Ausarbeitung Message Queue RabbtiMQ

Was ist eine RabbitMQ?

- Stabiler Datentransfer für Anwendungen
- Einfach zu gebrauchen
- Läuft auf allen gängigen Betriebssystemen
- Unterstützt eine Vielzahl von Entwicklerplattformen
- Unterstützt Open Source und kommerziell

Arten von RabbitMQ

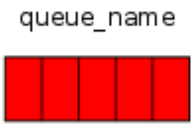
- Hello World
- Work queues
- Publish/Subscribe
- Routing
- Topics
- RPC

HELLO WORLD

- RabbitMQ ist ein Nachrichtenverteiler
- Es akzeptiert Nachrichten vom Hersteller und übermittelt diese an den Empfänger
- Es leitet, puffert und erhält die Nachrichten in Bezug auf die erstellten Regeln
- RabbitMQ und Nachrichtenübermittlung im Allgemeinen benutzen eine Fachsprache
- Herstellen bedeutet Senden
- Ein Programm das Nachrichten sendet ist ein Hersteller
- Wird dargestellt durch „P“
- Eine Warteschlange ist der Name für eine Mailbox
- Nachrichten können nur in einer Warteschlange gespeichert werden (nicht in RabbitMQ und Anwendungen)
- Keine Speichergrenzen für Warteschlangen
- Viele Hersteller können Nachrichten an eine Warteschlange schicken

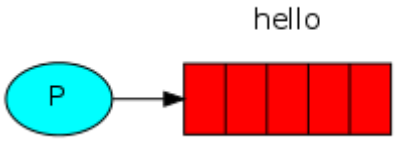
- Viele Empfänger können versuchen die Daten von einer Warteschlage zu empfangen

- Der Name einer Warteschlanger steht darüber:



- Ein Empfänger ist ein Programm das darauf wartet Nachrichten zu empfangen
- Wird mit „C“ dargestellt
- Hersteller, Empfänger und Vermittler müssen nicht in der gleichen Maschine sein

Sending



- Den Sender nennen wir Send und den Empfänger Recv
- Der Sender verbindet sich mit dem RabbitMQ
- Sendet eine einzelne Nachricht und beendet es dann
    - import com.rabbitmq.client.ConnectionFactory;
    - import com.rabbitmq.client.Connection;
    - import com.rabbitmq.client.Channel;

- Erstelle eine Klasse und benenne die Warteschlange

```
 public class Send {
private final static String QUEUE_NAME = "hello";

public static void main(String[] argv)
   throws java.io.IOException {
   ...
 }
}
```

- Dann kann eine Verbindung zum Server erstellt werden

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();
```

- Anschlussbuchse und kümmert sich um die Protokollversion, Aushandlung und Authentifizierung usw.
- verbinden mit einen Vermittler auf der lokalen Maschine - daher der Localhost.

- Wenn wir uns mit einem anderen Vermittler auf einem anderen Computer verbinden möchten brauchen wir einfach den Namen oder die IP-Adresse einzugeben.

- Zum Senden müssen wir eine Warteschlage deklarieren an die man Senden kann

- Dann können wir die Nachricht an die Warteschlange veröffentlichen:

```
channel.queueDeclare(QUEUE_NAME, false, false, false, null);
String message = "Hello World!";
channel.basicPublish("", QUEUE_NAME, null, message.getBytes());
System.out.println(" [x] Sent '" + message + "'");
```

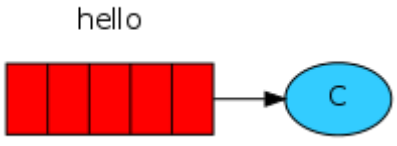
- Die Deklaration einer Warteschlange wird nur dann erstellt, wenn nicht bereits eine vorhanden ist.

- Schließen des Kanals und der Verbindung

```
channel.close ();
connection.close ();
```

Receiving (Empfang)

- Der Empfänger erhält eine Nachricht vom RabbitMQ, die einzelnen Nachrichten können veröffentlich werden und bleiben ständig erhalten, um Nachrichten zu empfangen und auszugeben




```
import com.rabbitmq.client.ConnectionFactory;
import com.rabbitmq.client.Connection;
import com.rabbitmq.client.Channel;
import com.rabbitmq.client.Consumer;
import com.rabbitmq.client.DefaultConsumer;
```

Die extra DefaultConsumer Klasse ist die Umsetzung der Consumer-Schnittstelle diese Schnittstelle wird verwendet, um die Nachrichten zu puffern um diese auf den Server zu schieben.

Wir öffnen eine Verbindung und einen Kanal, man deklariert der Warteschlange von wo gesendet wird und überprüft, ob es mit der Warteschlange übereinstimmt.

```
public class Recv {
  private final static String QUEUE_NAME = "hello";

  public static void main(String[] argv)
      throws java.io.IOException,
             java.lang.InterruptedException {

    ConnectionFactory factory = new ConnectionFactory();
    factory.setHost("localhost");
    Connection connection = factory.newConnection();
    Channel channel = connection.createChannel();

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

Vorher möchte man Sichterstellen, ob die Warteschlange vorhanden ist

Server wird angewiesen, die Nachrichten aus der Warteschlange zu liefern

Da die Nachrichten asynchron ausgegeben werden, wird eine Bestätigung in Form eines Objektes gesendet, welches die Nachrichten puffert bis man bereit ist diese zu nutzen

```
    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 + "'");
      }
    };
    channel.basicConsume(QUEUE_NAME, true, consumer);
```