**Creating SQL Script to execute while application loads**

By default, data.sql scripts are now run before Hibernate is initialized. This aligns the behavior of basic script-based initialization with that of Flyway and Liquibase.

spring.jpa.defer-datasource-initialization=true

**Calling properties from the properties file**

"@ConfigurationProperties(prefix = "sfg.brewery", ignoreUnknownFields = false)" , we can define properties starting with sfg.brewery in application.properties and it will map to field of the class.

Ex : sfg.brewery.beer-inventory-service-host=http://localhost:8082

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Caching**

1.dependency:

<**dependency**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-starter-cache</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>javax.cache</**groupId**>

<**artifactId**>cache-api</**artifactId**>

</**dependency**>

<**dependency**>

<**groupId**>org.ehcache</**groupId**>

<**artifactId**>ehcache</**artifactId**>

</**dependency**>

2.xml file in resource (ehcache.xml)

<**config**

xmlns:jsr107=*'http://www.ehcache.org/v3/jsr107'*

xmlns=*'http://www.ehcache.org/v3'*>

<**service**>

<**jsr107:defaults** enable-management=*"true"* enable-statistics=*"true"*/>

</**service**>

<**cache** alias=*"beerCache"* uses-template=*"config-cache"*/>

<**cache** alias=*"beerListCache"* uses-template=*"config-cache"*/>

<**cache-template** name=*"config-cache"*>

<**expiry**>

<**ttl** unit=*"minutes"*>5</**ttl**>

</**expiry**>

<**resources**>

<**heap**>1</**heap**>

<**offheap** unit=*"MB"*>1</**offheap**>

</**resources**>

</**cache-template**>

</config>

3. application.properties

spring.cache.jcache.config=classpath:ehcache.xml

4. creat a class

*@Configuration*

*@EnableCaching*

public class CacheConfig {

}

5. using in the method

@Cacheable(cacheNames = "beerCache", key = "#beerId",

condition = "#showInventoryOnHand == false ")

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**JaxB dependencies**

Jakarta XML Binding [4.0](https://jakarta.ee/specifications/xml-binding/4.0/) (requires Java SE 11 or >

<**dependency**>

<**groupId**>jakarta.xml.bind</**groupId**>

<**artifactId**>jakarta.xml.bind-api</**artifactId**>

<**version**>4.0.0</**version**>

</**dependency**>

<**dependency**>

<**groupId**>com.sun.xml.bind</**groupId**>

<**artifactId**>jaxb-impl</**artifactId**>

<**version**>4.0.0</**version**>

<**scope**>runtime</**scope**>

</**dependency**>

-----------------------------------------------------------

<**properties**>

<**jaxb.version**>2.3.0</**jaxb.version**>

</**properties**>

<**dependency**>

<**groupId**>javax.xml.bind</**groupId**>

<**artifactId**>jaxb-api</**artifactId**>

<**version**>${jaxb.version}</**version**>

</**dependency**>

<**dependency**>

<**groupId**>com.sun.xml.bind</**groupId**>

<**artifactId**>jaxb-core</**artifactId**>

<**version**>${jaxb.version}</**version**>

</**dependency**>

<**dependency**>

<**groupId**>com.sun.xml.bind</**groupId**>

<**artifactId**>jaxb-impl</**artifactId**>

<**version**>${jaxb.version}</**version**>

</**dependency**>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Plugin to auto clean**

