# 12-rabbitmq-远程过程调用-spring

## 先决条件

本教程假定RabbitMQ已在标准端口（5672）上的localhost上安装并运行。如果使用不同的主机，端口或凭据，连接设置将需要调整。

## 远程过程调用

在第二个教程中，我们学习了如何使用工作队列在多个工作人员之间分配耗时的任务。

但是，如果我们需要在远程计算机上运行功能并等待结果怎么办？那是一个不同的问题。此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：一个客户机和一个可扩展的RPC服务器。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斐波纳契数字的虚拟RPC服务。

## 客户端界面

为了说明如何使用RPC服务，我们将从“Sender”和“Receiver”更改“Client”和“Server”的名称，当我们调用服务器时，我们将获得参数的fibonacci值。

|  |
| --- |
| Integer response = (Integer) template.convertSendAndReceive(  exchange.getName(), "rpc", start++);  System.out.println(" [.] Got '" + response + "'"); |

## RPC注意事项

虽然RPC是一个很常见的计算模式，但它经常被批评。当程序员不知道函数调用是本地函数还是缓慢的RPC时，出现问题。这样的混乱导致了一个不可预测的系统，并增加了调试的不必要的复杂性。而不是简化软件，滥用RPC可能导致不可维护的意大利面条代码。

铭记这一点，请考虑以下建议：

* 确保显而易见哪个函数调用是本地的，哪个是远程的。
* 记录您的系统。使组件之间的依赖关系清除。
* 处理错误情况。当RPC服务器停机很长时间后，客户端应该如何反应？

当有疑问避免RPC。如果可以的话，您应该使用异步管道 - 而不是类似RPC的阻塞，将异步推送到下一个计算阶段。

## 回调队列

一般来说RPC对RabbitMQ来说很容易。客户端发送请求消息，服务器回复一条响应消息。为了收到响应，我们需要发送一个'回调'队列地址与请求。当我们使用上述'convertSendAndReceive（）'方法时，Spring-amqp的RabbitTemplate处理我们的回调队列。使用RabbitTemplate时，无需进行任何其他设置。有关详细解释，请参阅[请求/回复](http://docs.spring.io/spring-amqp/reference/htmlsingle/" \l "request-reply)消息。

## 消息属性

AMQP 0-9-1协议预先定义了一组14个随附消息的属性。大多数属性很少使用，除了以下内容：

* deliveryMode：将消息标记为persistent（值为2）或transient（任何其他值）。您可能会从第二个教程中记住此属性。
* contentType：用于描述mime类型的编码。例如对于经常使用的JSON编码，将此属性设置为：application / json是一个很好的做法。
* replyTo：通常用来命名一个回调队列。
* correlationId：用于将RPC响应与请求相关联。

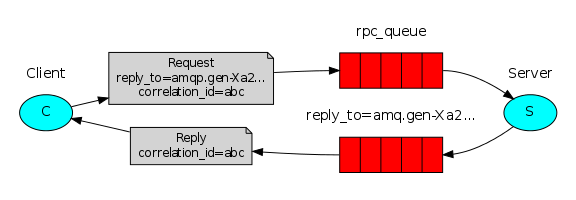
## 相关标识（Correlation Id）

Spring-amqp允许您专注于您正在使用的消息样式，并隐藏支持此样式所需的消息管道的详细信息。例如，通常，本机客户端将为每个RPC请求创建一个回调队列。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回调队列。

这引发了一个新问题，在该队列中收到响应，响应所属的请求不清楚。那就是在使用correlationId属性的时候 。Spring-amqp为每个请求自动设置唯一的值。此外，它还处理使用正确的correlationID匹配响应的细节。

spring-amqp使rpc样式更容易的一个原因是有时您可能想忽略回调队列中的未知消息，而不是失败。这是由于在服务器端发生竞争条件的可能性。虽然不太可能，RPC服务器可能会在发送答复之后，但在发送请求的确认消息之前死亡。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，RPC应该理想地是幂等的。

