An Introduction to Redis in PHP using Predis

Redis is an open source data structure server with an in-memory dataset that does much more than simple key/value storage thanks to its built-in data types.

It was started in 2009 by Salvatore Sanfilippo and because of its popularity quickly grew, being chosen by big companies like VMware (who later hired Sanfilippo to work on the project full time), GitHub, Craigslist, Disqus, Digg, Blizzard, Instagram, and more (see redis.io/topics/whos-using-redis).

You can use Redis as a session handler, which is especially useful if you are using a multi-server architecture behind a load balancer. Redis also has a publish/subscribe system, which is great for creating an online chat or a live booking system. Documentation and more information on Redis and all of its commands can be found on the project's website, redis.io.

There is a lot of argument whether Redis or Memcache is better, though as the benchmarks show they perform pretty much on par with each other for basic operations. Redis has more features than Memcache, such as in-memory and disk persistence, atomic commands and transactions, and not logging every change to disk but rather server-side data structures instead.

In this article we'll take a look at some of the basic but powerful commands that Redis has to offer using the Predis library.

Easy Install

Redis is easy to install, and brief installation instructions are published on the product's download page. From my own experience, if you are running Ubuntu then you will get an error if you do not have TCL installed (simply run sudo apt-get install tcl). Once Redis is installed, you can run the server:

```
gafitescu@ubun2:~$ /usr/local/bin/redis-server
* The server is now ready to accept connections on port 6379
```

There are Redis client libraries available for many languages and are listed on the Redis website, often times with several available for each language! For PHP,

there are five. For this article I'll be using the Predis library, but you may also want to look into phpredis which gets compiled and installed as a PHP module.

If you have Git installed on your machine like I do, all you need to do is clone the Predis repository. Otherwise you'll need to download the ZIP archive and unpack it.

```
gafitescu@ubun2:~$ git clone git://github.com/nrk/predis.git
```

To test everything out, create the file `test.php` with the following content to test if you can successfully connect to the running Redis server with Predis:

```
<?php
require "predis/autoload.php";
PredisAutoloader::register();

// since we connect to default setting localhost
// and 6379 port there is no need for extra
// configuration. If not then you can specify the
// scheme, host and port to connect as an array
// to the constructor.
try {
    $redis = new PredisClient();
/*
    $redis = new PredisClient(array(
        "scheme" => "tcp",
        "host" => "127.0.0.1",
        "port" => 6379));
*/
    echo "Successfully connected to Redis";
}
catch (Exception $e) {
    echo "Couldn't connected to Redis";
    echo $e->getMessage();
}
```

When you run it, you should hopefully see the message "Successfully connected to Redis".

Using Redis

In this section you'll gain an overview of the most commonly used commands that Redis has to offer. Memcache has an equivalent for most of them, so if you are familiar with Memcache then this listing will look familiar to you.

SET, GET, and EXISTS

The most important commands used with Redis are `SET`, `GET`, and `EXISTS`. You can use these commands to store and check on temporary information that is going to be accessed multiple times, typically in a key/value manner. For example:

```
<?php
$redis->set("hello_world", "Hi from php!");
$value = $redis->get("hello_world");
var_dump($value);


echo ($redis->exists("Santa Claus")) ? "true" : "false";
```

The `set()` method is used to set a value to a particular key, in this case the key is "hello_world" and the value is "Hi from php!" The `get()` method retrieves the value for the key, again in this case "hello_world". The `exists()` method reports back whether the provided key is found or not in Redis' storage.

The key is not restricted to alphanumeric characters and the underscore. The following will work just as well:

```
<?php
$redis->set("I 2 love Php!", "Also Redis now!");
$value = $redis->get("I 2 love Php!");
```

INCR (INCRBY) and DECR (DECRBY)

The `INCR` and `DECR` commands are used to increment and decrement values and are a great way to maintain counters. INCR and DECR increment/decrement their values by 1; you can also use `INCRBY` and `DECRBY` to adjust by larger intervals. Here's an example:

```
<?php
// increment the number of views by 1 for an article
// with id 234
$redis->incr("article_views_234");


// increment views for article 237 by 5
```

```
$redis->incrby("article_views_237", 5);

// decrement views for article 237
$redis->decr("article_views_237");

// decrement views for article 237 by 3
$redis->decrby("article_views_237", 3);
```

Redis Data Types

As I mentioned earlier, Redis has built-in data types. You may be thinking that it's odd to have data types in a NoSQL key-value storage system such as Redis, but this would be useful for developers to structure the information in a more meaningful way and perform specific operations which is typically much faster when the data is typed. Redis' data types are:

- **String** – the basic data type used in Redis in which you can store from few characters to the content of an entire file.
- **List** – a simple list of strings order by the insertion of its elements. You can add and remove elements from both the list's head and tail, so you can use this data type to implement queues.
- **Hash** – a map of string keys and string values. In this way you can represent objects (think of it as a one-level deep JSON object).
- **Set** – an unordered collection of strings where you can add, remove, and test for existence of members. The one constraint is that you are not allowed to have repeated members.
- **Sorted set** – a particular case of the set data type. The difference is that every member has and associated score that is used order the set from the smallest to the greatest score.

