**RabbitMQ学习笔记**

1. **概念：**
2. rabbitmq是开源的AMQP（高级消息队列协议），服务端使用erlang语言编写，支持多种语言的客户端；具有消息持久化、可扩展性等特点；
3. **消息中间件：**

消息队列提供的是异步通信：当消息生产者发送消息时，不需要等待消费者接收消息后才返回，而是发送完消息后立刻返回。消息队列对该消息进行发送，当网络通信出现故障时，该消息由消息队列暂存，等待网络通信恢复时再进行发送；

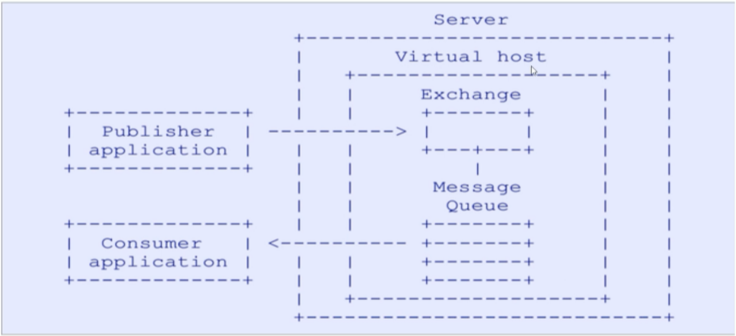
**作用：提升并发性、对各个服务进行解耦**（因为消息的接收者只需要进行订阅/取消操作，就可以实现消息的接收与否）

1. **MQ常见的应用场景：**
2. 异步通信：
3. 流量削峰：秒杀活动等等；
4. 日志处理：
5. 应用解耦：添加服务之间的消息传输，不需要考虑接收、发送因素，只需要订阅相应消息队列即可。
6. **常见的rabbitmq端口：**
7. 4369：是erlang端口/节点名称映射程序，用来跟踪节点名称监听地址，在集群中起到一个类似DNS的作用。
8. **5672/5671：**AMQP 0-9-1和1.0**客户端端口**，没有使用SSL和使用SSL的端口。
9. 25672：用于rabbitMQ节点间和CLI工具通信，配合4369使用。
10. **15672：HTTP\_API端口，具有管理界面**，管理员用户才能访问，用于管理rabbitmq，**需要启用management插件**。
11. 61613/61614：当STOMP插件启用的时候打开，作为STOMP客户端端口（根据是否使用TLS选择）
12. 1883/8883：当MQTT插件启用的时候打开，作为MQTT看客户端端口（根据是否使用TLS选择）
13. 15674：基于websocket的STOMP客户端端口（当插件web STOMP启用的时候打开）
14. 15674：基于websocket的MQTT客户端端口（当插件web MQTT启用的时候打开）
15. **AMQP概念：**
16. **Server：**接收客户端的连接，实现AMQP实体服务。
17. **Connection：**连接，**应用程序与Server的网络连接，TCP连接**。
18. **Channel：**信道，**消息读写等操作在信道中进行**。**客户端可以建立多个信道，每个信道代表一个会话任务，使用TCP建立连接后，通过创建多个信道来保持消息的传输，防止多次创建连接占用资源、产生延时**。
19. **Message：**消息，应用程序和服务器之间传送的数据，由Properties和Body组成。

* Properties：为外包装，对消息进行修饰，eg：消息的优先级、延迟等高级特性；
* Body：是消息体内容。

1. **Virtual Host：**虚拟主机，用于逻辑隔离。**一个虚拟主机里面可以有若干个Exchange和Queue，同一个虚拟主机里面不能有相同名称的Exchange或Queue**。
2. **Exchange：**交换器，接收消息，**按照路由规则将消息路由到一个或者多个队列**。如果路由不到，或者返回给生产者，或者直接丢弃。RabbitMQ常用的交换器常用类型有direct、topic、fanout、headers四种，后面详细介绍。
3. **Binding：**绑定，**捆绑交换器和消息队列**，绑定中可以包含一个或者多个RoutingKey。
4. **RoutingKey：**路由键，生产者将消息发送给交换器的时候，会发送一个RoutingKey，用来**指定路由规则，这样交换器就知道把消息发送到哪个队列**。路由键通常为一个“.”分割的字符串，例如“com.rabbitmq”。
5. **Queue：**消息队列，**用来保存消息**，供消费者消费。

