

Manipulação de string com stringr :: Folha de Referência



O pacote stringr fornece um conjunto de ferramentas consistentes para trabalhar com cadeia de caracteres (strings), ou seja, sequência de caracteres entre aspas.

Detectar Encontros

`str_detect(string, pattern, negate = FALSE)`
Detecta a presença de um padrão em uma string. Ver também `str_like()`. `str_detect(fruit, "a")`

`str_starts(string, pattern, negate = FALSE)`
Detecta a presença de um padrão no início da string. Ver também `str_ends()`. `str_starts(fruit, "a")`

`str_which(string, pattern, negate = FALSE)`
Find the indexes of strings that contain a pattern match. `str_which(fruit, "a")`

`str_locate(string, pattern)` Localiza a posição que o padrão foi encontrado em uma string. Ver também `str_locate_all()`. `str_locate(fruit, "a")`

`str_count(string, pattern)` Conta quantas vezes o padrão foi encontrado na string. `str_count(fruit, "a")`

Partes da String

`str_sub(string, start = 1L, end = -1L)` Extrair partes de uma string. `str_sub(fruit, 1, 3)`; `str_sub(fruit, -2)`

`str_subset(string, pattern, negate = FALSE)`
Retorna apenas as strings que contêm um padrão encontrado. `str_subset(fruit, "p")`

`str_extract(string, pattern)` Retorna apenas o primeiro padrão encontrado em cada string, como um vetor. Ver também `str_extract_all()` para retornar todos os padrões encontrados. `str_extract(fruit, "[aeiou]")`

`str_match(string, pattern)` Retorna o primeiro padrão encontrado em cada string como uma matriz com uma coluna para cada grupo do padrão. Ver também `str_match_all()`. `str_match(sentences, "(a[the] ([^+])")`

Gerenciar Comprimento

`str_length(string)` Retorna o comprimento da string (ou seja, número de pontos de código que em geral é igual ao número de caracteres). `str_length(fruit)`

`str_pad(string, width, side = c("left", "right", "both"), pad = " ")` Ajusta a string à um comprimento constante. `str_pad(fruit, 17)`

`str_trunc(string, width, side = c("right", "left", "center"), ellipsis = "...")` Trunca a string à um comprimento, colocando o conteúdo restante como elipse (...). `str_trunc(sentences, 6)`

`str_trim(string, side = c("both", "left", "right"))`
Remove espaços em branco do início/fim da string. `str_trim(str_pad(fruit, 17))`

`str_squish(string)` Corta espaços em branco das extremidades e remove os espaços em branco duplicados no meio. `str_squish(str_pad(fruit, 17, "both"))`

Modificar Strings

`str_sub() <- value`. Substitui partes da string pelo padrão identificado com `str_sub()`, com o valor atribuído. `str_sub(fruit, 1, 3) <- "str"`

`str_replace(string, pattern, replacement)`
Substitui o primeiro padrão encontrado em cada string. Ver também. `str_replace(fruit, "p", "-")`

`str_replace_all(string, pattern, replacement)`
Substitui todos os padrões encontrados em cada string. Ver também `str_remove_all()`. `str_replace_all(fruit, "p", "-")`

`str_to_lower(string, locale = "en")1`
Converte strings para minúsculo. `str_to_lower(sentences)`

`str_to_upper(string, locale = "en")1`
Converte strings para maiúsculo. `str_to_upper(sentences)`

`str_to_title(string, locale = "en")1` Converte strings para títulos. Ver também `str_to_sentence()`. `str_to_title(sentences)`

Juntar e Dividir

`str_c(..., sep = "", collapse = NULL)` Junta várias strings em uma única. `str_c(letters, LETTERS)`

`str_flatten(string, collapse = "")` Combina em uma única separada pelo collapse. `str_flatten(fruit, ",")`

`str_dup(string, times)` Repete a string n vezes. Ver também `str_unique()` para remover duplicadas. `str_dup(fruit, times = 2)`

`str_split_fixed(string, pattern, n)` Divide um vetor de strings em uma matriz de partes da string (dividindo cada ocorrência do padrão encontrado). Ver também `str_split()` para retornar uma lista de partes da string e `str_split_i()` para dividir em i partes. `str_split_fixed(sentences, " ", n=3)`

`str_glue(..., .sep = "", .envir = parent.frame())`
Cria uma string juntando strings e {expressões} {expressions}. `str_glue("Pi is {pi}")`

`str_glue_data(x, ..., .sep = "", .envir = parent.frame(), .na = "NA")` Use um data frame, lista, ou ambiente para criar uma string de strings ou {expressões}. `str_glue_data(mtcars, "{rownames(mtcars)} has {hp} hp")`

Ordenar Strings

`str_order(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)1`
Retorna um vetor de índices com a ordem de cada vetor de caracteres. `fruit[str_order(fruit)]`

`str_sort(x, decreasing = FALSE, na_last = TRUE, locale = "en", numeric = FALSE, ...)1`
Ordena um vetor de caracteres. `str_sort(fruit)`

Auxiliares

`str_conv(string, encoding)` Sobrescreve a codificação de uma string. `str_conv(fruit, "ISO-8859-1")`

`str_view_all(string, pattern, match = NA)`
Mostra uma renderização HTML dos padrões regulares (regex) encontrados. Ver `str_view()` para mostrar apenas o primeiro padrão encontrado. `str_view_all(sentences, "[aeiou]")`

`str_equal(x, y, locale = "en", ignore_case = FALSE, ...)1` Determina se duas strings são equivalentes. `str_equal(c("a", "b"), c("a", "c"))`

`str_wrap(string, width = 80, indent = 0, exdent = 0)` Encapsula strings em uma formatação agradável de parágrafos. `str_wrap(sentences, 20)`

¹ Ver bit.ly/ISO639-1 para a lista completa de localizações.



