Use JMS for Request-Reply Communication

In this practical you will learn how implement request-reply communication using JMS messages. This practical consists of the following steps:

Table of Contents

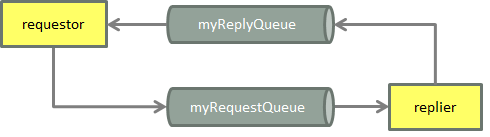
[1. Make necessary channels. 1](#_Toc526235203)

[2. Match replies with requests. 1](#_Toc526235204)

[2. Deal with multiple requestors. 2](#_Toc526235205)

# 1. Make necessary channels.

Note that request-reply communication with JMS requires two channels:



# 2. Match replies with requests.

By default, the requestor cannot match reply messages to request messages. Implement pattern Correlation Identifier in order to enable the requestor application to math each reply to the right request.



Implementing Correlation Identifier in JMS is trivial. The replier application must sent the JMSCorrelationID to be equal to JMSMessageID of the request message:

**public** **void** sendReply(MyRequest request, MyReply reply){

Message requestMsg = …

Message replyMsg = …

replyMsg.setJMSCorrelationID(requestMsg.getJMSMessageID());

}

Note that, in order to fully implement the Correlation Identifier you will need some more programming (e.g. serialization, maybe using some HashMaps, etc.).

When you are finished, it should be possible to match the replies to the right requests, even f the replies are sent in different order than the requests:

|  |  |
| --- | --- |
|  |  |

# 2. Deal with multiple requestors.

Each requestor must have its own reply queue. As a consequence, the replier must send reply messages to multiple reply queues: each reply must be sent to the reply queue of the requestor application.



Implementing Return Address in JMS is trivial. Each requestor application must include its reply queue name in the request message:

**public** **void** sendRequest(MyRequest request){

Message requestMsg = …

Destination receiverDestination = …

requestMsg.setJMSReplyTo(receiverDestination);

producer.sendMessage(requestMsg);

}

producer = session.createProducer(**null**); //create producer with a NULL destination

...

**public** **void** sendReply(MyRequest request, MyReply reply){

Message requestMsg = …

Message replyMsg = …

Destination returnAddress = requestMsg.getJMSReplyTo();

producer.send(replyMsg, returnAddress);

}

Note that, in order to fully implement the ReturnAddress you will need some more programming (e.g. serialization, maybe using some HashMaps, etc.).

When you are finished, each requestor should receive its replies. Adding a new requestor does not require any changes in the replier. It should be even possible to add a new requestor while the replier is being executed.

|  |  |
| --- | --- |
|  |  |
|  |