

MidEng 7.3 Message Oriented Middleware

Einführung

Diese Übung soll die Funktionsweise und Implementierung von einer Message Oriented Middleware (MOM) mit Hilfe des **Frameworks Apache Kafka** demonstrieren. **Message Oriented Middleware (MOM)** ist neben InterProcessCommunication (IPC), Remote Objects (RMI) und Remote Procedure Call (RPC) eine weitere Möglichkeit um eine Kommunikation zwischen mehreren Rechnern umzusetzen.

Die Umsetzung basiert auf einem praxisnahen Beispiel eines Warenlagers. Die Zentrale des Warenlagers möchte jede Stunde den aktuellen Lagerstand aller Lagerstandorte abfragen.

Mit diesem Ziel soll die REST-Applikation aus MidEng 7.1 Warehouse REST and Dataformats bei einem entsprechenden Request `http://warehouse/send` die Daten (JSON oder XML) in eine Message Queue der Zentrale übertragen. In regelmäßigen Abständen werden alle Message Queues der Zentrale abgefragt und die Daten aller Standorte gesammelt.

Die gesammelten Lagerstände werden über eine REST-Schnittstelle (in XML oder JSON) dem Berichtswesen des Managements zur Verfügung gestellt.

1.1 Ziele

Das Ziel dieser Übung ist die Implementierung einer Kommunikationsplattform für Warenlager. Dabei erfolgt ein Datenaustausch von mehreren Lagerstandorten mit der Zentrale unter Verwendung einer Message Oriented Middleware (MOM). Die einzelnen Daten des Warenlagers sollen an die Zentrale übertragen werden. Es sollen **nachrichtenbasierten Protokolle mit Message Queues** verwendet werden. Durch diese lose Kopplung kann gewährleistet werden, dass in Zukunft weitere Standorte hinzugefügt bzw. Kooperationspartner eingebunden werden können.

Für die REST-Schnittstelle in der Zentrale müssen die Datenstrukturen der einzelnen Lagerstandorte zusammengefasst werden. Um die Datenintegrität zu garantieren, sollen jene Daten, die mit der Middleware übertragen werden in einer LOG-Datei abgespeichert werden.

1.2 Voraussetzungen

- Grundlagen Architektur von verteilten Systemen

- Grundlagen zur nachrichtenbasierten Systemen / Message Oriented Middleware
- Verwendung des Message Brokers Apache Kafka
- Verwendung der XML- oder JSON Datenstruktur des Wahllokals
- Verwendung der Demo-Applikation MOMApplication (inklusive MOMReceiver und MOMSender) (siehe Repo)

1.3 Aufgabenstellung

Implementieren Sie die Lager-Kommunikationsplattform mit Hilfe des Java Message Service. Verwenden Sie Apache Kafka (<https://kafka.apache.org>) als Message Broker Ihrer Applikation. Das Programm soll folgende Funktionen beinhalten:

- Installation von Apache Kafka in der Zentrale.
- Jeder Lagerstandort hat eine Message Queue mit einer ID am zentralen Rechner.
- Jeder Lagerstandort legt in regelmässigen Abständen die Daten des Lagers in der Message Queue ab.
- Bei einer erfolgreichen Übertragung sendet die Zentrale die Nachricht "SUCCESS" an den Lagerstandort retour.
- Der zentrale Rechner fragt in regelmässigen Abständen alle Message Queues ab.
- Der Zentralrechner fügt alle Daten aller Lagerstandorte zusammen und stellt diese an einer REST Schnittstelle im JSON/XML Format zur Verfügung.

1.4 Demo Applikation

- Installation und starten des Message Broker Apache Kafka (Container)
[Apache Kafka](#)
- Erstellen einer Message Queue "quickstart-events" (Terminal/Container)

```
cd /opt/kafka  
bin/kafka-topics.sh --create --topic quickstart-events --bootstrap-server  
localhost:9092
```
- Senden von Nachrichten (via Terminal)

```
bin/kafka-console-producer.sh --topic quickstart-events --bootstrap-server  
localhost:9092  
> Hallo Spencer, hier ist Nachricht 1.  
> Hallo Spencer, hier ist Nachricht 2  
> Hallo Spencer, hier ist Nachricht 3.
```

- Lesen von Nachrichten (via Terminal)

```
bin/kafka-console-consumer.sh --topic quickstart-events --from-beginning --bootstrap-server localhost:9092
```

1.4.1 warehouse_demo

Demo 1 beinhaltet eine Implementierung, die alle Einzelschritte zur Implementierung von Java und JMS beinhaltet und übersichtlich darstellt.

