#Redis读写锁机制

##redis实现lock互斥访问资源

Redis是当前很流行的一种开源键值数据库。目前睿思的后台架构在数据库层采用了Redis和MySQL组合的形式，其中Redis主要用来存储状态信息（比如当前种子的peer）和读写频繁的数据。Redis完全运行在内存之上，无lock设计，速度非常快！通过实测，在睿思服务器上读写速度达到3万次/s。   
 在高并发的应用中，很多时候我们需要对某些资源进行竞争访问，比如在很多人[下载](http://www.2cto.com/soft" \t "_blank)一个热门资源，就可能存在很多请求去修改某个资源的peer信息（就是保存了当前保种人的ip地址和端口号），需要保证某个请求修改peer信息的时候，不允许其他请求修改，否则就会出现数据覆盖的问题。但是Redis没有提供对数据的加锁，所以需要我们通过Redis提供的命令自己实现：   
**思路一：通过get 和set 命令实现** 这种方式很容易想到，就是当每次请求到来时通过get判断这个锁是否存在，如果不存在则set创建。这种方法有一个弊端，由于get和set是两次Redis请求，二者之间有延时，在高并发的环境下，有可能在get检测到锁不存之后在set之前已经被其他线程set，这时当前线程再set，这样锁就失效了。所以这种方法只能应对并发量不是很高的情况。   
**思路二：通过setnx 和 expire命令实现** 在访问需要互斥访问的资源时，通过setnx命令去设置一个lock 键，setnx的作用是判断锁是否存在，如果不存在则创建，返回成功，如果存在则返回失败，服务器返回给客户端，指示客户端稍后重试。expire命令用于给该锁设定一个过期时间，用于防止线程crash，导致锁一直有效，从而导致死锁。例如：设定锁的有效期为100秒，那么即使线程奔溃，在100秒后锁会自动失效。

1. setnx lock “lock”
2. 如果锁设置成功则创建该键并设置值

为此锁设置一个过期时间

expire lock 100

访问互斥资源后要及时删除锁

del lock

**思路三：通过watch和Redis的事务命令实现** 这种方式的效果和思路二类似。在请求到时先watch该资源锁，然后再通过在事务执行 创建锁的过程，锁的键值能唯一标识改请求（比如用时间+用户标识）。如果当前还有其他线程请求该资源，当判断该锁存在时则返回错误重试（例如睿思BT tracker返回“服务器过载，自动重试的”的提示就属于此类情况）。如果有多个请求同时判断该锁不存在而创建锁，这样也会由于watch了这个锁，导致之前watch的线程执行事务失败，返回客户端自动重试。这样达最终达到了锁的目的。

## 利用锁机制来放置缓存过期产生的惊群现象

缓存惊群现象，在各种缓存中都会存在这种现象，这里以Redis为例，提供一种解决思路，留作参考~

首先，所谓的缓存过期引起的“惊群”现象是指，在大并发情况下，我们通常会用缓存来给数据库分压，但是会有这么一种情况发生，那就是当一个缓存数据失效之后会导致同时有多个并发线程去向后端数据库发起请求去获取同一个数据，这样如果在一段时间内同时生成了大量的缓存，然后在另外一段时间内又有大量的缓存失效，这样就会导致后端数据库的压力突然增大，这种现象就可以称为“缓存过期产生的惊群现象”！

以下代码的思路，就是利用“锁机制”来防止惊群现象。先看代码：

class KomaRedis{

    private $redis; //redis对象

    private static $\_instance = null;

    private function \_\_construct($config = array())

    {

        if (empty($config)) {

            return false;

        }

        $this->redis = new Redis();

        $this->redis->connect($config['server'], $config['port']);

        return $this->redis;

    }

    /\*\*

     \* @param array $config

     \* @return redis操作类对象

     \*/

    public static function getInstance($config = array())

    {

        if (!(self::$\_instance instanceof self)) {//self是本类

            self::$\_instance = new self ($config);

        }

        return self::$\_instance;

    }

    /\*\*

     \* 获取缓存，从下面的方法设置的缓存格式里好的缓存

     \* @param $key string $name

     \* @return array,object,number,string,boolean

     \* @desc 此方法使用了锁机制来防止缓存过期时所产生的惊群现象，保证只有一个进程不获取数据，可以更新，其他进程仍然获取过期数据

     \*/

    public function getByLock ($key)//根据锁的情况来获取值

    {

        $sth = $this->redis->get($key);

        if ($sth === false) {

            return $sth;

