**声明：此文档为个人翻译官方文档，未经允许请勿外传。**

**翻译者：杨德军**

Remote procedure call (RPC)

**(using the spring-amqp client)**

**前置条件**

本教程假定RabbitMQ在默认端口（5672）上的本地主机上安装并运行。 如果您使用不同的host, port or credentials, connections设置将需要调整。

**从哪里获得帮助**

如果您在阅读本教程时遇到困难，可以通过邮件与我们联系[contact us](https://groups.google.com/forum/#!forum/rabbitmq-users)。

在 [second tutorial](http://www.rabbitmq.com/tutorials/tutorial-two-spring-amqp.html)中，我们学习了如何使用*Work Queues* 在多个workers之间分配time-consuming的tasks。

但是如果我们需要在远程计算机上运行某个功能并等待结果呢？ 那么，这是一个不同的故事。 这种模式通常称为远程过程调用或RPC。

在本教程中，我们将使用RabbitMQ构建一个RPC系统：一个client和一个scalable RPC server。 由于我们没有任何值得分发的耗时任务，我们将创建一个返回Fibonacci numbers的虚拟RPC服务。

**Client interface**

为了说明如何使用RPC服务，我们将把我们的配置文件名称从“Sender”和“Receiver”更改为“Client”和“Server”。当我们调用服务器时，我们将返回参数的fibonacci。

Integer response = (Integer) template.convertSendAndReceive

(exchange.getName(), "rpc", start++);

System.out.println(" [.] Got '" + response + "'");

**关于RPC**

虽然RPC是计算中很常见的模式，但它经常受到批评。 当程序员不知道函数调用是本地的还是慢速的RPC时会出现这些问题。 像这样的混乱导致不可预测的系统，并增加了调试的不必要的复杂性。 而不是简化软件，滥用RPC会导致不可维护的spaghetti code。

铭记这一点，请考虑以下建议：

* 确保显而易见哪个函数条用是本地的，哪个是远程的。
* 记录你的系统。清楚组件之间的依赖关系。
* 处理错误情况。当RPC服务长时间关闭时，客户端应该如何反应?

有疑问时避免RPC。 如果可以的话，你应该使用asynchronous pipeline - 而不是类似于RPC的blocking，结果被异步推送到下一个计算阶段。

**Callback queue**

一般来说，通过RabbitMQ来执行RPC是很容易的。 客户端发送请求消息，服务器回复响应消息。 为了收到回应，我们需要发送一个'callback' queue address与request。 当我们使用上面的'convertSendAndReceive()'方法时，Spring-amqp的RabbitTemplate为我们处理callback queue。 使用RabbitTemplate时无需做任何其他设置。 有关详细解释，请参阅 [Request/Reply Message](http://docs.spring.io/spring-amqp/reference/htmlsingle/#request-reply)。

**Message properties**

AMQP 0-9-1协议预定义了一组包含14个属性的消息。 大多数属性很少使用，但以下情况除外：

* deliveryMode: 将消息标记为persistent (with a value of 2) 或 transient (any other value). 你可能会记得 [the second tutorial](http://www.rabbitmq.com/tutorials/tutorial-two-spring-amqp.html)中的这个属性。
* contentType: 用于描述编码的mime-type 。例如，对于经常使用JSON编码，将此属性设置为application/json是一种很好的做法。
* replyTo: 通常用于name a callback queue
* correlationId: 用于将RPC responses 与requests 关联起来

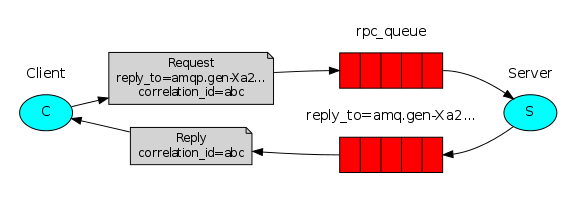
**Correlation Id**

Spring-amqp允许您专注于您正在使用的消息风格，并隐藏支持此风格所需的消息管道的详细信息。 例如，本地客户端通常会为每个RPC请求创建一个callback queue。 这是非常低效的，所以另一种方法是为每个客户端创建一个callback queue。

这引发了一个新问题，在该队列中收到回复后，不清楚回复属于哪个请求。 那是什么时候使用correlationId属性。 Spring-amqp自动为每个请求设置一个唯一值。 此外，它还处理将响应与正确的correlationID进行匹配的详细信息。

spring-amqp使得rpc风格更简单的一个原因是，有时您可能想忽略callback queue中的未知消息，而不是因为错误而失败。 这是由于服务器端可能出现竞争状况。 尽管不太可能，但在发送给我们答案之后，但在发送请求的确认消息之前，RPC服务器可能会死亡。 如果发生这种情况，重新启动的RPC服务器将再次处理该请求。 spring-amqp客户端优雅地处理重复的响应，并且RPC理想情况下应该是幂等的。

**Summary**



我们的RPC 会像这样工作:

* Tut6Config 将设置一个新的DirectExchange 和一个client
* client 将利用convertSendAndReceive passing the exchange name, the routingKey, and the message.
* request 被发送到 rpc\_queue ("tut.rpc") queue.
* RPC worker (又称为: server) 正在等待该队列上的requests. 当出现请求时，它执行并使用replyTo 字段中的队列将结果发送回客户端的消息
* client 等待callback queue中的数据. 当出现消息时，它会检查correlationId属性。如果它匹配来自请求的值，则返回对应用程序的响应。再次，这是通过RabbitTemplate自动完成的。

