Grails + messaging with AMQP/ RabbitMQ

Peter Ledbrook - SpringSource
A history of messaging
A history of messaging
A history of messaging

Royal Mail

Email
Java messaging - JMS

• Java API
• Synchronous and asynchronous messaging
• Point-to-point and broadcast
  – P2P via Queues
  – Broadcast via Topics
• No standard for communication between client and broker
JMS Queues

Producer → Broker → Queue → Consumer
The challenger - AMQP

- Advanced Message Queuing Protocol
- Wire-level protocol
  - Any type of client
  - Client-broker communication standardised
- Synchronous and asynchronous messaging
- Point-to-point, broadcast, and more
  - Single, flexible model
- Simple management part of the protocol
  - Create exchanges and queues
Basic structure

Producer

Exchange: accepts and routes messages from producer

Exchange

Broker

Queue

Queue

Consumer

Consumer

Consumer

Consumer
Basic structure

Producer → Exchange: accepts and routes messages from producer

Exchange → Queue

Queue: a FIFO queue of messages - each message can only go to one consumer

Queue → Consumer

Consumer
Basic structure

Exchange: accepts and routes messages from producer

Binding: rule for routing messages to associated queue

Queue: a FIFO queue of messages - each message can only go to one consumer
Exchanges

• Only producers talk to the exchange directly
• Message routing depends on
  – Exchange type
  – Message’s ‘routing key’, e.g. “stocks.nasdaq.vmw”
  – Binding between exchange and queue
• Routing and binding keys are typically strings
  – Allow for filtering - similar to JMS selectors
Exchange types

- **Fanout**
  - Messages go to all bound queues
  - Routing and binding keys are ignored

- **Direct**
  - Messages only go to queues with a binding key that exactly matches the routing key
  - Typically routing key is the queue name

- **Topic**
  - Like Direct exchange but binding key can have wildcards
    - ‘#’ like regex ‘*’, ‘*’ like regex ‘?’

- **Headers**
  - Routing based on message headers
Example: JMS-like Queue

Producer → Direct Exchange → myQueue → Consumer

Broker
Example: JMS-like Queue

Producer \rightarrow Direct Exchange \rightarrow myQueue \rightarrow Consumer

routing-key = myQueue
Example: JMS-like Queue

Producer → Direct Exchange → Consumer

binding = myQueue

routing-key = myQueue
Example: JMS-like Topic

Producer

Broker

Fanout Exchange

Queue

Queue

Queue

Consumer

Consumer

Consumer
Example: JMS-like Topic

Producer

Fanout Exchange

Routing key not specified

Broker

Queue

Queue

Queue

Consumer

Consumer

Consumer

Friday, 22 October 2010
Example: JMS-like Topic

- Producer
- Fanout Exchange
- Binding not specified
- Queue
- Queue
- Queue
- Consumer
- Consumer
- Consumer

Routing key not specified
Example: broadcast with filtering
Example: broadcast with filtering

Producer

Broker

Topic Exchange

Consumer

Queue

Consumer

Queue

Consumer

Queue

routing-key = shares.nyse.vmw
Example: broadcast with filtering

Producer

Topic Exchange

Broker

Queue

Queue

Queue

Consumer

Consumer

Consumer

binding = shares.#

routing-key = shares.nyse.vmw
Example: broadcast with filtering

Producer

Topic Exchange

Broker

Queue

Queue

Queue

Consumer

Consumer

Consumer

routing-key = shares.nyse.vmw

binding = shares.*
Example: broadcast with filtering

Producer -> Topic Exchange

- routing-key = shares.nyse.vmw

Broker

- Queue
- Queue
- Queue

Consumer

- Consumer
- Consumer
- Consumer

binding = shares.nyse.vmw

Friday, 22 October 2010
Example: RPC

Producer -> Direct Exchange -> Broker (myQueue) -> Consumer
Producer -> Direct Exchange -> Broker (replyQ)
Example: RPC