Importante Saber

O argumento de padrões (pattern) no stringr são interpretados como uma expressão regular (regex) depois que qualquer caractere especial seja processado.

No R, você escreve expressões regulares como strings, ou seja, como sequência de caracteres entre aspas duplas ("") ou simples ('').

Alguns caracteres não podem ser representados diretamente como um string no R. Estes devem ser representados como um caractere especial, ou seja, uma sequência de caracteres com significado especial, e.x.

Caractere Especial	Representa
\\	\
\"	"
\\n	nova linha

Execute ?"" para ver a lista completa

Devido a isto, sempre que ver \ em uma expressão regular, você deve escrevê-la como \\ na string que representa a expressão.

Use writeLines() para ver como o R vê sua string depois que os caracteres especiais são processados.

```
writeLines("\\|. ")
#|.
```

```
writeLines("\\| é uma barra invertida ")
#| é uma barra invertida
```

INTERPRETADORES

Padrões no stringr são interpretados como regex. Para mudar isto, encapsule o padrão em umas das funções:

regex(pattern, ignore_case = FALSE, multiline = FALSE, comments = FALSE, dotall = FALSE, ...)
Modifica o regex para ignorar maiúsculas/minúsculas, encontrar fim de linha como fim da string, permitir comentários do R dentro do regex. Encontrar tudo incluindo \n. str_detect("i", regex("i", TRUE))

fixed() Encontra bytes básicos mas irá perder alguns caracteres que podem estar representados de outras formas (rápido). str_detect("\u0130", fixed("i"))

coll() Encontra bytes básicos e usa a localização para reconhecer os caracteres que podem ser representados de várias formas (lento). str_detect("\u0130", coll("i", TRUE, locale = "tr"))

boundary() Encontra fronteiras entre caracteres, quebra de linhas, sentenças ou palavras. str_split(sentences, boundary("word"))

Expressões Regulares-

Expressões regulares, ou *regexps*, é uma linguagem concisa para descrever padrões em strings.

ENCONTRAR CARACTERES

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

string (digite)	regexp (para dizer)	encontra (que encontra isto)	exemplo
	a (etc.)	a (etc.)	see("a")
\\.	\\.	.	see("\\.")
\\!	\\!	!	see("\\!")
\\?	\\?	?	see("\\?")
\\	\\		see("\\ ")
\\(\\((see("\\(")
\\)	\\))	see("\\)")
\\{	\\{	{	see("\\{")
\\}	\\}	}	see("\\}")
\\n	\\n	nova linha (return)	see("\\n")
\\t	\\t	tab	see("\\t")
\\s	\\s	espaço em branco (\\S para não-brancos)	see("\\s")
\\d	\\d	qualquer dígito (\\D para não-dígitos)	see("\\d")
\\w	\\w	qualquer letra (\\W para não-letras)	see("\\w")
\\b	\\b	limite das palavras	see("\\b")
	[:digit:]	dígitos	see("[:digit:]")
	[:alpha:]	letras	see("[:alpha:]")
	[:lower:]	letras minúsculas	see("[:lower:]")
	[:upper:]	letras maiúsculas	see("[:upper:]")
	[:alnum:]	letras e números	see("[:alnum:]")
	[:punct:]	pontuação	see("[:punct:]")
	[:graph:]	letras, números e pontuações	see("[:graph:]")
	[:space:]	caractere de espaço (ou seja, \\s)	see("[:space:]")
	[:blank:]	espaços e tabs (mas não nova linha)	see("[:blank:]")
	.	qualquer caractere exceto nova linha	see(".")

¹ Muitas funções do R básico exigem que as classes sejam colocadas com um segundo colchete, ou seja, [[:digit:]]

ALTERNADORES

```
alt <- function(rx) str_view_all("abcde", rx)
```

regexp	encontra	exemplo
ab d	ou	alt("ab d")
[abe]	um dos	alt("[abe]")
[^abe]	tudo menos	alt("[^abe]")
[a-c]	range	alt("[a-c]")

ANCORAGEM

```
anchor <- function(rx) str_view_all("aaa", rx)
```

regexp	encontra	exemplo
^a	início da string	anchor("^a")
a\$	fim da string	anchor("a\$")

OLHAR AO REDOR

```
look <- function(rx) str_view_all("bacad", rx)
```

regexp	encontra	exemplo
a(?=c)	seguido por	look("a(?=c)")
a(!c)	não seguido por	look("a(!c)")
(?<=b)a	precedido por	look("(?<=b)a")
(?!b)a	Não precedido por	look("(?!b)a")

QUANTIFICADORES

```
quant <- function(rx) str_view_all(".a.aa.aaa", rx)
```

regexp	matches	exemplo
a?	zero ou um	quant("a?")
a*	zero or mais	quant("a*")
a+	um ou mais	quant("a+")
a{n}	exatamente n	quant("a{2}")
a{n,}	n ou mais	quant("a{2,}")
a{n,m}	entre n e m	quant("a{2,4}")

GRUPOS

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

Use parênteses para definir precedência (ordem de avaliação) e criar grupos

regexp	matches	exemplo
(ab d)e	sets precedence	alt("(ab d)e")

Use um número com dupla barra invertida para referenciar ou duplicar grupos identificados anteriormente no padrão. Referencia cada grupo, pela sua ordem de aparição

string (digite)	regexp (significa)	encontra (que encontra isto)	exemplo (o resultado é o mesmo que ref("abba"))
\\1	\\1 (etc.)	first () group, etc.	ref("(a)(b)\\2\\1")

