**XcmqClient 使用文档**

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 作者 | 时间 | 备注 |
| V1.0 | 谢佐福 | 2016-5-5 | 初稿 |
| V1.1 | 谢佐福 | 2016-5-6 | 重连write部分注释修改 |
| V1.2 | 谢佐福 | 2016-7-9 | 对lib包v1.0版修改 |
| V1.3 | 谢佐福 | 2016-8-28 | 对lib包v1.2版新加close方法或者unset操作可以关闭连接 |
| v1.4 | 谢佐福 | 2016-9-29 | lib包v1.3-beta  新增单例方法  修复每次连接都打日志的bug |
| V1.5 | 谢佐福 | 2016-11-11 | lib包v1.4-beta  新增超时配置  新增exchange\_type配置  连接及消费数据的对错误进行抛出异常，更好的查错  相对上一版本文档，**红色为修改内容** |

[1.目录结构 3](#_Toc5546)

[2.使用案例 3](#_Toc23782)

[2.1 写操作案例 3](#_Toc13272)

[2.2 读操作案例 4](#_Toc16000)

[3.配置文件讲解 4](#_Toc28612)

[3.1 配置文件目录 4](#_Toc10549)

[3.2 共用配置 5](#_Toc4697)

[3.3 activemq 配置 6](#_Toc24309)

[3.4 rabbitmq和rabbitmqorg 配置 7](#_Toc18356)

[4.各框架中加载使用 8](#_Toc11696)

[4.1 zend 1.0 8](#_Toc14145)

[5. xcmq 对外接口方法定义 9](#_Toc21360)

[5.1 初始化 9](#_Toc20969)

[5.1 写入数据 9](#_Toc9311)

[5.2 事务操作 10](#_Toc11914)

[5.3 消费数据 11](#_Toc2514)

[5.4 连接关闭 11](#_Toc20302)

# 1.目录结构

xcmq xcmq客户端对外api目录

Base.inc.php 模块引用文件

Xcmq.class.php 对外接口文件

PhpAmqpLib rabbitmq官方提供的php写的客户端

module 对mq操作的实际类文件夹

Xcmq.ActiveMQ.class.php ActiveMQ操作类

Xcmq.RabbitMQ.class.php RabbitMQ php官方提供的amqp操作类

Xcmq.RabbitMQORG.class.php RabbitMQ 官方提供的php操作类

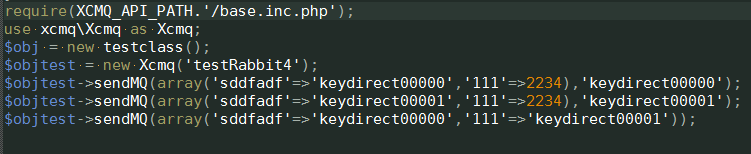
config xcmq配置文件存放文件夹

xcore 一些核心资源管理目录，如文件加，连接管理

xlogger 日志模块

# 2.使用案例

## 2.1 写操作案例



require(XCMQ\_API\_PATH.'/base.inc.php');

use xcmq\Xcmq as Xcmq;

$obj = new testclass();

$objtest = new Xcmq('testRabbit4'); // 或者 Xcmq::singleton('testRabbit4');

$objtest->sendMQ(array('sddfadf'=>'keydirect00000','111'=>2234),'keydirect00000');

$objtest->sendMQ(array('sddfadf'=>'keydirect00001','111'=>2234),'keydirect00001');

$objtest->sendMQ(array('sddfadf'=>'keydirect00000','111'=>'keydirect00001'));

$objtest->close();

unset($objtest-);

## 2.2 读操作案例

class testclass{

public function testmq($data,$obj){

$obj->ack();

print\_r($data);

}

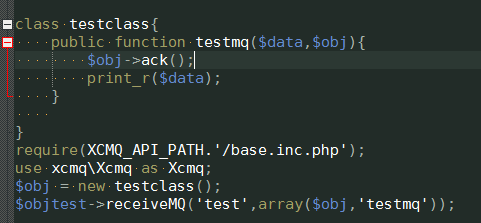
}

require(XCMQ\_API\_PATH.'/base.inc.php');

use xcmq\Xcmq as Xcmq;

$obj = new testclass();

$objtest->receiveMQ('test',array($obj,'testmq'));