1. **AMQP模型：**

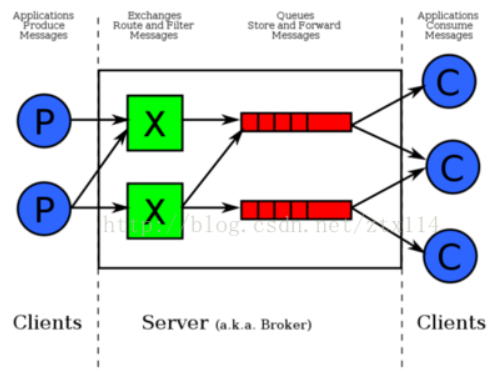


1. 生产者先接到server，开启信道，并声明交换机和消息队列，通过路由键将消息队列和交换机绑定；
2. 消费者也需要和sever建立连接，开启信道；
3. 生产者就可以将消息发送到server中的virtual host中，virtual host的交换机根据消息中的路由键选择路由规则，将消息发送到不同的消息队列中，由此消费者根据订阅的消息队列就可以取到对应的消息。
4. **RabbitMQ的核心概念：**

producer发送消息前需要建立连接connection，在connection中可以存在多个channel。producer发送的消息中有一个路由键routingKey，将这个消息发送到交换机exchange，然后exchange再根据routingKey将消息发送到对应的队列queue中（exchange和queue是通过routing key路由键绑定在一起的）。

1. **RabbitMQ的模型：**

RabbitMQ是基于AMQP实现的，因此基本和AMQP模型一样。

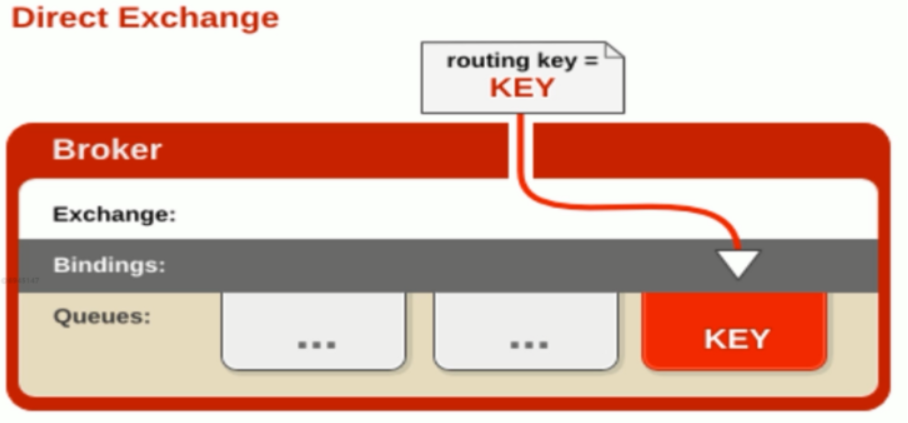


1. **RabbitMQ的交换机：**

**共有4中交换机：direct、topic、fanout、headers**

1. **direct：**

该交换器将所有发送到该交换器的消息被转发到**bingding绑定的RoutingKey指定的队列中**，即:路由到BindingKey和RoutingKey完全匹配的队列中。



