匠人学院
jiangren.com.au

Java Spring Boot
Message Queue

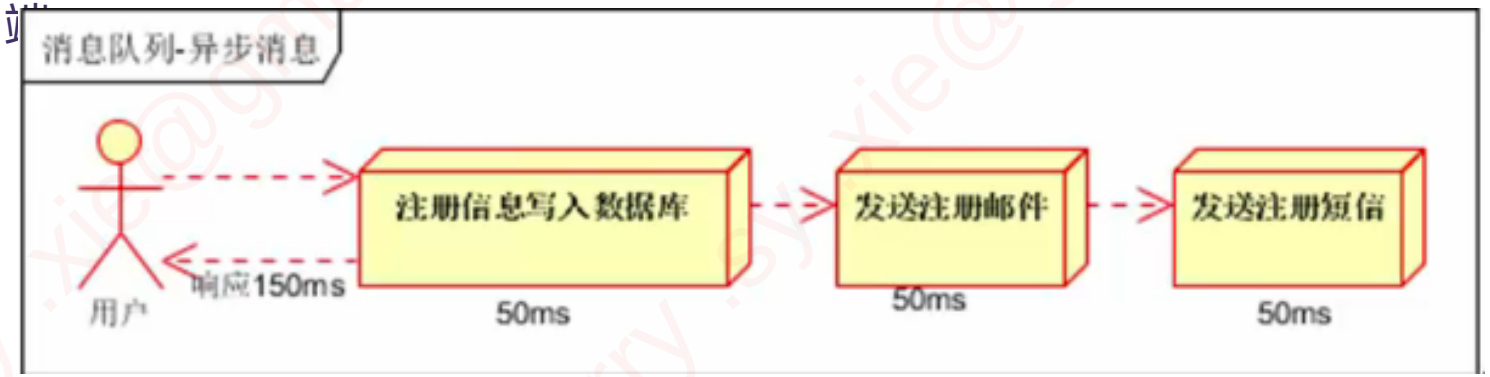
Message Queue

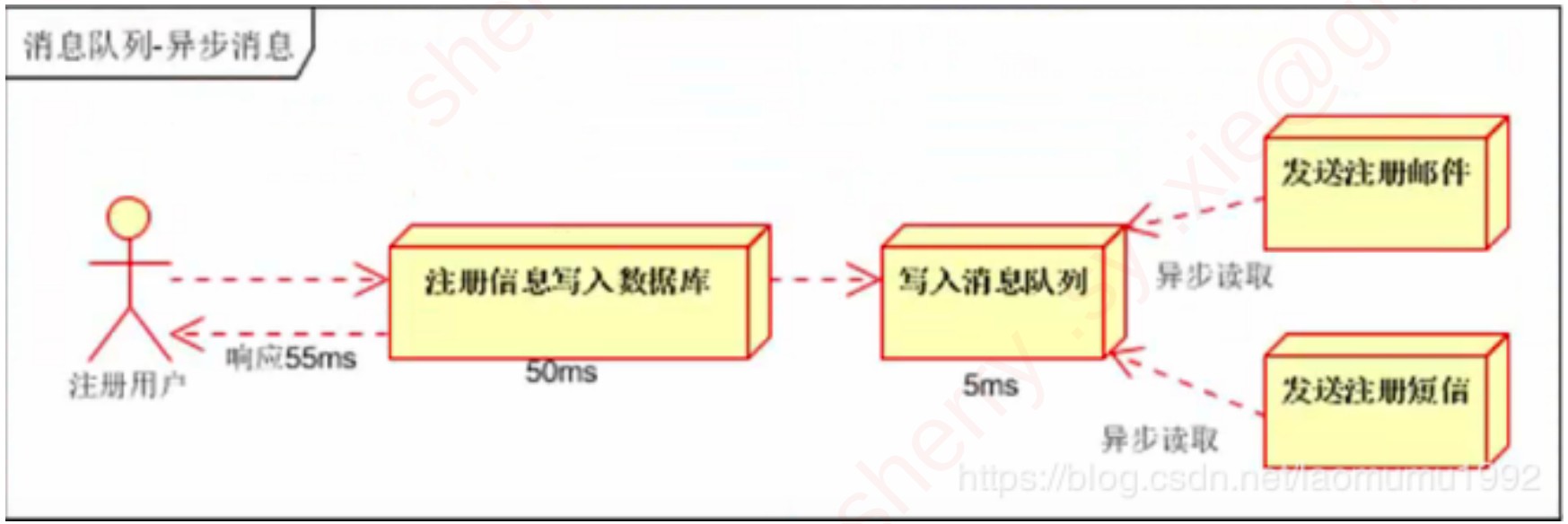
- 消息队列中间件，队列：FIFO
- 为什么要使用？
- 关键点：异步 + 多线程（多进程）
- 资源有限：可以开启的线程数量
- 性能有限：I/O（网络，文件，数据库等）
- 系统扩展：