<**plugin**>

<**groupId**>org.springframework.boot</**groupId**>

<**artifactId**>spring-boot-maven-plugin</**artifactId**>

</**plugin**>

<**plugin**>

<**artifactId**>maven-clean-plugin</**artifactId**>

<**executions**>

<**execution**>

<**id**>auto-clean</**id**>

<**phase**>initialize</**phase**>

<**goals**>

<**goal**>clean</**goal**>

</**goals**>

</**execution**>

</**executions**>

</**plugin**>

**Applying Rules to BOM**

<**plugin**>

<**groupId**>org.apache.maven.plugins</**groupId**>

<**artifactId**>maven-enforcer-plugin</**artifactId**>

<**executions**>

<**execution**>

<**id**>enforce-versions</**id**>

<**goals**>

<**goal**>enforce</**goal**>

</**goals**>

<**configuration**>

<**rules**>

<**requireMavenVersion**>

<**version**>[3.6.0,)</**version**>

</**requireMavenVersion**>

<**requireJavaVersion**>

<**version**>11</**version**>

</**requireJavaVersion**>

<**requireReleaseDeps**>

<**onlyWhenRelease**>true</**onlyWhenRelease**>

<**message**>Release builds must not have on

snapshot dependencies

</**message**>

</**requireReleaseDeps**>

</**rules**>

</**configuration**>

</**execution**>

</**executions**>

</**plugin**>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Excluding the specific class during springBoot class loading**

*@SpringBootApplication(****exclude = ClassName.class****)*

public class MsscBreweryApplication {

----

**MySQL DB Connection in properties file**

spring.datasource.url=jdbc:mysql://127.0.0.1:3306/beerservice?useUnicode=true&char

acterEncoding=UTF-8&serverTimezone=UTC

spring.datasource.username=beer\_service

spring.datasource.password=password

spring.jpa.database=mysql

spring.jpa.hibernate.ddl-auto=update

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

**Hikari Connection Pooling**

spring.datasource.hikari.maximum-pool-size=5

best for MqSql

spring.datasource.hikari.data-source-properties.cachePrepStmts=true

spring.datasource.hikari.data-source-properties.prepStmtCacheSize=250

spring.datasource.hikari.data-source-properties.prepStmtCacheSqlLimit=2048

spring.datasource.hikari.data-source-properties.useServerPrepStmts=true

spring.datasource.hikari.data-source-properties.useLocalSessionState=true

spring.datasource.hikari.data-source-properties.rewriteBatchedStatements=true

spring.datasource.hikari.data-source-properties.cacheResultSetMetadata=true

spring.datasource.hikari.data-source-properties.cacheServerConfiguration=true

spring.datasource.hikari.data-source-properties.elideSetAutoCommits=true

spring.datasource.hikari.data-source-properties.maintainTimeStats=false

Enable logging for config troubleshooting

logging.level.org.hibernate.SQL=DEBUG

logging.level.com.zaxxer.hikari.HikariConfig=DEBUG

logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE

JMS

**1.0 Apache ActiveMQ Artemis dependencies and usage vn:172**

By spring initializer it will add only core and client, it only gives ability to talk to servers

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-artemis</artifactId>  
</dependency>

**To run embedded servers we have to add extra dependencies (typical case)**

<dependency>  
 <groupId>org.apache.activemq</groupId>  
 <artifactId>artemis-server</artifactId>  
</dependency>  
<dependency>  
 <groupId>org.apache.activemq</groupId>  
 <artifactId>artemis-jms-server</artifactId>  
</dependency>

Inside main application

ActiveMQServer server = ActiveMQServers.*newActiveMQServer*(new ConfigurationImpl()  
 .setPersistenceEnabled(false)  
 .setJournalDirectory("target/data/journal")  
 .setSecurityEnabled(false)  
 .addAcceptorConfiguration("invm", "vm://0"));  
  
server.start();

***1.1 Hello world Example***

**Create a Message Model**

@Data  
@Builder  
@AllArgsConstructor  
@NoArgsConstructor  
public class HelloWorldMessage implements Serializable {  
  
 static final long *serialVersionUID* = 6056263104526923315L;  
  
 private UUID id;  
 private String message;  
}

**Enabling Task Scheduling**

@EnableScheduling  
@EnableAsync  
@Configuration  
public class TaskConfig {  
  
 @Bean  
 TaskExecutor taskExecutor(){  
 return new SimpleAsyncTaskExecutor();  
 }  
}

Img:1.0

**Message convertor to convert json ↔ object**

import org.springframework.jms.support.converter.\*;

@Configuration  
public class JmsConfig {  
  
 public static final String *MY\_QUEUE* = "my-hello-world";  
  
 @Bean  
 public MessageConverter messageConverter() {  
 MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();  
 converter.setTargetType(MessageType.*TEXT*);  
 converter.setTypeIdPropertyName("\_type"); //need tobe set in order to convert incoming massage to java object (default is "none")  
 return converter;  
 }  
}

Img:1.2

MessageConverter class:

Strategy interface that specifies a converter between Java objects and JMS messages.