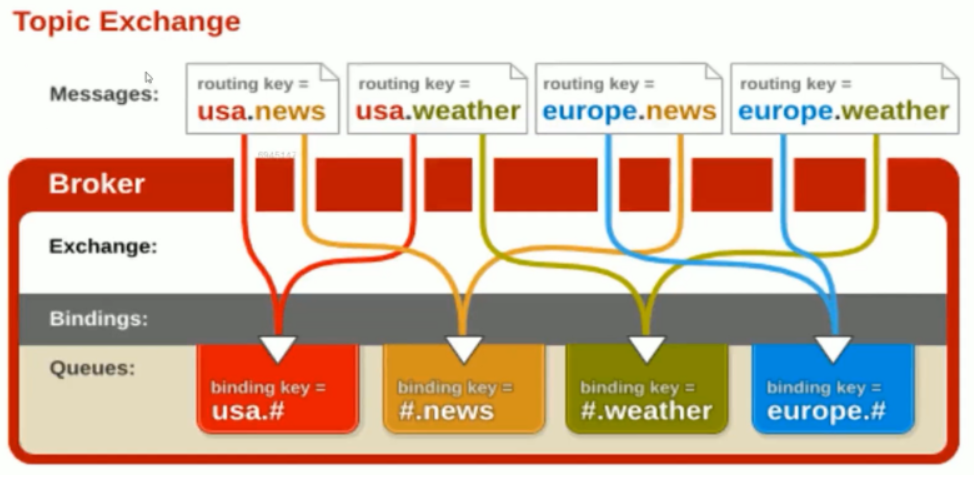
1. **topic：**

该交换器将所有消息转发到RoutingKey中指定的Topic的队列中。

通过RoutingKey和某Topic进行**模糊匹配**，其中**“\*”用来匹配一个词，“#”用于匹配一个或者多个词**。

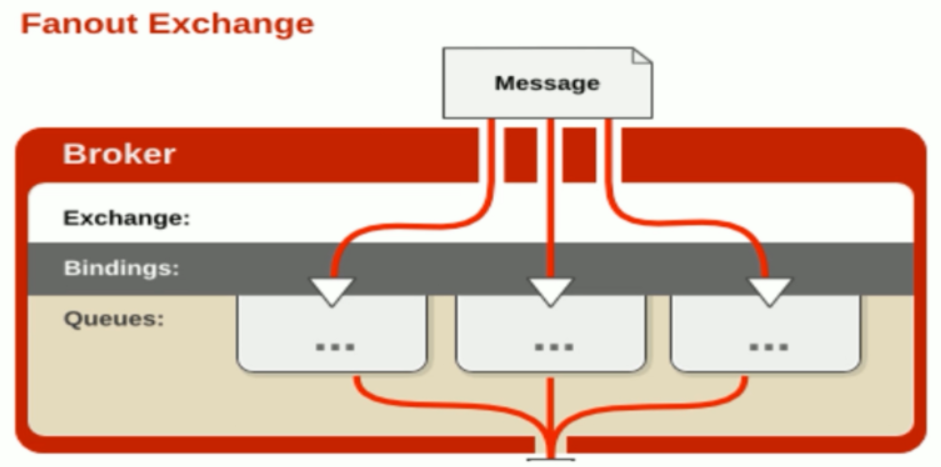
**eg：（以 . 为界限划分单词）**

1. “com.#”：能匹配到“com.rabbitmq.oa”和“com.rabbitmq”
2. “login.\*”：能匹配到“login.rabbitmq”



