# RabbitMQ实践场景示例

[mnstory.net](http://mnstory.net)

## 基本配置

RabbitMQ是一个成熟稳定的消息队列，对比过几款主流MessageQueue框架，以此框架最为成熟，OpenStack和VMWare这类的虚拟化和云平台都使用此软件作为消息通信，实际测试一个月，相当稳定。

利用MessageQueue可以实现，不同地方，不同时间，不同语言间的通信。

1. **不同地方**，容易理解，通过网络实现，网络可达的地方都能通信。
2. **不同时间**，是因为MQ本身是有缓存机制，包括内存缓存、文件缓存、数据库缓存等，假设Server端向Client发送消息，但是Client并没有启动，没有关系，等到Client启动的时候，从MQ里面取队列或交换里的消息即可，这种方式能很好地实现异步框架。
3. **不同语言**，所有终端只需要通过网络与MQ交互，换言之，只需要使用本语言的网络API实现Client端封装，即可与MQ交互，所以，使用MessageQueue，可以跨语言。

安装方法不再赘述，网上很多，最新版本下载：

<https://www.rabbitmq.com/download.html>

### 配置用户

记录下一些常用命令。

新增用户

[root sbin]# /opt/rabbitmq/sbin/rabbitmqctl add\_user username password

设置tags（类似用户所属组）

[root sbin]# /opt/rabbitmq/sbin/rabbitmqctl set\_user\_tags username administrator monitoring policymaker management

查看所有用户和其tags信息

[root sbin]# /opt/rabbitmq/sbin/rabbitmqctl list\_users

Listing users ...

username [administrator, monitoring, policymaker, management]

guest [administrator]

设置用户访问权限

[root sbin]# /opt/rabbitmq/sbin/rabbitmqctl set\_permissions -p / username ConfP WriteP ReadP

或

[root sbin]# /opt/rabbitmq/sbin/rabbitmqctl set\_permissions -p / admin '.\*' '.\*' '.\*'

Setting permissions for user "username" in vhost "/" ...

查看权限

[root sbin]# /opt/rabbitmq/sbin/rabbitmqctl list\_permissions

Listing permissions in vhost "/" ...

admin .\* .\* .\*

username ConfP WriteP ReadP

guest .\* .\* .\*

### 配置插件

mkdir /etc/rabbitmq

启用WEB管理UI插件

[root sbin]# /opt/rabbitmq/sbin/rabbitmq-plugins enable rabbitmq\_management

此插件详细介绍见： <https://www.rabbitmq.com/management.html>

启用后，可以访问15672端口查看rabbitmq状态：<http://127.0.0.1:15672/>

### 启动

/opt/rabbitmq/sbin/rabbitmq-server restart

启动后，可以通过ctl查看队列

[root sbin]# ./rabbitmqctl list\_queues

Listing queues ...

hello 1

或者查看交换机

[root sbin]# ./rabbitmqctl list\_exchanges

Listing exchanges ...

direct

amq.direct direct

amq.fanout fanout

amq.headers headers

amq.match headers

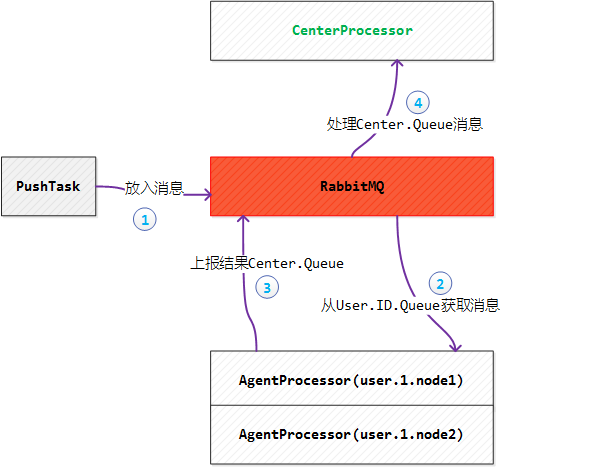
amq.rabbitmq.log topic

amq.rabbitmq.trace topic

amq.topic topic

## 示例

官网 <https://www.rabbitmq.com/getstarted.html> 有详细的入门教程，看一遍能入门，这里演示实际项目如何使用，下面这张多节点通信实例是我在一个实际项目中使用的架构：



我们定义了多个队列，包括中心处理端Center.Queue和节点处理端User.ID.Queue。

### AgentProcessor

AgentProcessor，每台主机一个实例，每台主机定义自己专门的队列，例如：

user.1.node1，表示用户ID为1且节点名称为node1上的队列。

此代理负责从自己的队列里面等待消息，并处理，然后将消息结果返回到Center.Queue。类似于客户端，收集或处理本机数据然后发送给服务器的后期处理队列。

**client.py:**

#!/usr/bin/env python

# coding=utf-8

import json

import pika

import action

*class* Connection(*object*):

