
GRAILS


spring
source

RabbitMQ Plugin - Reference Documentation

Authors: Jeff Brown

Version: 0.2

Table of Contents

1. Introduction To The RabbitMQ Plugin 3
2. Configuration 4
  2.1 Configuring Queues 4
3. Sending Messages 6
4. Consuming Messages 7
5. Using The RabbitTemplate Directly 8

1. Introduction To The RabbitMQ Plugin

The RabbitMQ plugin provides integration with the RabbitMQ highly reliable enterprise messaging system. The plugin relies on Spring AMQP as an implementation detail, which provides a high level abstraction for sending and receiving messages.
This guide documents configuration details and usage details for the plugin. More information on RabbitMQ itself is available at rabbitmq.com.

2. Configuration

The plugin supports a number of configuration options which all may be expressed in `grails-app/conf/Config.groovy`. A basic configuration might look something like this:

```
// grails-app/conf/Config.groovy
rabbitmq {
    connectionfactory {
        username = 'guest'
        password = 'guest'
        hostname = 'localhost'
    }
}
```

Those are settings which are necessary in order for the plugin to be able to connect to and communicate with a RabbitMQ server.
Following is a list of other configuration settings supported by the plugin.

Configuration Property	Description	Default
rabbitmq.connectionfactory.username	The user name for connection to the server	(none)
rabbitmq.connectionfactory.password	The password for connection to the server	(none)
rabbitmq.connectionfactory.hostname	The host name of the server	(none)
rabbitmq.connectionfactory.channelCacheSize	The connection channel cache size	10
rabbitmq.concurrentConsumers	The number of concurrent consumers to create per message handler. Raising the number is recommendable in order to scale the consumption of messages coming in from a queue. Note that ordering guarantees are lost when multiple consumers are registered.	1

2.1 Configuring Queues

Queues must be declared in the RabbitMQ server before consumers can be associated with those Queues and before messages may be sent to those Queues. If the Grails application will be sending messages to or receiving messages from Queues that may not already be declared in the RabbitMQ server, the application needs to declare those Queues up front. One way to do that is to add beans to the Spring application context of type `org.springframework.amqp.core.Queue`. That might look something like this:

```
// grails-app/conf/spring/resources.groovy
beans = {
    myQueue(org.springframework.amqp.core.Queue, 'myQueueName')
    myOtherQueue(org.springframework.amqp.core.Queue, 'myOtherQueueName') {
        autoDelete = false
        durable = true
        exclusive = false
        arguments = [arg1: 'val1', arg2: 'val2']
    }
}
```

The plugin also supports a DSL for describing these Queues. This DSL is expressed in `grails-app/conf/Config.groovy`. The code below shows how to describe same Queues that were described in the previous code sample.

```
// grails-app/conf/Config.groovy
rabbitmq {
    connectionfactory {
        username = 'guest'
        password = 'guest'
        hostname = 'localhost'
    }
    queues = {
        myQueueName()
        myOtherQueueName autoDelete: false, durable: true, exclusive: false, arguments: [arg
    }
}
```

With both techniques, the `autoDelete`, `durable` and `exclusive` attributes default to `false` and the `arguments` attribute defaults to null.

3. Sending Messages

The plugin adds a method named `rabbitSend` to all Grails artefacts (Controllers, Services, TagLibs, etc...). The `rabbitSend` method accepts 2 parameters. The first parameter is a queue name and the second parameter is the message being sent.

```
class MessageController {
    def sendMessage = {
        rabbitSend 'someQueueName', 'someMessage'
        …
    }
}
```

Messages may also be sent by interacting with the `RabbitTemplate` Spring bean directly. See the Using The RabbitTemplate Directly section for more information.

4. Consuming Messages

The plugin provides a convention based mechanism for associating a listener with a Queue. Any Grails Service may express that it wants to receive messages on a specific Queue by defining a static property named `rabbitQueue` and assigning the property a String which represents the name of a Queue.

```
package org.grails.rabbitmq.test
class DemoService {
    static rabbitQueue = 'someQueueName'
    void handleMessage(message) {
        // handle message…
    }
}
```

Messages are delivered to the Service by invoking the `handleMessage` method.
Different message types may be handled with overloaded versions of `handleMessage`.

```
package org.grails.rabbitmq.test
class DemoService {
    static rabbitQueue = 'someQueueName'
    void handleMessage(String textMessage) {
        // handle String message…
    }
    void handleMessage(Map mapMessage) {
        // handle Map message///
    }
}
```

5. Using The RabbitTemplate Directly

Most of the interaction with the RabbitMQ server is being handled by an instance of `org.springframework.amqp.rabbit.core.RabbitTemplate`. For many applications this is happening at a lower level than the application needs to be concerned with. The plugin does provide a Spring bean to the application context that is an instance of the `RabbitTemplate` class which may be used directly. The bean name is `rabbitTemplate`.

```
class MessageController {
    def rabbitTemplate
    def sendMessage = {
        rabbitTemplate.convertAndSend('someQueueName', 'someMessage)
        …
    }
}
```

The full API of the `RabbitTemplate` class is documented in The Spring AMQP API Docs.