## 概要



我们的RPC将像这样工作：

* Tut6Config将设置一个新的DirectExchange和一个客户端
* 客户端将利用convertSendAndReceive传递交换名称，routingKey和消息。
* 请求被发送到rpc\_queue（“tut.rpc”）队列。
* RPC 工作线程正在等待队列上的请求。当请求出现时，它执行任务，并使用replyTo字段中的队列将结果发送回客户端。
* 客户端等待回呼队列中的数据。当信息出现时，它检查correlationId属性。如果它与请求中的值相匹配，则返回对应用程序的响应。再次，这是通过RabbitTemplate自动完成的。

## 把它们放在一起

斐波纳契任务是一个@RabbitListener，定义为：

|  |
| --- |
| private static int fib(int n) {  if (n == 0) return 0;  if (n == 1) return 1;  return fib(n-1) + fib(n-2);  } |

我们声明我们的fibonacci功能。它只假定有效的正整数输入。（不要指望这个工作的大数字，这可能是最慢的递归实现）。

我们的Tut6Config的代码 如下所示：

|  |
| --- |
| **package** com.example.rabbitmq.tut6;  **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;  */\*\*  \* Author: 王俊超  \* Date: 2017-06-17 22:15  \* All Rights Reserved !!!  \*/* @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”。它还使用两个bean设置“客户端”配置文件;

1. 我们正在使用的DirectExchange，
2. Tut6Client本身。

我们还使用三个bean，“tut.rpc.requests”队列，与客户端交换器匹配的DirextExchange以及使用“rpc”路由密钥从队列到交换机的绑定来配置“服务器”的配置文件。

服务器代码相当简单：

像往常一样，我们开始用@RabbitListener注释我们的接收方法，并定义其监听的队列。

我们的fibanacci方法使用有效载荷参数调用fib（）并返回结果

我们的RPC客户端的代码Tut6Server.java：

|  |
| --- |
| **package** com.example.rabbitmq.tut6;  **import** org.springframework.amqp.rabbit.annotation.RabbitListener;  */\*\*  \* Author: 王俊超  \* Date: 2017-06-17 22:15  \* All Rights Reserved !!!  \*/* **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**));  } } |

客户端代码Tut6Client 与服务器一样简单：

* 我们自动连接Tut6Config中定义的RabbitTemplate和DirectExchange bean。
* 我们使用参数交换名称，路由密钥和消息调用template.convertSendAndReceive。
* 我们打印结果

客户端的请求非常简单的：

|  |
| --- |
| **package** com.example.rabbitmq.tut6;  **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;  */\*\*  \* Author: 王俊超  \* Date: 2017-06-17 22:15  \* All Rights Reserved !!!  \*/* **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** + **")"**);  *// 设置发送并且接收* Integer response = (Integer) **template**.convertSendAndReceive(**exchange**.getName(), **"rpc"**, **start**++);  System.***out***.println(**" [.] Got '"** + response + **"'"**);  } } |

## 运行

先运行服务器，再运行接收器

启动服务器

|  |
| --- |
| --spring.profiles.active=rpc,server --tutorial.client.duration=6000 |

运行接收器

|  |
| --- |
| --spring.profiles.active=rpc,client --tutorial.client.duration=6000 |

## 说明

这里提出的设计不是RPC服务的唯一可能的实现，而是具有一些重要的优点：

* 如果RPC服务器太慢，可以通过运行另一个RPC服务器进行扩展。尝试在新的控制台中运行第二个RPCServer。
* 在客户端，RPC需要发送和接收一条消息。不需要像queueDeclare这样的同步调用 。因此，RPC客户端只需要一个网络往返单个RPC请求。

我们的代码仍然非常简单，不会尝试解决更复杂（但重要的）问题，例如：

* 如果没有服务器运行，客户端应该如何反应？
* 客户端是否需要RPC的某种超时时间？
* 如果服务器发生故障并引发异常，应该将其转发给客户端？
* 在处理之前防止无效的传入消息（例如检查边界，类型）。