*def* \_\_init\_\_(*self*, *userToken*, *nodeIdentify*, *nodeType*, *action*):

        self.userToken = userToken

        self.nodeIdentify = nodeIdentify

        self.nodeType = nodeType

        # connect

        credentials = pika.PlainCredentials('username', 'password')

        self.connection = pika.BlockingConnection(pika.ConnectionParameters('server', *credentials*=credentials))

        # get channels

        self.channel = self.connection.channel()

        self.init()

        self.action = action

*def* init(*self*):

        self.nodeQueue  = "user.{0}.{1}".format(self.userToken, self.nodeIdentify)

        #self.channel.exchange\_declare(exchange='al', exchange\_type='direct')

        self.channel.queue\_declare(*queue*=self.nodeQueue)

        #self.channel.queue\_bind(exchange='al', queue=self.nodeQueue, routing\_key=self.nodeQueue)

        # for send back

        self.channel.exchange\_declare(*exchange*='al', *exchange\_type*='direct')

*def* report(*self*, *msg*):

        self.channel.basic\_publish(*exchange*='al', *routing\_key*="center.queue", *body*=msg)

*def* onMessage(*self*, *channel*, *method\_frame*, *header\_frame*, *body*):

        print *str*(method\_frame.delivery\_tag) + ":" + body

        # deal

        try:

            request = json.loads(body);

        except(*ValueError*):

            print "not json format"

            self.channel.basic\_ack(*delivery\_tag*=method\_frame.delivery\_tag)

            return False;

        # continue

        result = self.action.process(request)

        response = json.dumps(result);

        self.report(response)

        print *str*(method\_frame.delivery\_tag) + ":" + body + " -> " + response

        self.channel.basic\_ack(*delivery\_tag*=method\_frame.delivery\_tag)

*def* run(*self*):

        print "start consuming on " + self.nodeQueue

        self.channel.basic\_consume(self.onMessage, self.nodeQueue)

        try:

            self.channel.start\_consuming()

        except *KeyboardInterrupt*:

            self.channel.stop\_consuming()

*def* \_\_del\_\_(*self*):

        self.connection.close()

*def* entry():

    conn = Connection(1, 'node1', args.type, action.Action())

    conn.run()

entry()

**action.py:**

|  |
| --- |
| # coding=utf-8  import os  import json    *class* Action(*object*):  *def* \_\_init\_\_(*self*):          print "Action init"    *def* onAppsCheck(*self*, *data*):          return 0, 'apps check all ok', nil    *def* process(*self*, *request*):          if request['action'] == "AppsCheck" :              (status, msg, data) = self.onAppsCheck(request['data'])          else:              (status, msg, data) = (1, "undefined action " + *str*(request['action']), None)          return {'status': status, 'action':request['action'], 'msg': msg, 'data' : data} |

### CenterProcessor

CenterProcessor，一般在服务器上启用，其等待并处理Center.Queue队列中的消息，类似于服务端接收数据。处理了数据，可根据实际情况做进一步动作，例如，将数据写入数据库或存入文件。

**server.py:**

|  |  |
| --- | --- |
| # coding=utf-8  import pika  import alcommon    *class* Connection(*object*):  *def* \_\_init\_\_(*self*):          # connect          credentials = pika.PlainCredentials('username', 'password')          self.connection = pika.BlockingConnection(pika.ConnectionParameters('server', *credentials*=credentials))          # get channels          self.channel = self.connection.channel()          self.init()    *def* init(*self*):          self.channel.exchange\_declare(*exchange*='al', *exchange\_type*='direct')          # use anonymous queue          self.queue = self.channel.queue\_declare(*exclusive*=True)          # bind queue to exchange          self.channel.queue\_bind(*exchange*='al', *queue*=self.queue.method.queue, *routing\_key*="center.queue")    *def* onMessage(*self*, *channel*, *method\_frame*, *header\_frame*, *body*):          print(*str*(method\_frame.delivery\_tag) + ":" + body)          # TODO: deal messages          self.channel.basic\_ack(*delivery\_tag*=method\_frame.delivery\_tag)    *def* run(*self*):          print "start consuming on queue" + self.queue.method.queue + ", exchange 'al', route 'center.queue'"          self.channel.basic\_consume(self.onMessage, self.queue.method.queue)          try:              self.channel.start\_consuming()          except *KeyboardInterrupt*:              self.channel.stop\_consuming()    *def* \_\_del\_\_(*self*):          self.connection.close()    conn = Connection()  conn.run() |  |

### PushTask

PushTask，是一个接口，用于将任务分派到不同队列，假设，前端UI需要用户ID为1，节点名称为node1的主机执行命令，那么为了异步，我们需要将任务压入user.1.node1的队列里面，等待node1的AgentProcessor处理后，再发送给Center.Queue，Center.Queue再将数据整理放到数据库，下次前端请求的时候，可以自己读取数据库。

上面的两个示例是Python版本，由于实际项目我使用的是Laravel在做Web服务端，下面的示例实现的是PushTask和CenterProcessor，也是使用的Laravel框架，实现为Laravel Commands。

<?php

namespace App\Console\Commands;

//define('AMQP\_DEBUG', true);

use Illuminate\Console\Command;

use PhpAmqpLib\Connection\AMQPStreamConnection;

use PhpAmqpLib\Message\AMQPMessage;

use App\Opr;

use App\Node;

*class* MQServer extends *Command*

{

    private $exchange\_name = 'al';

    /\*\*

     \* The name and signature of the console command.

