Nhóm ref <- function(rx) str_view_all("abbaab", rx)

regex	Điều kiện	Ví dụ
<code>(ab d)e</code>	sets precedence	<code>alt("(ab d)e")</code>

Sử dụng dấu gạch sổ để lặp lại nhóm trong ngoặc đơn. Số thứ tự chính là thứ tự của nhóm.

Ký tự (câu lệnh)	regex (Ý nghĩa)	Diễn giải (Điều kiện lọc)	Ví dụ (Kết quả tương tự như ref("abba"))
<code>\\1</code>	<code>\\1 (etc.)</code>	Nhóm đầu tiên.	<code>ref("(a)(b)\\2\\1")</code>

