RabbitMQ基础知识

简介：RabbitMQ是实现了高级消息队列协议（AMQP）的开源消息代理软件（亦称面向消息的中间件）。RabbitMQ服务器是用Erlang语言编写的，而集群和故障转移是构建在开放电信平台框架上的。所有主要的编程语言均有与代理接口通讯的客户端库。

rabbitMq消息持久化：

目的：防止重启的时候，队列及消息丢失

1. 队列持久化需要在声明队列时添加参数 durable=True，这样在rabbitmq崩溃时也能保存队列
2. 仅仅使用durable=True ，只能持久化队列，不能持久化消息
3. 消息持久化需要在消息生成时，添加参数 properties=pika.BasicProperties(delivery_mode=2)

一、背景

RabbitMQ是一个由erlang开发的AMQP（Advanced Message Queue ）的开源实现。AMQP 的出现其实也是应了广大人民群众的需求，虽然在同步消息通讯的世界里有很多公开标准（如 COBAR的 IIOP ，或者是 SOAP 等），但是在异步消息处理中却不是这样，只有大企业有一些商业实现（如微软的 MSMQ ，IBM 的 Websphere MQ 等），因此，在 2006 年的 6 月，Cisco 、Redhat、iMatix 等联合制定了 AMQP 的公开标准。

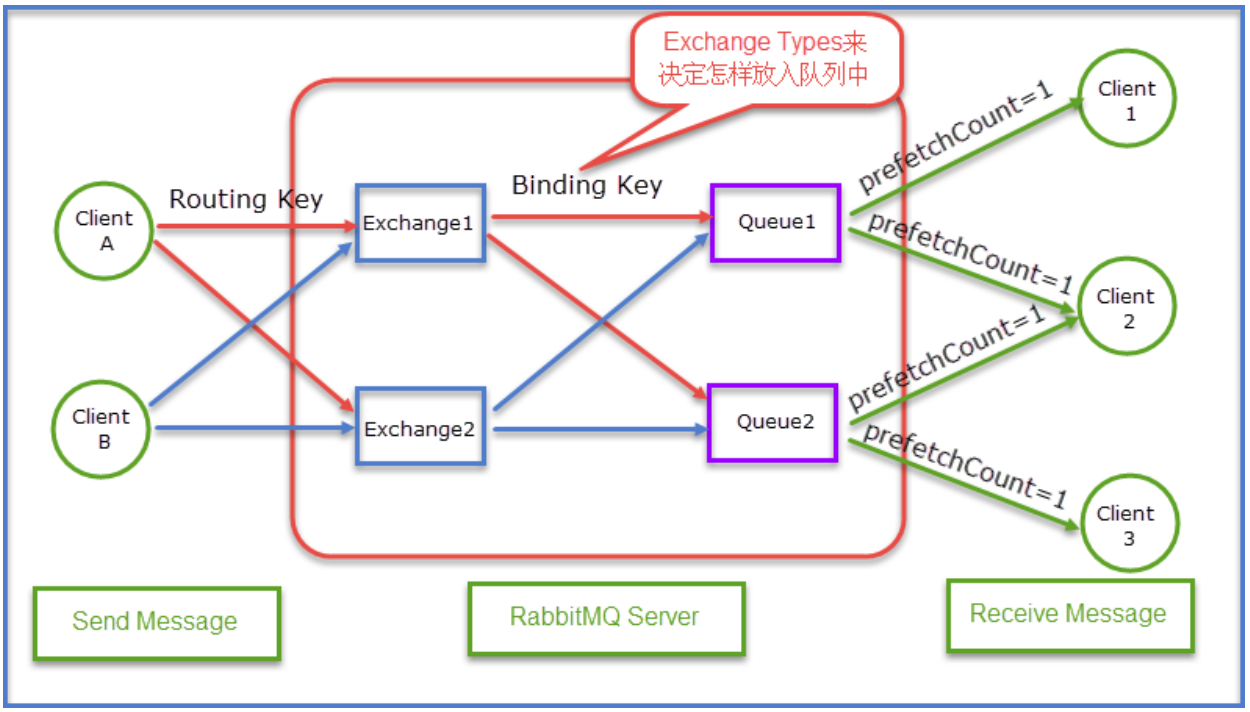
RabbitMQ是由RabbitMQ Technologies Ltd开发并且提供商业支持的。该公司在2010年4月被SpringSource（VMWare的一个部门）收购。在2013年5月被并入Pivotal。其实VMWare，Pivotal和EMC本质上是一家的。不同的是VMWare是独立上市子公司，而Pivotal是整合了EMC的某些资源，现在并没有上市。