        } else {

            $sth = json\_decode($sth, TRUE);//将缓存值的json数据解析成数组，该数组包括data键值对应值，和expire对应的是过期时间

            if (intval($sth['expire']) <= time()) {//如果已经过了有效期

                $lock = $this->redis->incr($key . ".lock");//当键不存在时会创建该键并赋值为0，再增加1得到结果1

                if ($lock === 1) {//说明是第一次加锁，可能没有更新时间，就当做数据过期

保证只有一个进程不获取数据，可以更新，其他进程仍然获取过期数据

                    return false;//值已经过期

                } else {//说明之前已经加过了锁更新了数据的有效期，那么可以直接返回值但不能设置值

                    return $sth['data'];

                }

            } else {//如果数据没有到过期时间，则直接返回数据

                return $sth['data'];

            }

        }

    }

原始数据的有效期已经过了，则如果更新的有效期（即设置缓存的有效期）即能拿到缓存则直接返回数据，如果不能拿到缓存则数据已经失效，如果原始数据没有过期则直接返回数据。

    /\*\*

     \* 设置缓存

     \* @param $key string $name 缓存键

     \* @param $value $string ,array,object,number,boolean $value 缓存值

     \* @param null $ttl $string ,number $ttl 过期时间，如果不设置，则使用默认时间，如果为 infinity 则为永久保存

     \* @return bool

     \* @desc 此方法存储的数据会自动加入一些其他数据来避免惊群现象，如需保存原始数据，请使用 set

     \*/

    public function setByLock($key, $value, $ttl = null)

    {

        if (is\_numeric($ttl) && intval($ttl) > 0) {

            $ttl = intval($ttl);

            $exp = time() + $ttl;

            $arg = array("data" => $value, "expire" => $exp);

        } else {

            $ttl = 300;//设置默认过期时间

            $exp = time() + $ttl;

        }

        empty($ttl) OR $ttl += 300; //增加redis缓存时间，使程序有足够的时间生成缓存

        $arg = array("data" => $value, "expire" => $exp);//这里设置的缓存键为data

        $rs = $this->redis->setex($key, $ttl, json\_encode($arg, TRUE));//值被变成了json格式，注意此时的json与之前的getByLock方法，解析json从里面获取数据

        $this->redis->del($key . ".lock");//删除之前设置的键锁

        return $rs;

    }

    /\*\*

     \* 返回redis对象

     \* redis有非常多的操作方法，我们只封装了一部分

     \* 拿着这个对象就可以直接调用redis自身方法

     \*/

    public function redis()

    {

        return $this->redis;

    }

}

原理就是：

首先，在存储数据的时候，设置数据的过期时间比实际设置的过期时间多300秒，然后存储的数据中，（setByLock方法）通过一个数组来存储数据，数组中一个键用来存放真实的数据，另外一个键用来存放数据的真实过期时间，这个留到后期获取数据的时候做校验，然后把对应这个数据的“锁”删除掉。

这里这么做的原因和读取数据的做法相关！

然后，在读取数据的时候，依然像平时一样直接读取，如果数据已经超过了有效期（注意：这里的有效期并非设置的有效期，而是更改之后的有效期（缓存的有效期）），那么就只能去读后端数据库。如果数据依然有效，则需要去判断，判断数据“在真正的有效期内是否失效”，如果没有失效，则直接返回数据！

重点是，假如数据“在伪造的有效期内没有失效，而在真正的有效期内已经失效”，那么这时就需要去判断“数据的锁”！

通过代码“$lock = $this->redis->incr($key . ".lock");”可以获取数据的锁，“$lock === 1”表示数据没有锁，那么这一次请求需要发送到后端数据库去读取最新的数据，否则的话表示该数据已经加了锁，也就是已经有一个线程去后端读取数据了（后来的线程不需要再到后端读取数据），那么后来的线程也就没有权限再去后端取数据，需要等到前面的那个线程执行结束，但是这次读取就只能读取“旧的数据”了！

通过上面的解释也就明白，为什么在存储数据的时候需要“删除数据的锁”！因为一旦数据被重新存储，那么说明已经有一个线程去后端得到了最新的数据，那么该数据的锁就可以释放，然后下一个线程在获取数据的时候如果有需要就可以得到这个锁，然后才有权限进入到后端去读取新数据！