PHP高级编程之守护进程

http://netkiller.github.io/journal/php.daemon.html

Mr. Neo Chen (陈景峯), netkiller, BG7NYT

中国广东省深圳市龙华新区民治街道溪山美地
518131
+86 13113668890

<netkiller@msn.com>

2014-09-01

摘要

2014-09-01 发表

2015-08-31 更新

2015-10-20 更新，增加优雅重启

目录

1. 什么是守护进程

守护进程是脱离于终端并且在后台运行的进程。守护进程脱离于终端是为了避免进程在执行过程中的信息在任何终端上显示并且进程也不会被任何终端所产生的终端信息所打断。

例如 apache, nginx, mysql 都是守护进程

2. 为什么开发守护进程

很多程序以服务形式存在，他没有终端或UI交互，它可能采用其他方式与其他程序交互，如TCP/UDP Socket, UNIX Socket, fifo。程序一旦启动便进入后台，直到满足条件他便开始处理任务。

3. 何时采用守护进程开发应用程序

以我当前的需求为例，我需要运行一个程序，然后监听某端口，持续接受服务端发起的数据，然后对数据分析处理，再将结果写入到数据库中; 我采用ZeroMQ实现数据收发。

如果我不采用守护进程方式开发该程序，程序一旦运行就会占用当前终端窗框，还有受到当前终端键盘输入影响，有可能程序误退出。

4. 守护进程的安全问题

我们希望程序在非超级用户运行，这样一旦由于程序出现漏洞被骇客控制，攻击者只能继承运行权限，而无法获得超级用户权限。

我们希望程序只能运行一个实例，不运行同事开启两个以上的程序，因为会出现端口冲突等等问题。

5. 怎样开发守护进程

例 1. 多线程守护进程例示

```
<?php
class ExampleWorker extends Worker {

        #public function __construct(Logging $logger) {
        #       $this->logger = $logger;
        #}

        #protected $logger;
        protected  static $dbh;
        public function __construct() {

        }
        public function run(){
                $dbhost = '192.168.2.1';                        // 数据库服务器
                $dbport = 3306;
            $dbuser = 'www';                            // 数据库用户名
        $dbpass = 'qwer123';                    // 数据库密码
                $dbname = 'example';                            // 数据库名

                self::$dbh  = new PDO("mysql:host=$dbhost;port=$dbport;dbname=$dbname", $dbuser, $dbpass, array(
                        /* PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES \'UTF8\'', */
                        PDO::MYSQL_ATTR_COMPRESS => true,
                        PDO::ATTR_PERSISTENT => true
                        )
                );

        }
        protected function getInstance(){
        return self::$dbh;
    }

}

/* the collectable class implements machinery for Pool::collect */
class Fee extends Stackable {
        public function __construct($msg) {
                $trades = explode(",", $msg);
                $this->data = $trades;
                print_r($trades);
        }

        public function run() {
                #$this->worker->logger->log("%s executing in Thread #%lu", __CLASS__, $this->worker->getThreadId() );

                try {
                        $dbh  = $this->worker->getInstance();

                        $insert = "INSERT INTO fee(ticket, login, volume, `status`) VALUES(:ticket, :login, :volume,'N')";
                        $sth = $dbh->prepare($insert);
                        $sth->bindValue(':ticket', $this->data[0]);
                        $sth->bindValue(':login', $this->data[1]);
                        $sth->bindValue(':volume', $this->data[2]);
                        $sth->execute();
                        $sth = null;

                        /* ...... */

                        $update = "UPDATE fee SET `status` = 'Y' WHERE ticket = :ticket and `status` = 'N'";
                        $sth = $dbh->prepare($update);
                        $sth->bindValue(':ticket', $this->data[0]);
                        $sth->execute();
                        //echo $sth->queryString;
                        //$dbh = null;
                }
                catch(PDOException $e) {
                        $error = sprintf("%s,%s\n", $mobile, $id );
                        file_put_contents("mobile_error.log", $error, FILE_APPEND);
                }
        }
}

class Example {
        /* config */
        const LISTEN = "tcp://192.168.2.15:5555";
        const MAXCONN = 100;
        const pidfile = __CLASS__;
        const uid       = 80;
        const gid       = 80;

        protected $pool = NULL;
        protected $zmq = NULL;
        public function __construct() {
                $this->pidfile = '/var/run/'.self::pidfile.'.pid';
        }
        private function daemon(){
                if (file_exists($this->pidfile)) {
                        echo "The file $this->pidfile exists.\n";
                        exit();
                }

                $pid = pcntl_fork();
                if ($pid == -1) {
                         die('could not fork');
                } else if ($pid) {
                         // we are the parent
                         //pcntl_wait($status); //Protect against Zombie children
                        exit($pid);
                } else {
                        // we are the child
                        file_put_contents($this->pidfile, getmypid());
                        posix_setuid(self::uid);
                        posix_setgid(self::gid);
                        return(getmypid());
                }
        }
        private function start(){
                $pid = $this->daemon();
                $this->pool = new Pool(self::MAXCONN, \ExampleWorker::class, []);
                $this->zmq = new ZMQSocket(new ZMQContext(), ZMQ::SOCKET_REP);
                $this->zmq->bind(self::LISTEN);

                /* Loop receiving and echoing back */
                while ($message = $this->zmq->recv()) {
                        //print_r($message);
                        //if($trades){
                                        $this->pool->submit(new Fee($message));
                                        $this->zmq->send('TRUE');
                        //}else{
                        //              $this->zmq->send('FALSE');
                        //}
                }
                $pool->shutdown();
        }
        private function stop(){

                if (file_exists($this->pidfile)) {
                        $pid = file_get_contents($this->pidfile);
                        posix_kill($pid, 9);
                        unlink($this->pidfile);
                }
        }
        private function help($proc){
                printf("%s start | stop | help \n", $proc);
        }
        public function main($argv){
                if(count($argv) < 2){
                        printf("please input help parameter\n");
                        exit();
                }
                if($argv[1] === 'stop'){
                        $this->stop();
                }else if($argv[1] === 'start'){
                        $this->start();
                }else{
                        $this->help($argv[0]);
                }
        }
}

$cgse = new Example();
$cgse->main($argv);
```

