Spring Boot ve Advanced Message Queueing Protocol /RabbitMQ

AMQP Nedir?

- Farklı mimari ve teknolojilerle geliştirilen uygulamaların birbirleri ile **haberleşme**lerini destekleyen **açık ve cross-platform bir spesifikasyon**dur
- Farklı sistemler birbirleri ile **point-to-point** ve **publish-subscribe** örüntüleri üzerinden asenkron biçimde haberleşebilirler
- Bütün bu sistemler AMQP protokolü üzerinden herhangi bir **AMQP** tabanlı **message broker** gerçekleştirimi ile çalışabilir

JMS vs AMQP

- AMQP çoğunlukla **JMS'e benzetilir** ve onunla kıyaslanır
- Her JMS sunucusunun haberleşme için **kendine ait bir wiring protokol**ü mevcuttur
- Mesaj formatı ve mesaj gönderme alma yöntemleri **message broker'a özgü**dür
- JMS API java uygulamalarının JMS sunucu ile bağlantı kurma, mesaj alma ve gönderme işlemlerini **standart bir API** üzerinden yapılmasını sağlar

Spring Boot ve AMQP

- Spring Boot içerisinde RabbitMQ/AMQP desteğini devreye almak için **amqp starter** bağımlılığı pom.xml'e eklenmiş olmalıdır

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

```
@SpringBootApplication
@EnableRabbit
public class PetClinicApplication {
    ...
}
```

RabbitMQ Nedir?

- AMQP protokolünü implement eden bir **mesaj broker**'dır
- Point-to-point, request/reply, pub-sub gibi değişik stillerde **haberleşme modelleri**ni destekler
- Farklı programlama dillerinde **istemci kütüphaneleri** sunar
- **Spring-rabbit** modülü RabbitMQ broker desteği sunmaktadır

Neden Spring AMQP?

- AMQP temelli mesajlaşma çözümleri üretmek için **Spring merkezli** bir yaklaşım sunar
- **Farklı broker'lar**da mesaj gönderme ve alma için **“template” tabanlı** bir AMQP'nin üstünde ayrı bir soyutlama sağlar
- **Message Driven POJO**'lar oluşturmaya imkan tanır

Spring AMQP Bileşenleri: Message

- AMQP spesifikasyonunda **ayrı bir mesaj sınıfı veya arayüzü yok**tur
- Mesaj içeriği **byte[]**, diğer özellikleri ise ayrı metot parametreleri şeklinde gönderilir
- Spring AMQP ise **ayrı bir Message sınıfı** tanımlar
- Amacı **mesaj içeriğini ve özelliklerini bir arada encapsule edebilmek** ve API'yi basitleştirmektir

Spring AMQP Bileşenleri: Exchange

- Mesaj üreticisinin (message producer) **mesajı gönderdiği yapı**dır
- Bir AMQP broker'ındaki her bir virtual host altında exchange nesneleri **benzersiz isim**e sahiptir
- Değişik **exchange tipleri** vardır
  - Direct, Topic, Fanout, Headers
  - Davranışları **queue binding'in nasıl yapıldığı**na göre değişiklik gösterir

