介绍

**前置条件**

本教程假定RabbitMQ在默认的端口(5672)上的本地主机上安装并运行。如果你使用不同的host,port 或credentials，则连接设置需要进行调整。

**从哪里可以获得帮助**

如果你在阅读本教程是遇到困难，可以通过邮件与我们取得联系[contact us](https://groups.google.com/forum/#!forum/rabbitmq-users)

RabbitMQ是一个message broker：它接受和转发消息。 你可以把它想象成一个邮局：当你把你想要发布的邮件放在邮箱中时，你可以确定邮差先生最终将邮件发送给你的收件人。 在这个比喻中，RabbitMQ是邮政信箱，邮局和邮递员。

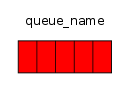
RabbitMQ和邮局的主要区别在于它不处理纸张，而是accepts, stores and forwards binary blobs of data ‒ *messages*。

RabbitMQ和一般的消息传递使用了一些术语。

* *Producing* 意味着sending. 一个发送messages的程序时一个*producer* :

http://www.rabbitmq.com/img/tutorials/producer.png

* *queue* 是RabbitMQ内的post box 的名称。尽管messages flow经过RabbitMQ和你的应用程序，但他们只能存储在*queue* 中。一个*queue* 只受host’s memory和disk limits，它本质上是一个很大的message buffer，许多*producers*  可以send messages到queue，并且许多consumers可以尝试从一个queue receive数据。这就是我们呈现的queue的方式。



* Consuming 与receiving有类似的意义。consumer 是一个主要等待接收messages的程序。

http://www.rabbitmq.com/img/tutorials/consumer.png

请注意，producer, consumer, and broker不必在同一主机上; 实际上在大多数应用程序中它们不是在同一主机上。

## "Hello World"

### (using the spring-amqp client)

在本教程的这一部分，我们将使用spring-amqp library编写两个程序; 一个发送单个消息的producer，以及接收消息并将其打印出来的consumer。 我们将详细介绍Spring-amqp API中的一些细节，并着重介绍这个非常简单的事情，以便开始使用。 这是一个messaging传递的“Hello World”。

在下图中，“P”是我们的producer，“C”是我们的consumer。 中间的盒子是一个queue - RabbitMQ代表consumer保存的message buffer。

(P) -> [|||] -> (C)

#### The Spring AMQP Framework

RabbitMQ提供多种protocols。 本教程使用AMQP 0-9-1，它是一种开源的通用消息传递协议。 RabbitMQ有许多不同的语言客户端。

Spring AMQP利用Spring Boot进行配置和dependency management。 Spring支持maven或gradle，但对于本教程，我们将选择使用Spring Boot 1.5.2的maven。 打开 [Spring Initializr](http://start.spring.io/)并提供 : group id（例如org.springframework.amqp.tutorials）artifact id（例如rabbitmq-amqp-tutorials）搜索amqp依赖关系并选择AMQP依赖关系。

生成项目并将生成的项目解压缩到您选择的位置。 现在可以将其导入到您最喜欢的IDE中。 或者，您可以使用您最喜爱的编辑器处理它。

### Configuring the project

Spring Boot提供了许多功能，但我们只会在这里强调一些。 首先，Spring Boot应用程序可以通过application.properties或application.yml文件提供属性。你会在生成的项目中找到一个没有任何内容的application.properties文件。将application.properties重命名为具有以下属性的application.yml文件：

spring:

profiles:

active: usage\_message

logging:

level:

org: ERROR

tutorial:

client:

duration: 10000

创建一个新的目录(package - tut1)，我们可以在其中放置教程代码。 现在我们将创建一个JavaConfig文件（Tut1Config.java）以下面的方式描述我们的bean：

package org.springframework.amqp.tutorials.tut1;

import org.springframework.amqp.core.Queue;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.context.annotation.Profile;

@Profile({"tut1","hello-world"})

@Configuration

public class Tut1Config {

@Bean

public Queue hello() {

return new Queue("hello");

}

@Profile("receiver")

@Bean

public Tut1Receiver receiver() {

return new Tut1Receiver();

}

@Profile("sender")

@Bean

public Tut1Sender sender() {

return new Tut1Sender();

}

}

请注意，我们已将第一个教程配置文件定义为tut1，package name或hello-world。 我们使用@Configuration让Spring知道这是一个Java Configuration，并且在其中创建Queue ("hello") 并定义我们的Sender and Receiver beans。

我们将通过我们的简单的配置文件运行整个我们的应用程序教程。要启用此功能，我们将修改RabbitAmqpTutorialsApplication.java并启动:

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Profile;

import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication

@EnableScheduling

public class RabbitAmqpTutorialsApplication {

@Profile("usage\_message")

@Bean

public CommandLineRunner usage() {

return new CommandLineRunner() {

@Override

public void run(String... arg0) throws Exception {

System.out.println("This app uses Spring Profiles to

control its behavior.\n");

