

# PHP-redis api 中文说明

**phpredis** 是 php 的一个扩展，效率是相当高有链表排序功能，对创建内存级的模块业务关系，很有用，以下是 redis 官方提供的命令使用技巧:

下载地址如下：

<https://github.com/owlient/phpredis> ( 支持 redis 2.0.4 )

**Redis::\_\_construct** 构造函数

```
$redis = new Redis();
```

**connect, open** 链接 redis 服务

**参数**

*host*: string , 服务地址

*port*: int, 端口号

*timeout*: float, 链接时长 (可选, 默认为 0 , 不限链接时间)

注: 在 redis.conf 中也有时间，默认为 300

**pconnect, popen** 不会主动关闭的链接

参考上面

**setOption** 设置 redis 模式

**getOption** 查看 redis 设置的模式

**ping** 查看连接状态

**get** 得到某个 key 的值 ( string 值 )

如果该 key 不存在 , return **false**

**set** 写入 key 和 value ( string 值 )

如果写入成功 , return **true**

**setex** 带生存时间的写入值

```
$redis->setex('key', 3600, 'value'); // sets key → value, with 1h TTL.
```

**setnx** 判断是否重复的 , 写入值

```
$redis->setnx('key', 'value');
```

```
$redis->setnx('key', 'value');
```

**delete** 删除指定 key 的值

返回已经删除 key 的个数 ( 长整数 )

```
$redis->delete('key1', 'key2');
```

```
$redis->delete(array('key3', 'key4', 'key5'));
```

## **ttl**

得到一个 key 的生存时间

## **persist**

移除生存时间到期的 key

如果 key 到期 true 如果不到期 false

**mset** ( redis 版本 1.1 以上才可以用 )

同时给多个 key 赋值

```
$redis->mset(array('key0' => 'value0', 'key1' => 'value1'));
```

## **multi, exec, discard**

进入或者退出事务模式

参数可选 Redis::MULTI 或 Redis::PIPELINE. 默认是 Redis::MULTI

Redis::MULTI : 将多个操作当成一个事务执行

Redis::PIPELINE: 让 ( 多条 ) 执行命令简单的, 更加快速的发送给服务器, 但是没有任何原

子性的保证

discard: 删除一个事务

返回值

multi(), 返回一个 redis 对象, 并进入 multi-mode 模式, 一旦进入 multi-mode 模式,

以后调用的所有方法都会返回相同的对象, 只到 exec() 方法被调用。

**watch, unwatch** （代码测试后，不能达到所说的效果）

监测一个 key 的值是否被其它的程序更改。如果这个 key 在 watch 和 exec（方法）间被修改，这个 MULTI/EXEC 事务的执行将失败（return false）

unwatch 取消被这个程序监测的所有 key

参数，一对 key 的列表

```
$redis->watch('x');
```

```
$ret = $redis->multi() ->incr('x') ->exec();
```

**subscribe \***

方法回调。注意，该方法可能在未来里发生改变

**publish \***

发表内容到某一个通道。注意，该方法可能在未来里发生改变

**exists**

判断 key 是否存在。存在 true 不在 false

**incr, incrBy**

key 中的值进行自增 1，如果填写了第二个参数，者自增第二个参数所填的值

```
$redis->incr('key1');
```

```
$redis->incrBy('key1', 10);
```

## **decr, decrBy**

做减法，使用方法同 incr

## **getMultiple**

传参

由 key 组成的数组

返回参数

如果 key 存在返回 value，不存在返回 false

```
$redis->set('key1', 'value1'); $redis->set('key2', 'value2'); $redis->set('key3',
```

```
'value3'); $redis->getMultiple(array('key1', 'key2', 'key3'));
```

```
$redis->lRem('key1', 'A', 2);
```

```
$redis->lRange('key1', 0, -1);
```

## **list 相关操作**

### **lPush**

```
$redis->lPush(key, value);
```

在名称为 key 的 list 左边（头）添加一个值为 value 的元素

### **rPush**

```
$redis->rPush(key, value);
```

在名称为 key 的 list 右边（尾）添加一个值为 value 的元素

### **lPushx/rPushx**

```
$redis->lPushx(key, value);
```

在名称为 key 的 list 左边(头)/右边(尾)添加一个值为 value 的元素,如果 value 已经存在,则不添加

### **lPop/rPop**

```
$redis->lPop('key');
```

输出名称为 key 的 list 左(头)起/右(尾)起的第一个元素,删除该元素

### **blPop/brPop**

```
$redis->blPop('key1', 'key2', 10);
```

lpop 命令的 block 版本。即当 timeout 为 0 时,若遇到名称为 key *i* 的 list 不存在或该 list 为空,则命令结束。如果 timeout>0,则遇到上述情况时,等待 timeout 秒,如果问题没有解决,则对 key<sub>*i*+1</sub>开始的 list 执行 pop 操作