1) 纵向扩展，即购买更好的机器，更多的磁盘、更多的内存等等；
2) 横向扩展，即购买更多的机器组成集群。

Message Queue 场景——异步处理

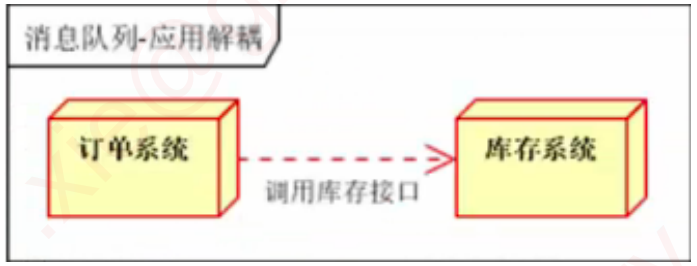
场景说明：用户注册后需要发送注册邮件和注册短信。同步做法：将注册的信息写入数据库成功后，发送注册邮件，再发送注册短信。以上三个任务全部完成后返回给客户端

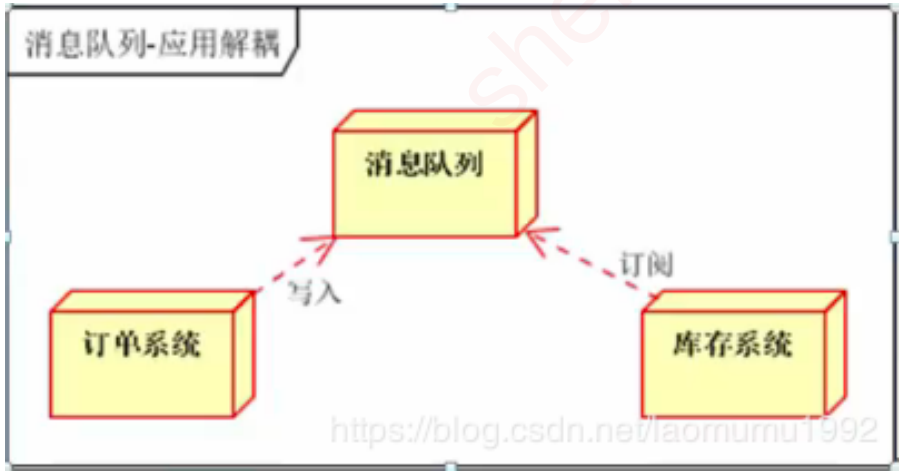




Message Queue 场景——系统解耦

用户下单后，订单系统需要通知库存系统。同步做法是，订单系统调用库存系统的接口。 假如库存系统无法访问，则订单减库存将失败，从而导致下单失败。两个系统耦合。





Message Queue 场景——流量削峰

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要在应用前端加入消息队列。

- 可以控制活动人数
- 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。假如消息队列的长度超过最大数量，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消息，再做后续处理。

Message Queue 的特性

消息队列中间件是分布式系统中重要的组件，主要解决应用耦合、异步消息、流量削锋等问题，实现高性能、高可用、可伸缩和最终一致性架构，是大型分布式系统不可缺少的中间件。

- 异步性，将耗时的同步操作通过以发送消息的方式进行了异步化处理，减少了同步等待的时间。
- 松耦合，消息队列减少了服务之间的耦合性，不同的服务可以通过消息队列进行通信，而不用关心彼此的实现细节，只要定义好消息的格式就行。
- 分布式，通过对消费者的横向扩展，降低了消息队列阻塞的风险，以及单个消费者产生单点故障的可能性（当然消息队列本身也可以做成分布式集群）。
- 可靠性，消息队列一般会把接收到的消息存储到本地硬盘上（当消息被处理完之后，存储信息根据不同的消息队列实现，有可能将其删除），这样即使应用挂掉或者消息队列本身挂掉，消息也能够重新加载。

