php zookeeper你需要知道的细节

作为一名php程序员，一直都是在做基础的开发工作，总感觉技能上没有什么实质性的突破。渐渐的了解了分布式开发的重要性以后，近期也逐渐将魔掌伸向了分布式领域。首先接触到的一款应用就是zookeeper。

相信能看到这篇文章的看官对zookeeper都有一定的了解，而在下目前刚接触zookeeper，可以说还是一个门外汉。所以对于zookeeper的描述以及zookeeper的优缺点实在是不敢妄下定论。有像在下一样，对于zookeeper还不是太清楚的可以去官网[Apache ZooKeeper](http://zookeeper.apache.org/)细细品读。

略过闲话不题，下面我们进入正文。本篇主要是想大家简单介绍以下几个方面：

php如何集成zookeeper

php zookeeper简单案例

php zookeeper运行中的问题以及代码分析

下面，请各位看官听我一一道来。

**php如何集成zookeeper**

话说，zookeeper是一个Java应用程序，性能强大可以说是名闻天下。可偏偏我们要用php来操作zookeeper。还好其提供了C客户端，既然php是用C扩展的，因而可以继续用C扩展一个php插件。进而产生了一个php扩展应用[PHP-ZooKeeper](https://github.com/andreiz/php-zookeeper)（可以点击链接从而下载该应用）。下面我们来介绍该应用的安装。

这里假设各位看官已经安装了zookeeper，我们在这不再做安装。但是在zookeeper中有一个C客户端需要我们安装。因为我们需要这个C客户端为PHP-Zookeeper提供类库。下面是安装过程（以zookeeper-3.4.9为例）。

# tar –zxvf zookeeper-3.4.9.tar.gz

# cd zookeeper-3.4.9/src/c //进入c源目录

# ./configure --prefix=/opt/zk\_c //安装目录

# make && make install //编译安装

至此，zookeeper中C客户端就已经安装完成。下面进行PHP-ZooKeeper的安装。

# cd php-zookeeper

# /usr/local/php5/bin/phpize //php安装目录下面的phpize命令

# ./configure –with-php-config=/usr/local/php5/bin/php-config –with-libzookeeper-dir=/opt/zk\_c //这里是c客户端安装目录

# make

# make test //此步骤可以省略，建议执行

# make install

此时php-zookeeper就已经安装成功。然后我们编辑php.ini文件，在里面加入zookeeper扩展

# vim /usr/local/php5/etc/php.ini

添加下面一行代码。

extension=zookeeper.so

到这里PHP-ZooKeeper已经安装成功了，该过程很顺利，一般情况下都不会出现什么问题。

下面继续。

**php zookeeper简单案例**

既然PHP-ZooKeeper已经安装成功，那各位，实该提供两个例子给大家赏玩。下面两个例子乃旁人所造，虽无实用，亦是经典，可说明问题，故在此引用。

例一：

# bin/zkCli.sh -server 127.0.0.1:2181

[zk: 127.0.0.1:2181(CONNECTED) 1] ls /

[cluster, zookeeper]

[zk: 127.0.0.1:2181(CONNECTED) 2] create /phpDemo 1

Created /phpDemo

[zk: 127.0.0.1:2181(CONNECTED) 3] ls /

[cluster, zookeeper, phpDemo]

[zk: 127.0.0.1:2181(CONNECTED) 4]

首先连接zookeeper服务，然后创建一个名为 /phpDemo——如上代码所示——的znode。然后新建phpZookeeperDemo.php文件，写入代码如下：

<?php

***class*** ZookeeperDemo ***extends*** Zookeeper{

***public function*** *\_\_construct*(*$host*,*$watcher\_cb* = ***null***, *$recv\_timeout* = 1000){ ***parent***::*\_\_construct*(*$host*,*$watcher\_cb*,*$recv\_timeout*);

}

***public function*** *watcher*(*$i*,*$type*,*$key*){

***echo*** "Insider Watcher\n";

//监听 /phpDemo 节点

*$this*->*get*('/phpDemo',***array***(*$this*,'watcher'));

}

}

*$zk\_php* = ***new*** ZookeeperDemo('127.0.0.1:2181');

*$zk\_php*->*get*('/phpDemo',***array***(*$zk\_php*,'watcher'));