### **lSize**

```
$redis->lSize('key');
```

返回名称为 key 的 list 有多少个元素

## **LIndex, LGet**

```
$redis->LGet('key', 0);
```

返回名称为 key 的 list 中 index 位置的元素

## **LSet**

```
$redis->LSet('key', 0, 'X');
```

给名称为 key 的 list 中 index 位置的元素赋值为 value

## **LRange, LGetRange**

```
$redis->LRange('key1', 0, -1);
```

返回名称为 key 的 list 中 start 至 end 之间的元素 ( end 为 -1 , 返回所有 )

## **LTrim, listTrim**

```
$redis->LTrim('key', start, end);
```

截取名称为 key 的 list , 保留 start 至 end 之间的元素

## **LRem, LRemove**

```
$redis->LRem('key', 'A', 2);
```

删除 count 个名称为 key 的 list 中值为 value 的元素。count 为 0 , 删除所有值为 value 的元素 ,count>0 从头至尾删除 count 个值为 value 的元素 ,count<0 从尾到头删除|count| 个值为 value 的元素

## lInsert

在名称为 `key` 的 list 中, 找到值为 *pivot* 的 value, 并根据参数 `Redis::BEFORE` | `Redis::AFTER`, 来确定, `newvalue` 是放在 `pivot` 的前面, 或者后面。如果 `key` 不存在, 不会插入, 如果 `pivot` 不存在, `return -1`

```
$redis->delete('key1');    $redis->lInsert('key1',    Redis::AFTER,    'A',    'X');  
  
$redis->lPush('key1', 'A'); $redis->lPush('key1', 'B'); $redis->lPush('key1', 'C');  
  
$redis->lInsert('key1', Redis::BEFORE, 'C', 'X');  
  
$redis->lRange('key1', 0, -1);  
  
$redis->lInsert('key1', Redis::AFTER, 'C', 'Y');  
  
$redis->lRange('key1', 0, -1);  
  
$redis->lInsert('key1', Redis::AFTER, 'W', 'value');
```

## rpoplpush

返回并删除名称为 `srckey` 的 list 的尾元素, 并将该元素添加到名称为 `dstkey` 的 list 的头部

```
$redis->delete('x', 'y');  
  
$redis->lPush('x', 'abc'); $redis->lPush('x', 'def'); $redis->lPush('y', '123');  
  
$redis->lPush('y', '456'); // move the last of x to the front of y.  
  
var_dump($redis->poplpush('x', 'y'));  
  
var_dump($redis->lRange('x', 0, -1));  
  
var_dump($redis->lRange('y', 0, -1));  
  
string(3) "abc"
```



```
array(1) { [0]=> string(3) "def" }
```

```
array(3) { [0]=> string(3) "abc" [1]=> string(3) "456" [2]=> string(3) "123" }
```

## **SET 操作相关**

### **sAdd**

向名称为 key 的 set 中添加元素 value,如果 value 存在 , 不写入 , return false

```
$redis->sAdd(key , value);
```

### **sRem, sRemove**

删除名称为 key 的 set 中的元素 value

```
$redis->sAdd('key1' , 'set1');
```

```
$redis->sAdd('key1' , 'set2');
```

```
$redis->sAdd('key1' , 'set3');
```

```
$redis->sRem('key1', 'set2');
```

### **sMove**

将 value 元素从名称为 srckey 的集合移到名称为 dstkey 的集合

```
$redis->sMove(seckey, dstkey, value);
```

### **sIsMember, sContains**

名称为 key 的集合中查找是否有 value 元素 , 有 true 没有 false

```
$redis->sIsMember(key, value);
```

### **sCard, sSize**

返回名称为 key 的 set 的元素个数

### **sPop**

随机返回并删除名称为 key 的 set 中一个元素

### **sRandMember**

随机返回名称为 key 的 set 中一个元素，不删除

### **sInter**

求交集

### **sInterStore**

求交集并将交集保存到 output 的集合

```
$redis->sInterStore('output', 'key1', 'key2', 'key3')
```

### **sUnion**

求并集

```
$redis->sUnion('s0', 's1', 's2');
```

s0,s1,s2 同时求并集

## **sUnionStore**

求并集并将并集保存到 output 的集合

```
$redis->sUnionStore('output', 'key1', 'key2', 'key3') ;
```

## **sDiff**

求差集

## **sDiffStore**

求差集并将差集保存到 output 的集合

## **sMembers, sGetMembers**

返回名称为 key 的 set 的所有元素

## **sort**

排序，分页等

参数

'by' => 'some\_pattern\_\*',

'limit' => array(0, 1),

'get' => 'some\_other\_pattern\_\*' or an array of patterns,

'sort' => 'asc' or 'desc',

'alpha' => TRUE,

'store' => 'external-key'

