

Package ‘mojson’

July 23, 2025

Title A Serialization-Style Flattening and Description for JSON

Version 0.1

Author Bo Wei <ckris@163.com>

Maintainer Bo Wei <ckris@163.com>

Description Support JSON flattening in a long data frame way, where the nesting keys will be stored in the absolute path. It also provides an easy way to summarize the basic description of a JSON list. The idea of 'mojson' is to transform a JSON object in an absolute serialization way, which means the early key-value pairs will appear in the heading rows of the resultant data frame. 'mojson' also provides an alternative way of comparing two different JSON lists, returning the left/inner/right-join style results.

License MIT + file LICENSE

URL <https://github.com/chriswweibo/mojson>

BugReports <https://github.com/chriswweibo/mojson/issues>

Encoding UTF-8

LazyData true

RoxygenNote 7.1.1

Imports RJSONIO, magrittr, tidyr, iterators, stringr, compareDF

NeedsCompilation no

Repository CRAN

Date/Publication 2021-02-11 09:30:02 UTC

Contents

alignj	2
descj	3
diffj	4
expanddf	5
flattenj	6
flattenj_one	7
loadj	8

alignj	<i>JSON Lists Alignment</i>
--------	-----------------------------

Description

Align the two JSON lists by specifying the primary path(keys), to support the left/inner/right-join style comparison.

Usage

```
alignj(json_new, json_old, sep = "@", primary)
```

Arguments

json_new	list. The new JSON list.
json_old	list. The old JSON list.
sep	character. A character/string passed to flattenj . Defaults to @ to avoid the occasional overriding. Not recommended to use some risky characters like . and \.
primary	character. The primary path(keys) for identifying a unique JSON object. The value provided should contain the sep value to specify the nesting information unless it is an outermost key.

Details

The function borrows the idea from the data set operation, and the result contains:

- new, contains the flattening result of json_new.
- old, contains the flattening result of json_old.
- common_primary, contains the primary paths both in json_new and json_old.
- new_primary, contains the primary paths only in json_new.
- old_primary, contains the primary paths only in json_old.

Value

list. The result list contains the alignment information of three types: the primary paths only in the new JSON, only in the old JSON, and in both.

Examples

```
library(mojson)
j1 <- list(list(id = list(x = 1 ,y = 2), gender = 'M'),
           list(id = list(x = 2 ,y = 2), gender = 'M'))
j2 <- list(list(id = list(x = 2 ,y = 2), gender = 'F'),
           list(id = list(x = 3 ,y = 2), gender = 'F'))
alignj(j1, j2, primary = 'idex')
```

descj	<i>JSON Description</i>
-------	-------------------------

Description

Provide descriptive information about the JSON list, such as the key frequency, the nesting information and the value distribution.

Usage

```
descj(dat, sep = "@")
```

Arguments

dat	list. Loaded result from a JSON file.
sep	character. A character/string passed to flattenj . Defaults to @ to avoid the occasional overriding. Not recommended to use some risky characters like . and \.

Details

The result contains three parts:

- `key_summary`, presents the description of keys, which contains all the keys and their respective frequencies.
- `value_summary`, presents the description of values, which contains all atomic values and their respective frequencies.
- `stream_summary`, presents the description of paths' direct upstream keys and downstream keys. The up data frame stores the upstream information about where the current key is nested. And the down data frame stores the downstream information about how the current key branches. It means no upstream or downstream if . value is empty.

Note that the mathematical logic of frequency is based on the flattening work, which means the occurrence of one key will be considered as repeated if it has multiple downstream keys. For example, `list(list(x = list(m = 1, n = 2), y = 2))`, and the frequency of x will be 2, because it has two nesting keys. It is recommended to interpret the upstream and downstream information in a relative way rather than an absolute way. Returning the absolute frequency is to preserve the raw information. Hence, it is easy to know that x will equally branches to m and n.

Value

list. The descriptive result.

See Also

[flattenj](#).

Examples

```
library(mojson)
j <- list(a = list(x = 1, y = 2),
          b = c(3, 4, list(z = 5, s = 6, t = list(m = 7, n = 8))))
j_multi <- list(j, j, j)
desc <- descj(j_multi)
desc$keys_summary
```

diffj

Multiple JSON Objects Diff

Description

Find the difference between multiple JSON objects yielded by create, delete and update operations.

Usage

```
diffj(json_new, json_old, sep = "@", primary)
```

Arguments

json_new	list. The new JSON objects.
json_old	list. The old JSON objects.
sep	character. A character/string used to separate keys in the nesting path. Defaults to @ to avoid the occasional overriding. Not recommended to use some risky characters like . and \. When compact = FALSE, it is unnecessary to assign sep explicitly, unless @ has been used in the keys.
primary	character. The primary path(keys) for identifying a unique JSON object. The value provided should contain the sep value to specify the nesting information unless it is an outermost key.

Details

This function finds out the difference between two JSON lists. And the difference is as follows:

- create, stores the flattened result of objects only in the json_new, that is some JSON objects have been created.
- delete, stores the flattened result of objects only in the json_old, that is some JSON objects have been deleted.
- change, stores the value update information in the common objects, reflected by '+ (add)', and '- (remove)' in the chng_type field.

The change_summary provides the general information of value change.

Value

list. Contains the difference result, including path create, path delete and value change results.