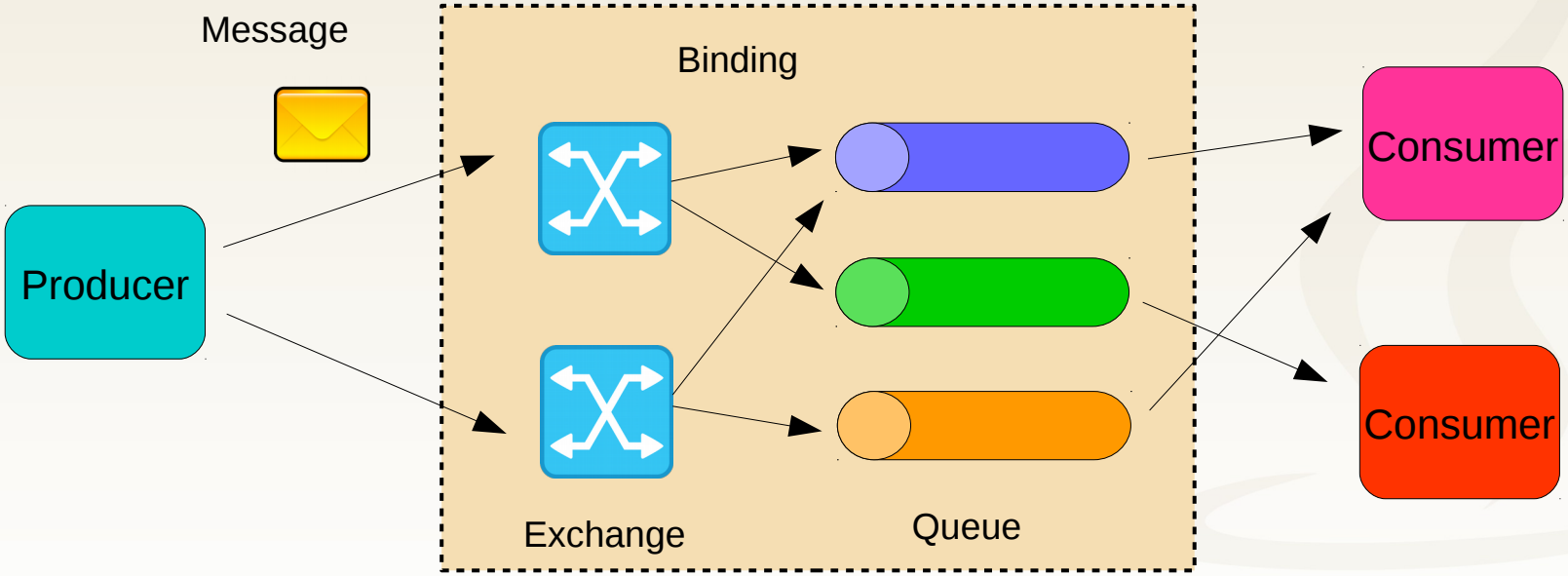
Spring AMQP Bileşenleri: Queue

- Mesaj consumer'ın **mesajı aldığı yapı**dır
- **Durable**, **exclusive** ve **auto-delete** özelliklerine sahip olabilir
- **Durable** queue, broker yeniden başlatıldığında da hayatta kalır
- **Exclusive** queue, spesifik bir connection için geçerlidir, connection sonlanınca queue'da sonlanır
- **Auto-delete** queue, son consumer silindiğinde otomatik silinir

Spring AMQP Bileşenleri: Binding/Routing

- Exchange ve queue yapıları **arasındaki bağlantı**yı ifade eder
- **Bağlantı ile ilgili bir veri** tutmaktadır, herhangi bir davranış sergilemez
- Queue-exchange **binding işlemi farklı opsiyonlarla** gerçekleştirilebilir

RabbitMQ Mimarisi



Exchange Türleri: Direct

- Mesajları queue'lara "**routing key**" ile eşleştirerek teslim eder
- Daha çok "**unicast**" mesajlaşma (1-1 mesajlaşma) için kullanılır
- Genellikle task'ları farklı worker'lara **round robin** mantığı ile dağıtmak için kullanılır

Exchange Türleri: Fanout

- Mesajları bind edilen queue'lara **doğrudan** teslim eder
- **Routing key** dikkate alınmaz
- Eğer bir exchange'e N tane queue bind edilmiş ise **bütün hepsi** publish edilen mesajı alırlar
- Daha çok mesaj **broadcast** amacı ile kullanılırlar

Exchange Türleri: Topic

- Mesajları bind edilen queue'lara **routing key**'i ve bind işlemi sırasında tanımlanan "**pattern**"ı dikkate alarak teslim eder
- Genellikle mesaj **multicast** amacı ile kullanılır
- Klasik **pub/sub** haberleşme mimarisini implement etmeyi sağlar

Exchange Türleri: Headers

- Routing key yerine **mesajın header değerleri**ne bakarak mesaj teslimini gerçekleştirir
- Direct exchange yöntemine benzer, ancak String routing key kullanmak yerine **farklı türde header değerleri** kullanma imkanı sunar
- Header değerlerinin **herhangi birisinin** eşleşmesi yeterli olabilir **veya hepsinin eşleşmesi** istenebilir