Convert a Java object to a JMS Message using the supplied session to create the message object.  
Params: object – the object to convert ,session – the Session to use for creating a JMS Message  
Returns the JMS Message.

MappingJackson2MessageConverter class : Message converter that uses Jackson 2.x to convert messages to and from JSON. Maps an object to a BytesMessage, or to a TextMessage if the targetType is set to MessageType.TEXT. Converts from a TextMessage or BytesMessage to an object.

**Sender**

@RequiredArgsConstructor  
@Component  
public class HelloSender {  
  
 private final JmsTemplate jmsTemplate; //JmsTemplate(for messaging) is pre configured like JdbcTemplate(for db)  
  
 @Scheduled(fixedRate = 2000)  
 public void sendMessage(){  
 System.*out*.println("I'm Sending a message");  
  
 HelloWorldMessage message = HelloWorldMessage  
 .*builder*()  
 .id(UUID.*randomUUID*())  
 .message("Hello World!")  
 .build();  
  
 jmsTemplate.convertAndSend(JmsConfig.*MY\_QUEUE*, message);  
  
 System.*out*.println("Message Sent!");  
 }  
}

**Note:** Jms maintains transactional. If client doesnot confirms receiving the message then that message gets requed and resent.

**Listener**

@Component  
public class HelloMessageListener {  
  
 @JmsListener(destination = JmsConfig.*MY\_QUEUE*)  
 public void listen(@Payload HelloWorldMessage helloWorldMessage,  
 @Headers MessageHeaders headers, Message message){  
 System.*out*.println("I Got a Message!!!!!");  
  
 System.*out*.println(helloWorldMessage);  
  
 // uncomment and view to see retry count in debugger  
 // throw new RuntimeException("foo");  
 }  
}

**O/P:**

I'm Sending a message

Message Sent!

I Got a Message!!!!!

HelloWorldMessage(id=68aa1f77-c379-49a1-acd4-20ccf1bbf229, message=Hello World!)

***1.11 Sending and Receiving Message Example*** vn:180

@RequiredArgsConstructor  
@Component  
public class HelloSender {

“”””””””””

private final ObjectMapper objectMapper;

“”””””””

@Scheduled(fixedRate = 2000)  
 public void sendAndRecieveMessage() throws JMSException {  
  
 HelloWorldMessage message = HelloWorldMessage  
 .*builder*()  
 .id(UUID.*randomUUID*())  
 .message("Hello ")  
 .build();  
  
 Message recievedMessage = jmsTemplate.sendAndReceive(JmsConfig.*MY\_SEND\_RCV\_QUEUE*, new MessageCreator() {  
 @Override  
 public Message createMessage(Session session) throws JMSException {  
 Message helloMessage = null;  
 try {  
 //we have to manually configure converter using objectMapper bcz  
 // our convertor(MessageConverter from JmsConfig.class) doesnot work here  
 helloMessage = session.createTextMessage(objectMapper.writeValueAsString(message));  
 //set property to convert to java Object  
 helloMessage.setStringProperty("\_type","guru.springframework.sfgjms.model.HelloWorldMessage");  
 System.*out*.println("Sending Hello");  
 return helloMessage;  
 }catch (JsonProcessingException e){  
 throw new JMSException(e.getMessage());  
 }  
 }  
 });  
  
 System.*out*.println(recievedMessage.getBody(String.class));  
  