Examples

```
library(mojson)
j1 <- list(list(x = 1, y = 2, b = list(m = 1, n = 1)),
           list(x = 2, y = 2, b = list(m = 1, n = 1)))
j2 <- list(list(x = 2, y = 3, b = list(m = 1)),
           list(x = 3, y = 2, b = list(m = 1, n = 1)))
diffj(j1, j2, primary = 'x')
```

 expanddf

Data Frame Expand

Description

Expand a data frame by splitting one column

Usage

```
expanddf(df, column, sep)
```

Arguments

df	data frame. The input to be expanded.
column	character. The column to be splitted.
sep	character. Separator for splitting a column.

Details

This function implements the data frame expansion if you need to split one column by the specific characters. The new data frame will generate the new columns named as 'level' appended by position-indexing numbers, such as 'level1', 'level2'. The maximum of appended numbers indicates the most splitting pieces for one cell. If the splitting results of one cell are fewer than the maximum, the row will be padded and corresponding cells will be filled with NAs.

Value

data frame. The resultant data frame with new columns.

Examples

```
library(mojson)
# levels are identical.
df1 <- data.frame(a = c('ab@gmail.com', 'cd@gmail.com'),
                 b = c(TRUE, FALSE))
expanddf(df1, 'a', '@')

# change the separator and treat various levels.
df2 <- data.frame(a = c('1-2-0', '1-2-0-3', '1-2'),
                 b = c(TRUE, FALSE, TRUE))
```

```
expanddf(df2, 'a', '-')
```

 flattenj

JSON Flatten

Description

Transform multiple JSON objects into a flattened data frame.

Usage

```
flattenj(dat, sep = "@", compact = TRUE)
```

Arguments

dat	list. Loaded result from a JSON file.
sep	character. A character/string passed to flattenj_one . Defaults to @ to avoid the occasional overriding. Not recommended to use some risky characters like . and \. When compact=FALSE, it is unnecessary to assign sep explicitly, unless @ has been used in the key fields.
compact	logical. Whether to generate the compact or completely expanded data frame. Defaults to TRUE.

Details

The function flattens multiple JSON objects into a new data frame. The result contains multiple columns. If compact=TRUE, it returns paths, values and index columns, otherwise level1, level2, ..., values and index. The index column stores the id of each JSON object.

Value

data frame. The flattened result.

See Also

[flattenj_one](#).

Examples

```
library(mojson)
j <- list(a = list(x = 1, y = 2),
         b = c(3, 4, list(z = 5, s = 6, t = list(m = 7, n = 8))))
j_multi <- list(j, j, j)
flattenj(j_multi)
flattenj(j_multi, compact=FALSE)
```

flattenj_one	<i>Single JSON Object Flatten</i>
--------------	-----------------------------------

Description

Transform a JSON object into a flattened data frame in a serialization way.

Usage

```
flattenj_one(dat, sep = "@", compact = TRUE)
```

Arguments

dat	list. The list from a JSON object.
sep	character. A character/string used to separate keys in the nesting path. Defaults to @ to avoid the occasional overriding. Not recommended to use some risky characters like . and \. When compact = FALSE, it is unnecessary to assign sep explicitly, unless @ has been used in the key fields.
compact	logical. Whether to generate the compact or completely expanded data frame. Defaults to TRUE.

Details

The function flattens a single JSON object into a data frame with two different schemas according to the compact value.

- For compact = TRUE, the data frame contains two columns. One is paths which stores the absolute path of each record. And the other is values which stores the corresponding values of each path.
- For compact = FALSE, the data frame has more columns based on the global nesting situation.

It actually applies the serialization way for flattening, which means the early values correspondingly appear in the heading rows of the data frame. And if the value is a list object in the original data or a non-named list/vector in the R environment, the path will be correspondingly appended with an integer to specify each list element. For example, in the raw JSON file, "{ 'a': [1, 2, 3] }" will be data.frame(paths = c('a1', 'a2', 'a3'), values = c(1, 2, 3)). Great credits to the answer of [Tommy](#).

Value

data frame. The flattened result.

See Also

[expanddf](#).

Examples

```
library(mojson)
j <- list(a = list(x = 1, y = 2),
          b = c(3, 4, list(z = 5, s = 6, t = list(m = 7, n = 8))))
flattenj_one(j)
flattenj_one(j, compact = FALSE)
```

loadj	<i>JSON Load</i>
-------	------------------

Description

Load a JSON file into an R list

Usage

```
loadj(file, encoding = "UTF-8")
```

Arguments

file	character. A JSON file connection.
encoding	character. Encoding method to use. Defaults to UTF-8.

Details

This function provides a simple interface to load a JSON file, meanwhile prints some loading information.

- `num_of_loaded_obj` tells the length of the JSON object.
- `duration_seconds` tells the loading duration.
- `speed_objs_sec` tells the loading speed in objects per second.
- `obj_len_summary` gives the length summary of each JSON object.

Value

list. The loading result.

Examples

```
library(mojson)
j <- list(a = list(1, 2), b = 3)
tf <- tempfile()
writeLines(RJSONIO::toJSON(j), tf)
loadj(tf)
```


Index

`alignj`, 2

`descj`, 3

`diffj`, 4

`expanddf`, 5, 7

`flattenj`, 2, 3, 6

`flattenj_one`, 6, 7

`loadj`, 8