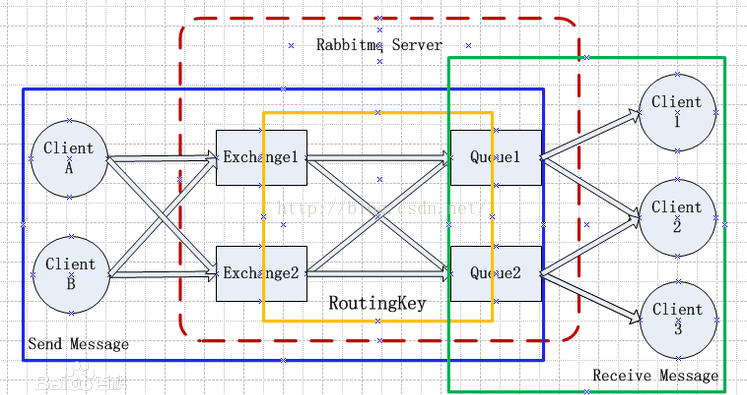
RabbitMQ 开发手册

## 一、RabbitMQ简介

（1）、RabbitMQ是一个由erlang开发的AMQP（Advanced Message Queue ）的流行的开源消息队列系统。RabbitMQ是AMQP（高级消息队列协议）的标准实现

RabbitMQ的结构图如下：

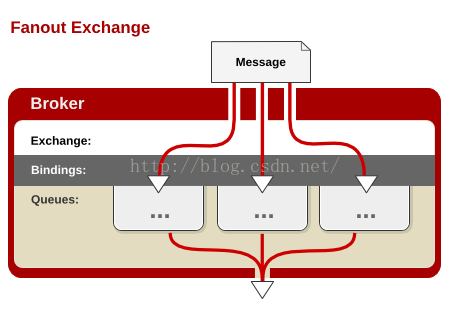


RabbitMQ几个概念说明：

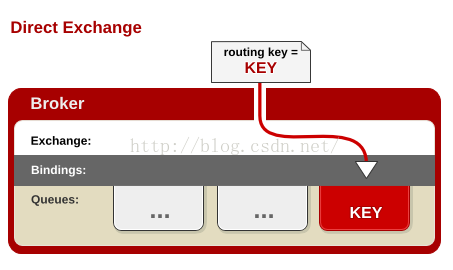
Broker：简单来说就是消息队列服务器实体。  
Exchange：消息交换机，它指定消息按什么规则，路由到哪个队列。  
Queue：消息队列载体，每个消息都会被投入到一个或多个队列。  
Binding：绑定，它的作用就是把exchange和queue按照路由规则绑定起来。  
Routing Key：路由关键字，exchange根据这个关键字进行消息投递。  
vhost：虚拟主机，一个broker里可以开设多个vhost，用作不同用户的权限分离。  
producer：消息生产者，就是投递消息的程序。  
consumer：消息消费者，就是接受消息的程序。  
channel：消息通道，在客户端的每个连接里，可建立多个channel，每个channel代表一个会话任务。

（2）、**Exchange----转发器，**类型有：Direct（直连）、Topic（主题模式）、Headers（标题）和Fanout

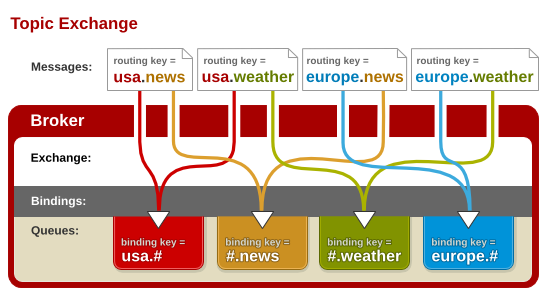
Fanout--广播模式，广播接受到的消息给所有的队列



driect---直接指定模式



Topic模式--匹配模式



Headers模式

Headers是一个键值对，可以定义成Hashtable。发送者在发送的时候定义一些键值对，接收者也可以再绑定时候传入一些键值对，两者匹配的话，则对应的队列就可以收到消息

## 二、应用场景：

#### 2.1异步处理

场景说明：用户注册后，需要发注册邮件和注册短信。传统的做法有两种 1.串行的方式；2.并行方式

（1）串行方式：将注册信息写入**[数据库](http://lib.csdn.net/base/mysql" \o "MySQL知识库" \t "_blank)**成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后，返回给客户端



