



Spring AMQP

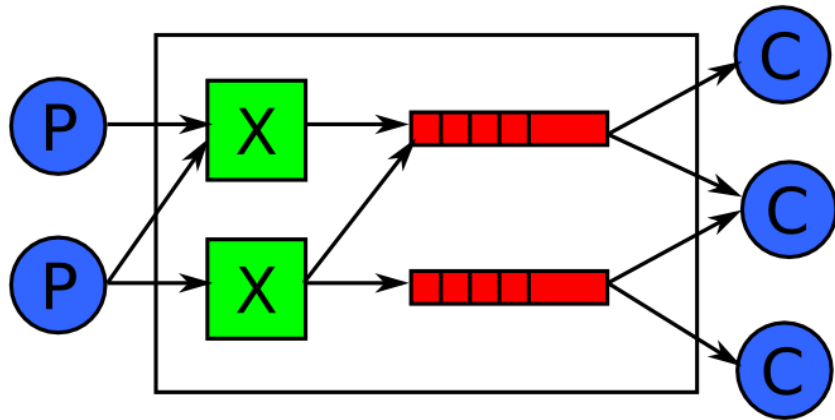
<http://www.springsource.org/spring-amqp>

Mark Fisher – *SpringSource/VMware*

Twitter: m_f_

AMQP: Quick Introduction

The AMQP Model



- Exchanges
 - Where producers send Messages
 - May also send a routing key
- Queues
 - Where consumers receive Messages
 - A named FIFO* buffer
- Bindings
 - Exchanges route to queues
 - Queues bind with routing keys/patterns

Exchange Types

- **Direct Exchange**
 - Publisher sends a routing key with Message
 - Queues bound with that routing key will receive it
- **Topic Exchange**
 - Queues bind with a pattern that may include wildcards
 - * matches a single word, # matches 0 or more words
 - examples: news.*.sports, news.us.*, news.#
- **Fanout Exchange**
 - No routing keys are used (not when binding or sending)
 - All bound Queues receive each Message
- **Headers Exchange**
 - Queue binding based on Message headers
 - Can match key=value pairs with 'x-match' as AND/OR
 - Can match the presence of a header regardless of value

Required Exchanges

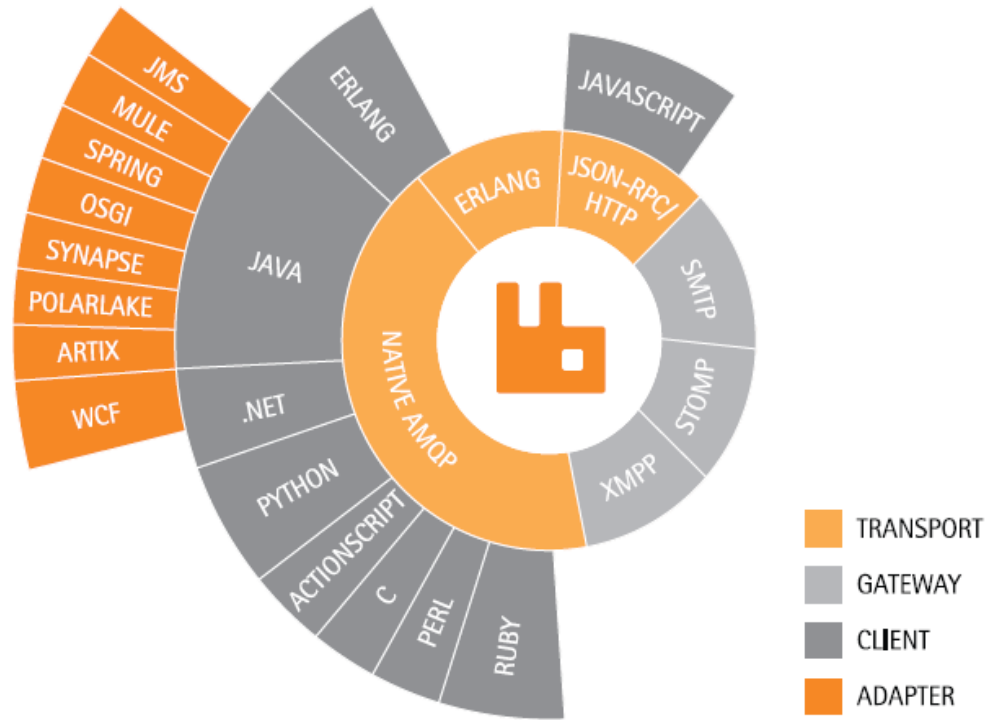
- The *nameless* Direct Exchange
 - All Queues are automatically bound to this with the Queue name as the routing key
- amq.direct
- amq.topic*
- amq.fanout
- amq.headers*
 - (amq.match is a legacy name)