AMQP

- 目前在生产环境中使用较多的消息队列有 ActiveMQ、RabbitMQ、ZeroM、MetaMQ、RocketMQ ，AWS SQS等。

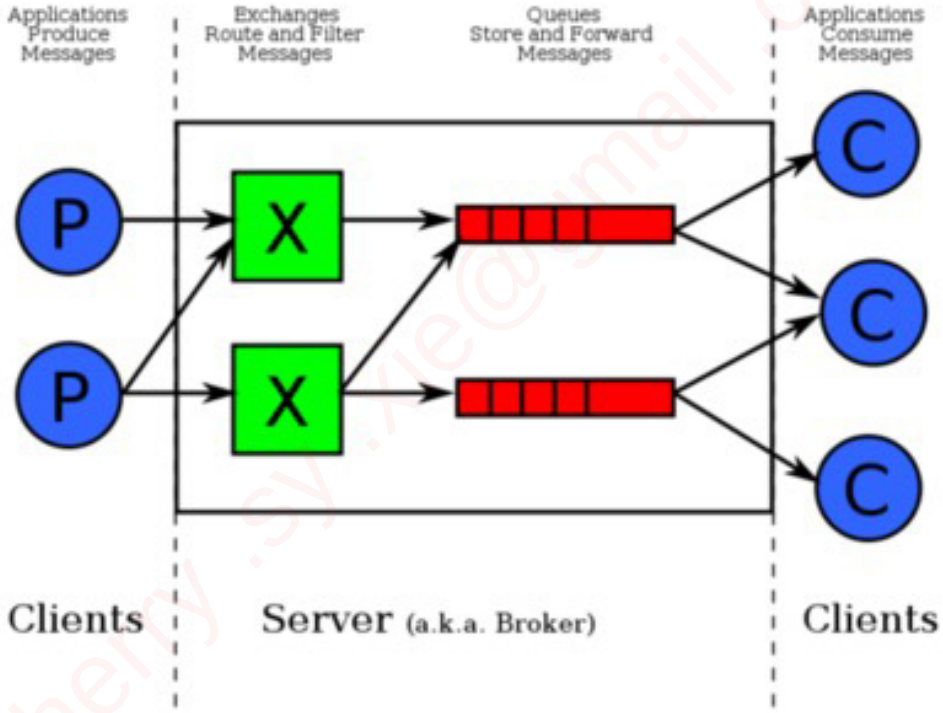
- RabbitMQ 是一个开源的 AMQP 实现，服务器端用 Erlang 语言编写，支持多种客户端，如 Python、Ruby、.NET、Java、JMS、C、PHP、ActionScript、XMPP、STOMP 等，支持 AJAX。用于在分布式系统中存储转发消息，在易用性、扩展性、高可用性等方面表现不俗。

- AMQP（Advanced Message Queuing Protocol）高级消息队列协议，是应用层协议的一个开放标准，为面向消息的中间件设计。消息中间件主要用于组件之间的解耦，消息的发送者无需知道消息使用者的存在，反之亦然。
- AMQP 的主要特征是面向消息、队列、路由（包括点对点和发布/订阅）、可靠性、安全。

相关概念

队列服务有三个概念：发消息者、队列、收消息者。RabbitMQ 在这个基本概念之上，多做了一层抽象，在发消息者和队列之间加入了交换器（Exchange）。这样发消息者和队列就没有直接联系，转而变成发消息者把消息给交换器，交换器根据调度策略再把消息再给队列。

- 左侧 P 代表生产者，也就是往 RabbitMQ 发消息的程序。
- 中间即是 RabbitMQ，其中包括了交换机和队列。
- 右侧 C 代表消费者，也就是往 RabbitMQ 拿消息的程序。



相关概念

- 虚拟主机：一个虚拟主机持有一组交换机、队列和绑定。为什么需要多个虚拟主机呢？RabbitMQ 当中，用户只能在虚拟主机的粒度进行权限控制。因此，如果需要禁止 A 组访问 B 组的交换机/队列/绑定，必须为 A 和 B 分别创建一个虚拟主机。每一个 RabbitMQ 服务器都有一个默认的虚拟主机“/”。
- 交换机：Exchange 用于转发消息，但是它不会做存储，如果没有 Queue bind 到 Exchange 的话，它会直接丢弃掉 Producer 发送过来的消息。这里有一个比较重要的概念：**路由键** 。消息到交换机的时候，交换机会转发到对应的队列中，那么究竟转发到哪个队列，就要根据该路由键的情况。
- 绑定：也就是交换机需要和队列相绑定，这其中如上图所示，是多对多的关系。

