Package ‘redux’

May 15, 2017

Title R Bindings to 'hiredis'
Version 1.0.0
Author Rich FitzJohn
Maintainer Rich FitzJohn <rich.fitzjohn@gmail.com>
Description A 'hiredis' wrapper that includes support for transactions, pipelining, blocking subscription, serialisation of all keys and values, 'Redis' error handling with R errors. Includes an automatically generated 'R6' interface to the full 'hiredis' API. Generated functions are faithful to the 'hiredis' documentation while attempting to match R's argument semantics. Serialisation must be explicitly done by the user, but both binary and text-mode serialisation is supported.
SystemRequirements hiredis
License GPL-2
LazyData true

URL https://github.com/richfitz/redux

BugReports https://github.com/richfitz/redux/issues
Depends R (>= 3.2.0)
Imports R6
Suggests knitr, rmarkdown, sys, testthat
VignetteBuilder knitr
RoxygenNote 6.0.1
NeedsCompilation yes
Repository CRAN
Date/Publication 2017-05-15 14:11:17 UTC
from_redis_hash

R topics documented:

from_redis_hash .................................................. 2
hiredis ................................................................. 3
object_to_string ..................................................... 4
parse_redis_url ....................................................... 5
redis ................................................................. 5
redis_api .............................................................. 6
redis_config .......................................................... 6
redis_connection ...................................................... 8
redis_info ............................................................. 9
redis_multi ........................................................... 9
redis_scripts ......................................................... 10
redis_time ............................................................ 10
scan_apply ............................................................ 11

Description

Convert a Redis hash to a character vector or list. This tries to bridge the gap between the way Redis returns hashes and the way that they are nice to work with in R, but keeping all conversions very explicit.

Usage

from_redis_hash(con, key, fields = NULL, f = as.character,
missing = NA_character_)
Examples

```r
if (redux::redis_available()) {
    # Using a random key so we don't overwrite anything in your database:
    key <- paste0("redux::", paste(sample(letters, 15), collapse = ""))
    r <- redux::hiredis()
    r$HSET(key, "a", "apple")
    r$HSET(key, "b", "banana")
    r$HSET(key, "c", "carrot")

    # Now we have a hash with three elements:
    r$HGETALL(key)

    # Ew, that's not very nice. This is nicer:
    redux::from_redis_hash(r, key)

    # If one of the elements was not a string, then that would not
    # have worked, but you can always leave as a list:
    redux::from_redis_hash(r, key, f = identity)

    # To get just some elements:
    redux::from_redis_hash(r, key, c("a", "c"))

    # And if some are not present:
    redux::from_redis_hash(r, key, c("a", "x"))
    redux::from_redis_hash(r, key, c("a", "z"), missing = "zebra")

    r$DEL(key)
}
```

---

**hiredis**  
**Interface to Redis**

**Description**

Create an interface to Redis, with a generated interface to all Redis commands.

**Usage**

- `hiredis(...)`, `version = NULL`
  - `redis_available(...)`

**Arguments**

- `...`  
  Named configuration options passed to `redis_config`, used to create the environment (notable keys include `host`, `port`, and the environment variable `REDIS_URL`). For `redis_available`, arguments are passed through to `hiredis`. 
version

Version of the interface to generate. If given as a string or numeric version, then only commands that exist up to that version will be included. If given as TRUE, then we will query the Redis server (with INFO) and extract the version number that way.

Examples

```r
# Only run if a Redis server is running
if (redux::redis_available()) {
  r <- redux::hiredis()
  r$PING()
  r$SET("foo", "bar")
  r$GET("foo")

  # There are lots of methods here:
  r
}
```

---

**object_to_string**

*Convert R objects to/from strings*

**Description**

Serialise/deserialise an R object into a string. This is a very thin wrapper around the existing R functions `serialize` and `rawToChar`. This is useful to encode arbitrary R objects as string to then save in Redis (which expects a string).

**Usage**

```r
object_to_string(obj)

string_to_object(str)

object_to_bin(obj, xdr = FALSE)

bin_to_object(bin)
```

**Arguments**

- `obj` An R object to convert into a string
- `str` A string to convert into an R object
- `xdr` Use the big-endian representation? Unlike, `serialize` this is disabled here by default as it is a bit faster (~ 20 microsecond roundtrip for a serialization of 100 doubles)
- `bin` A binary vector to convert back to an R object
parse_redis_url

Examples

```r
s <- object_to_string(1:10)
s
string_to_object(s)
identical(string_to_object(s), 1:10)
```

---

<table>
<thead>
<tr>
<th>parse_redis_url</th>
<th>Parse Redis URL</th>
</tr>
</thead>
</table>

Description

Parse a Redis URL

Usage

`parse_redis_url(url)`

Arguments

- `url` A URL to parse

---

<table>
<thead>
<tr>
<th>redis</th>
<th>Redis commands object</th>
</tr>
</thead>
</table>

Description

Primarily used for pipeling, the `redis` object produces commands the same way that the main `redis_api` objects do. If passed in as arguments to the `pipeline` method (where supported) these commands will then be pipelined. See the `redux` package for an example.

