# Centos 安装 RabbitMQ

## 安装Erlang

**1. 下载erlang**

下载地址：<http://download.csdn.net/detail/a15134566493/9517595>

官方下载地址：<http://erlang.org/download/otp_src_18.3.tar.gz>

**2.安装**

#解压

tar xvf otp\_src\_18.3.tar.gz

cd otp\_src\_18.3

#配置 '--prefix'指定的安装目录

./configure --prefix=/usr/local/erlang --with-ssl -enable-threads -enable-smmp-support -enable-kernel-poll --enable-hipe --without-javac

#安装

make && make install

**3.配置erlang环境变量**

vim /etc/profile

#在文件末尾添加下面代码 'ERLANG\_HOME'等于上一步'--prefix'指定的目录

ERLANG\_HOME=/usr/local/erlang

PATH=$ERLANG\_HOME/bin:$PATH

export ERLANG\_HOME

export PATH

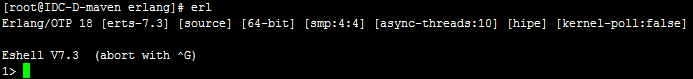
#使环境变量生效

source /etc/profile

#输入命令检验是否安装成功

erl

#如下输出表示安装成功



## 安装RabbitMQ

**1.下载RabbitMQ**

官方下载地址<http://www.rabbitmq.com/releases/rabbitmq-server/v3.6.1/rabbitmq-server-generic-unix-3.6.1.tar.xz>

**2.安装**   
RabbitMQ3.6版本无需make、make install 解压就可以用

#解压rabbitmq，官方给的包是xz压缩包，所以需要使用xz命令

xz -d rabbitmq-server-generic-unix-3.6.1.tar.xz

#xz解压后得到.tar包，再用tar命令解压

tar -xvf rabbitmq-server-generic-unix-3.6.1.tar

#移动目录 看个人喜好

cp -rf ./rabbitmq\_server-3.6.1 /usr/local/

cd /usr/local/

#修改文件夹名

mv rabbitmq\_server-3.6.1 rabbitmq-3.6.1

#开启管理页面插件

cd ./rabbitmq-3.6.1/sbin/

./rabbitmq-plugins enable rabbitmq\_management

**3.启动**

#启动命令，该命令ctrl+c后会关闭服务

./rabbitmq-server

#在后台启动Rabbit

./rabbitmq-server -detached

#关闭服务

./rabbitmqctl stop

#关闭服务(kill) 找到rabbitmq服务的pid [不推荐]

ps -ef|grep rabbitmq

kill -9 \*\*\*\*

**4. 添加管理员账号**

**（**出于安全的考虑，guest这个默认的用户只能通过http://localhost:15672 来登录，不能使用IP地址登录，也就是不能远程访问，这对于服务器上没有安装桌面的情况是无法管理维护的。要解决这个问题需要配置远程登录权限，添加用户，用于外网的访问**）**