交换机

交换机的功能主要是接收消息并且转发到绑定的队列，交换机不存储消息，在启用 ack 模式后，交换机找不到队列，会返回错误。交换机有四种类型：Direct、Topic、Headers 和 Fanout 。

- Direct：该类型的行为是“先匹配，再投送”，即在绑定时设定一个 **routing key**，消息的 **routing key** 匹配时，才会被交换器投送到绑定的队列中去。
- Topic：按规则转发消息（最灵活）。
- Headers：设置 header attribute 参数类型的交换机。
- Fanout：转发消息到所有绑定队列。

交换机——Direct Exchange

Direct Exchange 是 RabbitMQ 默认的交换机模式，也是最简单的模式，根据 key 全文匹配去寻找队列。

第一个 X - Q1 就有一个 binding key，名字为 orange；X - Q2 就有 两 个 binding key，名字为 black 和 green。当消息中的路由键和这个 binding key 对应上的时候，那么就知道了该消息去到哪一个队列中。

交换机——Topic Exchange

Topic Exchange 转发消息主要是根据通配符。在这种交换机下，队列和交换机的绑定会定义一种路由模式，那么，通配符就要在这种路由模式和路由键之间匹配后交换机才能转发消息。

- 路由键必须是一串字符，用句号（.）隔开，如 agreements.us，或者 agreements.eu.stockholm 等。
- 路由模式必须包含一个星号（*），主要用于匹配路由键指定位置的一个单词。例如，一个路由模式是这样子：agreements..b.*，那么就只能匹配路由键是这样子的：第一个单词是 agreements，第四个单词是 b。井号（#）就表示相当于一个或者多个单词。例如，一个匹配模式是agreements.eu.berlin.#，那么，以 agreements.eu.berlin 开头的路由键都是可以的。