- Starten der Demo Applikation `gradle clean bootRun`
- Senden einer Nachricht <http://localhost:8080/send?message=Hallo%20Spencer>
- Empfang der Nachricht auf der Konsole Hallo Spencer

1.5 Bewertung

- Gruppengröße: 1 Person
- Abgabemodus: per Protokoll, bei EK kann ein Abgabegespräch erforderlich sein
- Anforderungen "**Grundlagen**"
 - Implementierung der Kommunikation zwischen **EINEM** Lagerstandort und dem Zentralrechner (JMS Queue)
 - Ausgabe der empfangenen Daten am Zentralrechner (Konsole oder Log-Datei)
 - Beantwortung der Fragestellungen
- Anforderungen "**Erweiterte Grundlagen**"
 - Zusammensetzung der Daten aller Lagerstandorte in einer zentralen JSON/XML-Struktur
 - Implementierung der REST Schnittstelle am Zentralrechner
- Erweiterte Anforderungen "**Vertiefung**"
 - Implementierung der Kommunikation mit **MEHREREN** Lagerstandorte und dem Zentralrechner
 - Logging der Daten bei aller Lagerstandorte und dem Zentralrechner
 - Rückmeldung des Ergebnisses der Übertragung vom Zentralrechner an den einzelnen Lagerstandort (JMS Topic)

1.6 Fragestellung für Protokoll

1. Nennen Sie mindestens 4 Eigenschaften der Message Oriented Middleware:

- Asynchrone Nachrichtenübertragung

- Zuverlässige Zustellung
- Unterstützung verteilter Systeme
- Entkopplung von Sender und Empfänger
- Persistenz von Nachrichten

2. Was versteht man unter einer transienten und synchronen Kommunikation?

- **Transiente Kommunikation:** Nachricht nur im Speicher, geht bei Ausfall verloren.
- **Synchrone Kommunikation:** Sender wartet auf Bestätigung vom Empfänger, bis Nachricht verarbeitet ist.

3. Beschreiben Sie die Funktionsweise einer JMS Queue:

- Punkt-zu-Punkt-Kommunikation (P2P)
- Sender legt Nachricht in Queue ab
- Ein Empfänger liest die Nachricht
- Nachricht wird nach erfolgreicher Verarbeitung gelöscht

4. JMS Overview – Beschreiben Sie die wichtigsten JMS Klassen und deren Zusammenhang:

- **ConnectionFactory:** erzeugt Verbindungen zum JMS Provider
- **Connection:** stellt Verbindung zum Provider her
- **Session:** verwaltet Nachrichten, Producer und Consumer
- **Destination:** Ziel für Nachrichten (Queue oder Topic)
- **MessageProducer / MessageConsumer:** senden bzw. empfangen Nachrichten
- **Message:** enthält die Nutzdaten

5. Beschreiben Sie die Funktionsweise eines JMS Topic:

- Publish/Subscribe-Modell
- Nachricht wird an alle angemeldeten Subscriber verteilt
- Jeder Subscriber erhält eine Kopie der Nachricht

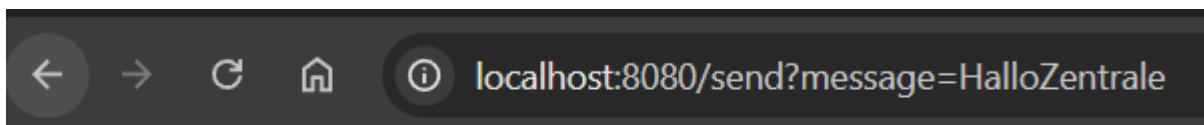
6. Was versteht man unter einem lose gekoppelten verteilten System? Nennen Sie ein Beispiel dazu. Warum spricht man hier von lose?

