Package: bank (via r-universe)

June 20, 2024

Title Extra Cache

Version 0.0.0.9002

Description More cache backends.

License MIT + file LICENSE

Imports attempt, digest, R6 (>= 2.5.0),

Suggests memoise, mongolite (>= 2.2.0), DBI, testthat (>= 3.0.0), redux (>= 1.1.0), RPostgres, withr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Repository https://thinkr-open.r-universe.dev

RemoteUrl https://github.com/ThinkR-open/bank

RemoteRef HEAD

RemoteSha c8ba84aaa0b6cc6f009344dbb4c99f0985ebb444

Contents

cache_mongo *A Caching object for MongoDB*

Description

A Caching object for MongoDB

A Caching object for MongoDB

Details

Create a cache backend with MongoDB.

Methods

Public methods:

- `cache_mongo$new()`
- `cache_mongo$get()`
- `cache_mongo$set()`
- `cache_mongo$has_key()`
- `cache_mongo$reset()`
- `cache_mongo$remove()`
- `cache_mongo$keys()`
- `cache_mongo$digest()`
- `cache_mongo$clone()`

Method `new()`**:** Start a new mongo cache

Usage:

```
cache_mongo$new(
  db = "test",
  url = "mongodb://localhost",
  prefix = "fs",
  options = mongolite::ssl_options(),
  algo = "sha512",
  compress = FALSE
)
```

Arguments:

`db` name of database

`url` address of the mongodb server in mongo connection string URI format

`prefix` string to prefix the collection name

`options` additional connection options such as SSL keys/certs.

`algo` for `{memoise}` compatibility. The `digest()` algorithm.

`compress` for `{memoise}` compatibility. Should the data be compressed?

Returns: A cache_mongo object

Method `get()`**:** Get a key from the cache

Usage:

```
cache_mongo$get(key)
```

Arguments:

`key` Name of the key.

Returns: The value stored using the `key`

Method `set()`**:** Set a key in the cache

Usage:

```
cache_mongo$set(key, value)
```

Arguments:

`key` Name of the key.

`value` Value to store

Returns: Used for side effect

Method `has_key()`**:** Does the cache contains a given key?

Usage:

```
cache_mongo$has_key(key)
```

Arguments:

`key` Name of the key.

Returns: TRUE/FALSE

Method `reset()`**:** Clear all the cache

Usage:

```
cache_mongo$reset()
```

Returns: Used for side-effect

Method `remove()`**:** Remove a key/value pair

Usage:

```
cache_mongo$remove(key)
```

Arguments:

`key` Name of the key.

Returns: Used for side-effect

Method `keys()`**:** List all the keys in the cache

Usage:

```
cache_mongo$keys()
```

Returns: A list of keys

Method `digest()`**:** Function that runs an hash algo. For compatibily with `{memoise}`.

Usage:

```
cache_mongo$digest(...)
```

Arguments:

... the value to hash

Returns: A function

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
cache_mongo$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`cache_postgres` *A Caching object for postgres*

Description

A Caching object for postgres

A Caching object for postgres

Details

Create a cache backend with postgres

Methods

Public methods:

- `cache_postgres$new()`
- `cache_postgres$has_key()`
- `cache_postgres$get()`
- `cache_postgres$set()`
- `cache_postgres$reset()`
- `cache_postgres$remove()`
- `cache_postgres$keys()`
- `cache_postgres$digest()`
- `cache_postgres$clone()`

Method `new()`**:** Start a new postgres cache

Usage:

```
cache_postgres$new(
  ...,
  cache_table = "bankrcache",
  algo = "sha512",
  compress = FALSE
)
```

Arguments:

`...` Parameters passes do DBI::dbConnect(RPostgres::Postgres(), ...)

`cache_table` On `initialize()`, the cache object will create a table to store the cache. Default name is `bankrcache`. Change it if you already have a table named `bankrcache` in your DB.

`algo` for `{memoise}` compatibility, the `digest()` algorithm

`compress` for `{memoise}` compatibility, should the data be compressed?

Returns: A cache_postgres object

Method `has_key()`**:** Does the cache contains a given key?

Usage:

```
cache_postgres$has_key(key)
```

Arguments:

`key` Name of the key.

Returns: TRUE/FALSE

Method `get()`**:** Get a key from the cache

Usage:

```
cache_postgres$get(key)
```

Arguments:

`key` Name of the key.

Returns: The value stored using the `key`

Method `set()`**:** Set a key in the cache

Usage:

```
cache_postgres$set(key, value)
```

Arguments:

`key` Name of the key.

`value` Value to store

Returns: Used for side effect

Method `reset()`**:** Clear all the cache

Usage:

```
cache_postgres$reset()
```

Returns: Used for side-effect

Method `remove()`**:** Remove a key/value pair

Usage:

```
cache_postgres$remove(key)
```

Arguments:

`key` Name of the key.

Returns: Used for side-effect

Method `keys()`**:** List all the keys in the cache

Usage:

```
cache_postgres$keys()
```

Returns: A list of keys

Method `digest()`**:** Function that runs an hash algo. For compatibily with `{memoise}`

Usage:

```
cache_postgres$digest(...)
```

Arguments:

`...` the value to hash

Returns: A function

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
cache_postgres$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`cache_redis` *A Caching object for redis*

Description

A Caching object for redis

A Caching object for redis

Details

Create a cache backend with redis

Methods

Public methods:

- `cache_redis$new()`
- `cache_redis$get()`
- `cache_redis$set()`
- `cache_redis$has_key()`
- `cache_redis$reset()`
- `cache_redis$remove()`
- `cache_redis$keys()`
- `cache_redis$digest()`
- `cache_redis$clone()`

Method `new()`**:** Start a new redis cache

Usage:

```
cache_redis$new(..., version = NULL, algo = "sha512", compress = FALSE)
```

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 to 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.

`algo` for `{memoise}` compatibility, the `digest()` algorithm

`compress` for `{memoise}` compatibility, should the data be compressed?

Returns: A cache_redis object

Method `get()`**:** Get a key from the cache

Usage:

```
cache_redis$get(key)
```

Arguments:

`key` Name of the key.

Returns: The value stored using the `key`

Method `set()`**:** Set a key in the cache

Usage:

```
cache_redis$set(key, value)
```

Arguments:

`key` Name of the key.

`value` Value to store

Returns: Used for side effect

Method `has_key()`**:** Does the cache contains a given key?

Usage:

```
cache_redis$has_key(key)
```

Arguments:

`key` Name of the key.

Returns: TRUE/FALSE

Method `reset()`**:** Clear all the cache

Usage:

```
cache_redis$reset()
```

Returns: Used for side-effect

Method `remove()`**:** Remove a key/value pair

Usage:

```
cache_redis$remove(key)
```

Arguments:

`key` Name of the key.

Returns: Used for side-effect

Method `keys()`**:** List all the keys in the cache

Usage:

```
cache_redis$keys()
```

Returns: A list of keys

Method `digest()`**:** Function that runs an hash algo. For compatibily with `{memoise}`

Usage:

```
cache_redis$digest(...)
```

Arguments:

`...` the value to hash

Returns: A function

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
cache_redis$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index