***while***(***true***){

***echo*** '.';

*sleep*(2);

}

现在运行该脚本

# php phpZookeeperDemo.php

这是应该会在命令行每隔2秒产生一个点。然后切换到zookeeper客户端，更新znode /phpDemo的值

[zk: 127.0.0.1:2181(CONNECTED) 4] set /phpDemo 2

这样就会触发PHP脚本中watcher方法然后输出‘Inside Watcher’，并继续监听。我们可以看到，其实zookeeper:get()方法的第二个参数就是监听事件的回调函数。当触发事件时，监视器会被销毁，所以我们需要在ZookeeperDemo:watcher方法中再次使用zookeeper:get()方法继续监听。

上面例子应该很好理解。下面我们来看一个分布式应用的例子。

例二：

其中的挑战是让这些独立的程序决定哪个（是leader）协调它们的工作，以及哪些（是worker）需要执行。这个处理过程叫做leader选举，在[ZooKeeper Recipes and Solutions](http://zookeeper.apache.org/doc/trunk/recipes.html#sc_leaderElection)你能看到相关的实现方法。

这里简单来说就是，每个处理（或服务器）紧盯着相邻的那个处理（或服务器）。如果一个已被监视的处理（也即Leader）退出或者崩溃了，监视程序就会查找其相邻（此时最老）的那个处理作为Leader。

在真实的应用程序中，leader会给worker分配任务、监控进程和保存结果。这里为了简化，我跳过了这些部分。

创建一个新的PHP文件，命名为worker.php：

<?php

class Worker extends Zookeeper{

const CONTAINER = '/cluster';

protected $acl = array(

array(

'perms' => Zookeeper::PERM\_ALL,

'scheme' => 'world',

'id' => 'anyone'

)

);

private $isLeader = false;

private $znode;

public function \_\_construct($host = '', $watcher\_cb = null, $recv\_timeout = 100){

parent::\_\_construct($host, $watcher\_cb, $recv\_timeout);

}

public function register(){

if (! $this->exists(self::CONTAINER)) {

$this->create(self::CONTAINER, null, $this->acl);

}

$this->znode = $this->create(self::CONTAINER . '/w-', null, $this->acl, Zookeeper::EPHEMERAL | Zookeeper::SEQUENCE);

$this->znode = str\_replace(self::CONTAINER . '/', '', $this->znode);

printf("I'm registerd as %s\n", $this->znode);

$watching = $this->watchPrevious();

if ($watching == $this->znode) {

printf("Nobody here , I'm the leader\n");

$this->setleader(true);

} else {

printf("I'm watching %s\n", $watching);

}

}

public function watchPrevious(){

$workers = $this->getChildren(self::CONTAINER);

sort($workers);

$size = sizeof($workers);

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

if ($this->znode == $workers[$i]) {

if ($i > 0) {

$res = $this->get(self::CONTAINER . '/' . $workers[$i - 1], array(

$this,

'watchNode'

));

return $workers[$i - 1];

}

return $workers[$i];

}

}

throw new Exception(sprintf("Something went very wrong! I can't find myself: %s/%s", self::CONTAINER, $this->znode));

}

public function watchNode(){

$watching = $this->watchPrevious();

if ($watching == $this->znode) {

printf("I'm the new leader\n");

$this->setLeader(true);

} else {

printf("Now I'm watching %s\n", $watching);

}

}

public function isLeader(){

return $this->isLeader;

}

public function setLeader($flag){

$this->isLeader = $flag;

}

public function run(){

$this->register();

while (true) {

if ($this->isLeader()) {

$this->doLeaderJob();

} else {

$this->doWorkerJob();

}

sleep(2);

}

}

public function doLeaderJob(){

echo "Leading\n";

}

public function doWorkerJob(){

echo "Working\n";

}

}

$worker = new Worker("127.0.0.1:2181");

$worker->run();

打开至少3个终端，在每个终端中运行以下脚本：

term1

# php worker.php

I'm registerd as w-0000000044

Nobody here , I'm the leader

Leading

term2

# php worker.php

I'm registerd as w-0000000045

I'm watching w-0000000044

Working

term3

# php worker.php

I'm registerd as w-0000000046

I'm watching w-0000000045

Working

现在模拟Leader崩溃的情形。使用Ctrl+c或其他方法退出第一个脚本。刚开始不会有任何变化，worker可以继续工作。后来，ZooKeeper会发现超时，并选举出新的leader。