* IF the Exchange type is supported by the broker, and according to the spec, it *SHOULD* be

Queues

- Each Message will only be received by a single Consumer on a given Queue (regardless of Exchange type)
 - Messages may be redelivered after failure or rejection
 - FIFO if there is only one Consumer and no redelivery
- Queue properties: durable, exclusive, auto-delete
- RabbitMQ supports auto-generation of name
 - often used with private, temporary queues (e.g. reply-to)

RabbitMQ



- Written in Erlang/OTP
 - Excellent protocol handling capabilities; simple, optimized bit operations
 - Safe, high-performance concurrency; immutable state and lightweight processes

Getting Started with RabbitMQ

- Install Erlang/OTP
- Install RabbitMQ Broker
- Start Broker

```
./sbin/rabbitmq-server
```

- Command-Line Administration

```
./sbin/rabbitmqctl add_user  
./sbin/rabbitmqctl list_exchanges  
./sbin/rabbitmqctl list_queues
```

- Install the Management Plugin
 - <http://localhost:55672/mgmt>
- Review the Tutorials and Documentation
 - <http://rabbitmq.com>

Comparing AMQP and JMS

Comparing AMQP and JMS

- JMS is an API for Java clients to access middleware
 - The provider may be proprietary, licensed software with a closed wire-format
 - Non-Java language bindings may exist (e.g. ActiveMQ NMS), but these are not covered by the specification
- AMQP is a wire-level protocol
 - Open standard backed by a Working Group to facilitate a wide variety of implementations
 - Enables interoperability between clients implemented in different languages across potentially heterogeneous platforms

Payload Types

- JMS defines a Message and several sub-types
 - BytesMessage
 - TextMessage
 - MapMessage
 - ObjectMessage
- AMQP message has a binary body
 - Define Content-Type property (e.g. “text/plain”)
 - Avoid language-specific serialization

Message Routing

- JMS defines two types of *Destination*
 - Queue
 - Topic
- AMQP defines Exchanges and Queues
 - Producers interact with Exchanges
 - Consumers interact with Queues
 - Bindings enable routing patterns within broker

Reliable Messaging

- Persistent Messages
 - JMS defines the `JMSDeliveryMode` property
 - AMQP messages have a boolean persistent property
- Acknowledgement
 - Both support auto and client (explicit) acks
- Transactions
 - JMS supports transactional Sessions and JTA
 - AMQP supports transactional publishing on a channel as well as undo of acks on consumer-side upon rollback
 - `basic.reject(requeue=true|false)` is also available

Spring AMQP

Spring AMQP for Java *and* .NET

- Common AMQP Model
 - Defines a Message and MessageProperties
 - Defines a common Exception hierarchy
- Abstraction for Client Libraries
 - AmqpTemplate and MessageListenerContainer
 - Encapsulates use of the RabbitMQ Java client library
- Support for Broker Administration
 - Declaration of Exchanges and Queues
 - Binding Queues to Exchanges
- Resource Management
 - ConnectionFactory strategy suitable for Dependency Injection
 - Support for caching Connections and Channels