1. **fanout：**

该交换机不处理路由键，会把所有发送到交换器的消息路由到**所有绑定的队列**中。优点是转发消息最快，性能最好。



1. **headers：（不常用）**

该交换器不依赖路由规则来路由消息，而是根据消息内容中的headers属性进行匹配。headers类型交换器性能差，实际中并不常用。

1. **RabbitMQ的工作模式：**

**p：producer，生产者——消息的发送者；**

**c：consumer，消费者——消息的接收者；**

**x：exchange，交换机——消息转发到特定队列**

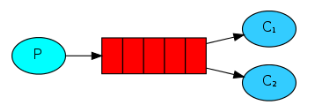
1. **简单队列：**

生产者和消费者是一对一的关系，生产者将消息发给消息队列后，消费者从消息队列中获取消息；



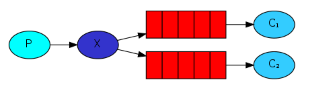
1. **工作队列：**

生产者和消费者是一对多的关系：一个生产者，多个消费者，每个消费者获取到的消息唯一。



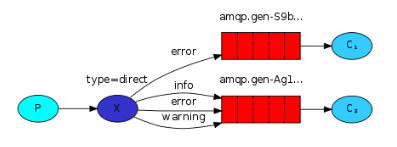
1. **发布订阅：**

一个生产者发送的消息会被多个消费者获取：消费者通过订阅的方式获取所订阅队列中的消息。



1. **路由模式：**

发送消息到交换机并且要指定routing key，消费者将队列绑定到交换机时需要指定routing key。交换机根据routing key将生产者的消息发送到指定的队列中，消费者才能获取到特定队列中的消息。

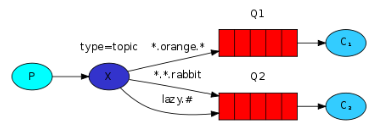


1. **主题模式：**

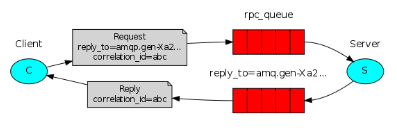
将路由键和某模式进行匹配，此时队列需要绑定在一个模式上。

1. “#”：匹配一个词或多个词；
2. “\*”：只匹配一个词。

**注意：**这里的**单词匹配**指的是**以 . 为分割**， .和.之间算是一个单词。



1. **RPC模式：**



1. **发布认证：**



1. **SpringBoot中使用RabbitMQ：**

（查看D:\Files\Code\JAVA\SpringBoot\RabbitMQTemplate 工程模板）

1. **Docker安装rabbitmq：**
2. rabbitmq**（带web界面）**：docker pull **rabbitmq:management**
3. rabbitmq**（没有web界面）**：docker pull **rabbitmq**
4. **启动rabbitmq命令：**