Usage

`redis`

Format

An object of class `redis_commands` of length 199.
Examples

```r
# This object creates commands in the format expected by the
# lower-level redis connection object:
redis$PING()

# For example to send two PING commands in a single transmission:
if (redux::redis_available()) {
  r <- redux::hiredis()
  r$pipeline(
    redux::redis$PING(),
    redux::redis$PING()
  )
}
```

---

**redis_api**  
Create a Redis API object

**Description**

Create a Redis API object. This function is designed to be used from other packages, and not designed to be used directly by users.

**Usage**

```r
redis_api(x, version = NULL)
```

**Arguments**

- **x**
  - An object that defines at least the function `command` capable of processing commands in the appropriate form.

- **version**
  - Version of the Redis API to generate. If given as a numeric version (or something that can be coerced into one). If given as TRUE, then we query the Redis server for its version and generate only commands supported by the server.

---

**redis_config**  
Redis configuration

**Description**

Create a set of valid Redis configuration options.

**Usage**

```r
redis_config(..., config = list(...))
```
Arguments

... See Details
config A list of options, to use in place of ...

Details

Valid arguments here are:

url The URL for the Redis server. See examples. (default: Look up environment variable REDIS_URL or NULL).
host The hostname of the Redis server. (default: 127.0.0.1).
port The port of the Redis server. (default: 6379).
path The path for a Unix socket if connecting that way.
pASSWORD The Redis password (for use with AUTH). This will be stored in plain text as part of the Redis object. (default: NULL).
db The Redis database number to use (for use with SELECT. Do not use in a redis clustering context. (default: NULL; i.e., don't switch).

The way that configuration options are resolved follows the design for redis-rb very closely.

1. First, look up (and parse if found) the REDIS_URL environment variable and override defaults with that.
2. Any arguments given (host, port, password, db) override values inferred from the url or defaults.
3. If path is given, that overrides the host/port settings and a socket connection will be used.

Examples

# default config:
redis_config()

# set values
redis_config(host = "myhost")

# url settings:
redis_config(url = "redis://p4ssw0rd@myhost:32000/2")

# override url settings:
redis_config(url = "redis://myhost:32000", port = 31000)
redis_config(url = "redis://myhost:32000", path = "/tmp/redis.conf")
redis_connection Create a Redis connection

Description

Create a Redis connection. This function is designed to be used in other packages, and not directly by end-users. However, it is possible and safe to use. See the hiredis package for the user friendly interface.

Usage

redis_connection(config = redis_config())

Arguments

config Configuration parameters as generated by redis_config

Details

This function creates a list of functions, appropriately bound to a pointer to a Redis connection. This is designed for package authors to use so without having to ever deal with the actual pointer itself (which cannot be directly manipulated from R anyway).

The returned list has elements, all of which are functions:

config() The configuration information
reconnect() Attempt reconnection of a connection that has been closed, through serialisation/deserialisation or through loss of internet connection.
command(cmd) Run a Redis command. The format of this command will be documented elsewhere.
pipeline(cmds) Run a pipeline of Redis commands.
subscribe(channel, pattern, callback, envir) Subscribe to a channel or pattern specifying channels. Here, channel must be a character vector, pattern a logical indicating if channel should be interpreted as a pattern, callback is a function to apply to each received message, returning TRUE when subscription should stop, and envir is the environment in which to evaluate callback. See below.

Subscriptions

The callback function must take a single argument; this will be the received message with named elements type (which will be message), channel (the name of the channel) and value (the message contents). If pattern was TRUE, then an additional element pattern will be present (see the Redis docs). The callback must return TRUE or FALSE; this indicates if the client should continue quit (i.e., TRUE means return control to R, FALSE means keep going).

Because the subscribe function is blocking and returns nothing, so all data collection needs to happen as a side-effect of the callback function.

