MQ

# 什么叫消息队列

消息（Message）是指在应用间传送的数据。消息可以非常简单，比如只包含文本字符串，也可以更复杂，可能包含嵌入对象。

消息队列（Message Queue）是一种应用间的通信方式，消息发送后可以立即返回，由消息系统来确保消息的可靠传递。消息发布者只管把消息发布到 MQ 中而不用管谁来取，消息使用者只管从 MQ 中取消息而不管是谁发布的。这样发布者和使用者都不用知道对方的存在。

https://www.zhihu.com/question/34243607/answer/140732170

# 为何用消息队列

四大好处：

1.解耦，2.提速，3.广播，4.削峰。

消息队列其成本：

1. 引入复杂度。
2. 暂时的不一致性。

# RabbitMQ特点

RabbitMQ 是一个由 Erlang 语言开发的 AMQP 的开源实现。

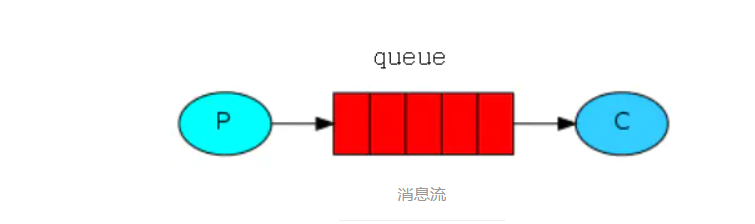
1. 可靠性（Reliability）  
   RabbitMQ 使用一些机制来保证可靠性，如持久化、传输确认、发布确认。
2. 灵活的路由（Flexible Routing）  
   在消息进入队列之前，通过 Exchange 来路由消息的。对于典型的路由功能，RabbitMQ 已经提供了一些内置的 Exchange 来实现。针对更复杂的路由功能，可以将多个 Exchange 绑定在一起，也通过插件机制实现自己的 Exchange 。
3. 消息集群（Clustering）  
   多个 RabbitMQ 服务器可以组成一个集群，形成一个逻辑 Broker 。
4. 高可用（Highly Available Queues）  
   队列可以在集群中的机器上进行镜像，使得在部分节点出问题的情况下队列仍然可用。
5. 多种协议（Multi-protocol）  
   RabbitMQ 支持多种消息队列协议，比如 STOMP、MQTT 等等。
6. 多语言客户端（Many Clients）  
   RabbitMQ 几乎支持所有常用语言，比如 Java、.NET、Ruby 等等。
7. 管理界面（Management UI）  
   RabbitMQ 提供了一个易用的用户界面，使得用户可以监控和管理消息 Broker 的许多方面。
8. 跟踪机制（Tracing）  
   如果消息异常，RabbitMQ 提供了消息跟踪机制，使用者可以找出发生了什么。
9. 插件机制（Plugin System）  
   RabbitMQ 提供了许多插件，来从多方面进行扩展，也可以编写自己的插件

## 参考链接：

<https://www.jianshu.com/p/79ca08116d57>

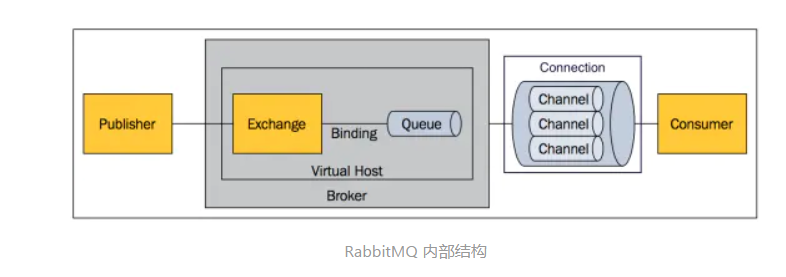
## RabbitMQ 中的概念模型

所有 MQ 产品从模型抽象上来说都是一样的过程：  
消费者（consumer）订阅某个队列。生产者（producer）创建消息，然后发布到队列（queue）中，最后将消息发送到监听的消费者。



### RabbitMQ 基本概念

AMQP 中的基本概念



1. Message

消息，消息是不具名的，它由消息头和消息体组成。消息体是不透明的，而消息头则由一系列的可选属性组成，这些属性包括routing-key（路由键）、priority（相对于其他消息的优先权）、delivery-mode（指出该消息可能需要持久性存储）等。