#进入RabbitMQ安装目录

cd /usr/local/rabbitmq-3.6.1/sbin

#添加用户

#rabbitmqctl add\_user Username Password

./rabbitmqctl add\_user rabbitadmin 123456

#分配用户标签

#rabbitmqctl set\_user\_tags User Tag

#[administrator]:管理员标签

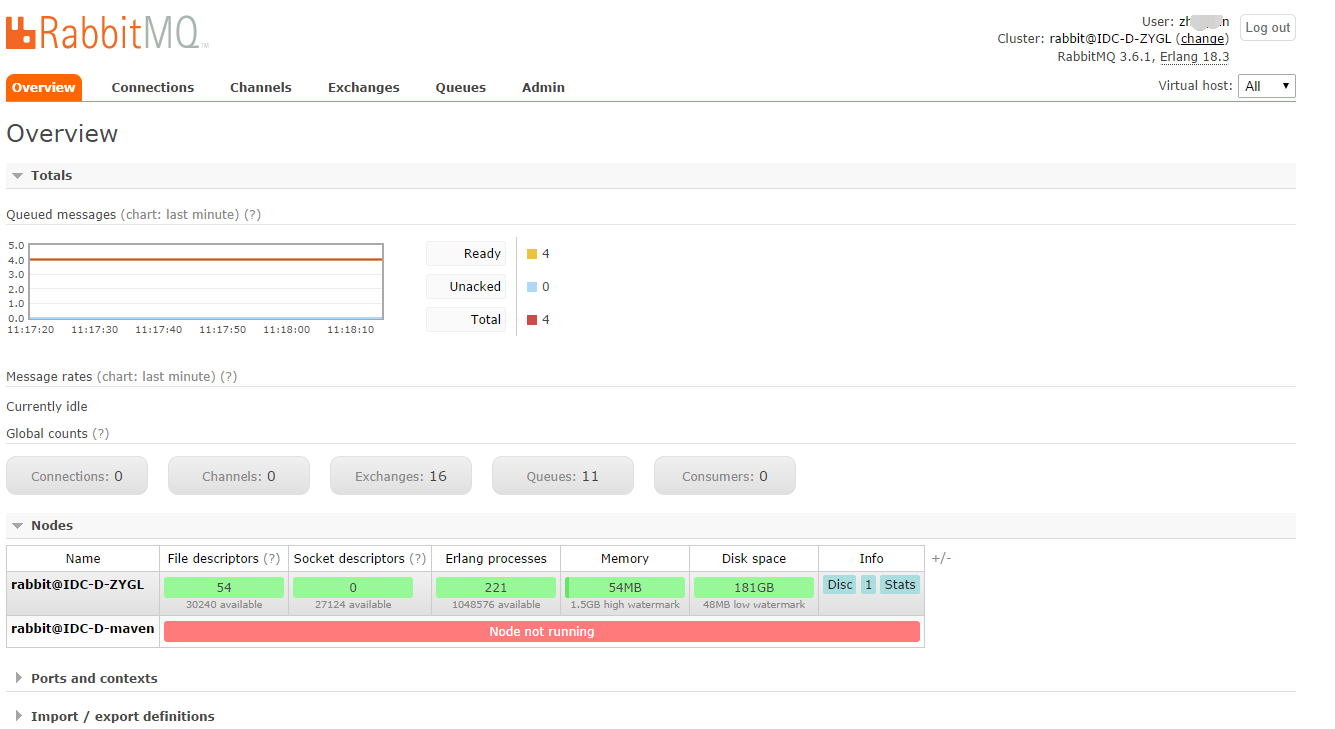
./rabbitmqctl set\_user\_tags rabbitadmin administrator

**5.登录管理界面**

浏览器输入地址：http://服务器IP地址:15672/



输入第4步添加的账号密码登录

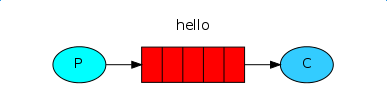


# 通过RabbitMQ实现对接

**要求：在对接过程中始终以结算系统为主（以注册为例）。**

**使用rabbitmq基于服务层面的对接**

## 第一种：结算注册至商城



双方约定消息格式和内容

//===mess\_array===//

/\* Service : 服务名称 \*/

/\* Method : 调用的方法\*/

/\* Args : 方法中的参数\*/

**结算（消息发送方） 将消息主体json编码后再base64编码**

**商城（消息接收方） 将消息主体base64解码后再json解码**

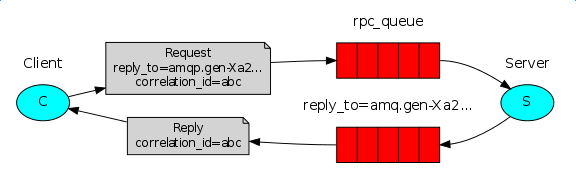
**1.**结算注册成功发送消息至商城，消息主体为商城注册所需参数数组。可在SaleController.class.php注册成功后调用，如下图：



**2.**商城启动消息接收服务, 在根目录service文件夹下增加AmqpconsumeService.php用来处理结算发送的消息（文档最后附上），在service.php里对它调用，启动之后程序在此处挂起等待生产者发布消息，一旦接收到消息通过process\_message方法来处理，根据消息主体内容来判断调用哪个服务，实现对接，如下图：