Putting it all together

Fibonacci task 是@RabbitListener，定义如下：:

public int fib(int n) {

return n == 0 ? 0 : n == 1 ? 1 : (fib(n - 1) + fib(n - 2));

}

我们declare我们的fibonacci function。 它只假定有效的正整数输入。 （不要指望这个版本适用于大数字，它可能是最慢的递归实现）。

我们的Tut6Config [Tut6Config](https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/spring-amqp/src/main/java/org/springframework/amqp/tutorials/tut6/Tut6Config.java)的代码如下所示：

import org.springframework.amqp.core.Binding;

import org.springframework.amqp.core.BindingBuilder;

import org.springframework.amqp.core.DirectExchange;

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({"tut6","rpc"})

@Configuration

public class Tut6Config {

@Profile("client")

private static class ClientConfig {

@Bean

public DirectExchange exchange() {

return new DirectExchange("tut.rpc");

}

@Bean

public Tut6Client client() {

return new Tut6Client();

}

}

@Profile("server")

private static class ServerConfig {

@Bean

public Queue queue() {

return new Queue("tut.rpc.requests");

}

@Bean

public DirectExchange exchange() {

return new DirectExchange("tut.rpc");

}

@Bean

public Binding binding(DirectExchange exchange,

Queue queue) {

return BindingBuilder.bind(queue)

.to(exchange)

.with("rpc");

}

@Bean

public Tut6Server server() {

return new Tut6Server();

}

}

}

它将我们的配置文件设置为“tut6”或“rpc”。 它还用两个bean建立了一个“client”配置文件; 1）我们使用的DirectExchange和2）Tut6Client本身。 我们还使用三个bean“tut.rpc.requests”队列，与客户端exchange匹配的DirextExchange以及使用“rpc” routing-key从队列到交换的绑定来配置“server”配置文件。

server code 非常简单:

* 像往常一样，我们使用@RabbitListener 开始注解receiver方法，并定义器listening queue
* 我们的fibanacci 方法使用payload参数调用fib() 并返回结果

我们的RPC client [Tut6Server.java](https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/spring-amqp/src/main/java/org/springframework/amqp/tutorials/tut6/Tut6Server.java) 代码如下:

package org.springframework.amqp.tutorials.tut6;

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

public class Tut6Server {

@RabbitListener(queues = "tut.rpc.requests")

*// @SendTo("tut.rpc.replies") used when the*

*// client doesn't set replyTo.*

public int fibonacci(int n) {

System.out.println(" [x] Received request for " + n);

int result = fib(n);

System.out.println(" [.] Returned " + result);

return result;

}

public int fib(int n) {

return n == 0 ? 0 : n == 1 ? 1 : (fib(n - 1) + fib(n - 2));

}

}

client code [Tut6Client](https://github.com/rabbitmq/rabbitmq-tutorials/blob/master/spring-amqp/src/main/java/org/springframework/amqp/tutorials/tut6/Tut6Client.java)  server code 一样简单:

* 我们按照Tut6Config中定义的autowire RabbitTemplate 和DirectExchange bean.
* 我们使用parameters exchange name key and message 调用template.convertSendAndReceive。
* 我们打印结果

提出client request很简单:

import org.springframework.amqp.core.DirectExchange;

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

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

import org.springframework.scheduling.annotation.Scheduled;

public class Tut6Client {

@Autowired

private RabbitTemplate template;

@Autowired

private DirectExchange exchange;

int start = 0;

@Scheduled(fixedDelay = 1000, initialDelay = 500)

public void send() {

System.out.println(" [x] Requesting fib(" + start + ")");

Integer response = (Integer) template.convertSendAndReceive

(exchange.getName(), "rpc", start++);

System.out.println(" [.] Got '" + response + "'");

}

}

使用start.spring.io和SpringInitialzr（参见 [tutorial one](http://www.rabbitmq.com/tutorials/tutorial-one-spring-amqp.html)）中定义的项目设置，准备运行时与其他教程相同：

mvn clean package

我们可以用以下方式启动server:

java -jar target/rabbit-tutorials-1.7.1.RELEASE.jar

--spring.profiles.active=rpc,server

--tutorial.client.duration=6000

request fibonacci number 运行客户端:

java -jar target/rabbit-tutorials-1.7.1.RELEASE.jar

--spring.profiles.active=rpc,server

java -jar target/rabbit-tutorials-1.7.1.RELEASE.jar

--spring.profiles.active=rpc,client

这里介绍的设计并不是RPC服务的唯一实现方式，但它有一些重要的优点：

* 如果RPC服务器速度太慢，可以通过运行另一个来扩展。尝试在新控制台中运行第二个RPCServer。
* 在客户端，RPC要求只用一种方法发送和接收一条消息。不需要像queueDeclare这样的同步调用。因此，RPC client仅需要一次网络往返即可获得单个RPC请求。

我们的代码仍然非常简单，不会尝试解决更复杂(但种种)的问题，如：

* 如果没有servers在运行，client应该如何应对?
* Client是否应该对RPC有某种超时?
* 如果server发生故障应引发异常，是否应将其转发给client?
* 在处理之前防止无效的传入消息(例如checking bounds, type)

如果你想进行试验，你可能会发现[management UI](http://www.rabbitmq.com/management.html) 对查看queues很有用。

RabbitMQ还有一个很好的功能。它作为服务在Pivotal Cloud Foundry(PCF)上被支持。