**RabbitMQ**

**Configurações gerais**

No application.properties/yml:

spring.rabbitmq.host = host.docker.internal  
spring.rabbitmq.port = 5672  
spring.rabbitmq.username = guest  
spring.rabbitmq.password = guest

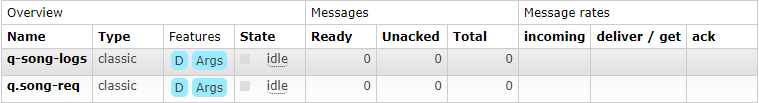
**Exchanges**

Uma **exchange** é a ferramenta que direciona as mensagens até as filas. Cada tipo de exchange tem as suas regras de direcionamento.

* **Fanout Exchange**

Espalha a mensagem para todas as filas bindadas.

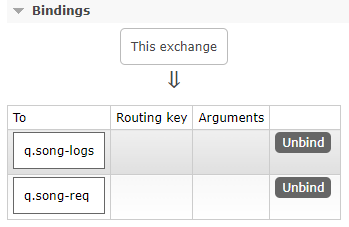
**Filas criadas:**



**Exchange criada:**

****

**Bindings:**

****

Código **producer** simples. A mensagem é enviada, em formato de json, para a exchange **x.song-req**, que será responsável por espalhar a mensagem às filas bindadas (**q.song-logs** e **q.song-req**):

@Service  
@AllArgsConstructor  
public class SongRequestService {  
  
 private RabbitTemplate rabbitTemplate;  
 private ObjectMapper objectMapper;  
  
 public SongRequest sendSongRequestToQueue(SongRequest songRequest) throws JsonProcessingException {  
 var json = objectMapper.writeValueAsString(songRequest);  
 rabbitTemplate.convertAndSend("x.song-req", "", json);  
 return songRequest;  
 }

Código **consumer** simples. Recebe a mensagem no formato json (String), e utiliza da biblioteca Gson para convertê-lo para uma instância do tipo SongRequest:

@Service  
@Slf4j  
@AllArgsConstructor  
public class SongPlayerService {  
  
 private Gson gson;  
  
 @RabbitListener(queues = "q.song-req")  
 public void listen(String msg){  
 SongRequest songRequest = gson.fromJson(msg, SongRequest.class);  
  
 playSong(songRequest);  
 }  
  
 private void playSong(SongRequest songRequest){  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 System.*out*.println("Playing " + songRequest.getSong() + "...");  
 System.*out*.println("Requested by: " + songRequest.getRequestedBy());  
 System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");  
 }  
}

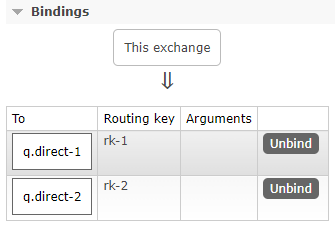
* **Direct Exchange**

Direciona a mensagem para filas baseado no routing-key da mensagem.

Segue o mesmo código da **Fanout Exchange**, mas utiliza **routing-keys** para direcionar as mensagens, que seriam passadas no segundo argumento da função **“convertAndSend(..., ..., ...)**”.

Note que a função convertAndSend(..., ..., ...), não recebe uma routing-key no **Fanout Exchange**. Isso porque ela é designada para espalhar a mensagem para mais de uma fila.

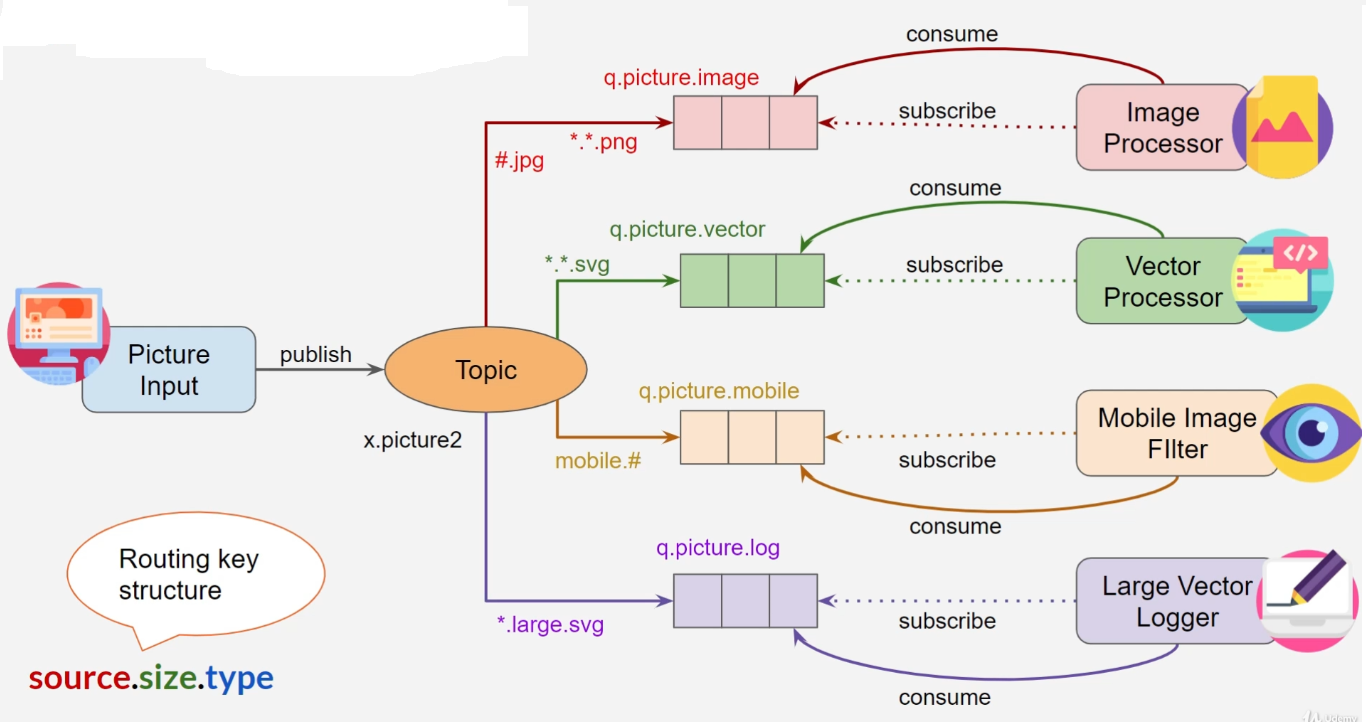
As routing-keys são configuradas na exchange.