RabbitMQ Support modeled after JMS Support

JMS	RabbitMQ
ConnectionFactory	ConnectionFactory
JmsTemplate	RabbitTemplate
MessageConverter	MessageConverter
*MessageListenerContainer	*MessageListenerContainer
MessageListenerAdapter	MessageListenerAdapter
SessionAwareMessageListener	ChannelAwareMessageListener

AmqpTemplate

```
public void send(Message message) // to template's exchange property
public void send(String routingKey, Message message)
public void send(String exchange, String routingKey, Message message)
public Message receive() // from template's queue property
public Message receive(String queueName)

// plain object methods, relying on MessageConverter:
public void convertAndSend(Object object)
public void convertAndSend(String routingKey, Object object)
public void convertAndSend(String exchange, String routingKey, Object o)
public Object receiveAndConvert()
public Object receiveAndConvert(String queueName)

// low-level callback, supports all higher-level operations:
public <T> T execute(ChannelCallback<T> action)
```

AmqpTemplate request/reply methods

```
public Message sendAndReceive(Message message)
```

```
public Message sendAndReceive(String routingKey, Message message)
```

```
public Message sendAndReceive(  
    String exchange, String routingKey, Message message)
```

```
// plain object methods, relying on MessageConverter:
```

```
public Object convertSendAndReceive(Object message)
```

```
public Object convertSendAndReceive(String routingKey, Object message)
```

```
public Object convertSendAndReceive(  
    String exchange, String routingKey, Object message)
```

RabbitAdmin

```
public void declareExchange(Exchange exchange)
```

```
public void deleteExchange(String exchangeName)
```

```
public Queue declareQueue()
```

```
public void declareQueue(Queue queue)
```

```
public void deleteQueue(String queueName)
```

```
public void deleteQueue(String queueName, boolean unused, boolean empty)
```

```
public void declareBinding(Binding binding)
```

BindingBuilder

```
BindingBuilder.from(new Queue("q")).to(new FanoutExchange("f"));
```

```
BindingBuilder.from(new Queue("q")).to(new DirectExchange("d")).with("r");
```

```
BindingBuilder.from(new Queue("q"))  
    .to(new DirectExchange("d")).withQueueName();
```

```
BindingBuilder.from(new Queue("q")).to(new TopicExchange("t")).with("r");
```

```
BindingBuilder.from(new Queue("q")).to(headersExchange).where("k").matches("v");
```

```
BindingBuilder.from(new Queue("q")).to(headersExchange).where("k").exists();
```

Namespace Support

```
<rabbit:direct-exchange name="demoExchange">
  <rabbit:bindings>
    <rabbit:binding queue="demoQueue"/>
  </rabbit:bindings>
</rabbit:direct-exchange>

<rabbit:queue name="demoQueue"/>
```

```
<rabbit:listener-container
  connection-factory="connectionFactory"
  acknowledge="manual">
  <rabbit:listener id="testListener"
    queue-names="foo, bar"
    ref="testBean"
    method="handle"/>
</rabbit:listener-container>
```

Spring Integration AMQP Adapters

```
<amqp:inbound-channel-adapter queue-name="demoQueue"  
    channel="amqpIn"/>
```

```
<amqp:outbound-channel-adapter channel="amqpOut"  
    exchange-name="demoExchange" routing-key="foo.bar"/>
```

```
<amqp:inbound-gateway request-channel="amqpRequests"  
    queue-name="demoQueue"/>
```

```
<amqp:outbound-gateway request-channel="amqpRequests"  
    reply-channel="amqpReplies"  
    exchange-name="demoExchange"  
    routing-key="foo.bar"/>
```

Demos

github.com/SpringSource/spring-amqp

[spring-amqp/spring-amqp-samples](#)

- *[helloworld](#)*

- *[stocks](#)*

git.springsource.org/spring-integration/samples

[basic/amqp](#)

Q&A