例 2. 消息队列与守护进程

```
<?php
declare(ticks = 1);
require_once( __DIR__.'/autoload.class.php' );
umask(077);
class EDM {
        protected $queue;
        public function __construct() {
                global $argc, $argv;
                $this->argc = $argc;
                $this->argv = $argv;
                $this->pidfile = $this->argv[0].".pid";
                $this->config = new Config('mq');
                $this->logging = new Logging(__DIR__.'/log/'.$this->argv[0].'.'.date('Y-m-d').'.log'); //.H:i:s
                //print_r( $this->config->getArray('mq') );
                //pcntl_signal(SIGHUP, array(&$this,"restart"));
        }
        protected function msgqueue(){
                $exchangeName = 'email'; //交换机名
                $queueName = 'email'; //队列名
                $routeKey = 'email'; //路由key
                //创建连接和channel
                $connection = new AMQPConnection($this->config->getArray('mq'));
                if (!$connection->connect()) {
                        die("Cannot connect to the broker!\n");
                }
                $this->channel = new AMQPChannel($connection);
                $this->exchange = new AMQPExchange($this->channel);
                $this->exchange->setName($exchangeName);
                $this->exchange->setType(AMQP_EX_TYPE_DIRECT); //direct类型
                $this->exchange->setFlags(AMQP_DURABLE); //持久化
                $this->exchange->declare();
                //echo "Exchange Status:".$this->exchange->declare()."\n";
                //创建队列
                $this->queue = new AMQPQueue($this->channel);
                $this->queue->setName($queueName);
                $this->queue->setFlags(AMQP_DURABLE); //持久化
                $this->queue->declare();
                //echo "Message Total:".$this->queue->declare()."\n";
                //绑定交换机与队列，并指定路由键
                $bind = $this->queue->bind($exchangeName, $routeKey);
                //echo 'Queue Bind: '.$bind."\n";
                //阻塞模式接收消息
                while(true){
                        //$this->queue->consume('processMessage', AMQP_AUTOACK); //自动ACK应答
                        $this->queue->consume(function($envelope, $queue) {
                                $msg = $envelope->getBody();
                                $queue->ack($envelope->getDeliveryTag()); //手动发送ACK应答
                                $this->logging->info('('.'+'.')'.$msg);
                                //$this->logging->debug("Message Total:".$this->queue->declare());
                        });
                        $this->channel->qos(0,1);
                        //echo "Message Total:".$this->queue->declare()."\n";
                }
                $conn->disconnect();
        }
        protected function start(){
                if (file_exists($this->pidfile)) {
                        printf("%s already running\n", $this->argv[0]);
                        exit(0);
                }
                $this->logging->warning("start");
                $pid = pcntl_fork();
                if ($pid == -1) {
                        die('could not fork');
                } else if ($pid) {
```

```
				//pcntl_wait($status); //等待子进程中断，防止子进程成为僵尸进程。
				exit(0);
			} else {
				posix_setsid();
				//printf("pid: %s\n", posix_getpid());
				file_put_contents($this->pidfile, posix_getpid());

				//posix_kill(posix_getpid(), SIGHUP);

				$this->msgqueue();
			}
	}
	protected function stop(){
			if (file_exists($this->pidfile)) {
				$pid = file_get_contents($this->pidfile);
				posix_kill($pid, SIGTERM);
				//posix_kill($pid, SIGKILL);
				unlink($this->pidfile);
				$this->logging->warning("stop");
			}else{
				printf("%s haven't running\n", $this->argv[0]);
			}
	}
	protected function restart(){
			$this->stop();
			$this->start();
	}
	protected function status(){
			if (file_exists($this->pidfile)) {
				$pid = file_get_contents($this->pidfile);
				printf("%s already running, pid = %s\n", $this->argv[0], $pid);
			}else{
				printf("%s haven't running\n", $this->argv[0]);
			}
	}
	protected function usage(){
			printf("Usage: %s {start | stop | restart | status}\n", $this->argv[0]);
	}
	public function main(){
			//print_r($this->argv);
			if($this->argc != 2){
				$this->usage();
			}else{
				if($this->argv[1] == 'start'){
					$this->start();
				}else if($this->argv[1] == 'stop'){
					$this->stop();
				}else if($this->argv[1] == 'restart'){
					$this->restart();
				}else if($this->argv[1] == 'status'){
					$this->status();
				}else{
					$this->usage();
				}
			}
	}
}
$edm = New EDM();
$edm->main();
```