具体代码发送的时候还是一样，第一个参数表示交换机，第二个参数表示 routing key，第三个参数即消息。如下：

topic 和 direct 类似，只是匹配上支持了“模式”，在“点分”的 routing_key 形式中，可以使用两个通配符：

- *表示一个词。
- #表示零个或多个词。

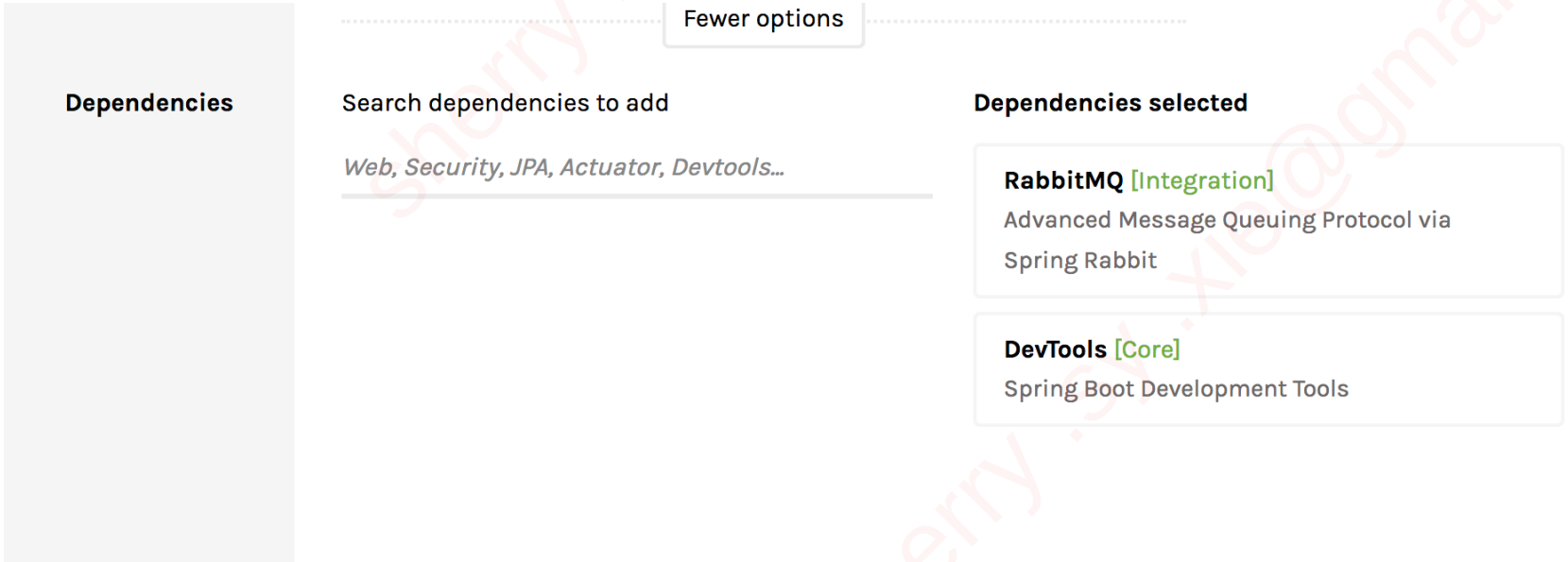
交换机——Headers & Fanout Exchange

- Headers 也是根据规则匹配，相较于 direct 和 topic 固定地使用 routing_key，Headers 则是一个自定义匹配规则的类型。在队列与交换器绑定时，会设定一组键值对规则，消息中也包括一组键值对(headers 属性)，当这些键值对有一对或全部匹配时，消息被投送到对应队列。
- Fanout Exchange 消息广播的模式，不管路由键或者是路由模式，**会把消息发给绑定给它的全部队列**，如果配置了routing_key会被忽略。

Spring Boot 集成RabbitMQ

- Spring Boot 集成 RabbitMQ 非常简单，仅需非常少的配置就可使用，Spring Boot 提供了 spring-boot-starter-amqp 组件对MQ消息支持。

https://start.spring.io/



发送简单消息

- Spring Boot 集成 RabbitMQ 非常简单，仅需非常少的配置就可使用，Spring Boot 提供了 spring-boot-starter-amqp 组件对MQ消息支持。

```yaml
version: '2.3'

services:
  rabbitmq:
    image: rabbitmq:management
    environment:
      RABBITMQ_DEFAULT_USER: "rabbitmq"
      RABBITMQ_DEFAULT_PASS: "rabbitmq"
    ports:
      - "5672:5672"
      - "15672:15672"
```

docker-compose.yml

启动rabbitmq：

- docker-compose up rabbitmq
- **http://127.0.0.1:15672/ 管理界面**

```
package com.fiona.queue.config;

import org.springframework.amqp.core.Queue;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class RabbitConfiguration {
  @Bean
  public Queue Queue() {
    return new Queue("hello");
  }
}
```

application.properties

```
spring.rabbitmq.host=127.0.0.1
spring.rabbitmq.port=5672
spring.rabbitmq.username=rabbitmq
spring.rabbitmq.password=rabbitmq
```

发送简单消息

```
package com.fiona.queue;
…
@Component
public class HelloSender {
  @Autowired
  private AmqpTemplate rabbitTemplate;

  public void send() {
    String context = "hello " + new Date();
    System.out.println("Sender : " + context);
    this.rabbitTemplate.convertAndSend("hello", context);
  }
}
```

```
package com.fiona.queue;
…
@Component
@RabbitListener(queues = "hello")
public class HelloReceiver {

  @RabbitHandler
  public void process(String hello) {
    System.out.println("Receiver  : " + hello);
  }

}
```

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class RabbitMqHelloTest {
  @Autowired
  private HelloSender helloSender;

  @Test
  public void hello() throws Exception {
    helloSender.send();
    Thread.sleep(1000l);
  }
}
```

高级使用——发送对象

- Spring Boot 已经完美的支持对象的发送和接收，不需要格外的配置。

```
public class User implements Serializable {
  private String name;

  private String pass;

  public String getName() {
    return name;
  }
  public void setName(String name) {
    this.name = name;
  }

  public String getPass() {
    return pass;
  }
  public void setPass(String pass) {
    this.pass = pass;
  }
}
```

高级使用—— Fanout Exchange

Fanout 就是我们熟悉的广播模式或者订阅模式，给 Fanout 交换机发送消息，绑定了这个交换机的所有队列都收到这个消息。

作业：自己试试写一个Fanout exchange的例子。

匠人学院
jiangren.com.au

Q & A