例子

```
$redis->delete('s'); $redis->sadd('s', 5); $redis->sadd('s', 4); $redis->sadd('s', 2);
```

```
$redis->sadd('s', 1); $redis->sadd('s', 3);
```

```
var_dump($redis->sort('s')); // 1,2,3,4,5
```

```
var_dump($redis->sort('s', array('sort' => 'desc'))); // 5,4,3,2,1
```

```
var_dump($redis->sort('s', array('sort' => 'desc', 'store' => 'out'))); // (int)5
```

## string 命令

### getSet

返回原来 key 中的值，并将 value 写入 key

```
$redis->set('x', '42');
```

```
$exValue = $redis->getSet('x', 'lol'); // return '42', replaces x by 'lol'
```

```
$newValue = $redis->get('x') // return 'lol'
```

### append

string，名称为 key 的 string 的值在后面加上 value

```
$redis->set('key', 'value1');
```

```
$redis->append('key', 'value2');
```

```
$redis->get('key');
```

### getRange（方法不存在）

返回名称为 key 的 string 中 start 至 end 之间的字符

```
$redis->set('key', 'string value');
```

```
$redis->getRange('key', 0, 5);
```

```
$redis->getRange('key', -5, -1);
```

### **setRange** ( 方法不存在 )

改变 key 的 string 中 start 至 end 之间的字符为 value

```
$redis->set('key', 'Hello world');
```

```
$redis->setRange('key', 6, "redis");
```

```
$redis->get('key');
```

### **strlen**

得到 key 的 string 的长度

```
$redis->strlen('key');
```

### **getBit/setBit**

返回 2 进制信息

### **zset ( sorted set ) 操作相关**

**zAdd(key, score, member)** : 向名称为 key 的 zset 中添加元素 member , score 用于排序。如果该元素已经存在 , 则根据 score 更新该元素的顺序。

```
$redis->zAdd('key', 1, 'val1');
```

```
$redis->zAdd('key', 0, 'val0');
```

```
$redis->zAdd('key', 5, 'val5');
```

```
$redis->zRange('key', 0, -1); // array(val0, val1, val5)
```

***zRange(key, start, end, withscores)*** : 返回名称为 key 的 zset ( 元素已按 score 从小到大排序 ) 中的 index 从 start 到 end 的所有元素

```
$redis->zAdd('key1', 0, 'val0');
```

```
$redis->zAdd('key1', 2, 'val2');
```

```
$redis->zAdd('key1', 10, 'val10');
```

```
$redis->zRange('key1', 0, -1); // with scores $redis->zRange('key1', 0, -1, true);
```

## **zDelete, zRem**

***zRem(key, member)*** : 删除名称为 key 的 zset 中的元素 member

```
$redis->zAdd('key', 0, 'val0');
```

```
$redis->zAdd('key', 2, 'val2');
```

```
$redis->zAdd('key', 10, 'val10');
```

```
$redis->zDelete('key', 'val2');
```

```
$redis->zRange('key', 0, -1);
```

***zRevRange(key, start, end, withscores)*** : 返回名称为 key 的 zset ( 元素已按 score 从大到小排序 ) 中的 index 从 start 到 end 的所有元素. *withscores*: 是否输出 score 的值 , 默认 false , 不输出

```
$redis->zAdd('key', 0, 'val0');
```

```
$redis->zAdd('key', 2, 'val2');
```

```
$redis->zAdd('key', 10, 'val10');
```

```
$redis->zRevRange('key', 0, -1); // with scores $redis->zRevRange('key', 0, -1, true);
```

### **zRangeByScore, zRevRangeByScore**

```
$redis->zRangeByScore(key, star, end, array(withscores , limit ));
```

返回名称为 key 的 zset 中  $score \geq star$  且  $score \leq end$  的所有元素

### **zCount**

```
$redis->zCount(key, star, end);
```

返回名称为 key 的 zset 中  $score \geq star$  且  $score \leq end$  的所有元素的个数

### **zRemRangeByScore, zDeleteRangeByScore**

```
$redis->zRemRangeByScore('key', star, end);
```

删除名称为 key 的 zset 中  $score \geq star$  且  $score \leq end$  的所有元素，返回删除个数

### **zSize, zCard**

返回名称为 key 的 zset 的所有元素的个数

### **zScore**

```
$redis->zScore(key, val2);
```

返回名称为 key 的 zset 中元素 val2 的 score

## **zRank, zRevRank**

```
$redis->zRevRank(key, val);
```

返回名称为 key 的 zset ( 元素已按 score 从小到大排序 ) 中 val 元素的 rank ( 即 index , 从 0 开始 ) , 若没有 val 元素 , 返回 "null" 。 zRevRank 是从大到小排序

## **zIncrBy**

```
$redis->zIncrBy('key', increment, 'member');
```

如果在名称为 key 的 zset 中已经存在元素 member , 则该元素的 score 增加 increment ; 否则向集合中添加该元素 , 其 score 的值为 increment

## **zUnion/zInter**

*参数*

*keyOutput*

*arrayZSetKeys*

*arrayWeights*

*aggregateFunction* Either "SUM", "MIN", or "MAX": defines the behaviour to use on duplicate entries during the zUnion.