## 第二种：商城注册至结算



**使用rabbitmq基于服务层面的对接**

双方约定消息格式和内容

/\*mess

\*\* Service : 对方系统服务名称

\*\* Method : 对方系统调用的方法

\*\* Args : 对方系统方法中的参数

\*\* oppositeArg:本系统功能实现所需参数 （同样包括Service Method Args）

\*/

**商城（消息发送方）**

**结算（消息接收方）**

1. 商城注册先发送rpc消息至结算，通过回调返回结果判断是否成功，如果成功也不是真正意义上的成功，结算注册成功后会另外发送注册消息给商城，商城再次消费消息来注册（相当于第一种注册方式）如下图:



1. 结算系统在根目录service文件夹下增加RpcserverService.php用来处理商城发送的rpc消息（文档最后附上），在service.php里对它调用，这样workman启动时rpcserver端的服务也就启动了（如果挂掉workman会自动重启）。

Function apiProcess()://接受对接信息并处理

Function run()://根据接受到的对接信息调用相应的服务

publisher()://处理回调成功后另外发送消息给对方

1. 商城另外启动消息接收服务, 在根目录service文件夹下增加AmqpconsumeService.php用来处理结算发送的消息（文档最后附上），在service.php里对它调用，启动之后程序在此处挂起等待生产者发布消息，一旦接收到消息通过process\_message方法来处理，根据消息主体内容来判断调用哪个服务（同第一种注册方式的第2步）