System.out.println("Sample usage: java -jar

rabbit-tutorials.jar

--spring.profiles.active=hello-world,sender");

}

};

}

@Profile("!usage\_message")

@Bean

public CommandLineRunner tutorial() {

return new RabbitAmqpTutorialsRunner();

}

public static void main(String[] args) throws Exception {

SpringApplication.run(RabbitAmqpTutorialsApplication.class, args);

}

}

并添加RabbitAmqpTutorialsRunner.java代码，如下所示：

package org.springframework.amqp.tutorials;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.beans.factory.annotation.Value;

import org.springframework.boot.CommandLineRunner;

import org.springframework.context.ConfigurableApplicationContext;

public class RabbitAmqpTutorialsRunner implements CommandLineRunner {

@Value("${tutorial.client.duration:0}")

private int duration;

@Autowired

private ConfigurableApplicationContext ctx;

@Override

public void run(String... arg0) throws Exception {

System.out.println("Ready ... running for " + duration + "ms");

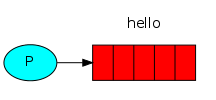
Thread.sleep(duration);

ctx.close();

}

}

### Sending



现在有很少的代码需要进入sender and receiver classes。 我们称他们为Tut1Receiver和Tut1Sender。 Sender利用我们的config和RabbitTemplate send the message。

*// Sender*

package org.springframework.amqp.tutorials.tut1;

import org.springframework.amqp.core.Queue;

import org.springframework.amqp.rabbit.core.RabbitTemplate;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.scheduling.annotation.Scheduled;

public class Tut1Sender {

@Autowired

private RabbitTemplate template;

@Autowired

private Queue queue;

@Scheduled(fixedDelay = 1000, initialDelay = 500)

public void send() {

String message = "Hello World!";

this.template.convertAndSend(queue.getName(), message);

System.out.println(" [x] Sent '" + message + "'");

}

}

您会注意到spring-amqp删除了boiler plate code，只留下messaging的逻辑以便于被关注(concerned)。我们的Tut1Config类的bean定义中配置的queue中autowire queue，并且像许多spring connection abstractions一样，我们使用可以autowired sender的RabbitTemplate封装样板rabbitmq client classes。 剩下的就是创建一条message并调用模板的convertAndSend方法，从我们定义的bean和我们刚创建的message传入queue name。

#### Sending doesn't work!

如果这是您第一次使用RabbitMQ，并且您没有看到“已发送”消息，那么您可能会抓住您的头脑，想知道会出现什么问题。 也许broker启动时没有足够的可用磁盘空间（默认情况下它至少需要200 MB空闲空间），因此拒绝接受消息。 检查broker日志文件以确认并在必要时减少限制。  [configuration file documentation](http://www.rabbitmq.com/tutorials/a%3E%20href=%22http:/www.rabbitmq.com/configure.html#config-items%22%3C/a)将告诉你如何设置disk\_free\_limit。

### Receiving

receiver同样是非常简单的。 我们使用@RabbitListener注解Receiver类并传入queue name。 然后我们使用@RabbitHandler注解我们的接收方法，传递已经pushed到queue的有效payload。

package org.springframework.amqp.tutorials.tut1;

import org.springframework.amqp.rabbit.annotation.RabbitHandler;

import org.springframework.amqp.rabbit.annotation.RabbitListener;

@RabbitListener(queues = "hello")

public class Tut1Receiver {

@RabbitHandler

public void receive(String in) {

System.out.println(" [x] Received '" + in + "'");

}

}

### Putting it all together

该应用程序使用Spring Profiles来控制它正在运行的教程，以及它是Sender or Receiver。 使用配置文件选择要运行的教程。 例如：

- {tut1|hello-world},{sender|receiver}

- {tut2|work-queues},{sender|receiver}

- {tut3|pub-sub|publish-subscribe},{sender|receiver}

- {tut4|routing},{sender|receiver}

- {tut5|topics},{sender|receiver}

- {tut6|rpc},{client|server}

当我们继续阅读其他五篇教程时，我们会回到这份清单。 使用maven构建之后，运行应用程序，但是您想运行boot apps（例如，从ide或命令行）。 我们将演示如何从命令行运行。

例如:

*# publisher*

java -jar rabbitmq-tutorials.jar --spring.profiles.active=hello-world,sender

*# consumer*

java -jar rabbitmq-tutorials.jar --spring.profiles.active=hello-world,receiver

#### Listing queues

您可能希望看到RabbitMQ有什么queues以及它们中有多少条messages。 您可以使用rabbitmqctl工具

sudo rabbitmqctl list\_queues

在Windows上省略sudo:

rabbitmqctl.bat list\_queues

有时间转到第2部分( [part 2](http://www.rabbitmq.com/tutorials/tutorial-two-spring-amqp.html))并构建一个简单的work queue模式。