5.1. 程序启动

下面是程序启动后进入后台的代码

通过进程ID文件来判断，当前进程状态，如果进程ID文件存在表示程序在运行中，通过代码file_exists($this->pidfile)实现，但而后进程被kill需要手工删除该文件才能运行

```
	private function daemon(){
			if (file_exists($this->pidfile)) {
				echo "The file $this->pidfile exists.\n";
				exit();
			}

			$pid = pcntl_fork();
			if ($pid == -1) {
```

```
                          die('could not fork');
                } else if ($pid) {
                        // we are the parent
                        //pcntl_wait($status); //Protect against Zombie children
                        exit($pid);
                } else {
                        // we are the child
                        file_put_contents($this->pidfile, getmypid());
                        posix_setuid(self::uid);
                        posix_setgid(self::gid);
                        return(getmypid());
                }
        }
```

程序启动后，父进程会推出，子进程会在后台运行，子进程权限从root切换到指定用户，同时将pid写入进程ID文件。

5.2. 程序停止

程序停止，只需读取pid文件，然后调用posix_kill($pid, 9); 最后将该文件删除。

```
        private function stop(){

                if (file_exists($this->pidfile)) {
                        $pid = file_get_contents($this->pidfile);
                        posix_kill($pid, 9);
                        unlink($this->pidfile);
                }
        }
```

5.3. 单例模式

所有线程共用数据库连接，在多线程中这个非常重要，如果每个线程建立以此数据库连接在关闭，这对数据库的开销是巨大的。

```
protected function getInstance(){
        return self::$dbh;
}
```

5.4. 实现优雅重启

所谓优雅重启是指进程不退出的情况加实现重新载入包含重置变量，刷新配置文件，重置日志等等

stop/start 或者 restart都会退出进程，重新启动，导致进程ID改变，同时瞬间退出导致业务闪断。所以很多守护进程都会提供一个reload功能，者就是所谓的优雅重启。