并行方式：将注册信息写入数据库成功后，发送注册邮件的同时，发送注册短信。以上三个任务完成后，返回给客户端。与串行的差别是，并行的方式可以提高处理的时间



引入消息队列，将不是必须的业务逻辑，异步处理



#### 2.2应用解耦

场景说明：用户下单后，订单系统需要通知库存系统。传统的做法是，订单系统调用库存系统的接口。如下图



引入应用消息队列后的方案



订单系统：用户下单后，订单系统完成持久化处理，将消息写入消息队列，返回用户订单下单成功

库存系统：订阅下单的消息，采用拉/推的方式，获取下单信息，库存系统根据下单信息，进行库存操作

假如：在下单时库存系统不能正常使用。也不影响正常下单，因为下单后，订单系统写入消息队列就不再关心其他的后续操作了。实现订单系统与库存系统的应用解耦

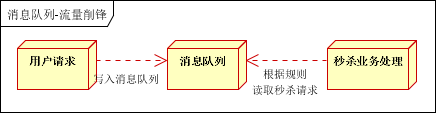
#### 2.3流量削锋

流量削锋也是消息队列中的常用场景，一般在秒杀或团抢活动中使用广泛

应用场景：秒杀活动，一般会因为流量过大，导致流量暴增，应用挂掉。为解决这个问题，一般需要在应用前端加入消息队列。

可以控制活动的人数

可以缓解短时间内高流量压垮应用



用户的请求，服务器接收后，首先写入消息队列。假如消息队列长度超过最大数量，则直接抛弃用户请求或跳转到错误页面

秒杀业务根据消息队列中的请求信息，再做后续处理

#### 2.4日志处理

日志处理是指将消息队列用在日志处理中，比如Kafka的应用，解决大量日志传输的问题。架构简化如下



日志采集客户端，负责日志数据采集，定时写受写入Kafka队列

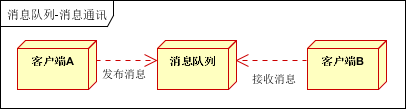
Kafka消息队列，负责日志数据的接收，存储和转发

日志处理应用：订阅并消费kafka队列中的日志数据

#### 2.5消息通讯

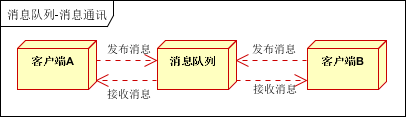
消息通讯是指，消息队列一般都内置了高效的通信机制，因此也可以用在纯的消息通讯。比如实现点对点消息队列，或者聊天室等

点对点通讯：



客户端A和客户端B使用同一队列，进行消息通讯。

聊天室通讯：



客户端A，客户端B，客户端N订阅同一主题，进行消息发布和接收。实现类似聊天室效果。

以上实际是消息队列的两种消息模式，点对点或发布订阅模式

## 三、RabbitMQ开发指导

1. 配置文件

（1）、修改本地配置文件rabbitmq.properties，设置rabbitMQ服务器地址，端口和用户信息

|  |
| --- |
| **ip**=**10.11.20.115 port**=**5672 user\_name**=**guest password**=**guest** |

（2）、添加pox.xml文件里依赖包

|  |
| --- |
| <**dependency**>  <**groupId**>org.springframework.amqp</**groupId**>  <**artifactId**>spring-rabbit</**artifactId**>  <**version**>1.5.6.RELEASE</**version**> </**dependency**> |

1. 消息提供方

（1）先将消息发送适配器实例化，并指定发送的消息队列名称

**final** SenderAdapter senderAdapter = SenderAdapter.APPLICATION\_CONTEXT.getBean(SenderAdapter.**class，"SendQueueName"**);

（2）调用消息发送适配器send方法，将要发送的内容发送到指定的队列上;

senderAdapter.send(userMessage);

1. 消息消费方
2. 先将消息发送适配器实例化

**final** SenderAdapter senderAdapter = SenderAdapter.APPLICATION\_CONTEXT.getBean(SenderAdapter.**class**);

1. 将指定channels上队列里的消息通过Consumer方法消费掉。参数一指定消费的队列名称；参数二指定消息体对象

**final** SenderAdapter senderAdapter = SenderAdapter.APPLICATION\_CONTEXT.getBean(SenderAdapter.**class**);

consumerAdapter.consume(**"SendTest"**,**new** TestMessageProcess());

1. 消息体方法封装，消息体要实现MessageProcess接口。在process方法中实现消息处理业务逻辑。

**public class** UserMessageProcess **implements** MessageProcess<T>

1. RabbitMQ开发实例
2. 消息提供方示例：

|  |
| --- |
| **public class** SendTest {  **public static void** main(String[] args) **throws** IOException {   **final** SenderAdapter senderAdapter = SenderAdapter.***APPLICATION\_CONTEXT***.getBean(SenderAdapter.**class**,**"test"**);   **int** id=0;  senderAdapter.send(**new** UserMessage(id++, **""** + System.*nanoTime*()));  } } |

1. 消息消费方示例：

MessageProcess接口实现：

|  |
| --- |
| **public class** UserMessageProcess **implements** MessageProcess<UserMessage> {  @Override  **public** DetailRes process(UserMessage userMessage) {  System.***out***.println(userMessage);   **return new** DetailRes(**true**, **""**);  } } |

启动消息接收线程：

|  |
| --- |
| **public static void** main(String[] args) **throws** IOException {   ConsumerPoolAdapter consumerPoolAdapter = ConsumerPoolAdapter.*ac*.getBean(ConsumerPoolAdapter.**class**,**"test"**,**new** UserMessageProcess());   consumerPoolAdapter.start(); } |