# 消息队列使用说明

### 目录

[药房网 – demo3.0项目相关接口文档 1](#_Toc28672)

[目录 1](#_Toc387)

[一、获取指定每康卡金额接口 1](#_Toc12372)

[二、测试curl抓取京东页面接口 2](#_Toc14982)

——2017.12.15 刘孝全

## 功能概述

http://blog.csdn.net/clh604/article/details/19118585

<http://blog.csdn.net/hzw19920329/article/details/54311277>

<https://www.cnblogs.com/jshen/p/5033517.html>

<http://blog.csdn.net/yangbutao/article/details/10982391> 集群

### 应用场景：

服务层、组件层应用的接口框架，每块业务需独立开发一套接口，为规范标准，特开发出此

Demo接口 作为样版，后续开发人员均以此为框架进行各自的开发

**更新说明:**

此接口已有两个老版本在线上，此为第三版。

解决问题与BUG：

1. 多业务接口混合调用时内部的命名冲突问题
2. 前后端分离，同一业务模块实现了前后端物理与逻辑上的分离，便于开发与维护
3. 降低了入口文件与业务逻辑文件的藕合度，无需再到处new 实例化类了
4. 优化了错误返回机制，实现了错误码统一配置
5. 核心功能更稳定，错误处理更加完善，代码结构与命名更加优雅
6. 自动兼容远程访问，无需额外处理url参数问题
7. 使用更方便，拷贝后只需批量替换demo 关键字为自己的接口名就行了

### 接口使用说明

1,binding key和routing key

　　binding key和routing key是都不过是自己设置的一组字符,只是用的地方不同,binding key是在绑定交换机和队列时候通过方法传递的字符串,routing key是在发布消息时候,顺便带上的字符串,有些人说这两个其实是一个东西,也对也不对,说对,是因为这两个可以完全一样,说不对,是因为这两个起的作用不同,一个交换机可以绑定很多队列,但是每个队列也许需要的消息类型不同,binding key就是这个绑定时候留在交换机和队列之间的提示信息,当消息发送出来后,随着消息一起发送的routing key如果和binding key一样就说明消息是这个队列要的东西,如果不一样那就不要给这个队列,交换机你找找下个队列看看要不要.明白了吧,这两个key就是暗号,对上了就是自己人,对不上那麻烦你再找找去.

　　binding key和routing key的配对其实也不是就要完全一样,还可以'相似'配对,建立交换机的时候,就要告诉MQ,我要声明的这个交换机和它上面的队列之间传输消息时候要求routing key和binding key完全一样,这种模式叫Direct,如果routing key和binding key可以'模糊'匹配,这种模式叫Topic,如果不需要匹配,尽管发,叫Fanout.

2,持久化

　　交换机和队列都可以在创建时候设置为持久化,重启以后会回复,但是其中的消息未不会,如果要消息也恢复,将消息发布到交换机的时候，可以指定一个标志“Delivery Mode”（投递模式）,　　1为非持久化,2为持久化.

3,流控机制

　　当消息生产的速度更快,而进程的处理能力低时,消息就会堆积起来,占用内存越来越多,导致MQ崩溃,所以rabbitmq有一个流控机制,当超过限定时候就会阻止接受消息,mq流控有三种机制

     　　1,主动阻塞住发消息太快的连接,这个无法调整,如果被阻塞了,在abbitmqctl 控制台上会显示一个blocked的状态。

     　　2,内存超过限量,会阻塞连接,在vm\_memory\_high\_watermark可调

    　　 3,剩余磁盘在限定以下mq会 主动阻塞所有的生产者,默认为50m,在disk\_free\_limit可调.

