[**redis并发问题**](http://www.cnblogs.com/iforever/p/5796902.html)

**redis中的并发问题**

使用redis作为缓存已经很久了，redis是以单进程的形式运行的，命令是一个接着一个执行的，一直以为不会存在并发的问题，直到今天看到相关的资料，才恍然大悟~~

**具体问题实例**

有个键，假设名称为myNum，里面保存的是阿拉伯数字，假设现在值为1，存在多个连接对myNum进行操作的情况，这个时候就会有并发的问题。假设有两个连接linkA和linkB，这两个连接都执行下面的操作，取出myNum的值，+1，然后再存回去，看看下面的交互：

linkA get myNum => 1

linkB get myNum => 1

linkA set muNum => 2

linkB set myNum => 2

执行完操作之后，结果可能是2，这和我们预期的3不一致。  
再看一个具体的例子：

<?php

require "vendor/autoload.php";

$client = new Predis\Client([

'scheme' => 'tcp',

'host' => '127.0.0.1',

'port' => 6379,

]);

for ($i = 0; $i < 1000; $i++) {

$num = intval($client->get("name"));

$num = $num + 1;

$client->setex("name", $num, 10080);

usleep(10000);

}

设置name初始值为0，然后同时用两个终端执行上面的程序，最后name的值可能不是2000，而是一个<2000的值，这也就证明了我们上面的并发问题的存在，这个该怎么解决呢？

**redis中的事务**

redis中也是有事务的，不过这个事务没有mysql中的完善，只保证了一致性和隔离性，不满足原子性和持久性。  
redis事务使用multi、exec命令

原子性，redis会将事务中的所有命令执行一遍，哪怕是中间有执行失败也不会回滚。kill信号、宿主机宕机等导致事务执行失败，redis也不会进行重试或者回滚。

持久性，redis事务的持久性依赖于redis所使用的持久化模式，遗憾的是各种持久化模式也都不是持久化的。

隔离性，redis是单进程，开启事务之后，会执行完当前连接的所有命令直到遇到exec命令，才处理其他连接的命令。  
一致性，看了文档，觉得挺扯的，但是貌似说的没有问题。

redis中的事务不支持原子性，所以解决不了上面的问题。

当然了redis还有一个watch命令，这个命令可以解决这个问题，看下面的例子，对一个键执行watch，然后执行事务，由于watch的存在，他会监测键a，当a被修该之后，后面的事务就会执行失败，这就确保了多个连接同时来了，都监测着a，只有一个能执行成功，其他都返回失败。

127.0.0.1:6379> set a 1

OK

127.0.0.1:6379> watch a

OK

127.0.0.1:6379> multi

OK

127.0.0.1:6379> incr a

QUEUED

127.0.0.1:6379> exec

1) (integer) 2

127.0.0.1:6379> get a

"2"

失败时候的例子，从最后可以看出，test的值被其他连接修改了：

127.0.0.1:6379> set test 1

OK

127.0.0.1:6379> watch test

OK

127.0.0.1:6379> multi

OK

127.0.0.1:6379> incrby test 11

QUEUED

127.0.0.1:6379> exec

(nil)

127.0.0.1:6379> get test

"100"

**我的问题如何解决**

redis中命令是满足原子性的，因此在值为阿拉伯数字的时候，我可以将get和set命令修改为incr或者incrby来解决这个问题，下面的代码开启两个终端同时执行，得到的结果是满足我们预期的2000。

<?php

require "vendor/autoload.php";

$client = new Predis\Client([

'scheme' => 'tcp',

'host' => '127.0.0.1',

'port' => 6379,

]);

for ($i = 0; $i < 1000; $i++) {

$client->incr("name");

$client->expire("name", 10800);

usleep(10000);

}

**@manzilu 提到的方法**

评论中[manzilu](http://www.cnblogs.com/manziluo/)提到的方法查了下资料，确实可行，效果还不错，这里写了个例子

<?php

require "vendor/autoload.php";

$client = new Predis\Client([

'scheme' => 'tcp',

'host' => '127.0.0.1',

'port' => 6379,

]);

class RedisLock

{

public $objRedis = null;

public $timeout = 3;