设定容器别名：Myrabbitmq

设定用户名：admin

设定密码：admin

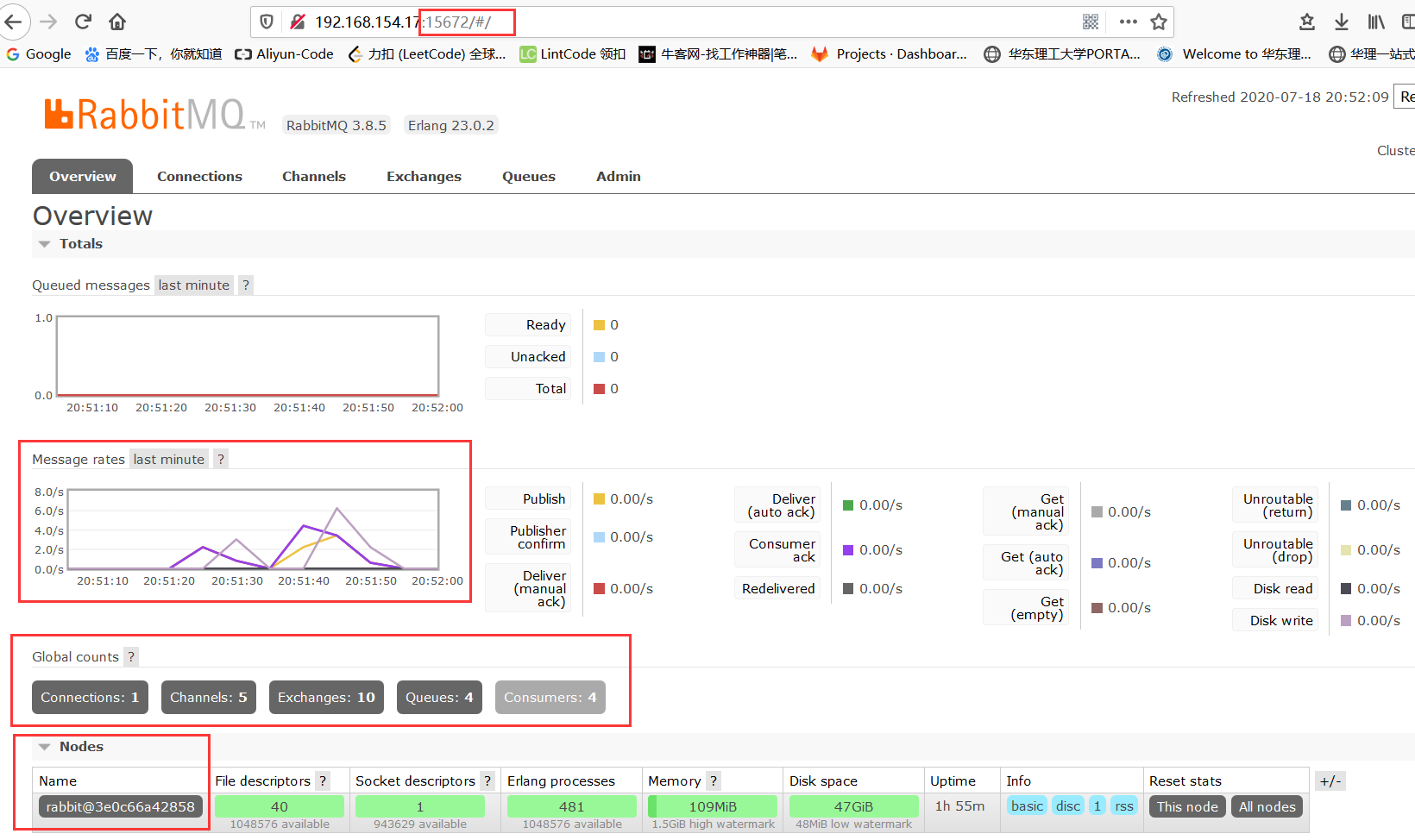
设定客户端端口（springboot中.yml配置的端口）：5672

设定浏览器登陆web端口：15672

docker run -dit --name Myrabbitmq

-e RABBITMQ\_DEFAULT\_USER=admin -e RABBITMQ\_DEFAULT\_PASS=admin -p 15672:15672 -p 5672:5672 **rabbitmq:management**

