
redtrine Documentation

Release 0.1

Ronny López, Marcos Quesada, Ricard Clau

September 22, 2013

CONTENTS

Welcome to the official documentation of the *Redtrine* project.

Redtrine is a high-level Redis library for PHP.

It aims to provide a clear interface to implement most of Redis cookbooks in an easy way using *Redtrine Structures*.

Redtrine is inspired by *Redback*, a high-level Redis library for Node.JS and its name is also inspired by *Doctrine*, an ORM that provides transparent persistence for PHP objects.

GETTING STARTED

Here we should provide some documentation on some easy use cases (for instance Cache, Queue, ...) with a toctree, etc...

STRUCTURES

Want to see all Structures supported by Redtrine? There's a tutorial for each one.

2.1 Cache

Cache structure makes it easy to use *Redis* as a cache backend.

You can set an **optional** expiration time in seconds.

This is an example on how to cache some database records for 1 minute.

```
use Redtrine\Structure\Cache;

// ...
$someDatabaseRecords = ...

$databseCache = new Cache('cachedRecords');
$databseCache->set(json_encode($someDatabaseRecords), 60);

// and to retrieve cached data in another HTTP Request
$databseCache = new Cache('cachedRecords');
$cachedRecords = $databseCache->get();
```

2.2 KeyPair

The *KeyPair* is a structure where unique values are assigned an ID.

You can think about this as a table with a primary auto-increment key and a single unique column.

Internally, the *KeyPair* uses two Redis hashes to provide O(1) lookup by both ID and value. It also uses a Redis key to store auto-increment counter.

Redis Structure:

- *(namespace:)key* = *hash(id => value)*
- *(namespace:)key:ids* = *hash(value => id)*
- *(namespace:)key:autoinc* = *integer*

```
use Redtrine\Structure\KeyPair;

$redisTable = new KeyPair('redisTable');
```

```
$redisTable->add('a');
$redisTable->add('b');
$redisTable->add('c');

// At this moment we have an structure like {"1":"a", "2":"b", "3":"c"}
// And we can perform queries by id, value, delete by id and value, etc...

$values = $redisTable->getById(array(1, 2)); // $values is array('a', 'b')

$id = $redisTable->get('a'); // $id is 1

// These 2 expressions would be equivalent in terms of structure
$redisTable->delete('b');
$redisTable->deleteById(2);
```

2.3 Queue

Queue structure makes it easy to use *Redis* as a queue.

Queues can be defined as **FIFO** or **LIFO** via a constructor argument.

You can add elements to the *Queue* with the *enqueue* method and extract elements from it with the *dequeue* method.

You can also send some elements to another Queues. This can be useful for instance if some records need to be processed with a different priority.

This is an example about how to send data to a Redis queue and consume it somewhere else

```
use Redtrine\Structure\Queue;

// ...
$processData = // ... some data to start a process

$processQueue = new Queue('process');
$processQueue->enqueue($processData);

// ... somewhere else in the code there should be a Queue consumer

$processQueue = new Queue('process');
$processData = $processQueue->dequeue();

// ... run a process with $processData
```

And this is an example about how to send data from one *Queue* to another

```
use Redtrine\Structure\Queue;

// ...
$processData = // ... some data to start a process

$processQueue = new Queue('process');
$processQueue->enqueue($processData);

// And we decide to move some queued data to a higher priority queue
$betterQueue = new Queue('highpriority');
$processQueue->dequeueEnqueue($betterQueue);
```

COOKBOOK

Same as getting started, some cookbooks easily solved, toctree, etc...