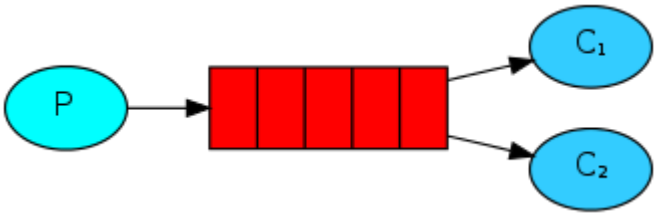
03-rabbitmq-工作队列

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。

工作队列



在第一个教程中，我们编写了程序来发送和接收来自命名队列的消息。在这一个中，我们将创建一个工作队列，用于在多个工作人员之间分配耗时的任务。

工作队列（又称：任务队列）背后的主要思想是避免立即执行资源密集型，并且必须等待完成的任务。相反，我们安排任务在后续完成。我们将任务封装成 消息，并将其发送到队列。在后台运行的工作进程将弹出任务并最终执行作业。当你运行很多工作进程时，这些任务将在它们之间共享。

这个概念在 Web 应用程序中特别有用，尤其适用在短时间 HTTP 请求窗口中无法处理复杂的任务。

准备

在本教程的前面部分，我们发送了一个包含“Hello World!”的消息。现在我们将发送的的字符串代表复杂任务。我们没有一个现实世界的任务，比如图像被调整大小，或者是要渲染的 pdf 文件来模拟，所以假设我们很忙 - 通过使用 Thread.sleep()函数来假冒它。我们将把字符串中的点数作为其复杂度; 每个点都将占“工作”的一秒钟。例如，由 Hello ...描述的假任务将需要三秒钟。

我们将稍微修改我们前面的例子中的 Send.java 代码，以允许从命令行发送任意消息。这个程序会将任务安排到我们的工作队列中，所以让我们命名为 NewTask.java：

```
String message = getMessage(argv);

channel.basicPublish("", "hello", null, message.getBytes());
System.out.println(" [x] Sent '" + message + "'");
```

有些帮助信息从命令行参数获取：

```
private static String getMessage(String[] strings){
    if (strings.length < 1)
        return "Hello World!";
    return joinStrings(strings, " ");
}

private static String joinStrings(String[] strings, String delimiter) {
    int length = strings.length;
    if (length == 0) return "";
    StringBuilder words = new StringBuilder(strings[0]);
    for (int i = 1; i < length; i++) {
        words.append(delimiter).append(strings[i]);
    }
    return words.toString();
}
```

我们的老 Recv.java 程序也需要进行一些更改：它需要为邮件正文中的每个点假一次工作。它将处理传递的消息并执行任务，所以让我们称之为 Worker.java：

```
final Consumer consumer = new DefaultConsumer(channel) {
  @Override
  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
properties, byte[] body) throws IOException {
    String message = new String(body, "UTF-8");

    System.out.println(" [x] Received '" + message + "'");
    try {
      doWork(message);
    } finally {
      System.out.println(" [x] Done");
    }
  }
};
boolean autoAck = true; // acknowledgment is covered below
channel.basicConsume(TASK_QUEUE_NAME, autoAck, consumer);
```

我们假模拟执行时间的任务：

```
private static void doWork(String task) throws InterruptedException {
    for (char ch: task.toCharArray()) {
        if (ch == '.') Thread.sleep(1000);
    }
}
```

循环调度

使用任务队列的优点之一是能够轻松地并行工作。如果我们正在建立积压的工作，我们可以增加更多的工作人员，这样可以轻松扩展。

首先，我们尝试在同一时间运行两个 worker 实例。他们都会从队列中获取消息，但是究竟如何？让我们来看看。

你需要三个控制台打开。两个将运行工作程序。这些控制台将是我们两个消费者 C1 和 C2。

在第三个我们将发布新的任务。一旦您开始使用消费者。

默认情况下，RabbitMQ 将按顺序将每条消息发送给下一个消费者。平均每个消费者将获得相同数量的消息。这种分发消息的方式叫做循环（round-robin）。与三名或更多的工作人员一起尝试。

消息确认

执行任务可能需要几秒钟。你可能会想，如果一个消费者开始一个长期的任务，并且仅仅部分地完成它，就会发生什么。使用我们当前的代码，一旦 RabbitMQ 向客户发送消息，它立即将其从内存中删除。在这种情况下，如果你杀死一个工作进程，我们将丢失正在处理的消息。我们还会丢失所有发送给该特定工作人员但尚未处理的消息。