RabbitMQ的官网是http://www.rabbitmq.com

花絮：本篇文章是一个系列的文章，本片是开篇，后续会陆陆续续的整理出来，我会现在我自己个人博客发表出来(www.battleheart.cn)，因为在自己的博客里面可以先修改了完善有些不对的地方，等完善后再发到博客园，免得误导大家。

二、基础概念

讲解基础概念的前面，我们先来整体构造一个结构图，这样会方便们更好地去理解RabbitMQ的基本原理。

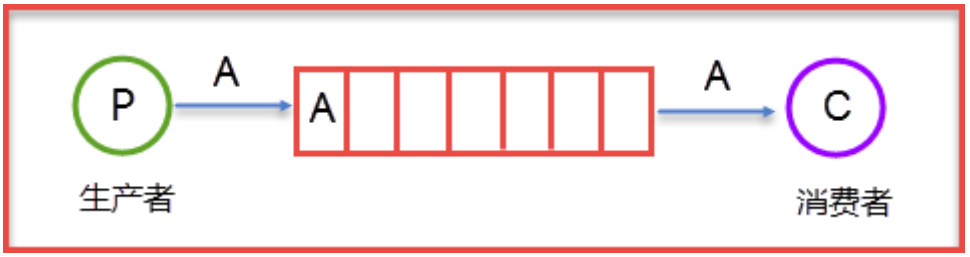


图一

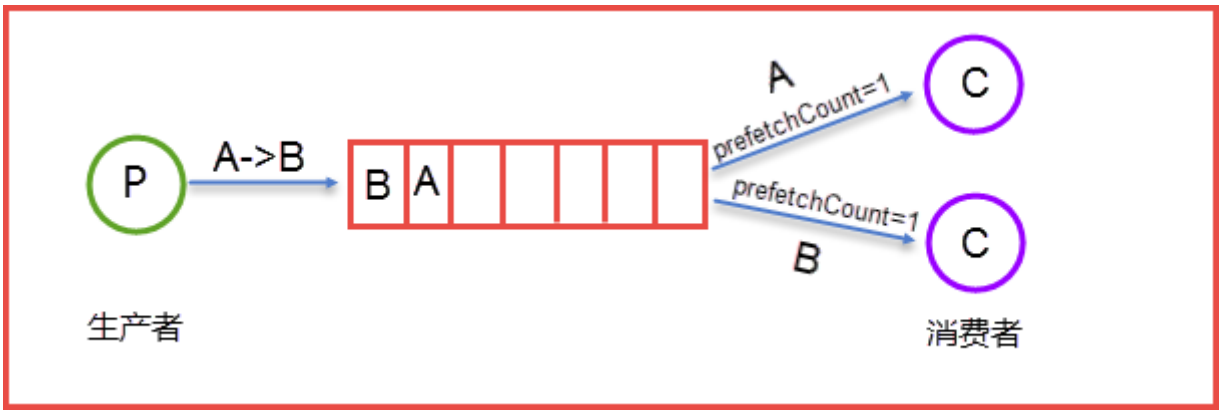
通过上面这张应用相结合的结构图既能够清晰的看清楚整体的send Message到Receive Message的一个大致的流程。当然上面有很多名词都相比还没有介绍到，不要着急接下来我们就开始对其进行详细的讲解。

Queue

Queue（队列）RabbitMQ的作用是存储消息，队列的特性是先进先出。上图可以清晰地看到Client A和Client B是生产者，生产者生产消息最终被送到RabbitMQ的内部对象Queue中去，而消费者则是从Queue队列中取出数据。可以简化成表示为：