1. Publisher

消息的生产者，也是一个向交换器发布消息的客户端应用程序。

1. Exchange

交换器，用来接收生产者发送的消息并将这些消息路由给服务器中的队列。

1. Binding

绑定，用于消息队列和交换器之间的关联。一个绑定就是基于路由键将交换器和消息队列连接起来的路由规则，所以可以将交换器理解成一个由绑定构成的路由表。

1. Queue

消息队列，用来保存消息直到发送给消费者。它是消息的容器，也是消息的终点。一个消息可投入一个或多个队列。消息一直在队列里面，等待消费者连接到这个队列将其取走。

1. Connection

网络连接，比如一个TCP连接。

1. Channel

信道，多路复用连接中的一条独立的双向数据流通道。信道是建立在真实的TCP连接内地虚拟连接，AMQP 命令都是通过信道发出去的，不管是发布消息、订阅队列还是接收消息，这些动作都是通过信道完成。因为对于操作系统来说建立和销毁 TCP 都是非常昂贵的开销，所以引入了信道的概念，以复用一条 TCP 连接。

1. Consumer

消息的消费者，表示一个从消息队列中取得消息的客户端应用程序。

1. Virtual Host

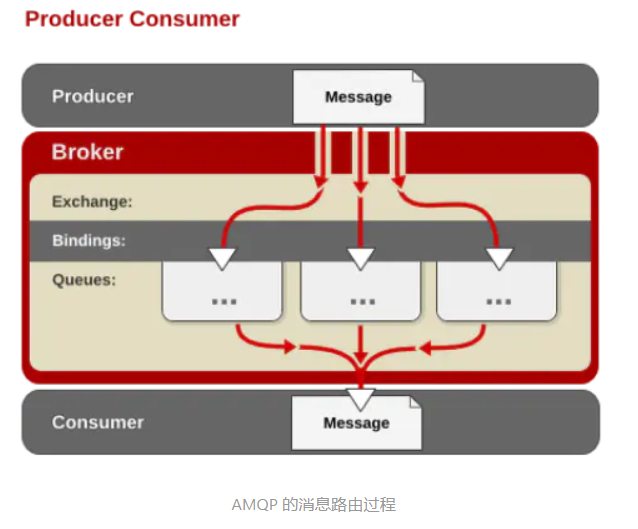
虚拟主机，表示一批交换器、消息队列和相关对象。虚拟主机是共享相同的身份认证和加密环境的独立服务器域。每个 vhost 本质上就是一个 mini 版的 RabbitMQ 服务器，拥有自己的队列、交换器、绑定和权限机制。vhost 是 AMQP 概念的基础，必须在连接时指定，RabbitMQ 默认的 vhost 是 / 。

1. Broker

表示消息队列服务器实体。

### AMQP 中的消息路由

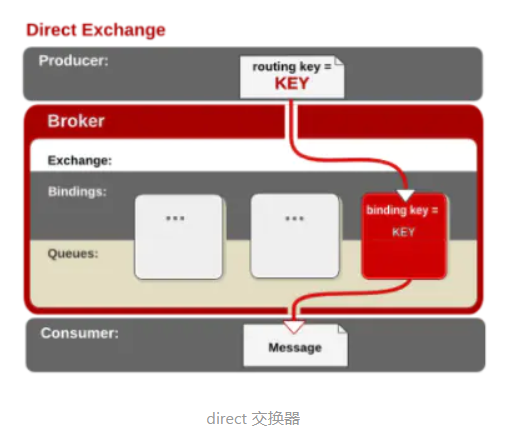
AMQP 中消息的路由过程和 Java 开发者熟悉的 JMS 存在一些差别，AMQP 中增加了 Exchange 和 Binding 的角色。生产者把消息发布到 Exchange 上，消息最终到达队列并被消费者接收，而 Binding 决定交换器的消息应该发送到那个队列。



Exchange 类型

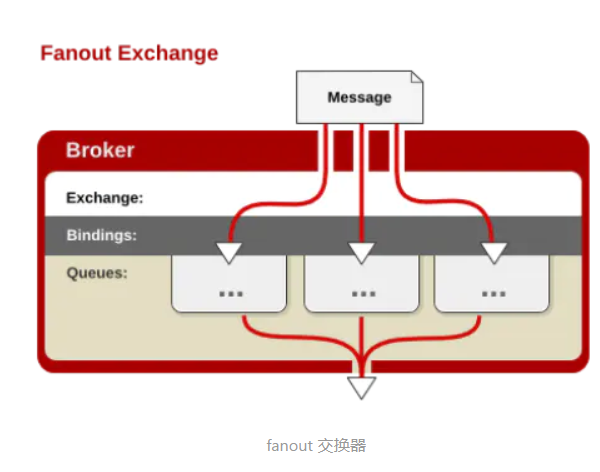
Exchange分发消息时根据类型的不同分发策略有区别，目前共四种类型：direct、fanout、topic、headers 。headers 匹配 AMQP 消息的 header 而不是路由键，此外 headers 交换器和 direct 交换器完全一致，但性能差很多，目前几乎用不到了，所以直接看另外三种类型：