例二是一个很经典的案例。但同时里面也有一些细节的问题需要注意。下面继续听我细细道来。

**php zookeeper运行中的问题以及代码分析**

在上面例二中，说使用ctrl+c退出第一个脚本来模拟Leader崩溃的情形。当Leader崩溃以后worker工作一段时间以后会选出新的leader。然则事实却并非如此，刚开始我曾花了1个小时的时间来等着重新选出leader。但是1个小时过后依然在做着worker的工作……

网上有人说这是php的问题，要把php换成非线程安全的就可以了。否则，监听器是没法工作的。但是上面的例一确确实实监听成功啊，而且我也检查了我的php发现就是非线程安全的。

# php -i | grep Thread

Thread Safety => disabled

说明php是没有问题的。那各位看官就要问了，究竟问题出在什么地方呢？其实各位看官只要再回去仔细看一下例一和例二就能发现其实这两个例子还是有一些不同的。不同之处就在于在例一中我们是先用zookeeper客户端创建了znode ‘/phpDemo’，然后运行了脚本程序，使用zookeeper:get()去监听znode，当改变/phpDemo值的时候调用回调函数。

而例二呢，

$this->znode = $this->create(self::CONTAINER . '/w-', null, $this->acl, Zookeeper::EPHEMERAL | Zookeeper::SEQUENCE);

是使用上面代码来创建的znode，也就是说该/cluster/w-num的节点没有值。所以当下面代码执行的时候在前一个znode中取出的值为null。

$res = $this->get(

self::CONTAINER . '/' . $workers[$i - 1],

array(

$this,

'watchNode'

)

);

猜想那问题可能就是出在这个地方了。下面改变了一下脚本运行的策略。

term1

//先运行终端1

# php worker.php

I'm registerd as w-0000000044

Nobody here , I'm the leader

Leading

//然后给w-0000000044设置值为1

[zk: 127.0.0.1:2181(CONNECTED) 5] set /cluster/w-0000000044 1

//接着再运行第二个终端

# php worker.php

I'm registerd as w-0000000045

I'm watching w-0000000044

Working

//同样给w-0000000045 设置值为2

[zk: 127.0.0.1:2181(CONNECTED) 6] set /cluster/w-0000000045 2

//最后运行第三个终端，值可以不设置

# php worker.php

I'm registerd as w-0000000046

I'm watching w-0000000045

Working

然后再使用ctrl+c模拟leader崩溃，发现很快就选出新的leader了。

还真是zookeeper:get()方法的问题。要找出问题原因还需要继续追踪，有兴趣的看官可以随我一起追踪。

找到PHP-ZooKeeper源码，里面有一个php\_zookeeper.c文件。该文件中定义了get()方法。

static PHP\_METHOD(Zookeeper, get)

{

char \*path;

int path\_len;

zend\_fcall\_info fci = empty\_fcall\_info;

zend\_fcall\_info\_cache fcc = empty\_fcall\_info\_cache;

zval \*stat\_info = NULL;

php\_cb\_data\_t \*cb\_data = NULL;

char \*buffer;

long max\_size = 0;

struct Stat stat;

int status = ZOK;

int length;

ZK\_METHOD\_INIT\_VARS;

if (zend\_parse\_parameters(ZEND\_NUM\_ARGS() TSRMLS\_CC, "s|f!zl", &path, &path\_len, &fci,&fcc, &stat\_info, &max\_size) == FAILURE) {

return;

}

ZK\_METHOD\_FETCH\_OBJECT;

if (fci.size != 0) {

cb\_data = php\_cb\_data\_new(&fci, &fcc, 1 TSRMLS\_CC);

}

if (max\_size <= 0) {

status = zoo\_exists(i\_obj->zk, path, 1, &stat);

if (status != ZOK) {

php\_cb\_data\_destroy(&cb\_data);

php\_error\_docref(NULL TSRMLS\_CC, E\_WARNING, "error: %s", zerror(status));

return;

}

length = stat.dataLength;

} else {

length = max\_size;