1. **使用浏览器的web界面：端口15672**

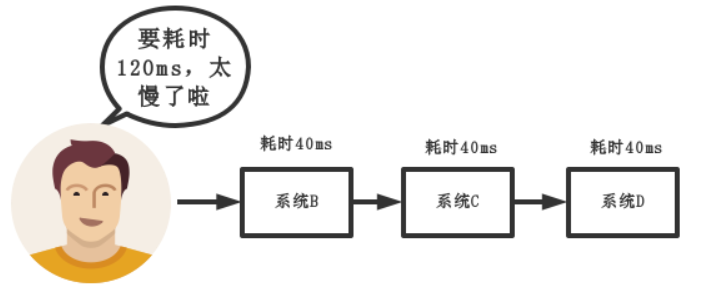


1. **为什么使用MQ：（MQ作用）**
2. **异步：**

串行变成并行，加快响应速度。

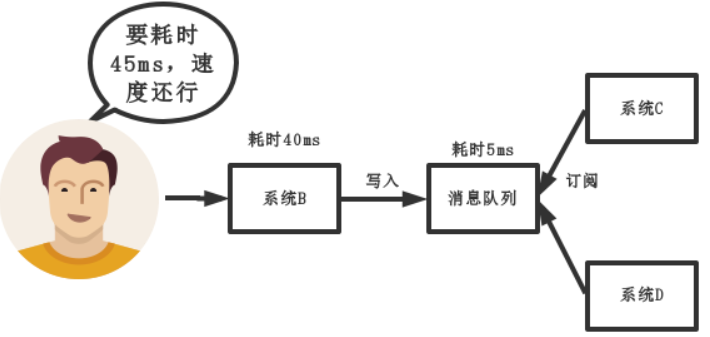
1. **同步情况：**

假设，有消息需要发送到3个系统上，使用串行，就需要依次给3个系统发送完消息后，才能够获取最终的返回结果。



1. **并行情况：**

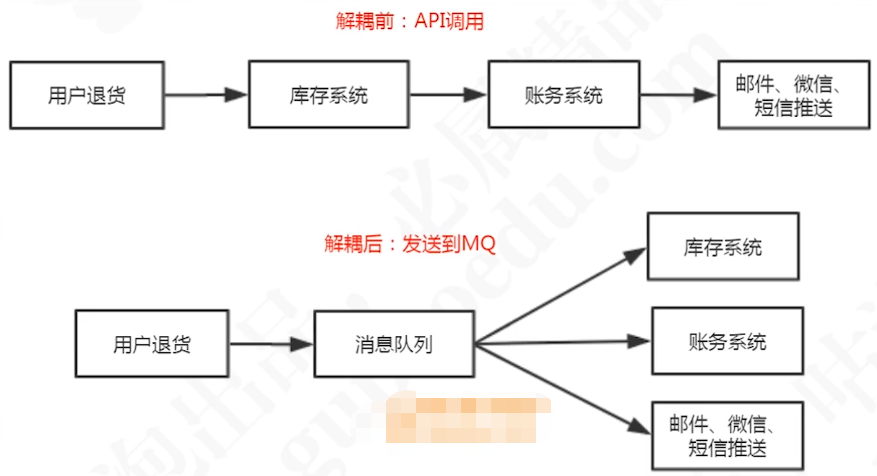
只需要将消息放到消息队列，其他的系统订阅之后就可以同时获得这个消息，进行并行处理。节约运行时间，获得返回结果的实践也就缩短了。



1. **解耦：**

把原来的串行调用API，改成并行调用。

1. **使用TCP通信：**如果使用线程池来实现API的并行调用，这样需要去管理多线程，而且每次新增了新的消费者还需要修改生产者的代码——增加消费者的信息、IP、端口。
2. **使用MQ：**只需要关注消息发送和接收，不用管理线程。生产者只需要记录MQ的主机IP，不需要关注其他消费者的IP。新增消费者时，只需要让消费者去连接MQ的消息队列即可获取消息。