1. Direct

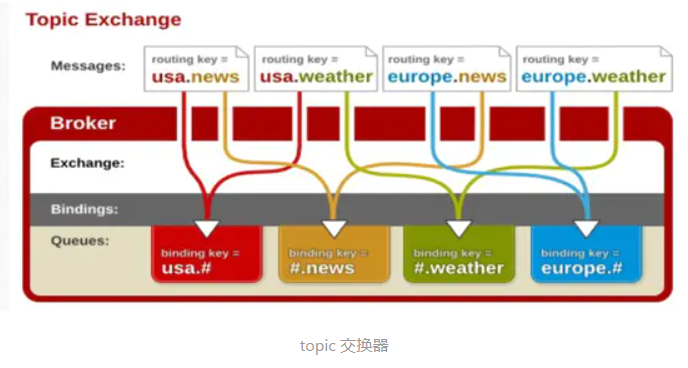


消息中的路由键（routing key）如果和 Binding 中的 binding key 一致， 交换器就将消息发到对应的队列中。路由键与队列名完全匹配，如果一个队列绑定到交换机要求路由键为“dog”，则只转发 routing key 标记为“dog”的消息，不会转发“dog.puppy”，也不会转发“dog.guard”等等。它是完全匹配、单播的模式。

1. Fanout



每个发到 fanout 类型交换器的消息都会分到所有绑定的队列上去。fanout 交换器不处理路由键，只是简单的将队列绑定到交换器上，每个发送到交换器的消息都会被转发到与该交换器绑定的所有队列上。很像子网广播，每台子网内的主机都获得了一份复制的消息。fanout 类型转发消息是最快的。

1. topic  
   

topic 交换器通过模式匹配分配消息的路由键属性，将路由键和某个模式进行匹配，此时队列需要绑定到一个模式上。它将路由键和绑定键的字符串切分成单词，这些单词之间用点隔开。它同样也会识别两个通配符：符号“#”和符号“*”。#匹配0个或多个单词，*匹配不多不少一个单词。

# RabbitMQ 安装

课程学习，demo测试。

# 正则表达式学习

1. + 号代表前面的字符必须至少出现一次（1次或多次）

runoo+b，可以匹配 runoob、runooob、runoooooob 等

1. \* 号代表前面的字符可以不出现，也可以出现一次或者多次（0次、或1次、或多次）。

runoo\*b，可以匹配 runob、runoob、runoooooob 等

1. ? 问号代表前面的字符最多只可以出现一次（0次、或1次）

colou?r 可以匹配 color 或者 colour

1. 匹配 [...] 中的所有字符

例如 [aeiou] 匹配字符串 "google runoob taobao" 中所有的 e o u a 字母。

1. 匹配除了 [...] 中字符的所有字符

例如 [^aeiou] 匹配字符串 "google runoob taobao" 中除了 e o u a 字母的所有字母。

1. [A-Z] 表示一个区间，匹配所有大写字母，[a-z] 表示所有小写字母。
2. 匹配除换行符（\n、\r）之外的任何单个字符，相等于 [^\n\r]。

### 参考链接：

<https://www.codejiaonang.com/#/course/Java_Basic_chapter6/0/0>

https://regexr-cn.com/

# java 中的泛型

<https://blog.csdn.net/Beyondczn/article/details/107093693>

泛型的本质是为了参数化类型（在不创建新的类型的情况下，通过泛型指定的不同类型来控制形参具体限制的类型）。也就是说在泛型使用过程中，操作的数据类型被指定为一个参数，这种参数类型可以用在类、接口和方法中，分别被称为泛型类、泛型接口、泛型方法。

# 泛型类型<T>不能用于静态方法

## 泛型只在编译阶段有效

Java中的泛型，只在编译阶段有效。在编译过程中，正确检验泛型结果后，会将泛型的相关信息擦出，并且在对象进入和离开方法的边界处添加类型检查和类型转换的方法。也就是说，泛型信息不会进入到运行时阶段。

## **? extends T**

? extends T 描述了通配符上界, 即具体的泛型参数需要满足条件: 泛型参数必须是 T 类型或它的子类,

## **? super T**

? super T 描述了通配符下界, 即具体的泛型参数需要满足条件: 泛型参数必须是 T 类型或它的父类

1. List<? super Integer> l1

正确的理解: ? super Integer 限定的是泛型参数. 令 l1 的泛型参数是 T, 则 T 是 Integer 或 Integer 的父类, 因此 Integer 或 Integer 的子类的对象就可以添加到 l1 中.