There is currently no way of interrupting the client while it is waiting for a message.
redis_info

**Parse Redis INFO**

**Description**
Parse and return Redis INFO data.

**Usage**
- `redis_info(con)`
- `parse_info(x)`
- `redis_version(con)`

**Arguments**
- `con` A Redis connection
- `x` character string

**Examples**
```c
if (redux::redis_available()) {
  r <- redux::hiredis()

  # Redis server version:
  redux::redis_version(r)
  # This is a 'numeric_version' object so you can compute with it
  # if you need to check for minimum versions
  redux::redis_version(r) >= numeric_version("2.1.1")

  # Extensive information is given back by the server:
  redux::redis_info(r)

  # Which is just:
  redux::parse_info(r$INFO())
}
```

redis_multi

**Helper for Redis MULTI**

**Description**
Helper to evaluate a Redis MULTI statement. If an error occurs then, DISCARD is called and the transaction is cancelled. Otherwise EXEC is called and the transaction is processed.
redis_time

Usage

redis_time(con)

format_redis_time(x)

redis_time_to_r(x)

redis_multi

Usage

redis_multi(con, expr)

Arguments

con        A Redis connection object
expr       An expression to evaluate

redis_scripts

Load Lua scripts into Redis

Description

Load Lua scripts into Redis, providing a convenience function to call them with. Using this function
means that scripts will be available to use via EVALSHA, and will be preloaded on the Redis server.
Scripts are then accessed by name rather than by content or SHA. See the vignette for details and
an example.

Usage

redis_scripts(con, ..., scripts = list(...))

Arguments

con        A Redis connection
...        A number of scripts
scripts    Alternatively, a list of scripts

redis_time

Get time from Redis

Description

Get time from Redis and format as a string.

Usage

redis_time(con)

format_redis_time(x)

redis_time_to_r(x)
scan_apply

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>con</td>
<td>A Redis connection object</td>
</tr>
<tr>
<td>x</td>
<td>a list as returned by TIME</td>
</tr>
</tbody>
</table>

Examples

```r
if (redux::redis_available()) {
  r <- redux::hiredis()

  # The output of Redis' TIME command is not the *most* useful
  # thing in the world:
  r$TIME()

  # We can get a slightly nicer representation like so:
  redux::redis_time(r)

  # And from that convert to an actual R time:
  redux::redis_time_to_r(redux::redis_time(r))
}
```

Description

Support for iterating with SCAN. Note that this will generalise soon to support collecting output, SSCAN and other variants, etc.

Usage

```r
scan_apply(con, callback, pattern = NULL, ..., count = NULL,
            type = "SCAN", key = NULL)

scan_del(con, pattern, count = NULL, type = "SCAN", key = NULL)

scan_find(con, pattern, count = NULL, type = "SCAN", key = NULL)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>con</td>
<td>A redis_api object</td>
</tr>
<tr>
<td>callback</td>
<td>Function that takes a character vector of keys and does something useful to it. con$DEL is one option here to delete keys that match a pattern. Unlike R's *apply functions, callback is called for its side effects and its return values will be ignored.</td>
</tr>
<tr>
<td>pattern</td>
<td>Optional pattern to use.</td>
</tr>
<tr>
<td>...</td>
<td>additional arguments passed through to callback. Note that if used, pattern must be provided (at least as NULL).</td>
</tr>
</tbody>
</table>
count

Optional step size (default is Redis’ default which is 10)

type

Type of SCAN to run. Options are "SCAN" (the default), "HSCAN" (scan through keys of a hash), "SSCAN" (scan through elements of a set) and "ZSCAN" (scan through elements of a sorted set). If type is not "SCAN", then key must be provided. HSCAN and ZSCAN currently do not work usefully.

details

Key to use when running a hash, set or sorted set scan.

details

The functions scan_del and scan_find are example functions that delete and find all keys corresponding to a given pattern.
Index

*Topic **datasets**
  redis, 5

bin_to_object (object_to_string), 4

format_redis_time (redis_time), 10
from_redis_hash, 2

hiredis, 3, 8

object_to_bin (object_to_string), 4
object_to_string, 4

parse_info (redis_info), 9
parse_redis_url, 5

rawToChar, 4
redis, 5
redis_api, 5, 6
redis_available (hiredis), 3
redis_config, 3, 6, 8
redis_connection, 8
redis_info, 9
redis_multi, 9
redis_scripts, 10
redis_time, 10
redis_time_to_r (redis_time), 10
redis_version (redis_info), 9

scan_apply, 11
scan_del (scan_apply), 11
scan_find (scan_apply), 11
serialize, 4
string_to_object (object_to_string), 4