# 3.配置文件讲解

## 3.1 配置文件目录

xcmq/config/mq\_config.php



## 3.2 共用配置

不管什么驱动，通用配置

#用什么存储，有rabbitmq,rabbitmqorg,activemq 选择

'mq'=>'rabbitmq',

#连接参数，不同的驱动，参数不一样，查详查查看各自驱动配置，不管什么驱动，连接处会进行一次单例操作，只要参数一致，第二次连接，则会直接返回上一次连接的实例对象

'base'=>array( 'host'=>'192.168.6.231' , 'port'=> '5672', 'login'=>'admin' , 'password'=> 'admin','vhost' =>'/'),

#这个queue这个配置rabbitmq和rabbitmqorg是一致的，activemq 'durable'=>true,'auto\_delete'=>false, 这两个无效

#routingkey 是和queue\_name 进行绑定的，一个queue\_name 只能一个routingkey，但是一个routingkey可以绑定多个queue\_name, 当写入的时候 ，sendMQ($data,routingkey) 的时候，没有指定 routingkey 将会写入到当前配置中所有queue中。如果指定了routingkey 则只写入到 和当前routingkey绑定的queue中，并且这个操作并不保重是在一个事务中，如果一定要求要么全部成功要么全部失败，建议开启事务

'queue'=>array(

'test'=>array('queue\_name'=>'direct0000','durable'=>true,'auto\_delete'=>false,'routingkey'=>'keydirect00000'),

'test2'=>array('queue\_name'=>'direct0001','durable'=>true,'auto\_delete'=>true,'routingkey'=>'keydirect00001'),

),

#serialize(数据所有数据源有效),json(对数据进行json转换，数据源是数组的情况，否则失败) , string(不进行任何处理)

'format'=>'json',

#重连配置

'reconnect'=>array(

'write'=>array('count'=>0,'sleep'=>0.1), //如果当前写入提交重连次数在 count次数内写入成功后，后面再次进行写入，失败又重新计算，默认为 0，sleep 默认为0.1

// 消费线程重连次数，与sleep时间,默认0次，sleep 1秒

'read' =>array('count'=>0,'sleep'=>1),

),

## 3.3 activemq 配置

'test'=>array(

'mq'=>'activemq',//用什么存储

'base'=>'tcp://192.168.7.177:61613', // 连接activemq 信息

'queue'=>array(

'test'=>array('queue\_name'=>'/queue/test','routingkey'=>'keydirect00000'),

'mytest2'=>array('queue\_name'=>'/queque/mytest2','routingkey'=>'keydirect00001'),

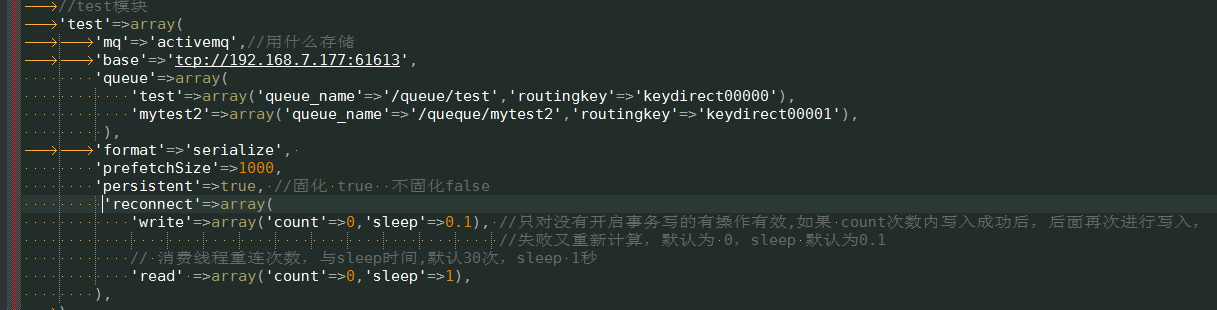
),

'format'=>'serialize',

'prefetchSize'=>1000,

'persistent'=>true, //固化 true 不固化false

),



## 3.4 rabbitmq和rabbitmqorg 配置

'testRabbit4'=>array(

'mq'=>'rabbitmq',//用什么存储

'base'=>array( 'host'=>'10.40.6.46' , 'port'=> '5672', 'login'=>'admin' , 'password'=> 'admin','vhost' =>'/'),