错误的理解: ? super Integer限定的是插入的元素的类型, 因此只要是 Integer 或 Integer 的父类的对象都可以插入 l1 中

1. List<? extends Integer> l2:

正确的理解: ? extends Integer 限定的是泛型参数. 令 l2 的泛型参数是 T, 则 T 是 Integer 或 Integer 的子类, 进而我们就不能找到一个类 X, 使得 X 是泛型参数 T 的子类, 因此我们就不可以向 l2 中添加元素. 不过由于我们知道了泛型参数 T 是 Integer 或 Integer 的子类这一点, 因此我们就可以从 l2 中读取到元素(取到的元素类型是 Integer 或 Integer 的子类), 并可以存放到 Integer 中.

错误的理解: ? extends Integer 限定的是插入元素的类型, 因此只要是 Integer 或 Integer 的子类的对象都可以插入 l2 中

## **使用场景**

PECE 原则: Producer Extends, Consumer Super

Producer extends: 如果我们需要一个 List 提供类型为 T 的数据(即希望从 List 中读取 T 类型的数据), 那么我们需要使用 ****? extends T****, 例如 List<? extends Integer>. 但是我们不能向这个 List 添加数据.

Consumer Super: 如果我们需要一个 List 来消费 T 类型的数据(即希望将 T 类型的数据写入 List 中), 那么我们需要使用 ? super T, 例如 List<? super Integer>. 但是这个 List 不能保证从它读取的数据的类型.

如果我们既希望读取, 也希望写入, 那么我们就必须明确地声明泛型参数的类型, 例如 List<Integer>.

例子:

public **class** **Collections** {

public **static** <T> **void** copy(List<? **super** T> dest, List<? **extends** T> src)

{

**for** (int i=0; i<src.size(); i++)

dest.**set**(i,src.**get**(i));

}

}

上面的例子是一个拷贝数据的代码, src 是 List<? extends T> 类型的, 因此它可以读取出 T 类型的数据(读取的数据类型是 T 或是 T 的子类, 但是我们不能确切的知道它是什么类型, 唯一能确定的是读取的类型 is instance of T), , dest 是 List<? super T> 类型的, 因此它可以写入 T 类型或其子类的数据.

## 参考地址：

https://segmentfault.com/a/1190000008423240

# Thymeleaf

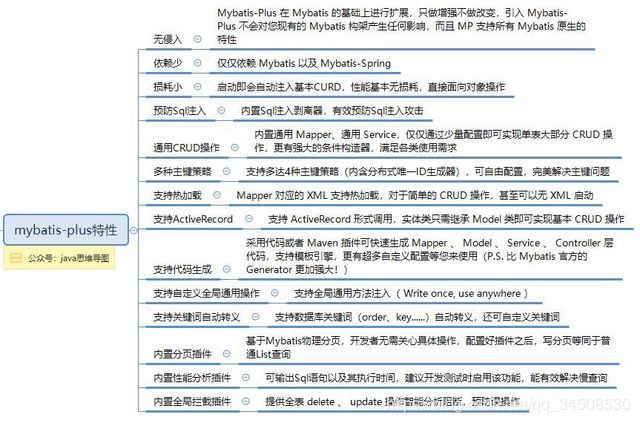
# MYBATIS\_PLUS

官网: https://baomidou.com/guide/#%E6%A1%86%E6%9E%B6%E7%BB%93%E6%9E%84

## Mybatis 和 Mybatis Plus 的区别

[MyBatis-Plus (opens new window)](https://github.com/baomidou/mybatis-plus" \t "https://baomidou.com/guide/_blank)（简称 MP）是一个 [MyBatis (opens new window)](http://www.mybatis.org/mybatis-3/" \t "https://baomidou.com/guide/_blank)的增强工具，在 MyBatis 的基础上只做增强不做改变，为简化开发、提高效率而生。





Mybis-Plus提供了一些很有意思的插件，比如SQL性能监控、乐观锁、执行分析等

使用Mybatis时会发现，每当要写一个业务逻辑的时候都要在DAO层写一个方法，再对应一个SQL，即使是简单的条件查询、即使仅仅改变了一个条件都要在DAO层新增一个方法，针对这个问题，MP就提供了一个很好的解决方案，之后我会进行介绍。另外，MP的代码生成器也是一个很有意思的东西，它可以让我们避免许多重复性的工作，下面我将介绍如何在你的项目中集成MP。

### 参考链接：

https://www.jb51.net/article/194512.htm