### RPC流程图

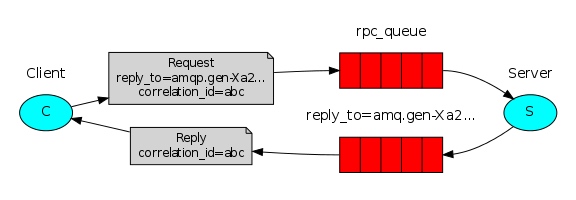
优势

1.每个调用方只需要提出需求，不需要了解对方具体的业务。

2.各自模块各自业务隔离开来，满足面向对象思维，各自封装各自的业务逻辑，不会因为不熟悉业务的人的修改而导致系统崩溃。

3.可以实现跨语言跨平台调用，.NET可以调用JAVA服务，反之亦可。

4.易于拓展，易于复用，当然满足了面向对象的特性肯定有具有面向对象的优势了。此处举个实例，当然的项目因为业务分布不一样，后端服务逻辑一样，有.NET平台的

￼

* 当客户端启动时，它创建了匿名的 callback queue
* 客户端发起 RPC 请求时将同时设置两个 properties：reply\_to 设置为 callback queue；correlation\_id 设置为每个请求都是独一无二的值
* 请求将被发送到一个 rpc\_queue 队列
* RPC 端或者说 Server 一直在等待那个 queue 的请求，当请求到达时，它将通过在 reply\_to 指定的 queue 回复一个 Message 给客户端
* client 一直等待 callback queue 的数据，当 Message 到达时，它将检查 correlation\_id 的值，如果值和它 request 发送时的一致那么就将返回响应

我们的RPC将会类似下面的流程进行工作：

* 当客户端启动时，其将会创建一个匿名的特定的回调队列。
  + 注：RabbitMQ默认有一个exchange， 叫default exchange, 它用一个空字符串表示，它是direct exchange类型，
  + 任何发往此exchange的消息都会被路由到 队列名与routing key同名的队列上，如果没有对应的队列，则消息会被丢弃
* 对于一个RPC调用，客户端发送出的消息都带有两个属性：replyTo，设置回调队列。关联ID（correlationId），对请求设置唯一的id值。
* 请求被发送到一个称为“rpc\_queue”的队列当中。
* RPC worker（也称之为：server）在队列上一直等待请求的发生。当发生请求时，其处理该任务，并使用relayTo指定的队列，将请求结果以消息的形式发送到客户端当中。
* 客户端在回调队列当中等待数据。当有一条消息出现时，就会检查关联ID属性（correlationId）。如果其能够匹配到请求当中的任何一个，那么就会将响应返回给应用程序。

1. CI框架中可直接调用（需装配套扩展，具体问同事），如：

加载配置

$m = require\_once(‘component.yaofang.cn/resque/resque.php’);

$c = call\_user\_func($m, array());

交换机类型

Fanout: 广播模式, 不处理路由键，将消息广播给绑定到该交换机的所有队列 (速度最快)

Direct: 单播模式，处理路由键，对消息路径进行全文匹配。消息路由键 "dog" 只能匹配 "dog" 绑定。default exchange: 特殊的direct交换机，声明时名字为空串， 无需绑定，客户端直接将消息投递到队列, 一个routing\_key也是支持应用到多个队列中的

Topic: 主题模式，处理路由键，按模式匹配路由键。模式符号 "#" 表示一个或多个单词，"\*" 仅匹配一个单词。如 "audit.#" 可匹配 "audit.irs.corporate"，但 "audit.\*" 只匹配 "audit.irs"。

Header: 不建义使用

交换机属性

Php5.x

Durable

Passive

Php 7.x

Auto\_delete

队列类型：

持久性队列：一直存在，不管有没有消费者，可以被多个消费者共享

临时队列： 只绑定到一个消费者的私有队列，该消费者断开，队列删除

队列需要绑定到交换机， 同一交换机可以绑定多个队列

* Durable（消息代理重启后，队列依旧存在）
* Exclusive（只被一个连接（connection）使用，而且当连接关闭后队列即被删除）
* Auto-delete（当最后一个消费者退订后即被删除）
* Arguments（一些消息代理用他来完成类似与TTL的某些额外功能）