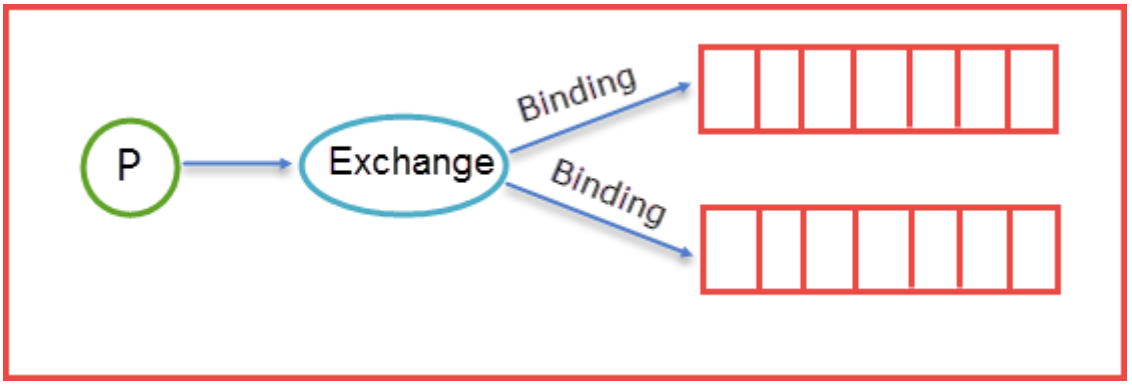
生产者Send Message “A”被传送到Queue中，消费者发现消息队列Queue中有订阅的消息，就会将这条消息A读取出来进行一些列的业务操作。这里只是一个消费正对应一个队列Queue，也可以多个消费者订阅同一个队列Queue，当然这里就会将Queue里面的消息平分给其他的消费者，但是会存在一个一个问题就是如果每个消息的处理时间不同，就会导致某些消费者一直在忙碌中，而有的消费者处理完了消息后一直处于空闲状态，因为前面已经提及到了Queue会平分这些消息给相应的消费者。这里我们就可以使用prefetchCount来限制每次发送给消费者消息的个数。详情见下图所示：



这里的prefetchCount=1是指每次从Queue中发送一条消息来。等消费者处理完这条消息后Queue会再发送一条消息给消费者。

Exchange

我们在开篇的时候就留了一个坑，就是那个应用结构图里面，消费者Client A和消费者Client B是如何知道我发送的消息是给Queue1还是给Queue2，有没有过这个问题，那么我们就来解开这个面纱，看看到底是个什么构造。首先明确一点就是生产者产生的消息并不是直接发送给消息队列Queue的，而是要经过Exchange（交换器），由Exchange再将消息路由到一个或多个Queue，当然这里还会对不符合路由规则的消息进行丢弃掉，这里指的是后续要谈到的Exchange Type。那么Exchange是怎样将消息准确的推送到对应的Queue的呢？那么这里的功劳最大的当属Binding，RabbitMQ是通过Binding将Exchange和Queue链接在一起，这样Exchange就知道如何将消息准确的推送到Queue中去。简单示意图如下所示：