**注意事项：**

1. **如果两套系统在同一台服务器上那么workman服务需要设置不同的端口，比如结算使用1236，商城可用1237，只要不同即可**

**附上代码：**

**./Service/AmqpconsumeService.php**

**<?php**

**use Workerman\Worker;**

**$cur\_dir = dirname(\_\_FILE\_\_);**

**chdir($cur\_dir);**

**ini\_set('memory\_limit','10000M');**

**//RPC消息队列**

**define('AMQP\_HOST', '192.168.177.154');**

**define('AMQP\_PORT', 5672);**

**define('AMQP\_USER', 'rabbit');**

**define('AMQP\_PASS', '123456');**

**define('AMQP\_VHOST', '/');**

**//If this is enabled you can see AMQP output on the CLI**

**define('AMQP\_DEBUG', false);**

**require\_once \_\_DIR\_\_ . '/../ThinkPHP/Library/Vendor/PhpAmqpLib/Autoloader/autoload.php';**

**use \Vendor\PhpAmqpLib\Connection\AMQPStreamConnection;**

**$consumer = new Worker();**

**// 慢任务，消费者的进程数可以开多一些**

**$consumer->count = 1;**

**/\*\***

**\* 进程启动阻塞式的从队列中读取数据并处理**

**\*/**

**$consumer->closeTime = 0;**

**$consumer->onWorkerStart = function($consumer)**

**{**

**$exchange = 'router';**

**$queue = 'msgs';**

**$consumerTag = 'consumer';**

**$connection = new AMQPStreamConnection(AMQP\_HOST, AMQP\_PORT, AMQP\_USER, AMQP\_PASS, AMQP\_VHOST);**

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

**/\***

**The following code is the same both in the consumer and the producer.**

**In this way we are sure we always have a queue to consume from and an**

**exchange where to publish messages.**

**\*/**

**/\***

**name: $queue**

**passive: false**

**durable: true // the queue will survive server restarts**

**exclusive: false // the queue can be accessed in other channels**

**auto\_delete: false //the queue won't be deleted once the channel is closed.**

**\*/**

**$channel->queue\_declare($queue, false, true, false, false);**

**/\***

**name: $exchange**

**type: direct**

**passive: false**

**durable: true // the exchange will survive server restarts**

**auto\_delete: false //the exchange won't be deleted once the channel is closed.**

**\*/**

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

**$channel->queue\_bind($queue, $exchange);**

**/\***

**queue: Queue from where to get the messages**

**consumer\_tag: Consumer identifier**

**no\_local: Don't receive messages published by this consumer.**

**no\_ack: Tells the server if the consumer will acknowledge the messages.**

**exclusive: Request exclusive consumer access, meaning only this consumer can access the queue**

**nowait:**

**callback: A PHP Callback**

**\*/**

**$channel->basic\_consume($queue, $consumerTag, false, false, false, false, 'process\_message');**

**register\_shutdown\_function('shutdown', $channel, $connection);**

**/\*\***

**\* 循环等待消息**

**\* Loop as long as the channel has callbacks registered**

**\*/**

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

**$channel->wait();**

**}**

**};**

**/\*\* 关闭连接回调**

**\* @param \PhpAmqpLib\Channel\AMQPChannel $channel**

**\* @param \PhpAmqpLib\Connection\AbstractConnection $connection**

**\*/**

**function shutdown($channel, $connection)**

**{**

**$channel->close();**

**$connection->close();**

**}**

**/\*\* 消息回调函数**

**\* @param \PhpAmqpLib\Message\AMQPMessage $message**

**\*/**

**function process\_message($message)**

**{**

**//获取消息内容**

**$messbody = $message->body;**

**//解码**

**$mess = json\_decode(base64\_decode($messbody), true);**

**/\*===mess===//**

**\*\* Service : 当前系统服务名称**

**\*\* Method : 当前系统调用的方法**

**\*\* Args : 当前系统方法中的参数**

**\*/**

**$res = run($mess['Service'],$mess['Method'],$mess['Args']);**

**if($res !== true){**

**echo $res;**

**return false;//必须return false 否则就被消费了**

**}else{**

**echo '成功！';**

**}**

**$message->delivery\_info['channel']->basic\_ack($message->delivery\_info['delivery\_tag']);**

**// Send a message with the string "quit" to cancel the consumer.**

**if ($message->body === 'quit') {**

**$message->delivery\_info['channel']->basic\_cancel($message->delivery\_info['consumer\_tag']);**

**}**

**}**

**/\*执行推送服务**

**\*\* 如果双方在同一台服务器上 那么tcp端口不能重复使用**

**\*\* 比如说商城使用1236端口，结算可使用1237**

**\*/**

**function run($class,$method,$para=array())**

**{**

**//由于与服务端通信都经过此地，所以$\_SESSION和$\_POST的东西都在这个地方传入；**

**if(isset($para['data']['SESSION'])) return 'SESSION不能作为键';**

**if(isset($para['data']['POST'])) return 'POST不能作为键';**

**$para['data']['SESSION']=(isset($\_SESSION) && $\_SESSION)?