但是我们不想失去任何任务。如果一个工作进程死亡，我们希望把这个任务交给另一个工作人员。

为了确保消息永远不会丢失，RabbitMQ 支持消息确认。从消费者发送一个确认信息（告示）告诉 RabbitMQ 已经收到，处理了特定的消息，并且 RabbitMQ 可以自由删除它。

如果消费者死机（其通道关闭，连接关闭或 TCP 连接丢失），而不发送确认信息，RabbitMQ 将会明白消息未被完全处理并将重新排队。如果同时有其他消费者在线，则会迅速将其重新提供给另一个消费者。这样就可以确保没有消息丢失，即使工作人员偶尔也会死亡。

没有任何消息超时; 当消费者死亡时，RabbitMQ 将重新发送消息。即使处理消息需要非常长的时间，这很好。

消息确认默认情况下打开。在前面的例子中，我们通过 autoAck = true 标志明确地将它们关闭。现在是一旦完成任务，将此标志设置为 false，并向工作人员发送正确的确认。

```
channel.basicQos(1); // accept only one unack-ed message at a time (see below)

final Consumer consumer = new DefaultConsumer(channel) {
  @Override
  public void handleDelivery(String consumerTag, Envelope envelope, AMQP.BasicProperties
properties, byte[] body) throws IOException {
    String message = new String(body, "UTF-8");

    System.out.println(" [x] Received '" + message + "'");
    try {
      doWork(message);
    } finally {
```

```
      System.out.println(" [x] Done");
      channel.basicAck(envelope.getDeliveryTag(), false);
    }
  }
};
boolean autoAck = false;
channel.basicConsume(TASK_QUEUE_NAME, autoAck, consumer);
```

使用这个代码，我们可以确定即使在处理消息时，使用 CTRL + C 杀死一个工作进程，也不会丢失任何东西。工作进程死亡之后不久，所有未确认的消息将被重新发送。

忘记确认

错过 basicAck 是一个常见的错误。这是一个容易的错误，但后果是严重的。当您的客户端退出（可能看起来像随机重新传递）时，消息将被重新传递，但是 RabbitMQ 将会消耗越来越多的内存，因为它将无法释放任何未包含的消息。

为了调试这种错误，您可以使用 rabbitmqctl 打印 messages_unacknowledged 字段：

sudo rabbitmqctl list_queues name message_ready messages_unacknowledgedged

在 Windows 上，删除 sudo：

rabbitmqctl.bat list_queues name message_ready messages_unacknowledgedged

消息持久化

我们已经学会了如何确保即使消费者死亡，任务也不会丢失。但是如果 RabbitMQ 服务器停止，我们的任务仍然会丢失。

当 RabbitMQ 退出或崩溃时，它会忘记队列和消息，除非你不告诉它。需要两件事来确保消息不会丢失：我们需要将队列和消息标记为持久。

首先，我们需要确保 RabbitMQ 不会失去我们的队列。为了这样做，我们需要将其声明为持久的：

```
boolean durable = true ;
channel.queueDeclare（“hello”，durable，false，false，null）;
```

虽然这个命令本身是正确的，但是在我们目前的设置中是不行的。这是因为我们已经定义了一个非持久化的名为 hello 的队列。RabbitMQ 不允许您重新定义具有不同参数的现有队列，并会向尝试执行此操作的任何程序返回错误。但是有一个快速的解决方法 - 让我们用不同的名称声明一个队列，例如 task_queue：

```
boolean durable = true;
channel.queueDeclare("task_queue", durable, false, false, null);
```

这个 queueDeclare 更改需要应用于生产者和消费者代码。

在这一点上，我们确信，即使 RabbitMQ 重新启动，task_queue 队列也不会丢失。现在我们需要通过将 MessageProperties（实现 BasicProperties）设置为值 PERSISTENT_TEXT_PLAIN 来标记我们的消息。

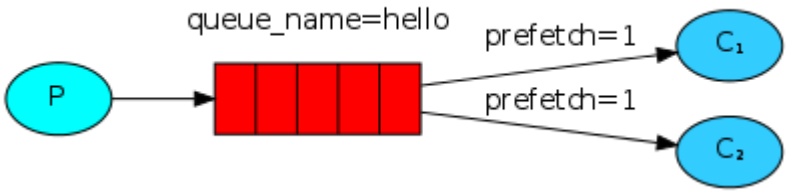
注意消息持久性

将消息标记为持久性不能完全保证消息不会丢失。虽然它告诉 RabbitMQ 将消息保存到磁盘，但是当 RabbitMQ 接受消息并且尚未保存消息时，仍然有一个很短的时间窗口。此外，RabbitMQ 不会对每个消息执行 fsync（2） - 它可能只是保存到缓存中，而不是真正写入磁盘。持久性保证不强，但对我们的简单任务队列来说已经足够了。如果您需要更强大的保证，那么您可以使用发布者确认。