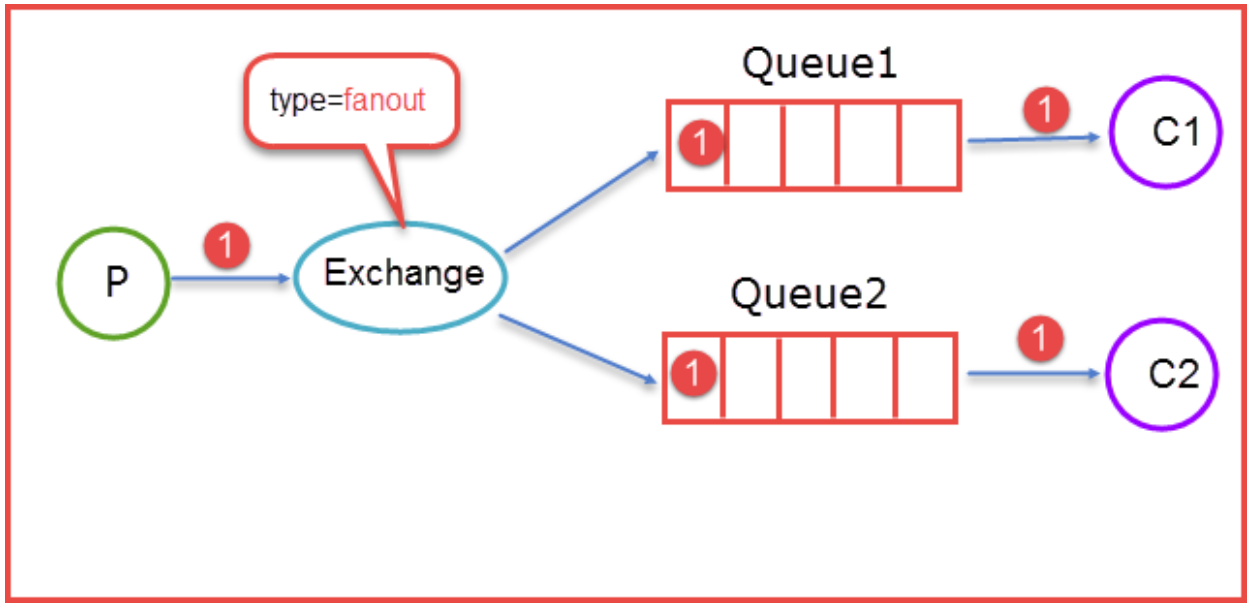
在绑定（Binding）Exchange和Queue的同时，一般会指定一个Binding Key，生产者将消息发送给Exchange的时候，一般会产生一个Routing Key，当Routing Key和Binding Key对应上的时候，消息就会发送到对应的Queue中去。那么Exchange有四种类型，不同的类型有着不同的策略。也就是表明不同的类型将决定绑定的Queue不同，换言之就是说生产者发送了一个消息，Routing Key的规则是A，那么生产者会将Routing Key=A的消息推送到Exchange中，这时候Exchange中会有自己的规则，对应的规则去筛选生产者发来的消息，如果能够对应上Exchange的内部规则就将消息推送到对应的Queue中去。那么接下来就来详细讲解下Exchange里面类型。

Exchange Type

我来用表格来描述下类型以及类型之间的区别。

- **fanout**

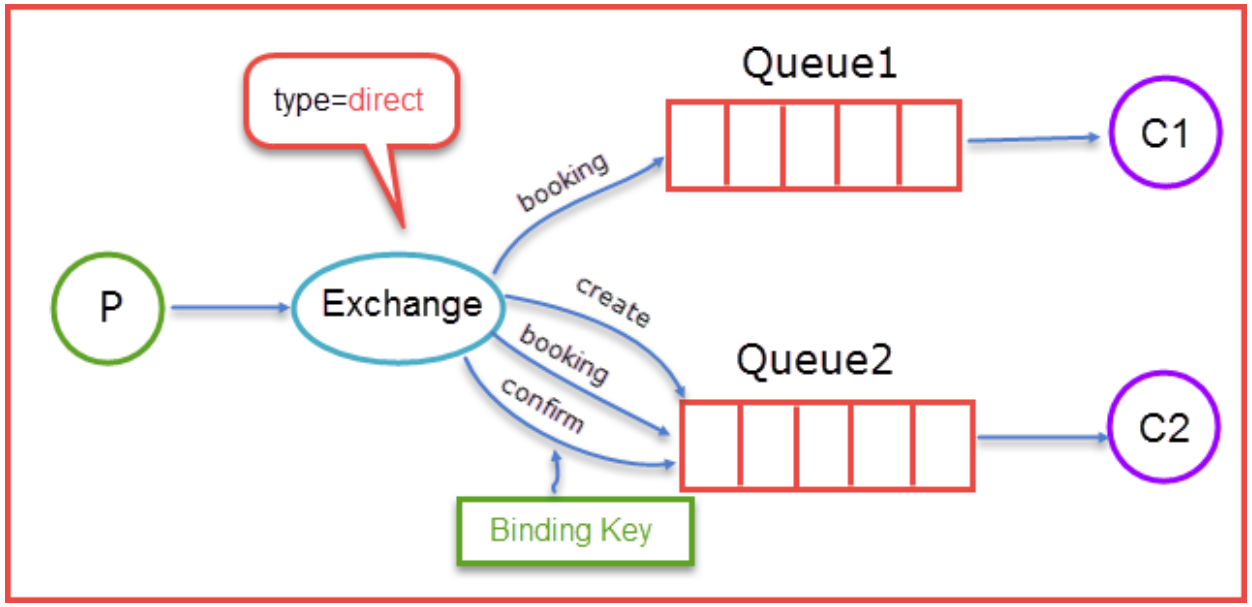
fanout类型的Exchange路由规则非常简单，它会把所有发送到该Exchange的消息路由到所有与它绑定的Queue中。