/\*\*

\* @desc 设置redis实例

\*

\* @param obj object | redis实例

\*/

public function \_\_construct($obj)

{

$this->objRedis = $obj;

}

/\*\*

\* @desc 获取锁键名

\*/

public function getLockCacheKey($key)

{

return "lock\_{$key}";

}

/\*\*

\* @desc 获取锁

\*

\* @param key string | 要上锁的键名

\* @param timeout int | 上锁时间

\*/

public function getLock($key, $timeout = NULL)

{

$timeout = $timeout ? $timeout : $this->timeout;

$lockCacheKey = $this->getLockCacheKey($key);

$expireAt = time() + $timeout;

$isGet = (bool)$this->objRedis->setnx($lockCacheKey, $expireAt);

if ($isGet) {

return $expireAt;

}

while (1) {

usleep(10);

$time = time();

$oldExpire = $this->objRedis->get($lockCacheKey);

if ($oldExpire >= $time) {

continue;

}

$newExpire = $time + $timeout;

$expireAt = $this->objRedis->getset($lockCacheKey, $newExpire);

if ($oldExpire != $expireAt) {

continue;

}

$isGet = $newExpire;

break;

}

return $isGet;

}

/\*\*

\* @desc 释放锁

\*

\* @param key string | 加锁的字段

\* @param newExpire int | 加锁的截止时间

\*

\* @return bool | 是否释放成功

\*/

public function releaseLock($key, $newExpire)

{

$lockCacheKey = $this->getLockCacheKey($key);

if ($newExpire >= time()) {

return $this->objRedis->del($lockCacheKey);

}

return true;

}

}

$start\_time = microtime(true);

$lock = new RedisLock($client);

$key = "name";

for ($i = 0; $i < 10000; $i++) {

$newExpire = $lock->getLock($key);

$num = $client->get($key);

$num++;

$client->set($key, $num);

$lock->releaseLock($key, $newExpire);

}

$end\_time = microtime(true);

echo "花费时间 ： ". ($end\_time - $start\_time) . "\n";

执行shell php setnx.php & php setnx.php&，最后会得到结果：

$ 花费时间 ： 4.3004920482635

[2] + 72356 done php setnx.php

# root @ ritoyan-virtual-pc in ~/PHP/redis-high-concurrency [20:23:41]

$ 花费时间 ： 4.4319710731506

[1] + 72355 done php setnx.php

同样循环1w次，去掉usleep，使用incr直接进行增加，耗时在2s左右。  
而获取所得时候取消usleep，时间不但没减少，反而增加了，这个usleep的设置要合理，免得进程做无用的循环

**总结**

看了这么多，简单的总结下，其实redis本事是不会存在并发问题的，因为他是单进程的，再多的command都是one by one执行的。我们使用的时候，可能会出现并发问题，比如get和set这一对。

2016-8-22 20:31:30

本文版权归作者iforever(luluyrt＠163.com)所有，未经作者本人同意禁止任何形式的转载，转载文章之后必须在文章页面明显位置给出作者和原文连接，否则保留追究法律责任的权利。

分类: [php(后台)](http://www.cnblogs.com/iforever/category/629912.html)

[好文要顶](javascript:void(0);)

[https://pic.cnblogs.com/face/675818/20161225095349.png](http://home.cnblogs.com/u/iforever/)

[奔跑的Man](http://home.cnblogs.com/u/iforever/)  
[关注 - 15](http://home.cnblogs.com/u/iforever/followees)  
[粉丝 - 89](http://home.cnblogs.com/u/iforever/followers)

[+加关注](javascript:void(0);)

16

0

[«](http://www.cnblogs.com/iforever/p/5736142.html) 上一篇：[短信验证——上行和下行](http://www.cnblogs.com/iforever/p/5736142.html)[»](http://www.cnblogs.com/iforever/p/5857898.html) 下一篇：[说说APP接口中的版本控制](http://www.cnblogs.com/iforever/p/5857898.html)

posted @ 2016-08-22 20:30 [奔跑的Man](http://www.cnblogs.com/iforever/) 阅读(28684) 评论(14) [编辑](https://i.cnblogs.com/EditPosts.aspx?postid=5796902) [收藏](https://www.cnblogs.com/iforever/p/5796902.html)