 }  
}

import org.springframework.jms.support.converter.\*;

@Configuration  
public class JmsConfig {

“””””””

public static final String *MY\_SEND\_RCV\_QUEUE* = "replybacktome";

@Bean  
 public MessageConverter messageConverter() {

“””””””””””  
 }  
}

@RequiredArgsConstructor  
@Component  
public class HelloMessageListener {  
  
 private final JmsTemplate jmsTemplate;  
  
 “””””””””””””””  
  
 @JmsListener(destination = JmsConfig.*MY\_SEND\_RCV\_QUEUE*)  
 public void listenForHello(@Payload HelloWorldMessage helloWorldMessage,  
 @Headers MessageHeaders headers, Message message) throws JMSException {  
  
 HelloWorldMessage payloadMsg = HelloWorldMessage  
 .*builder*()  
 .id(UUID.*randomUUID*())  
 .message("World!")  
 .build();  
  
 jmsTemplate.convertAndSend(message.getJMSReplyTo(),payloadMsg);  
  
 }  
}

**O/P:**

Sending Hello

{"id":"9e77dd6f-c3a1-49f7-b46e-99c88b613d85","message":"World!"}

**1.2 ActiveMQ Artemis Docker** vn:181

GitHub Link: <https://github.com/vromero/activemq-artemis-docker>

Docker command to run the image: docker run -it --rm -p 8161:8161 -p 61616:61616 vromero/activemq-artemis

Error!!!

Ports are not available: listen tcp 0.0.0.0:61616: bind: An attempt was made to access a socket in a way forbidden by its access permissions.

Solution**:** In windows as administrator

> net stop winnat

> net start winnat

Port: <http://127.0.0.1:8161>

To use ActiveMQ Artemis from our spring boot app (client) (disable embedded server)

In application.properties

spring.artemis.user=artemis  
spring.artemis.password=simetraehcapa

Img:1.3

**Note**: ActiveMQ is the implementation of Message (I) from javax.jms package.

Example to use Spring **Message** Type

* Easy to swap b/w kafka, rabbit, …

jmsTemplate.convertAndSend((Destination) springMessage.getHeaders().get("jms\_replyTo"), "got it!");

Example to use Jms **Message** Type

jmsTemplate.convertAndSend(jmsMessage.getJMSReplyTo(), payloadMsg);

**1.3 JMS Messaging b/w MicroServices**

***Service: Mssc-beer-service2***

1.create jms config file: refer Img:1.2

@Configuration  
public class JmsConfig {  
  
 public static final String BREWING\_REQUEST\_QUEUE = "brewing-request";

//Convert Message to Object (or vice-versa)  
 “”””””””  
}

2.add ActiveMQ credentials to application.properties file: refer Img:1.3

3.add events

@Data  
@RequiredArgsConstructor  
@Builder  
public class BeerEvent implements Serializable {  
  
 static final long *serialVersionUID* = -6612649371973922530L;  
  
 private final BeerDto beerDto;  
}

public class BrewBeerEvent extends BeerEvent{  
 public BrewBeerEvent(BeerDto beerDto) {  
 super(beerDto);  
 }  
}

public class NewInventoryEvent extends BeerEvent{  
 public NewInventoryEvent(BeerDto beerDto) {  
 super(beerDto);  
 }  
}

4.enable task scheduling: refer Img:1.0

5.create brewing service

@Slf4j  
@RequiredArgsConstructor  
@Service  
public class BrewingService {  
 private final BeerRepository beerRepository;  
 private final BeerInventoryService beerInventoryService;  
 private final JmsTemplate jmsTemplate;  
 private final BeerMapper beerMapper;  
  