$\_SESSION:null;**

**$para['data']['POST']=$\_POST?I('post.'):null;**

**$client = stream\_socket\_client("tcp://127.0.0.1:1236", $err\_no, $err\_msg, 5);**

**if(!$client)**

**{**

**exit($err\_msg);**

**}**

**// 一个邮件任务**

**$message = array(**

**'class' => $class,**

**'method' => $method,**

**'args' => $para,**

**);**

**// 数据末尾加一个换行，使其符合Text协议。使用json编码**

**$message = json\_encode($message)."\n";**

**$message = strlen($message)."\n".$message;**

**// 向队列发送任务，让队列慢慢去执行**

**fwrite($client, $message);**

**// 队列返回的结果，这个结果是立即返回的**

**$read=fread($client, 8192);**

**fclose($client);**

**return unserialize($read);**

**}**

**./Service/RpcserverService.php**

**<?php**

**use Workerman\Worker;**

**$cur\_dir = dirname(\_\_FILE\_\_);**

**chdir($cur\_dir);**

**ini\_set('memory\_limit','10000M');**

**//RPC消息队列**

**define('AMQP\_HOST', '192.168.177.154');**

**define('AMQP\_PORT', 5672);**

**define('AMQP\_USER', 'rabbit');**

**define('AMQP\_PASS', '123456');**

**define('AMQP\_VHOST', '/');**

**//If this is enabled you can see AMQP output on the CLI**

**define('AMQP\_DEBUG', false);**

**require\_once \_\_DIR\_\_ . '/../ThinkPHP/Library/Vendor/PhpAmqpLib/Autoloader/autoload.php';**

**use \Vendor\PhpAmqpLib\Connection\AMQPStreamConnection;**

**use \Vendor\PhpAmqpLib\Message\AMQPMessage;**

**use \Vendor\PhpAmqpLib\Message\MsgPublisher;**

**$consumer = new Worker();**

**// 慢任务，消费者的进程数可以开多一些**

**$consumer->count = 1;**

**/\*\***

**\* 进程启动阻塞式的从队列中读取数据并处理**

**\*/**

**$consumer->closeTime = 0;**

**$consumer->onWorkerStart = function($consumer)**

**{**

**//定义队列名称**

**$queue = 'rpc\_queue';**

**//开启连接**

**$connection = new AMQPStreamConnection(AMQP\_HOST, AMQP\_PORT, AMQP\_USER, AMQP\_PASS, AMQP\_VHOST);**

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

**$channel->queue\_declare($queue, false, false, false, false);**

**//echo " [x] Awaiting RPC requests\n";**

**$callback = function($req) {**

**$messageBody = $req->body;//获取消息内容**

**//echo " [.] apiProcess(", $messageBody, ")\n";**

**$msg = new AMQPMessage(**

**(string) apiProcess($messageBody),**

**array('correlation\_id' => $req->get('correlation\_id'))**

**);**

**$req->delivery\_info['channel']->basic\_publish(**

**$msg, '', $req->get('reply\_to'));**

**$req->delivery\_info['channel']->basic\_ack(**

**$req->delivery\_info['delivery\_tag']);**

**};**

**//限制：每次最多给一个消费者发送1条消息**

**$channel->basic\_qos(null, 1, null);**

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

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

**$channel->wait();**

**}**

**$channel->close();**

**$connection->close();**

**};**

**//对接信息处理**

**function apiProcess($messbody) {**

**//解码**

**$mess = json\_decode(base64\_decode($messbody), true);**

**/\*===mess===//**

**\*\* Service : 当前系统服务名称**

**\*\* Method : 当前系统调用的方法**

**\*\* Args : 当前系统方法中的参数**

**\*\* oppositeArg:对方系统参数**

**\*/**

**$res = run($mess['Service'],$mess['Method'],$mess['Args']);**

**if(gettype($res)=='string'){**

**return $res;**

**}else{**

**//需要另外推送给对方的消息重新编码再传输 (这个消息主体是对方在发送rpc消息时一并传过来的)**

**$oppositeMessBody = base64\_encode(json\_encode($mess['oppositeArg']));**

**//rpc消息回到处理成功后另外推送消息至对方**

**$msgPublisher = new MsgPublisher();**

**$msgPublisher->publisher($oppositeMessBody);**

**return true;**

**}**

**}**

**/\*执行推送服务**

**\*\* 如果双方在同一台服务器上 那么tcp端口不能重复使用**

**\*\* 比如说商城使用1236端口，结算可使用1237**

**\*/**

**function run($class,$method,$para=array())**

**{**

**//由于与服务端通信都经过此地，所以$\_SESSION和$\_POST的东西都在这个地方传入；**

**if(isset($para['data']['SESSION'])) return 'SESSION不能作为键';**

**if(isset($para['data']['POST'])) return 'POST不能作为键';**

**$para['data']['SESSION']=(isset($\_SESSION) && $\_SESSION)?$\_SESSION:null;**

**$para['data']['POST']=$\_POST?I('post.'):null;**

**$client = stream\_socket\_client("tcp://127.0.0.1:1237", $err\_no, $err\_msg, 5);**

**if(!$client)**

**{**

**exit($err\_msg);**

**}**

**// 一个邮件任务**

**$message = array(**

**'class' => $class,**

**'method' => $method,**

**'args' => $para,**

**);**

**// 数据末尾加一个换行，使其符合Text协议。使用json编码**

**$message = json\_encode($message)."\n";**

**$message = strlen($message)."\n".$message;**

**// 向队列发送任务，让队列慢慢去执行**

**fwrite($client, $message);**

**// 队列返回的结果，这个结果是立即返回的**

**$read=fread($client, 8192);**

**fclose($client);**

**return unserialize($read);**

**}**