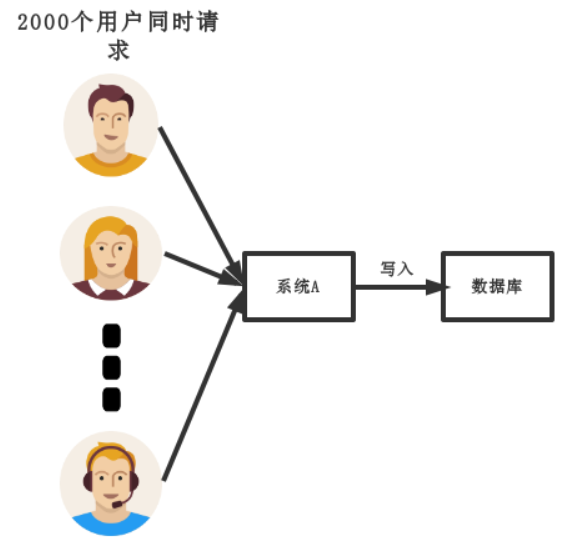


1. **削锋：**

基于MQ先进先出的特性。控制请求进入消息队列的数量，进而达到控制流量的作用。常用于流量削峰。

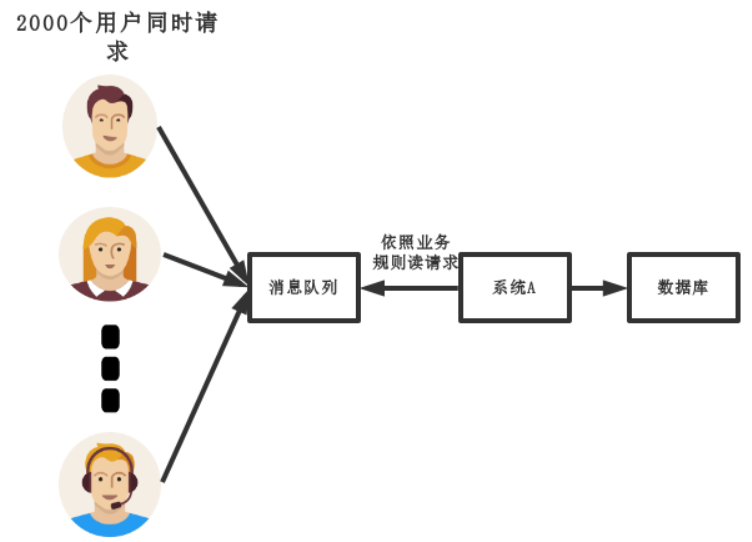
1. **传统并发请求模式：**

同时有2000个请求发送过来，直接输入到数据库，可能回导致数据库崩溃。

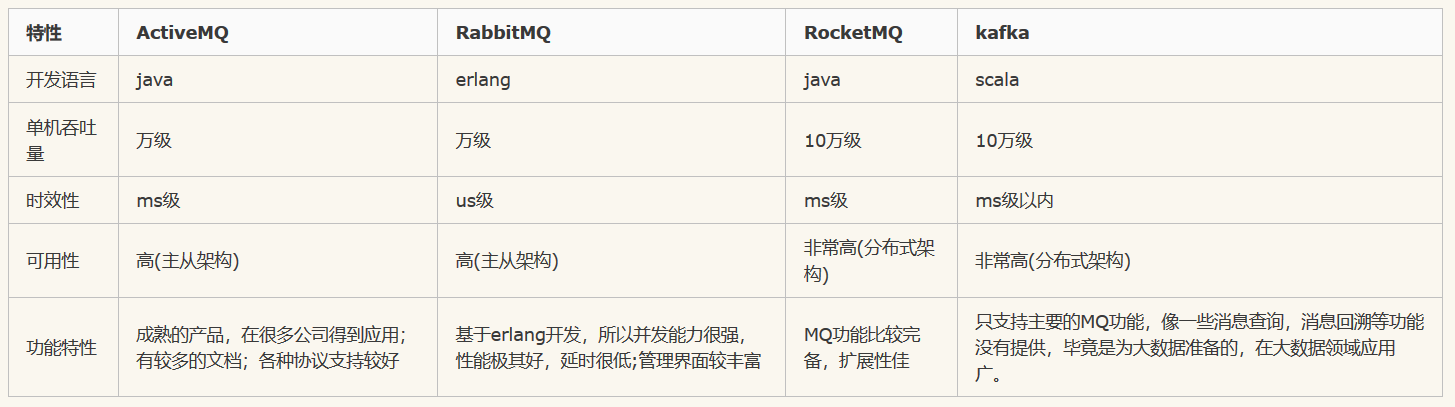


1. **使用MQ：**

使用消息队列来存放请求，按照先进先出的原则，所有的请求都只能逐个访问数据库，不会存在由于请求过多的因素，导致数据库崩溃的情况。



1. **不同MQ的特性：**



1. **死信队列：**
2. **什么是死信队列：**

死信队列是用于当消息队列中的数据由于：**消息被拒、消息超时（TTL）、消息队列达到最大长度**时，不能放到原来的消息队里中。

此时为了保证消息不会丢失，需要将这个消息经由**死信交换机**（**DLX**-dead letter exchange）放到**死信队列**中，最终由**死信消费者**去处理这些信息。

1. **死信队列的使用流程：**
2. 创建死信交换机：
3. 创建死信队列：
4. 绑定死信队列、死信交换机