消息确认模式：

自动确认： 消息发完， broker就从队列里将它移除

显式确认模式： 有消费者决定何时发ack, broker接到ack后才从队列中删除消息， 如果发ack前， 客户端断了， broker将重新投递消息

应用也可以发送拒绝消息，指示broker丢弃或重传

$c->setExchange(‘交换机’,AMQP\_EX\_TYPE\_DIRECT, AMQP\_DURABLE|AMQP\_AUTODELETE)->addQueue(‘队列名’,’路由键’, AMQP\_DURABLE)->consume()->push();

### 二、接口说明

#### 1、发送消息

* 功能描述：根据每康卡的卡号，获取每康卡金额
* 请求地址：<http://component.yaofang.cn/demo/cy_demo/get>
* 请求方式：GET
* 传输格式：JSON
* 测试地址：<http://component.yaofang.cn/demo/cy_demo/get?card_id=test0001&limit=1>
* 请求参数：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **参数** | **类型** | **是否必须** | 默认值 | **说明** |
| card\_id | String | 是 | “” | 每康卡卡号 |
| limit | int | 是 | 1 | 限制获取每康卡数量 |

返回结果：

基本参数（Json格式）：

|  |  |  |
| --- | --- | --- |
| **属性** | **类型** | **说明** |
| return\_code | String | 请求是否成功，0成功 >0失败(可自定义) |
| msg | String | 请求成功信息: success (与err\_msg不能同时存在) |
| err\_msg | String | 请求失败信息 |

## 2、消费消息

* 功能描述：定期同步京卫药房网药品信息
* 请求地址：<http://component.yaofang.cn/demo/cy_demo/testCurl>
* 请求方式：GET
* 传输格式：JSON
* 测试地址：<http://component.yaofang.cn/demo/cy_demo/testCurl>

返回结果：

基本参数（Json格式）：

|  |  |  |
| --- | --- | --- |
| **属性** | **类型** | **说明** |
| return\_code | String | 请求是否成功，0成功 >0失败(可自定义) |
| Msg | String | 请求成功信息: success (与err\_msg不能同时存在) |
| err\_msg | String | 请求失败信息 |
| Data | String | 京东页面html |

/\*\*

\* Passing in this constant as a flag will forcefully disable all other flags.

\* Use this if you want to temporarily disable the amqp.auto\_ack ini setting.

\* 传递这个参数作为标志将完全禁用其他标志,如果你想临时禁用amqp.auto\_ack设置起效

\*/

define('AMQP\_NOPARAM', 0);

/\*\*

\* Durable exchanges and queues will survive a broker restart, complete with all of their data.

\* 持久化交换机和队列,当代理重启动后依然存在,并包括它们中的完整数据

\*/

define('AMQP\_DURABLE', 2);

/\*\*

\* Passive exchanges and queues will not be redeclared, but the broker will throw an error if the exchange or queue does not exist.

\* 被动模式的交换机和队列不能被重新定义,但是如果交换机和队列不存在,代理将扔出一个错误提示

\*/

define('AMQP\_PASSIVE', 4);

/\*\*

\* Valid for queues only, this flag indicates that only one client can be listening to and consuming from this queue.

\* 仅对队列有效,这个人标志定义队列仅允许一个客户端连接并且从其消费消息

\*/

define('AMQP\_EXCLUSIVE', 8);

/\*\*

\* For exchanges, the auto delete flag indicates that the exchange will be deleted as soon as no more queues are bound

\* to it. If no queues were ever bound the exchange, the exchange will never be deleted. For queues, the auto delete

\* flag indicates that the queue will be deleted as soon as there are no more listeners subscribed to it. If no

\* subscription has ever been active, the queue will never be deleted. Note: Exclusive queues will always be

\* automatically deleted with the client disconnects.

\* 对交换机而言,自动删除标志表示交换机将在没有队列绑定的情况下被自动删除,如果从没有队列和其绑定过,这个交换机将不会被删除.

\* 对队列而言,自动删除标志表示如果没有消费者和你绑定的话将被自动删除,如果从没有消费者和其绑定,将不被删除,独占队列在客户断

\* 开连接的时候将总是会被删除

\*/

define('AMQP\_AUTODELETE', 16);

/\*\*

\* Clients are not allowed to make specific queue bindings to exchanges defined with this flag.