reload 实现原理是给进程发送SIGHUP信号，可以通过kill命令发送 kill -s SIGHUP 64881，也可以通过库函数实现 posix_kill(posix_getpid(), SIGUSR1);

```

<?php
pcntl_signal(SIGTERM,  function($signo) {
    echo "\n This signal is called. [$signo] \n";
    Status::$state = -1;
});

pcntl_signal(SIGHUP,  function($signo) {
    echo "\n This signal is called. [$signo] \n";
    Status::$state = 1;
        Status::$ini = parse_ini_file('test.ini');
});

class Status{
    public static $state = 0;
        public static $ini = null;
}

$pid = pcntl_fork();
```

```
if ($pid == -1) {
    die('could not fork');
}

if($pid) {
    // parent
} else {
        $loop = true;
        Status::$ini = parse_ini_file('test.ini');
    while($loop) {
                print_r(Status::$ini);
        while(true) {
                        // Dispatching...
                        pcntl_signal_dispatch();
                        if(Status::$state == -1) {
                                // Do something and end loop.
                                $loop = false;
                                break;
                        }

                        if(Status::$state == 1) {
                                printf("This program is reload.\r\n");
                                Status::$state = 0;
                                break;
                        }
            echo '.';
            sleep(1);
        }
        echo "\n";
    }

    echo "Finish \n";
    exit();
}
```

创建配置文件

```
[root@netkiller pcntl]# cat test.ini
[db]
host=192.168.0.1
port=3306
```

测试方法，首先运行该守护进程

```
# php signal.reload.php
Array
(
    [host] => 192.168.0.1
    [port] => 3306
)
```

现在修改配置文件，增加user=test配置项

```
[root@netkiller pcntl]# cat test.ini
[db]
host=192.168.0.1
port=3306
user=test
```

发送信号，在另一个终端窗口，通过ps命令找到该进程的PID，然后使用kill命令发送SIGHUP信号，然后再通过ps查看进程，你会发现进程PID没有改变

```
[root@netkiller pcntl]# ps ax | grep reload
64881 pts/0    S      0:00 php -c /srv/php/etc/php-cli.ini signal.reload.php
65073 pts/1    S+     0:00 grep --color=auto reload

[root@netkiller pcntl]# kill -s SIGHUP 64881

[root@netkiller pcntl]# ps ax | grep reload
64881 pts/0    S      0:00 php -c /srv/php/etc/php-cli.ini signal.reload.php
65093 pts/1    S+     0:00 grep --color=auto reload
```

配置文件被重新载入

```
This signal is called. [1]
This program is reload.

Array
(
    [host] => 192.168.0.1
    [port] => 3306
    [user] => test
)
```

