Getting Started with RabbitMQ in Java

dzone.com/articles/getting-started-rabbitmq-java

RabbitMQ is a popular message broker typically used for building integration between applications or different components of the same application using messages. This post is a very basic introduction on how to get started using RabbitMQ and assumes you already have setup the rabbitmq server.

RabbitMQ is written in Erlang and has drivers/clients available for most major languages. We are using Java for this post therefore we will first get hold of the java client. The maven dependency for the java client is given below.

```
<dependency>
        <groupId>com.rabbitmq</groupId>
        <artifactId>amqp-
client</artifactId>
        <version>3.0.4</version>
</dependency>
```

While message brokers such as RabbitMQ can be used to model a variety of schemes such as one to one message delivery or publisher/subscriber, our application will be simple enough and have two basic components, a single producer, that will produce a message and a single consumer that will consume that message.

In our example, the producer will produce a large number of messages, each message carrying a sequence number while the consumer will consume the messages in a separate thread.

The EndPoint Abstract class:

Let’s first write a class that generalizes both producers and consumers as ‘endpoints’ of a queue. Whether you are a producer or a consumer, the code to connect to a queue remains the same therefore we can generalize it in this class.

```
package co.syntx.examples.rabbitmq;

import java.io.IOException;

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

/**
 * Represents a connection with a queue
 * @author syntx
 *
 */
public abstract class EndPoint{

    protected Channel channel;
    protected Connection connection;
    protected String endPointName;

    public EndPoint(String endpointName) throws IOException{
         this.endPointName = endpointName;

         //Create a connection factory
         ConnectionFactory factory = new ConnectionFactory();

         //hostname of your rabbitmq server
         factory.setHost("localhost");

         //getting a connection
         connection = factory.newConnection();

         //creating a channel
         channel = connection.createChannel();

         //declaring a queue for this channel. If queue does not exist,
         //it will be created on the server.
         channel.queueDeclare(endpointName, false, false, false, null);
    }


    /**
     * Close channel and connection. Not necessary as it happens implicitly any
way.
     * @throws IOException
     */
     public void close() throws IOException{
         this.channel.close();
         this.connection.close();
     }
}
```

The Producer:

The producer class is what is responsible for writing a message onto a queue. We are using Apache Commons

Lang to convert a Serializable java object to a byte array. The maven dependency for commons lang is

```
<dependency>
 <groupId>commons-lang</groupId>
 <artifactId>commons-lang</artifactId>
 <version>2.6</version>
</dependency>
```

```
package co.syntx.examples.rabbitmq;

import java.io.IOException;
import java.io.Serializable;

import org.apache.commons.lang.SerializationUtils;


/**
 * The producer endpoint that writes to the queue.
 * @author syntx
 *
 */
public class Producer extends EndPoint{

 public Producer(String endPointName) throws IOException{
  super(endPointName);
 }

 public void sendMessage(Serializable object) throws IOException {
     channel.basicPublish("",endPointName, null,
SerializationUtils.serialize(object));
 }
}
```

The Consumer:

The consumer, which can be run as a thread, has callback functions for various events, most important of which is the availability of a new message.

```
package co.syntx.examples.rabbitmq;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import org.apache.commons.lang.SerializationUtils;

import com.rabbitmq.client.AMQP.BasicProperties;
import com.rabbitmq.client.Consumer;
import com.rabbitmq.client.Envelope;
import com.rabbitmq.client.ShutdownSignalException;


/**
```

```
 * The endpoint that consumes messages off of the queue. Happens to be runnable.
 * @author syntx
 *
 */
public class QueueConsumer extends EndPoint implements Runnable, Consumer{

 public QueueConsumer(String endPointName) throws IOException{
  super(endPointName);
 }

 public void run() {
  try {
   //start consuming messages. Auto acknowledge messages.
   channel.basicConsume(endPointName, true,this);
  } catch (IOException e) {
   e.printStackTrace();
  }
 }

 /**
  * Called when consumer is registered.
  */
 public void handleConsumeOk(String consumerTag) {
  System.out.println("Consumer "+consumerTag +" registered");
 }

 /**
  * Called when new message is available.
  */
 public void handleDelivery(String consumerTag, Envelope env,
   BasicProperties props, byte[] body) throws IOException {
  Map map = (HashMap)SerializationUtils.deserialize(body);
     System.out.println("Message Number "+ map.get("message number") + "
received.");

 }

 public void handleCancel(String consumerTag) {}
 public void handleCancelOk(String consumerTag) {}
 public void handleRecoverOk(String consumerTag) {}
 public void handleShutdownSignal(String consumerTag, ShutdownSignalException arg1)
{}
}
```

Putting it together:

In our driver class, we start a consumer thread and then proceed to generate a large number of messages that will be consumed by the consumer.

```
package co.syntx.examples.rabbitmq;

import java.io.IOException;
import java.sql.SQLException;
import java.util.HashMap;

public class Main {
 public Main() throws Exception{

  QueueConsumer consumer = new QueueConsumer("queue");
  Thread consumerThread = new Thread(consumer);
  consumerThread.start();

  Producer producer = new Producer("queue");

  for (int i = 0; i < 100000; i++) {
   HashMap message = new HashMap();
   message.put("message number", i);
   producer.sendMessage(message);
   System.out.println("Message Number "+ i +" sent.");
  }
 }

 /**
  * @param args
  * @throws SQLException
  * @throws IOException
  */
 public static void main(String[] args) throws
Exception{
   new Main();
 }
}
```

The Integration Zone is brought to you in partnership with CA Technologies. Create app backends instantly with REST APIs and reactive logic using CA Live API Creator .

Published at DZone with permission of Faheem Sohail , DZone MVB .

Opinions expressed by DZone contributors are their own.