\* 这个标志标识不允许自定义队列绑定到交换机上

\*/

define('AMQP\_INTERNAL', 32);

/\*\*

\* When passed to the consume method for a clustered environment, do not consume from the local node.

\* 在集群环境消费方法中传递这个参数,表示将不会从本地站点消费消息

\*/

define('AMQP\_NOLOCAL', 64);

/\*\*

\* When passed to the {@link AMQPQueue::get()} and {@link AMQPQueue::get()} methods as a flag,

\* the messages will be immediately marked as acknowledged by the server upon delivery.

\* 当在队列get方法中作为标志传递这个参数的时候,消息将在被服务器输出之前标志为acknowledged (已收到)

\*/

define('AMQP\_AUTOACK', 128);

/\*\*

\* Passed on queue creation, this flag indicates that the queue should be deleted if it becomes empty.

\* 在队列建立时候传递这个参数,这个标志表示队列将在为空的时候被删除

\*/

define('AMQP\_IFEMPTY', 256);

/\*\*

\* Passed on queue or exchange creation, this flag indicates that the queue or exchange should be

\* deleted when no clients are connected to the given queue or exchange.

\* 在交换机或者队列建立的时候传递这个参数,这个标志表示没有客户端连接的时候,交换机或者队列将被删除

\*/

define('AMQP\_IFUNUSED', 512);

/\*\*

\* When publishing a message, the message must be routed to a valid queue. If it is not, an error will be returned.

\* 当发布消息的时候,消息必须被正确路由到一个有效的队列,否则将返回一个错误

\*/

define('AMQP\_MANDATORY', 1024);

/\*\*

\* When publishing a message, mark this message for immediate processing by the broker. (High priority message.)

\* 当发布消息时候,这个消息将被立即处理.

\*/

define('AMQP\_IMMEDIATE', 2048);

/\*\*

\* If set during a call to {@link AMQPQueue::ack()}, the delivery tag is treated as "up to and including", so that multiple

\* messages can be acknowledged with a single method. If set to zero, the delivery tag refers to a single message.

\* If the AMQP\_MULTIPLE flag is set, and the delivery tag is zero, this indicates acknowledgement of all outstanding

\* messages.

\* 当在调用AMQPQueue::ack时候设置这个标志,传递标签将被视为最大包含数量,以便通过单个方法标示多个消息为已收到,如果设置为0

\* 传递标签指向单个消息,如果设置了AMQP\_MULTIPLE,并且传递标签是0,将所有未完成消息标示为已收到

\*/

define('AMQP\_MULTIPLE', 4096);

/\*\*

\* If set during a call to {@link AMQPExchange::bind()}, the server will not respond to the method.The client should not wait

\* for a reply method. If the server could not complete the method it will raise a channel or connection exception.

\* 当在调用AMQPExchange::bind()方法的时候,服务器将不响应请求,客户端将不应该等待响应,如果服务器无法完成该方法,将会抛出一个异常

\*/

define('AMQP\_NOWAIT', 8192);

/\*\*

\* If set during a call to {@link AMQPQueue::nack()}, the message will be placed back to the queue.

\* 如果在调用AMQPQueue::nack方法时候设置,消息将会被传递回队列

\*/

define('AMQP\_REQUEUE', 16384);

/\*\*

\* A direct exchange type.

\* direct类型交换机

\*/

define('AMQP\_EX\_TYPE\_DIRECT', 'direct');

/\*\*

\* A fanout exchange type.

\* fanout类型交换机

\*/

define('AMQP\_EX\_TYPE\_FANOUT', 'fanout');

/\*\*

\* A topic exchange type.

\* topic类型交换机

\*/

define('AMQP\_EX\_TYPE\_TOPIC', 'topic');

/\*\*

\* A header exchange type.

\* header类型交换机

\*/

define('AMQP\_EX\_TYPE\_HEADERS', 'headers');

/\*\*

\* socket连接超时设置

\*/

define('AMQP\_OS\_SOCKET\_TIMEOUT\_ERRNO', 536870947);