上图所示，生产者（P）生产消息1将消息1推送到Exchange，由于Exchange Type=fanout这时候会遵循fanout的规则将消息推送到所有与它绑定Queue，也就是图上的两个Queue最后两个消费者消费。

- **direct**

direct类型的Exchange路由规则也很简单，它会把消息路由到那些binding key与routing key完全匹配的Queue中

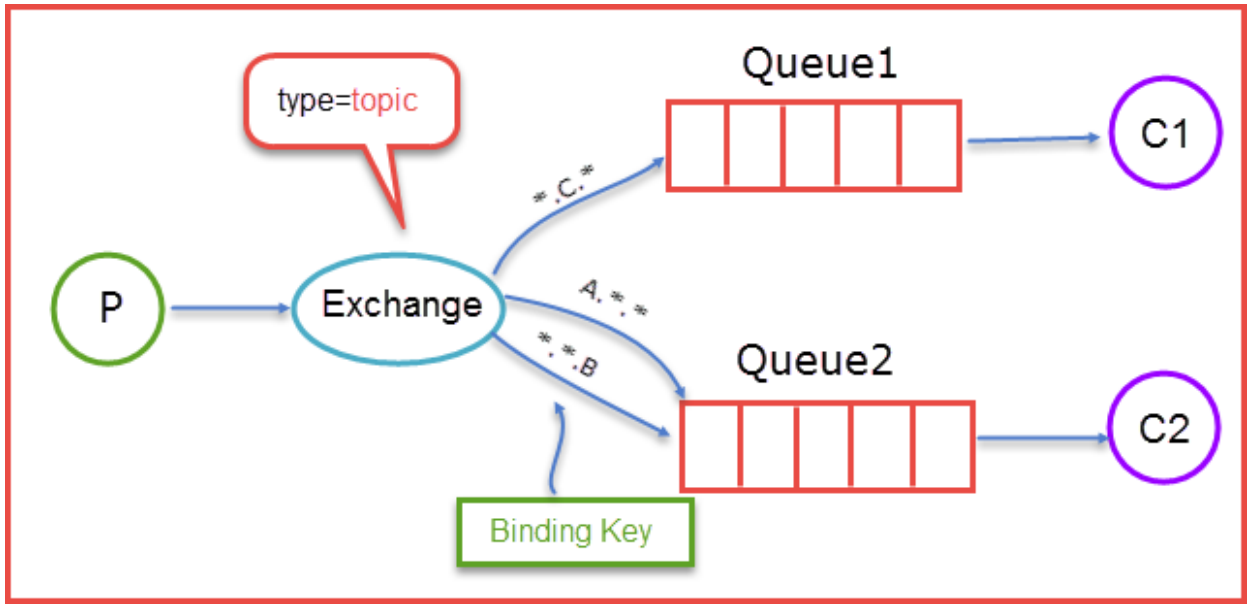


当生产者（P）发送消息时Rotuing key=booking时，这时候将消息传送给Exchange，Exchange获取到生产者发送过来消息后，会根据自身的规则进行与匹配相应的Queue，这时发现Queue1和Queue2都符合，就会将消息传送给这两个队列，如果我们以Rotuing key=create和Rotuing key=confirm发送消息时，这时消息只会被推送到Queue2队列中，其他Routing Key的消息将会被丢弃。