'read\_timeout'=>3, // rabbitmqorg 默认为3 rabbitmq，默认为90

'write\_timeout'=>3, // 默认为3

'heartbeat' =>1, // rabbitmqorg 默认为90 rabbitmq 无效

//在这去除了connect\_timeout 参数配置

'queue'=>array(

'test'=>array('queue\_name'=>'direct0000','durable'=>true,'auto\_delete'=>false,'routingkey'=>'keydirect00000'),

'test2'=>array('queue\_name'=>'direct0001','durable'=>true,'auto\_delete'=>true,'routingkey'=>'keydirect00001'),

),

'delivery\_mode'=>2, //2 消息持久化, 1 非持化，默认为 2

'format'=>'json', // serialize(数据所有数据源有效),json(对数据进行json转换，数据源是数组的情况，否则失败) , string(不进行任何处理)

'exchange'=>array('exchange\_name'=>'ExChangedirect0000','durable'=>true,'auto\_delete'=>false,’exchange\_type’=>’direct’),

//'amqp\_ex\_type'=>'direct', //topic,headers(暂未知) fanout(最快,和routingkey没有任何关系) direct(按routingkey接收和发送) 1.4-beta还支持这个，新版本也兼容这个

//'prefetchSize'=>1000, //客户端将预取数据到大小

//'prefetchCount'=>1, //可以同时接收多少 条未ack的数据，默认为1，预分配数量

'reconnect'=>array(

'write'=>array('count'=>0,'sleep'=>0.1), //只对没有开启事务写的有操作有效,如果 count次数内写入成功后，后面再次进行写入，失败又重新计算，默认为 0，sleep 默认为0.1

// 消费线程重连次数，与sleep时间,默认30次，sleep 1秒

'read' =>array('count'=>0,'sleep'=>1),

),

),



# 4.各框架中加载使用

## 4.1 zend 1.0

Download :

svn up [http://erpsvn.hqygou.com/svn/infrastructure/rabbitmq/tag/XcmqClient/v1.4-beta](http://erpsvn.hqygou.com/svn/infrastructure/rabbitmq/tag/XcmqClient/v1.0)

guest:guest

把v1.3-beta拷贝到 zend 1.0/Extend 文件夹下，并重命名为 XcmqClient

把XcmqClient/example/config 拷贝到 Apps\Global\config

程序中配置配置文件 路径:

public/index.php 及 daemon/\_autoload.php 文件中新增常量:

Xcmq的配置文件

define('XCMQ\_CONF\_FILE', APPLICATION\_PATH . '/Global/config/mq\_config.php');

Xlogger的配置文件目录

define('XLOGGER\_CONF\_PATH', APPLICATION\_PATH . '/Global/config/xlogger/');

然后修改mq\_config.php 相对应的配置文件

修改 config/xlogger/xcmq.php base 修改为 APPLICATION\_PATH.'/../logs/xcmq/'

手工创建APPLICATION\_PATH.'/../logs/xcmq/' 目录（暂时xlogger不支持自动创建目录）

$XcmqClient = new XcmqClient('testRabbit4'); //或者 XcmqClient::singleton('testRabbit4');

其中 参数 testRabbit4 是 mq\_config.php 中第一维数组中的key

$XcmqClient->sendMQ(array('sddfadf'=>'keydirect00000','111'=>2234),'keydirect00000');

第一个参数是要发送的内容，可以是任何类型，不过受mq\_config 配置中的 format 参数影响

第二个参数是指：往哪个队列中写，是queue[key][routingkey] 的值

'queue'=>array(

'test'=>array('queue\_name'=>'queuenametest ','durable'=>true,'auto\_delete'=>false,'routingkey'=>'testrountkey'),

),

假如没有指名routingkey 则把写到当前 配置中queue里的所有队列中，多个队列可以相同routingkey，相同rountingkey 意味着接收一样的数据

使用场景:

假如有多少个网站需求同步产品信息，但不同网站要同步的产品分类是不一样的，但是同步的时候，产品内容是一致的，A网站只要 手机分类的，B网站，只要衣服分类的，这样在写入队列中的时候，就不写2套逻辑，只要在逻辑中判断一下写到哪个routingkey中，然后在配置中routingkey绑定哪些队列就可以了

$XcmqClient->receiveMQ('test',array($this,'ackdata'));