优雅重启完成。

6. Example

```

<?php
/*
 * PHP Daemon sample.
 * Home: http://netkiller.github.io
 * Author: netkiller<netkiller@msn.com>
 *
*/
class Logger {

        public function __construct(/*Logging $logger*/) {
        }

        public function logger($type, $message) {
                $log = sprintf ( "%s\t%s\t%s\n", date ( 'Y-m-d H:i:s' ), $type, $message );
                file_put_contents ( sprintf(__DIR__."/../log/sender.%s.log", date ( 'Y-m-d' )), $log, FILE_APPEND );
        }

}

final class Signal{
    public static $signo = 0;
        protected static $ini = null;
        public static function set($signo){
                self::$signo = $signo;
        }
        public static function get(){
                return(self::$signo);
        }
        public static function reset(){
                self::$signo = 0;
        }
}

class Test extends Logger {
        //public static $signal = null;

        public function __construct() {
                //self::$signal == null;
        }
        public function run(){
                while(true){
                        pcntl_signal_dispatch();
                        printf(".");
                        sleep(1);
                        if(Signal::get() == SIGHUP){
                                Signal::reset();
                                break;
                        }
                }
                printf("\n");
        }
}

class Daemon extends Logger {
        /* config */
        const LISTEN = "tcp://192.168.2.15:5555";
        const pidfile   = __CLASS__;
	const uid		= 80;
	const gid		= 80;
	const sleep	= 5;

	protected $pool		= NULL;
	protected $config	= array();

	public function __construct($uid, $gid, $class) {
		$this->pidfile = '/var/run/'.basename(get_class($class), '.php').'.pid';
		//$this->config = parse_ini_file('sender.ini', true); //include_once(__DIR__."/config.php");
		$this->uid = $uid;
		$this->gid = $gid;
		$this->class = $class;
		$this->classname = get_class($class);

		$this->signal();
	}
	public function signal(){

		pcntl_signal(SIGHUP,  function($signo) /*use ()*/{
			//echo "\n This signal is called. [$signo] \n";
			printf("The process has been reload.\n");
			Signal::set($signo);
		});

	}
	private function daemon(){
		if (file_exists($this->pidfile)) {
			echo "The file $this->pidfile exists.\n";
			exit();
		}

		$pid = pcntl_fork();
		if ($pid == -1) {
			 die('could not fork');
		} else if ($pid) {
			 // we are the parent
			 //pcntl_wait($status); //Protect against Zombie children
			exit($pid);
		} else {
			file_put_contents($this->pidfile, getmypid());
			posix_setuid(self::uid);
			posix_setgid(self::gid);
			return(getmypid());
		}
	}
	private function run(){

		while(true){

			printf("The process begin.\n");
			$this->class->run();
			printf("The process end.\n");

		}
	}
	private function foreground(){
		$this->run();
	}
	private function start(){
		$pid = $this->daemon();
		for(;;){
			$this->run();
			sleep(self::sleep);
		}
	}
	private function stop(){

		if (file_exists($this->pidfile)) {
			$pid = file_get_contents($this->pidfile);
			posix_kill($pid, 9);
			unlink($this->pidfile);
		}
	}
	private function reload(){
		if (file_exists($this->pidfile)) {
			$pid = file_get_contents($this->pidfile);
			//posix_kill(posix_getpid(), SIGHUP);
			posix_kill($pid, SIGHUP);
		}
	}
	private function status(){
		if (file_exists($this->pidfile)) {
				$pid = file_get_contents($this->pidfile);
				system(sprintf("ps ax | grep %s | grep -v grep", $pid));
			}
	}
	private function help($proc){
			printf("%s start | stop | restart | status | foreground | help \n", $proc);
	}
	public function main($argv){

			if(count($argv) < 2){
				$this->help($argv[0]);
				printf("please input help parameter\n");
				exit();
			}
			if($argv[1] === 'stop'){
				$this->stop();
			}else if($argv[1] === 'start'){
				$this->start();
	}else if($argv[1] === 'restart'){
				$this->stop();
	    $this->start();
			}else if($argv[1] === 'status'){
				$this->status();
			}else if($argv[1] === 'foreground'){
				$this->foreground();
			}else if($argv[1] === 'reload'){
				$this->reload();
			}else{
				$this->help($argv[0]);
			}
	}
}

$daemon = new Daemon(80,80, new Test());
$daemon->main($argv);
?>
```

7. 进程意外退出解决方案

如果是非常重要的进程，必须要保证程序正常运行，一旦出现任何异常退出，都需要做即时做处理。下面的程序可能检查进程是否异常退出，如果退出便立即启动。

```
#!/bin/sh

LOGFILE=/var/log/$(basename $0 .sh).log
PATTERN="my.php"
RECOVERY="/path/to/my.php start"

while true
do
	TIMEPOINT=$(date -d "today" +"%Y-%m-%d_%H:%M:%S")
	PROC=$(pgrep -o -f ${PATTERN})
	#echo ${PROC}
	if [ -z "${PROC}" ]; then
		${RECOVERY} >> $LOGFILE
		echo "[${TIMEPOINT}] ${PATTERN} ${RECOVERY}" >> $LOGFILE

	#else
		#echo "[${TIMEPOINT}] ${PATTERN} ${PROC}" >> $LOGFILE
	fi
sleep 5
done &
```

8. 延伸阅读

PHP高级编程之消息队列

PHP高级编程之多线程

1条评论 **Netkiller Technology Document** Neo Chan

推荐 分享 按评分高低排序

加入讨论……

kylesean • 1个月前
感谢博主，感谢大师。谢谢你的无私奉献，让我这个菜鸟在学习的道路上走得更快。
回复 • 分享 ›

在 NETKILLER TECHNOLOGY DOCUMENT 上还有

第 3 章 Systems architecture(系统架构)
2条评论 • 4年前•
Neo Chan — 呵呵， 我手工画的。没有任何工具。

第 3 章 Nginx
3条评论 • 4年前•
Rambone — 对 这是正确的做法～～

2. 開發語言及平台
1条评论 • 5年前•
无忌 — 像php/perl/python这种动态语言，开发速度快，周期端，对服务器性能要求低，出错率低周期短

PHP高级编程之守护进程
2条评论 • 3年前•
何勇 — 太帮了，先收藏！

订阅 在您的网站上使用 Disqus添加 Disqus添加 隐私