- **topic**

前面提到的direct规则是严格意义上的匹配，换言之Routing Key必须与Binding Key相匹配的时候才将消息传送给Queue，那么topic这个规则就是模糊匹配，可以通过通配符满足一部分规则就可以传送。它的约定是：

1. routing key为一个句点号“.”分隔的字符串（我们将被句点号“.”分隔开的每一段独立的字符串称为一个单词），
如“stock.usd.nyse”、“nyse.vmw”、“quick.orange.rabbit”
2. binding key与routing key一样也是句点号“.”分隔的字符串
3. binding key中可以存在两种特殊字符“*”与“#”，用于做模糊匹配，其中“*”用于匹配一个单词，“#”用于匹配多个单词（可以是零个）



当生产者发送消息Routing Key=F.C.E的时候，这时候只满足Queue1，所以会被路由到Queue中，如果Routing Key=A.C.E这时候会被同是路由到Queue1和Queue2中，如果Routing Key=A.F.B时，这里只会发送一条消息到Queue2中。

- **headers**

headers类型的Exchange不依赖于routing key与binding key的匹配规则来路由消息，而是根据发送的消息内容中的headers属性进行匹配。
在绑定Queue与Exchange时指定一组键值对；当消息发送到Exchange时，RabbitMQ会取到该消息的headers（也是一个键值对的形式），对比其中的键值对是否完全匹配Queue与Exchange绑定时指定的键值对；如果完全匹配则消息会路由到该Queue，否则不会路由到该Queue。
该类型的Exchange没有用到过（不过也应该很有用武之地），所以不做介绍。
这里在对其进行简要的表格整理：

类型名称	类型描述
fanout	把所有发送到该Exchange的消息路由到Queue中
direct	Routing Key==Binding Key

topic	我这里自己总结的简称模糊匹配
headers	Exchange不依赖于routing key与bind 则来路由消息，而是根据发送的消息内 性进行匹配。

补充说明：

ConnectionFactory、Connection、Channel

ConnectionFactory、Connection、Channel都是RabbitMQ对外提供的API中最基本的对象。Connection是RabbitMQ的socket链接，它封装了socket协议相关部分逻辑。ConnectionFactory为Connection的制造工厂。

Channel是我们与RabbitMQ打交道的最重要的一个接口，我们大部分的业务操作是在Channel这个接口中完成的，包括定义Queue、定义Exchange、绑定Queue与Exchange、发布消息等。

Connection就是建立一个TCP连接，生产者和消费者的都是通过TCP的连接到RabbitMQ Server中的，这个后续会再程序中体现出来。

Channel虚拟连接，建立在上面TCP连接的基础上，数据流动都是通过Channel来进行的。为什么不是直接建立在TCP的基础上进行数据流动呢？如果建立在TCP的基础上进行数据流动，建立和关闭TCP连接有代价。频繁的建立关闭TCP连接对于系统的性能有很大的影响，而且TCP的连接数也有限制，这也限制了系统处理高并发的能力。但是，在TCP连接中建立Channel是没有上述代价的。

三、结束语

整理就到这里了。参考内容：

http://blog.csdn.net/whycold/article/details/41119807

http://blog.csdn.net/anzhsoft/article/details/19563091

相信学过这篇文章的人都会对RabbitMQ有一个初步的了解。本篇文章是建立在上面两篇文章的基础上加上自己的理解而形成的，收获很多。其实我个人感觉写一篇文章并不容易，因为要查很多资料以及想很多面，要严谨不能说写出来的东西没有通过验证就发出来这样给别人带来了误导，那我觉得是我的损失。所以如果又看了文章说那些又错误了就请指正，小丁会将它改过来。