- **Producer**
  - routing-key = myQueue
  - reply-to = replyQ

- **Direct Exchange**
- **myQueue**
- **replyQ**

- **Consumer**
- **Broker**
Example: RPC

Producer

Direct Exchange

myQueue

binding = myQueue

Consumer

replyQ

routing-key = myQueue
reply-to = replyQ
Example: RPC

- Producer
- Direct Exchange
- myQueue
- replyQ
- Consumer

routing-key = replyQ
Example: RPC

- **Producer**
- **Direct Exchange**
- **Broker**
- **Consumer**
  - myQueue
  - routing-key = replyQ
  - binding = replyQ

**Interaction:**
- Producer publishes a message to the Direct Exchange.
- The message is sent to the myQueue via the Broker.
- The Consumer, bound to the replyQ, consumes the message.
Example: work distribution
Example: work distribution

Producer -> Direct Exchange -> Queue -> Consumer

routing-key = some.task
Example: work distribution

Producer

Direct Exchange

Queue

Consumer

Consumer

Consumer

Broker

binding = some.task

routing-key = some.task

Friday, 22 October 2010
Messages

- Headers
  - routing-key
  - reply-to
  - content-type, etc.
- Custom properties
- Body
  - Byte data
  - Producer and consumer must agree on the format of the content
  - ... or use content-type header
  - AMQP does not define a meaning for content-type!
Queue and exchange properties

• **Durable**
  – Survives a broker restart
  – Applies to exchanges and queues

• **Auto delete**
  – Exchange will be deleted when all its bindings are gone
  – Queue will be deleted when all consumers are gone

• **Exclusive**
  – Only the owner can read messages from the queue
  – Doesn’t apply to exchanges
The Grails integration

• RabbitMQ plugin
• Declare exchanges and queues
• Configure services as queue consumers
  – Simple static properties
• Dynamic method for sending AMQP messages
class ListenerService {
    // Declare name of queue to listen to
    static rabbitQueue = "msgs"

    void handleMessage(msg) {
        // Do something with the message
    }
}

class AnotherListenerService {
    // Subscribe to a topic exchange
    static rabbitSubscribe = "sharesExchange"

    void handleMessage(msg) {
        // Do something with the message
    }
}
Consuming messages

```
class ListenerService {
    // Declare name of queue to listen to
    static rabbitQueue = "msgs"

    void handleMessage(msg) {
        // Do something with the message
    }
}

class AnotherListenerService {
    // Subscribe to a topic exchange
    static rabbitSubscribe = {
        name: "myEx",
        routingKey: "shares.#"
    }

    void handleMessage(msg) {
        // Do something with the message
    }
}
```
Sending messages

class PublisherService {
    def notify() {
        rabbitSend "msgs", "app.event", "The event details"
    }
}

Friday, 22 October 2010
class PublisherService {
    def notify() {
        rabbitSend "msgs", "app.event", "The event details"
    }
}
class PublisherService {

    def notify(String itemName) {
        rabbitSend "msgs", "app.event", [event: "publish", item: itemName ]
    }
}

Sending messages
Declaring exchanges and queues

// Config.groovy
rabbitmq {
    connectionfactory {
        ...
    }
    queues = {
        msgs durable: false, autoDelete: true
        exchange name: "shares", type: topic, durable: true, {
            allShares durable: true, autoDelete: false, binding: 'shares.#'
        }
    }
}
Declaring exchanges and queues

// Config.groovy
rabbitmq {
  connectionfactory {
    ...
  }
  queues = {
    msgs durable: false, autoDelete: true
    exchange name: "shares", type: topic, durable: true, {
      allShares durable: true, autoDelete: false, binding: 'shares.#'
    }
  }
}

- Standalone queue (msgs) - bound to default direct exchange
- Topic exchange (shares)
- Queue (allShares) bound to exchange (shares) with routing key ('shares.#')
A word about message content

• In the broker, it’s just byte data
• Plugin interprets data based on content-type header
  – Spring AMQP SimpleMessageConverter
  – String → text/plain; charset=utf-8
  – Serializable → application/x-java-serialized-object
  – Otherwise, just byte[]
• Producers & consumers typically agree on format
• Not all clients set the content-type!
• You still have to agree on format even if you use JSON or XML message content
Demo