No exemplo acima, observamos que mensagens com a **routing-key** **rk-1** serão direcionadas para a fila **q.direct-1**, quanto as com a **rk-2** serão direcionadas para a fila **q.direct-2**.

* **Topic Exchange**

Direciona a mensagem para filas baseado em um **routing-key** composto de dois ou mais parâmetros.



No exemplo acima, observamos que a **routing-key** é composta de 3 parâmetros: **source, size e type.**

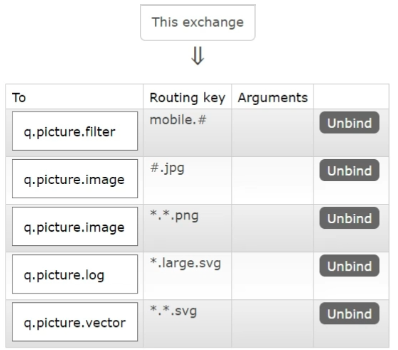
Também observamos que o símbolo **\*** substitui 1 parâmetro, enquanto o **#** substitui 1 ou mais parâmetros.

**Exemplos: \*.\*.png** - **#.jpg** - **\*.\*.svg**

**Implementando um Topic Exchange:**

1. Criar as filas normalmente
2. Criar a exchange normalmente, definindo o tipo para **Topic**
3. Clicar na exchange e configurar os bindings de acordo com a **routing-key** escolhida

No caso acima, a configuração do exchange ficaria da seguinte maneira:



Ou seja, todas as **routing-keys** que começarem com a palavra **mobile** serão direcionadas para a fila **q.picture.filter**, assim como todas as **routing-keys** que terminarem com **png** irão para a fila **q.picture.image**.

Código **producer** simples. A mensagem é enviada, em formato de json, para a exchange **x.picture2**, que será responsável por direcionar as mensagens às filas bindadas baseado no **routing-key** composto:

@Service  
public class PictureProducerTwo {  
  
 @Autowired  
 private RabbitTemplate rabbitTemplate;  
  
 @Autowired  
 private ObjectMapper objectMapper;  
  
 public void sendMessage(Picture picture) throws JsonProcessingException {  
 var json = objectMapper.writeValueAsString(picture);

//Criação da routing-key para passá-la no segundo parâmetro do método convertAndSend(...)  
 var sb = new StringBuilder();  
 sb.append(picture.getSource());  
 sb.append(".");  
 sb.append(picture.getSize() > 4000 ? "large" : "small");  
 sb.append(".");  
 sb.append(picture.getType());  
  
 rabbitTemplate.convertAndSend("x.picture2", sb.toString(), json);  
 }  
}

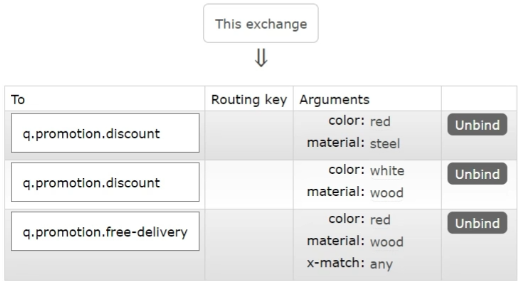
Para o **consumer**, utiliza-se o mesmo código do Fanout, adaptando somente o nome da fila desejada.