 @Scheduled(fixedRate = 5000) //every 5 seconds  
 public void checkForLowInventory(){  
 List<Beer> beers = beerRepository.findAll();  
  
 beers.forEach(beer -> {  
 Integer invQOH = beerInventoryService.getOnhandInventory(beer.getId());  
 *log*.debug("Checking Inventory for: " + beer.getBeerName() + " / " + beer.getId());  
 *log*.debug("Min Onhand is: " + beer.getMinOnHand());  
 *log*.debug("Inventory is: " + invQOH);  
  
 if(beer.getMinOnHand() >= invQOH){  
 jmsTemplate.convertAndSend(JmsConfig.*BREWING\_REQUEST\_QUEUE*, new BrewBeerEvent(beerMapper.beerToBeerDto(beer)));  
 }  
 });  
 }  
}

6.create the listeners

@Service  
@RequiredArgsConstructor  
@Slf4j  
public class BrewBeerListener {  
  
 private final BeerRepository beerRepository;  
 private final JmsTemplate jmsTemplate;  
  
 @JmsListener(destination = JmsConfig.*BREWING\_REQUEST\_QUEUE*)  
 public void listen(BrewBeerEvent brewBeerEvent){  
 BeerDto beerDto = brewBeerEvent.getBeerDto();  
  
 Beer beer = beerRepository.getOne(beerDto.getId());  
 beerDto.setQuantityOnHand(beer.getQuantityToBrew());  
 NewInventoryEvent newInventoryEvent = new NewInventoryEvent(beerDto);  
  
 *log*.debug("Brewed beer " + beer.getBeerName() + " : QOH: " + beerDto.getQuantityOnHand());  
  
 jmsTemplate.convertAndSend(JmsConfig.*NEW\_INVENTORY\_QUEUE*, newInventoryEvent);  
 }  
}

6.1 add new Constant to JmsConfig class

@Configuration  
public class JmsConfig {  
  
 “”””””””””

public static final String *NEW\_INVENTORY\_QUEUE* = "new-inventory";  
  
 //Convert Message to Object (or vice-versa)  
 “”””””””  
}

Error!!!

org.springframework.jms.listener.adapter.ListenerExecutionFailedException: Listener method 'public void com.vtech.msscbrewery.services.brewing.BrewBeerListener.listen(com.vtech.msscbrewery.events.BrewBeerEvent)' threw exception; nested exception is org.springframework.jms.support.converter.MessageConversionException: Failed to convert JSON message content; nested exception is com.fasterxml.jackson.databind.exc.InvalidDefinitionException: Cannot construct instance of `**java.time.OffsetDateTime**` (no Creators, like default construct, exist): cannot deserialize from Object value (no delegate- or property-based Creator)

reason: MessageConvertor doesnot use the springboot managed instance of jaxon ObjectMapper, so we should configure it.

Solution:

@Configuration  
public class JmsConfig {  
  
 public static final String *BREWING\_REQUEST\_QUEUE* = "brewing-request";  
 public static final String *NEW\_INVENTORY\_QUEUE* = "new-inventory";  
  
 //Strategy interface that specifies a converter between Java objects and JMS messages.  
 //Serialize message content to json using TextMessage  
 @Bean  
 public MessageConverter messageConverter(ObjectMapper objectMapper) {  
 MappingJackson2MessageConverter converter = new MappingJackson2MessageConverter();  
 converter.setTargetType(MessageType.*TEXT*);  
 converter.setTypeIdPropertyName("\_type"); //need tobe set in order to convert incoming messge to java object (default is "none")  
 converter.setObjectMapper(objectMapper);  
 return converter;  
 }  
}

Error!!!

org.springframework.jms.listener.adapter.ListenerExecutionFailedException: Listener method 'public void com.vtech.msscbrewery.services.brewing.BrewBeerListener.listen(com.vtech.msscbrewery.events.BrewBeerEvent)' threw exception; nested exception is org.hibernate.LazyInitializationException: could not initialize proxy [com.vtech.msscbrewery.domain.Beer#7a08bdbb-e944-4762-966e-5822d4e61e3d] - no Session

reason: Hibernate was doing lazy initialization of property and we were running outside of transactional scope, so there was no hibernate session to work with.