消费数据

第一个参数是指要消费哪一个队列的数据，mq\_config.php 中queue[key] 中的key值

第二个参数是callback类型。可以是直接function 名也可以是object的method名

# 5. xcmq 对外接口方法定义

## 5.1 初始化

\_\_construct($configKey)

$configKey 对应配置文件中第一层数组的key值 ，指定哪一个模块进行操作

注：如果是连接参数有问题，连接不成功会进行抛异常

## 5.1 写入数据

sendMQ($data,$routingKey='')

参数：

$data : 可以是任意数据格式，假如配置文件中配置format=>serialize ，$data 数据将会在写入之前进行序列化操作，反之在消费的时候，会进行反序列化操作，如果配置 format=>json，则要求$data数据为数组格式，否则报错。如果format=>string或者其他则不会对$data进行任何操作，$data必须为string格式，或者报错

$routingKey ： 默认为空，和配置文件中的queue=array(‘routingkey’=>’val’) 进行对应，如果指定了routingKey 进将写入到和当前routingkey绑定的queue\_name队列中，如果为空则写入到当前模块中所有队列中，如果指定的$routingKey 并不在配置中，则返回false

返回值：

true | false

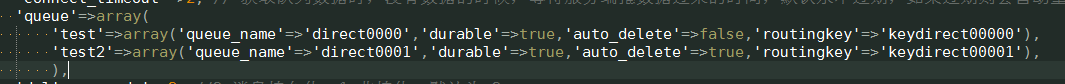
如果在开启了事务的情况下，先把数据写到arraylist的缓存中，待commit的时候再执行写入到mq中

否则将调用mq的接口进行写入，在activemq中，如果写入到多个队列中，在没有开启事务的情况下，会自动开启事务与提交，rabbitmq 不会自动开启事务。并且如果写到入多个routingkey里，不保证所有成功或者失败。

如果写入的过程中遇到报错，则返回false, 并且会进行写入错误日志到sendMQ模块。

eg:

配置文件：



$objtest->sendMQ(array('sddfadf'=>'keydirect00000','111'=>2234),'keydirect00000');

结果是写入到direct0000 队列中， 而 direct0001 中并没有写入数据

$objtest->sendMQ(array('sddfadf'=>'keydirect00000','111'=>'keydirect00001'));

结果是两个队列中都有写数据。

备注：如果写入到多个队列中，并不保证全部成功，所以如果要全部成功或者失败，请开始事务

## 5.2 事务操作

开启事务，如果开启了事务，sendMQ操作的时候，会先写入到php缓存一个arraylist中，待commit操作的时候，一起顺序执行

begin()

返回值 ：true ，没有false

事务回滚

rollback()

返回值 ： false 或者，如果成功则返回 $this

事务提交

commit()

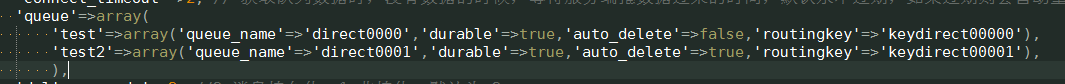
返回值 ： false 可重新提交，每次提交会受配置 reconnect[‘write’] 参数影响 ，默认每次提交会进0次重连操作

true 提交成功，当前事务完成

## 5.3 消费数据

receiveMQ($queuekey,$callback)

$queuekey : 和配置中queue[key] 中的key对应，比如



receiveMQ(‘test’,$callback) 则进行订阅queue\_name = direct0000队列的数据，则和 direct0001 队列没有任何关系

$callback : 回调函数或者对象方法，如果是方法则直接写方法名，如果是对象方法则是数组形式，可以查看Callback 回调类型

<http://php.net/manual/zh/language.types.callable.php>

回调到方法名 function\_name 的函数里

receiveMQ(‘test’,’function\_name’)

回调到当前实例对象中 方法名为 function\_name 的函数里

receiveMQ(‘test’,array($this,’function\_name’))

**回调方法参数要求：**

Callbakcfunction($data,$obj)

$data : 返回写入之前的数据。受format 参数影响

$obj : mq操作对象，可以$obj->ack() 进行ack操作，这个参数不受不同mq影响

eg:

class testclass{

public function testmq($data,$obj){

$obj->ack();

print\_r($data);

}

}

注: 如果是callback 参数有问题，会进行抛异常

## 5.4 连接关闭

close();

$obj->close();方法可以关闭连接，也可以通过unset($obj) 操作，当unset计数器为0的时候，也会进行连接关闭