So far I've only demonstrated strings, but there are commands that make working with data in other data types just as easy.

HSET, HGET and HGETALL, HINCRBY, and HDEL

These commands are used to work with Redis' hash data type:

- **HSET** – sets the value for a key on the the hash object.
- **HGET** – gets the value for a key on the hash object.

- **HINCRBY** – increment the value for a key of the hash object with a specified value.
- **HDEL** – remove a key from the object.
- **HGETALL** – get all keys and data for a object.

Here's an example that demonstrates their usage:

```
<?php
$redis->hset("taxi_car", "brand", "Toyota");
$redis->hset("taxi_car", "model", "Yaris");
$redis->hset("taxi_car", "license number", "RO-01-PHP");
$redis->hset("taxi_car", "year of fabrication", 2010);
$redis->hset("taxi_car", "nr_starts", 0);
/*
$redis->hmset("taxi_car", array(
    "brand" => "Toyota",
    "model" => "Yaris",
    "license number" => "RO-01-PHP",
    "year of fabrication" => 2010,
    "nr_stats" => 0)
);
*/
echo "License number: " .
    $redis->hget("taxi_car", "license number") . "<br>";

// remove license number
$redis->hdel("taxi_car", "license number");

// increment number of starts
$redis->hincrby("taxi_car", "nr_starts", 1);

$taxi_car = $redis->hgetall("taxi_car");
echo "All info about taxi car";
echo "<pre>";
var_dump($taxi_car);
echo "</pre>";
```

LPUSH, RPUSH, LPOP, RPOP, LLEN, LRANGE

These are the important commands for working with the list type in Redis. A Redis list is similar to an array in PHP, and offer a great support for implementing queues, stacks, or a capped collection of a certain number of elements.

- **LPUSH** – prepends element(s) to a list.
- **RPUSH** – appends element(s) to a list.
- **LPOP** – removes and retrieves the first element of a list.
- **RPOP** – removes and retrieves the last element of a list.
- **LLEN** – gets the length of a list.
- **LRANGE** – gets elements from a list.

```
<?php
$list = "PHP Frameworks List";
$redis->rpush($list, "Symfony 2");
$redis->rpush($list, "Symfony 1.4");
$redis->lpush($list, "Zend Framework");

echo "Number of frameworks in list: " . $redis->llen($list) . "<br>";

$arList = $redis->lrange($list, 0, -1);
echo "<pre>";
print_r($arList);
echo "</pre>";

// the last entry in the list
echo $redis->rpop($list) . "<br>";

// the first entry in the list
echo $redis->lpop($list) . "<br>";
```

EXPIRE , EXPIREAT , TTL, and PERSIST

Most likely, when you set a key you don't want it to be saved forever because after a certain period of time it's not likely to be relevant anymore. You'll need to update its value or delete it to reduce memory usage for better performance. Redis offers four commands that let you handle data persistence easily.

- **EXPIRE** – sets an expiration timeout (in seconds) for a key after which it and its value will be deleted.

- **EXPIREAT** – sets and expiration time using a unix timestamp that represents when the key and value will be deleted.
- **TTL** – gets the remaining time left to live for a key with an expiration.
- **PERSIST** – removes the expiration on the given key.

```
<?php
// set the expiration for next week
$redis->set("expire in 1 week", "I have data for a week");
$redis->expireat("expire in 1 week", strtotime("+1 week"));
$ttl = $redis->ttl("expire in 1 week"); // will be 604800 seconds

// set the expiration for one hour
$redis->set("expire in 1 hour", "I have data for an hour");
$redis->expire("expire in 1 hour", 3600);
$ttl = $redis->ttl("expire in 1 hour"); // will be 3600 seconds

// never expires
$redis->set("never expire", "I want to leave forever!");
```

Summary

We looked at just a short list of Redis commands in this article, but you can check the whole list of commands on the Redis website. Indeed Redis has much more to offer than just being a Memcache replacement.

Redis is here for the long run; it has a growing community, support for all major languages, and offers durability and high-availability with master-slave replication. Redit is open source, so if you're a C guru then you can fork its source code from GitHub and become a contributor.

If you're looking for more information beyond the project site, you might want to consider checking out two great Redis books, Redis Cookbook and Redis: The Definitive Guide.