     \*

     \* @var string

     \*/

    protected $signature = 'mq

                            {route=center.queue : which route?}

                            {msg=listen : messages want to send, "listen" means listen on route}

                            {--ex= : Whether the send to exchange}';

    /\*\*

     \* The console command description.

     \*

     \* @var string

     \*/

    protected $description = 'use MQ to send message, or listen on server queue at center.queue';

    public *function* \_\_construct()

    {

*parent*::\_\_construct();

        $this->connection = new *AMQPStreamConnection*('server', 5672, 'username', 'password');

        $this->channel = $this->connection->channel();

    }

    public *function* \_\_destruct()

    {

        $this->channel->close();

        $this->connection->close();

    }

*function* onNodeReport($user, $params) {

        $user\_id = $user['token']; //TODO CHECK USER VAILD

        $params['user\_id'] = $user\_id;

        $node = new *Node*;

        $node->updateInsert($params);

    }

*function* dispatchMessage($request) {

        print\_r($request);

        if(isset($request['cookies'])){

            $cookies = $request['cookies'];

            if(isset($cookies['opr'])) {

                $opr = *Opr*::find($cookies['opr']);

                if($opr) {

                    if (0 != $request['status']) {

                        $opr->fail($request['msg']);

                        return true;

                    } else if(isset($request['data']) && isset($request['data']['percent'])) {

                        $opr->updatePercent($request['data']['percent'], $request['data']['desc']);

                        return true;

                    } else {

                        if (method\_exists(*Opr*::class, 'on'.$opr['name'].'Response')) {

                            call\_user\_func\_array(array(*Opr*::class, 'on'.$opr['name'].'Response'), array($opr, $request['data']));

                            return true;

                        } else {

                            $this->error("unexist opr Response for ".$opr['name']);

                        }

                    }

                } else {

                    $this->error("unknow opr by id " .$cookies['opr']);

                }

                return false;

            }

        }

        $data = $request['data'];

        if (!$data) {

            $this->warn($request['msg']." with empty data");

            return true;

        }

        if (method\_exists($this, 'on'.$data['action'])) {

             call\_user\_func\_array(array($this, 'on'.$data['action']), array($data['user'], $data['params']));

        }

    }

*function* server($route) {

        // declare a queue to consume

        $this->channel->exchange\_declare($this->exchange\_name, 'direct', false, false, false);

        list($queue\_name, ,) = $this->channel->queue\_declare("", false, false, true, false);

        // bind queue to exchange and set route filter

        $this->channel->queue\_bind($queue\_name, $this->exchange\_name, $route);

        $callback = *function*($msg){

            $this->info(' [x] '. $msg->delivery\_info['routing\_key']. ':'. $msg->body);

            $request = json\_decode($msg->body, true);

            $this->dispatchMessage($request);

        };

        $this->channel->basic\_consume($queue\_name, '', false, false, false, false, $callback);

        $this->info("listen on queue $queue\_name, exchange $this->exchange\_name, route $route");

        while(count($this->channel->callbacks)) {

            $this->channel->wait();

        }

        $this->warn("exit listen $route!");

    }

*function* client($route, $msg, $exchange = '') {

        if(!empty($exchange)) {

            $this->channel->exchange\_declare($exchange, 'direct', false, false, false);

        }

        $this->channel->basic\_publish(new *AMQPMessage*($msg), $exchange, $route);

        $this->info("$exchange $route -> $msg");

    }

    /\*\*

     \* Execute the console command.

     \*

     \* @return mixed

     \*/

    public *function* handle()

    {

        $route = $this->argument('route');

        $msg = $this->argument('msg');

        $exchange = $this->option('ex');

        if('listen' == $msg) {

            $this->server($route);

        } else {

            //php artisan mq "user.1.00163e0213fe" "hello world"

            $this->client($route, $msg, $exchange);

        }

    }

}

示例可以通过命令运行：

|  |
| --- |
| [root]# ./artisan mq --help  Usage:    mq [options] [--] [<route>] [<msg>]  Arguments:    route                 which route? [default: "center.queue"]    msg                   messages want to send, "listen" means listen on route [default: "listen"]  Options:        --ex[=EX]         Whether the send to exchange    -h, --help            Display this help message    -q, --quiet           Do not output any message    -V, --version         Display this application version        --ansi            Force ANSI output        --no-ansi         Disable ANSI output    -n, --no-interaction  Do not ask any interactive question        --env[=ENV]       The environment the command should run under.    -v|vv|vvv, --verbose  Increase the verbosity of messages: 1 for normal output, 2 for more verbose output and 3 for debug  Help:  use MQ to send message, or listen on server queue at center.queue |

源码不保证编译运行正确，部分库缺失。

2016/1/18