- Komponenten interagieren über standardisierte Schnittstellen, ohne direkte Abhängigkeit
- **Beispiel:** JMS-basierte Anwendung mit mehreren Services
- **Lose gekoppelt:** Änderungen an einer Komponente beeinflussen andere kaum

GK

Vorgehensweise:

- `WarehouseData` -Klasse in beiden Projekten (Producer & Consumer) implementiert.
- Producer auf Port 8082, Consumer auf Port 8081.
- Producer:
 - Endpoint `/send` erstellt.
 - JSON-Daten (`WarehouseData`) per `KafkaTemplate.send()` an Topic `warehouse-001` gesendet.
 - Rückgabe als JSON via `ResponseEntity<WarehouseData>` an den Client (Browser / Bruno).
- Consumer:
 - `@KafkaListener` auf Topic `warehouse-001`.
 - Empfangene Objekte in `List<WarehouseData>` gespeichert.
 - Logging der empfangenen `WarehouseID`



Message 'HalloZentrale' sent.

```
Read from Message Queue: HalloZentrale  
<=====--> 80% EXECUTING [4m 48s]
```

EK

Vorgehensweise:

- Consumer auf mehreren Topics (warehouse-001 , warehouse-002 , warehouse-003) gleichzeitig hören.
- ErrorHandlingDeserializer für Key und Value konfiguriert:
-

The screenshot shows the Postman interface with the following details:

- Testing** tab is selected.
- POST T1** is the current collection.
- POST** method is selected, pointing to <http://localhost:8080/send>.
- Body** tab is active, showing the following JSON payload:

```
1 {  
2   "warehouseID": "W1",  
3   "warehouseName": "Lager Wien",  
4   "productData": [  
5     {  
6       "productID": "P1",  
7       "productName": "Schraube",  
8       "productCategory": "Bauteile",  
9       "productQuantity": 500,  
10      "productUnit": "Stk"  
11    }  
12  ]  
13 }
```

- Response** tab shows the result of the request:
 - Status: 200 OK
 - Time: 2.40s
 - Size: 20B
 - Content: "Warehouse 'W1' sent."

Vertiefung

Vorgehensweise:

- KafkaTemplate für Feedback an Topic warehouse-feedback eingerichtet.
- Nach Empfang einer Nachricht im Consumer:
 - Warehouse-Daten verarbeiten (Liste speichern, Logging).

- Feedback an Lagerstandort senden (über Kafka) → aktuell auskommentiert wegen Deserialization-Problemen.
- Logging der Feedback-Sendung für Nachverfolgung.

Herausforderungen:

- Feedback schlägt fehl, wenn Producer/Consumer unterschiedliche Packages oder keine korrekten `trusted.packages` konfiguriert sind.
- Fehlerhafte Nachrichten würden Listener stoppen → `ErrorHandlingDeserializer` nötig.

Consumer.yml

```

server:
  port: 8081

spring:
  application:
    name: "Warehouse-Consumer"

kafka:
  bootstrap-servers: localhost:9092

  consumer:
    group-id: myGroup
    key-deserializer:
      org.springframework.kafka.support.serializer.ErrorHandlingDeserializer
      value-deserializer:
        org.springframework.kafka.support.serializer.ErrorHandlingDeserializer

    properties:
      spring.deserializer.key.delegate.class:
        org.apache.kafka.common.serialization.StringDeserializer
      spring.deserializer.value.delegate.class:
        org.springframework.kafka.support.serializer.JsonDeserializer
      spring.json.trusted.packages: "at.ac.tgm.mwali.shared"
      spring.json.value.default.type: at.ac.tgm.mwali.shared.WarehouseData
  
```

```
/ $ cd /opt/kafka  
/opt/kafka $ bin/kafka-topics.sh --create --topic warehouse-001 --bootstrap-server localhost:9092  
Created topic warehouse-001.  
/opt/kafka $ bin/kafka-topics.sh --create --topic warehouse-002 --bootstrap-server localhost:9092  
Created topic warehouse-002.  
/opt/kafka $ bin/kafka-topics.sh --create --topic warehouse-003 --bootstrap-server localhost:9092  
Created topic warehouse-003.  
/opt/kafka $ bin/kafka-topics.sh --create --topic warehouse-feedback --bootstrap-server localhost:9092  
Created topic warehouse-feedback.  
/opt/kafka $ []
```

POST ▾ http://localhost:8082/send

Params	Body *	Headers	Auth *	Vars	Script	Response	Headers 5	Timeline	Tests
Assert	Tests	Docs	Settings		JSON ▾ Prettify	1 ▾ { 2 "warehouseID": "WH-001", 3 "warehouseName": "Vienna Central", 4 "warehouseCity": "Wien", 5 "warehouseCountry": "AT", 6 "productData": [] 7 }	2 "warehouseID": "WH-001", 3 "warehouseName": "Vienna Central", 4 "timestamp": "2025-12-16 15:26:41.195", 5 "warehouseAddress": null, 6 "warehousePostalCode": null, 7 "warehouseCity": "Wien", 8 "warehouseCountry": "AT", 9 "productData": [] 10 }	200 OK	