对 N 个 zset 求并集和交集 , 并将最后的集合保存在 dstkeyN 中。对于集合中每一个元素的 score , 在进行 AGGREGATE 运算前 , 都要乘以对于的 WEIGHT 参数。如果没有提供 WEIGHT , 默认为 1。默认的 AGGREGATE 是 SUM , 即结果集合中元素的 score 是所有集合对应元素进行 SUM 运算的值 , 而 MIN 和 MAX 是指 , 结果集合中元素的 score 是所有



集合对应元素中最小值和最大值。

## Hash 操作

### hSet

```
$redis->hSet('h', 'key1', 'hello');
```

向名称为 h 的 hash 中添加元素 key1—>hello

### hGet

```
$redis->hGet('h', 'key1');
```

返回名称为 h 的 hash 中 key1 对应的 value ( hello )

### hLen

```
$redis->hLen('h');
```

返回名称为 h 的 hash 中元素个数

### hDel

```
$redis->hDel('h', 'key1');
```

删除名称为 h 的 hash 中键为 key1 的域

### hKeys

```
$redis->hKeys('h');
```

返回名称为 key 的 hash 中所有键

## **hVals**

```
$redis->hVals('h')
```

返回名称为 h 的 hash 中所有键对应的 value

## **hGetAll**

```
$redis->hGetAll('h');
```

返回名称为 h 的 hash 中所有的键 ( field ) 及其对应的 value

## **hExists**

```
$redis->hExists('h', 'a');
```

名称为 h 的 hash 中是否存在键名字为 a 的域

## **hIncrBy**

```
$redis->hIncrBy('h', 'x', 2);
```

将名称为 h 的 hash 中 x 的 value 增加 2

## **hMset**

```
$redis->hMset('user:1', array('name' => 'Joe', 'salary' => 2000));
```

向名称为 key 的 hash 中批量添加元素

## **hMGet**

```
$redis->hmGet('h', array('field1', 'field2'));
```

返回名称为 h 的 hash 中 field1,field2 对应的 value

## redis 操作相关

### flushDB

清空当前数据库

### flushAll

清空所有数据库

### randomKey

随机返回 key 空间的一个 key

```
$key = $redis->randomKey();
```

### select

选择一个数据库

### move

转移一个 key 到另外一个数据库

```
$redis->select(0); // switch to DB 0
```

```
$redis->set('x', '42'); // write 42 to x
```

```
$redis->move('x', 1); // move to DB 1
```

```
$redis->select(1); // switch to DB 1
```

```
$redis->get('x'); // will return 42
```

### **rename, renameKey**

给 key 重命名

```
$redis->set('x', '42');
```

```
$redis->rename('x', 'y');
```

```
$redis->get('y'); // → 42
```

```
$redis->get('x'); // → `FALSE`
```

### **renameNx**

与 remane 类似，但是，如果重新命名的名字已经存在，不会替换成功

### **setTimeout, expire**

设定一个 key 的活动时间 ( s )

```
$redis->setTimeout('x', 3);
```

### **expireAt**

key 存活到一个 unix 时间戳时间

```
$redis->expireAt('x', time() + 3);
```

### **keys, getKeys**

返回满足给定 pattern 的所有 key

```
$keyWithUserPrefix = $redis->keys('user*');
```

## **dbSize**

查看现在数据库有多少 key

```
$count = $redis->dbSize();
```

## **auth**

密码认证

```
$redis->auth('foobared');
```

## **bgrewriteaof**

使用 aof 来进行数据库持久化

```
$redis->bgrewriteaof();
```

## **slaveof**

选择从服务器

```
$redis->slaveof('10.0.1.7', 6379);
```

## **save**

将数据同步保存到磁盘

## **bgsave**

将数据异步保存到磁盘

## **lastSave**

返回上次成功将数据保存到磁盘的 Unix 时戳

## **info**

返回 redis 的版本信息等详情

## **type**

返回 key 的类型值

string: Redis::REDIS\_STRING

set: Redis::REDIS\_SET

list: Redis::REDIS\_LIST

zset: Redis::REDIS\_ZSET

hash: Redis::REDIS\_HASH

other: Redis::REDIS\_NOT\_FOUND