* **Header Exchange**

O **Header Exchange** funciona de forma semelhante ao **Topic Exchange**, tendo também diversos parâmetros fazendo parte de sua **routing-key** (que na verdade é localizada no header da mensagem, não sendo denominada como **routing-key**. Tal nome fora dado de forma a facilitar o entendimento)

**Implementando um Header Exchange:**

1. Criar as filas normalmente
2. Criar a exchange normalmente
3. Clicar na exchange e definir os bindings, seguindo a seguinte estrutura:

****

Perceba que o argumento **x-match** não foi informado nos dois primeiros bindings. Isso acontece porque, segundo a regra definida na vídeo-aula, caso o objeto tenha os parâmetros **color: red E material: steel (BIND 1) OU color: white E material: wood (BIND 2)**, sua mensagem será direcionada à fila **q.promotion.discount**. O valor do **x-match** por padrão é **all, significando que todas as condições informadas naquele binding devem ser satisfeitas para que o direcionamento aconteça ali**. Caso queíramos aplicar um **||** (OU), **utilizamos o argument x-match:any**. No caso acima, a mensagem só seria direcionada à fila **q.promotion.free-delivery** se: OU a cor fosse red OU o material fosse wood.

Código **producer** simples. A mensagem é enviada, em formato de json, para a exchange **x.promotion**, que será responsável por direcionar as mensagens às filas bindadas baseado no headerdefinido na classe MessageProperties. Note que, neste caso, devemos criar uma instância da classe **Message** e inserir o **MessageProperties** contendo o header e a mensagem - com o método **json.getBytes()** - nela.

@Service  
public class FurnitureProducer {  
  
 @Autowired  
 private RabbitTemplate rabbitTemplate;  
  
 @Autowired  
 private ObjectMapper objectMapper;  
  
 public void sendMessage(Furniture furniture) throws JsonProcessingException {  
  
 var messageProperties = new MessageProperties();  
 messageProperties.setHeader("color", furniture.getColor());  
 messageProperties.setHeader("material", furniture.getMaterial());  
  
 var json = objectMapper.writeValueAsString(furniture);  
  
 var message = new Message(json.getBytes(), messageProperties);  
 rabbitTemplate.convertAndSend("x.promotion", "", message);  
 }  
}

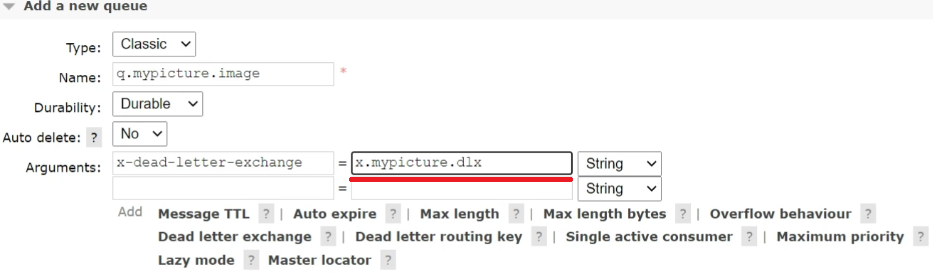
Para o **consumer**, utiliza-se o mesmo código do Fanout, adaptando somente o nome da fila desejada.

**Dead-Letter Exchange**

É para onde as mensagens que lançaram alguma exceção devem ser encaminhadas, caso contrário, entrarão em um looping infinito.

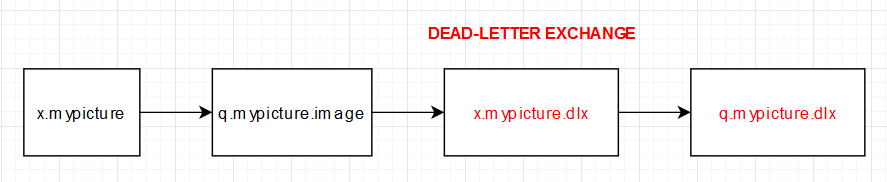
* **Método 1**

1. Criar a **dead-letter exchange** x.mypicture.dlx e a fila q.mypicture.dlx para receber as mensagens falhas, além de bindá-las
2. Criar a exchange x.mypicture
3. Na hora de criar a q.mypicture.image (fila convencional), selecionar a opção **Dead-Letter Exchange**, e informar a **Dead-Letter Exchange** criada (x.mypicture.dlx)

****

1. Bindar a x.mypicture à q.mypicture.image

A estrutura deve ficar mais ou menos assim:



1. Após isso, basta que lancemos a exceção **AmqpRejectAndDontRequeueException** no **consumer** para que a mensagem seja enviada à **Dead-Letter Exchange**

@RabbitListener(queues = "q.mypicture.image")  
public void listen(String message) throws IOException {  
 var picture = objectMapper.readValue(message, Picture.class);  
  
 if(Integer.*valueOf*(picture.getSize()) > 9000){  
 throw new AmqpRejectAndDontRequeueException("Picture's size is too large");  
 }  
 *LOG*.info("On image: {}", picture);  
}