Solution: add @Transactional for listener

import org.springframework.transaction.annotation.Transactional;

@Service  
@RequiredArgsConstructor  
@Slf4j  
public class BrewBeerListener {  
 private final BeerRepository beerRepository;  
 private final JmsTemplate jmsTemplate;  
  
 @Transactional  
 @JmsListener(destination = JmsConfig.*BREWING\_REQUEST\_QUEUE*)  
 public void listen(BrewBeerEvent brewBeerEvent){  
 BeerDto beerDto = brewBeerEvent.getBeerDto();  
   
 Beer beer = beerRepository.getOne(beerDto.getId());  
 beerDto.setQuantityOnHand(beer.getQuantityToBrew());  
 NewInventoryEvent newInventoryEvent = new NewInventoryEvent(beerDto);  
  
 *log*.info("Brewed beer " + beer.getBeerName() + " : QOH: " + beerDto.getQuantityOnHand());  
  
 jmsTemplate.convertAndSend(JmsConfig.*NEW\_INVENTORY\_QUEUE*, newInventoryEvent);  
 }  
}

Img:1.4

**Note:reffer** [**https://stackoverflow.com/questions/1099025/spring-transactional-what-happens-in-background**](https://stackoverflow.com/questions/1099025/spring-transactional-what-happens-in-background) **to what happnes in the background when we use @Transactional**

**Use: rollback feature,…**

**Setting log levels**

logging.level.guru.springframework=debug or info or …

**Talking to Different MicroService**

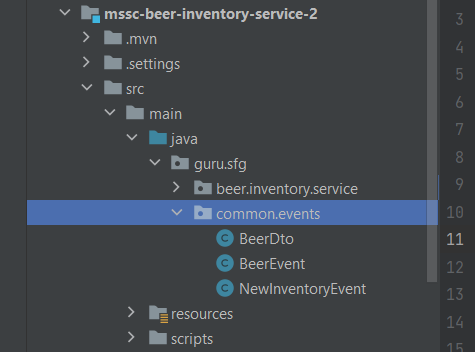
Talking from mssc-beer-service2 to mssc-beer-inventory-service(listener)

***mssc-beer-inventory-service*** changes

1.create services package and create NewInventoryListener.java class and also add constant to JmsConfig(NEW\_INVENTORY\_QUEUE)

@RequiredArgsConstructor  
@Component  
public class NewInventoryListener {  
 private final BeerInventoryRepository beerInventoryRepository;  
  
 @JmsListener(destination = JmsConfig.*NEW\_INVENTORY\_QUEUE*) // *NEW\_INVENTORY\_QUEUE= “*new-inventory”  
 public void listen(NewInventoryEvent event){  
 *log*.debug("Got Inventory: " + event.toString());  
 beerInventoryRepository.save(BeerInventory.*builder*()  
 .beerId(event.getBeerDto().getId())  
 .upc(event.getBeerDto().getUpc())  
 .quantityOnHand(event.getBeerDto().getQuantityOnHand()).build());   
 }}

2.create a new package(common.events) outside normal package and copy the required event classes from mssc-beer-service2



3.do the same for mssc-beer-service2 also

4.add ObjectMapper config in JmsConfig. refer Img:1.4

**Note:**We are maintaining same package name for events because in the mssc-beer-service2 we are trying to deserialize these event classes in different service(mssc-beer-inventory-service).

**2.0 Spring State Machine (SSM)**

Intorduction: reffer “[IntroductionToSpringStateMachine.pdf](pdf-notes/IntroductionToSpringStateMachine.pdf)” in NOTES > pdf-notes

Exmaple: refer “[paymentStateMachine](pdf-notes/PaymentsStateMachine.pdf)” in NOTES > pdf-notes

**Spring State Machine Dependecy**

<dependency>  
 <groupId>org.springframework.statemachine</groupId>  
 <artifactId>spring-statemachine-core</artifactId>  
 <version>3.1.0</version>  
</dependency>