公平调度

您可能已经注意到，调度仍然无法正常工作。例如在两个工人的情况下，当所有奇怪的信息都很重，甚至信息很轻的时候，一个工作人员将不断忙碌，另一个工作人员几乎不会做任何工作。那么，RabbitMQ 不知道什么，还会平均分配消息。

这是因为当消息进入队列时，RabbitMQ 只会分派消息。它不看消费者的未确认消息的数量。它只是盲目地向第 n 个消费者发送每个第 n 个消息。



为了打破这种方式，我们可以使用 basicQos 方法与 prefetchCount = 1 设置。这告诉 RabbitMQ 不要一次给一个工作者多个消息。或者换句话说，在处理并确认前一个消息之前，不要向工作进程发送新消息。相反，它将发送到下一个还不忙的工作进程。

```
int prefetchCount = 1 ;
channel.basicQos（prefetchCount）;
```

注意队列大小

如果所有的工人都忙，你的队列可以填满。你会想要注意的是，也许增加更多的工人，或者有其他的策略

完整代码

我们的 NewTask.java 类的最终代码：

```
package com.example.rabbitmq;

import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.MessageProperties;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

/**
 * Author: 王俊超
 * Date: 2017-06-09 08:08
 * All Rights Reserved !!!
 */
public class NewTask {
    private static final String TASK_QUEUE_NAME = "task_queue";

    public static void main(String[] args) throws IOException, TimeoutException {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        Connection connection = factory.newConnection();
        Channel channel = connection.createChannel();

        channel.queueDeclare(TASK_QUEUE_NAME, true, false, false, null);

        String message = getMessage(args);

        channel.basicPublish("", TASK_QUEUE_NAME,
                MessageProperties.PERSISTENT_TEXT_PLAIN,
                message.getBytes("UTF-8"));
        System.out.println(" [x] Sent '" + message + "'");

        channel.close();
        connection.close();
    }

    private static String getMessage(String[] strings) {
        if (strings.length < 1) {
            return "Hello World!";
        }
        return joinStrings(strings, " ");
```

```
    }

    private static String joinStrings(String[] strings, String delimiter) {
        int length = strings.length;
        if (length == 0) {
            return "";
        }
        StringBuilder words = new StringBuilder(strings[0]);
        for (int i = 1; i < length; i++) {
            words.append(delimiter).append(strings[i]);
        }
        return words.toString();
    }
}
```

和我们的 Worker.java：

```
package com.example.rabbitmq;

import com.rabbitmq.client.*;

import java.io.IOException;
import java.util.concurrent.TimeoutException;

/**
 * Author: 王俊超
 * Date: 2017-06-09 08:02
 * All Rights Reserved !!!
 */
public class Worker {
    public static final String TASK_QUEUE_NAME = "task_queue";

    public static void main(String[] args) throws IOException, TimeoutException {
        ConnectionFactory factory = new ConnectionFactory();
        factory.setHost("localhost");
        final Connection connection = factory.newConnection();
        final Channel channel = connection.createChannel();

        channel.queueDeclare(TASK_QUEUE_NAME, true, false, false, null);
        System.out.println(" [*] Waiting for messages. To exit press CTRL+C");

        // 指定从消息通道中每次取的消息数量
        channel.basicQos(1);

        final Consumer consumer = new DefaultConsumer(channel) {
            @Override
            public void handleDelivery(String consumerTag, Envelope envelope,
```

```
                AMQP.BasicProperties properties, byte[] body) throws IOException
{
            String message = new String(body, "UTF-8");

            System.out.println(" [x] Received '" + message + "'");
            try {
                doWork(message);
            } finally {
                System.out.println(" [x] Done");
                channel.basicAck(envelope.getDeliveryTag(), false);
            }
        }
    };

    channel.basicConsume(TASK_QUEUE_NAME, false, consumer);
  }

  private static void doWork(String task) {
    for (char ch : task.toCharArray()) {
      if (ch == '.') {
        try {
          Thread.sleep(1000);
        } catch (InterruptedException _ignored) {
          Thread.currentThread().interrupt();
        }
      }
    }
  }
}
```

使用消息确认和 prefetchCount 可以设置工作队列。即使 RabbitMQ 重新启动，耐久性选项也让任务生存下去。

运行

先运行接收者，需要添加运行参数：--spring.profiles.active=hello-world,receiver

再运行发送者，需要添加运行参数：--spring.profiles.active=hello-world,sender