Mesaj Gönderme ve Mesaj Alma

- Mesaj gönderme ve mesaj alma işlemleri öncesinde ApplicationContext içerisinde **Queue, Exchange, Binding bean tanımları** yapılmış olmalıdır

Mesaj Gönderme

- Mesaj göndermek için **RabbitTemplate** aşağıdaki metotları sağlamaktadır

```
void send(String exchange, String routingKey, Message message)
                                    throws AmqpException;

void send(String routingKey, Message message)
                                    throws AmqpException;

void send(Message message) throws AmqpException;
```

Exchange ve routingKey parametreleri sürekli tekrarlıyorsa RabbitTemplate düzeyinde set edilerek doğrundan mesaj gönderimi de gerçekleştirilebilir

Mesaj Alma

- Mesaj almak için **RabbitTemplate** aşağıdaki metotları sunmaktadır

```
Message receive(String queueName, long timeoutMillis)
                                          throws AmqpException;

Message receive(long timeoutMillis) throws AmqpException;

Message receive(String queueName) throws AmqpException;

Message receive() throws AmqpException;
```

receiveTimeout, queue gibi değerler mesaj alımında tekrarlıyor ise RabbitTemplate düzeyinde set edilerek doğrudan mesaj alımı da gerçekleştirilebilir

MessageBuilder API

- **MessageBuilder** ve **MessagePropertiesBuilder** sınıfları vasıtası ile **Message** nesneleri “fluent bir API” ile **yaratılabilir**

```
MessageProperties props = MessagePropertiesBuilder
        .newInstance()
        .setContentType(MessageProperties.CONTENT_TYPE_TEXT_PLAIN)
        .setMessageId("123")
        .setHeader("confidentiality", "top-secret")
        .build();

Message message = MessageBuilder
        .withBody("top secret message content".getBytes())
        .andProperties(props)
        .build();
```

MessageConverter

- AMQP **mesaj içeriğini byte[]** olarak ele almaktadır
- Mesaj gönderirken ve alırken nesne-mesaj dönüşümü için **MessageConverter**'lar kullanılmaktadır
- RabbitTemplate default olarak **String, byte[] ve Serializable tipteki Java nesneleri**ni **SimpleMessageConverter** kullanarak dönüştürmektedir

MessageListener

- Asenkron mesaj alımlarında **MessageListener** arayüzü kullanılır

```
public interface MessageListener {
    void onMessage(Message message);
}
```

- Ancak bu arayüzü implement etmeden **@RabbitListener** anotasyonu ile mesajların alımı sağlanabilir

Spring Boot ve AMQP

- Spring Boot içerisinde RabbitMQ/AMQP desteğini devreye almak için **amqp starter** bağımlılığı pom.xml'e eklenmiş olmalıdır

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-amqp</artifactId>
</dependency>
```

```
@SpringBootApplication
@EnableRabbit
public class PetClinicApplication {
    ...
}
```

Spring Boot AMQP Konfigürasyon Örneği

```
@Configuration
public class AmqpConfig {
    @Bean
    public TopicExchange topicExchange1() {
        TopicExchange exchange = new TopicExchange("topic-exchange-1");
        return exchange;
    }

    @Bean
    public Queue queue1() {
        Queue queue = new Queue("queue-1");
        return queue;
    }

    @Bean
    public Binding topicExchange1Queue1Binding() {
    return BindingBuilder.bind(queue1())
                    .to(topicExchange1()).with("routing-key-1");
    }
}
```

Mesaj Gönderme ve Alma Örneği

JAVA Eğitimleri

Rabbit Template ile Mesaj Gönderme

```
Message message = MessageBuilder
                    .withBody("hello world!".getBytes()).build();
rabbitTemplate.send("topic-exchange-1","routing-key-1", message);
```

@RabbitListener ile Mesaj Alma

```
@Component
public class BarBean {
    @RabbitListener(queues= {"queue-1"})
    public void handle(Message message) throws Exception {
        String msgBody = new String(message.getBody(),"utf-8");
        System.out.println(">>>message received :" + msgBody);
    }
}
```

İletişim

- Harezmi Bilişim Çözümleri
- Kurumsal Java Eğitimleri
- http://www.java-egitimleri.com
- info@java-egitimleri.com