}

if (length <= 0) /\* znode carries a NULL \*/

RETURN\_NULL();

buffer = emalloc (length+1);

status = zoo\_wget(i\_obj->zk, path, (fci.size != 0) ? php\_zk\_watcher\_marshal : NULL,

cb\_data, buffer, &length, &stat);

buffer[length] = 0;

if (status != ZOK) {

efree (buffer);

php\_cb\_data\_destroy(&cb\_data);

php\_error\_docref(NULL TSRMLS\_CC, E\_WARNING, "error: %s", zerror(status));

/\* Indicate data marshalling failure with boolean false so that user can retry \*/

if (status == ZMARSHALLINGERROR) {

RETURN\_FALSE;

}

return;

}

if (stat\_info) {

zval\_dtor(stat\_info);

php\_stat\_to\_array(&stat, stat\_info);

}

/\* Length will be returned as -1 if the znode carries a NULL \*/

if (length == -1) {

RETURN\_NULL();

}

RETURN\_STRINGL(buffer, length, 0);

}

代码比较多，但是没关系，我们只看主要的部分。zend\_parse\_parameters这是php扩展所用到的解析参数的函数。有兴趣的可以去研究一下关于php的扩展的知识。这里我们不去深究，我们只需要知道其最后一个参数&max\_size的值为0。因此会走下面的代码

if (max\_size <= 0) {

status = zoo\_exists(i\_obj->zk, path, 1, &stat);

if (status != ZOK) {

php\_cb\_data\_destroy(&cb\_data);

php\_error\_docref(NULL TSRMLS\_CC, E\_WARNING, "error: %s", zerror(status));

return;

}

length = stat.dataLength;

}

接着会执行zoo\_exists()函数。因为该znode是存在的，所有status!=ZOK。但是因为该znode没有值，所以stat.dataLength等于0。因此下面的代码就被执行了

if (length <= 0) /\* znode carries a NULL \*/

RETURN\_NULL();

返回了null，程序结束。我们继续往下看，假如length>0 那么程序会继续向下执行的。也就是说下面代码肯定会执行

status = zoo\_wget(i\_obj->zk, path, (fci.size != 0) ? php\_zk\_watcher\_marshal : NULL,

cb\_data, buffer, &length, &stat);

zoo\_wget()方法其实就是zookeeper的get。而zoo\_exists()就是zookeeper的exists。get和exists都是会监听znode的。所以说，即使zoo\_wget()没有执行，那zoo\_exists()是执行了的，为什么没有监听成功呢。但是在我们第二种运行脚本的方式来说，zoo\_wget()是执行了的，所以zoo\_wget()方法肯定能监听成功。那就是zoo\_exists()的问题了。

我们仔细看一下代码中zoo\_get()和zoo\_exists()的区别

status = zoo\_exists(i\_obj->zk, path, 1, &stat);

status = zoo\_wget(i\_obj->zk, path, (fci.size != 0) ? php\_zk\_watcher\_marshal : NULL,

cb\_data, buffer, &length, &stat);

这两个函数的第三个参数都是监听znode的回调方法，在zoo\_wget()中第三个参数其实是传递了具体的回调方法的，因为fci.size不等于0，所以传参伟php\_zk\_watcher\_marshal（这是一个回调函数解析方法）。而在zoo\_exists()中第三个参数只是传了一个1。所以说不是zoo\_exists()没有监听成功，而是监听以后没有回调函数可以执行，暂时可以这么认为。

下面继续追踪zoo\_exists()方法的实现，在zookeeper源文件中有src/c/src/zookeeper.c

int zoo\_exists(zhandle\_t \*zh, const char \*path, int watch, struct Stat \*stat)

{

return zoo\_wexists(zh,path,watch?zh->watcher:0,zh->context,stat);

}

追到这我们应该大概清楚了，在PHP-Zookeeper中的get方法定义中zoo\_exists()并没有指定监视器的回调函数。有兴趣的可以继续追踪zh->watcher的值。因为我们在传参的时候传了一个1，所以zoo\_wexists()中第三个参数就是zh->watcher。但是这个zh->watcher并不是我们在php代码中的方法

array(

$this,

'watchNode'

)

到这就清楚了。不知道这是PHP-ZooKeeper遗留的问题，还是这样自有其它的作用。感兴趣的可以继续关注，总之我们清楚了为什么出现watcher不能生效的原因，使用的时候尽量避免就行了。