Spring Framework & Microservices

In Java

* Interfaces
* Collection Framework

Framework: It is like a semi-implemented application which takes care of lot of common features every application needs so that developers need not write code on these common features, which are:

1. Design Pattern
2. Type Conversion
3. Exception Handling

Spring Framework

It is a Java framework that helps you to create various types of applications like web, webservices, cloud based applications

It gives you many modules

1. Spring Core / Spring Context: Takes care of design patterns & Dependency Injection (IoC)
2. Spring Web MVC: Create web & REST webservice application
3. Spring Boot: Quickly automate configurations
4. Spring Cloud: Create spring applications that can be run on cloud environments like microservices
5. Spring AOP: Aspect Oriented Programming
6. Spring JPA: To interact with the Database

Spring Core

It gives you all the design patterns & Dependency Injection feature

In Maven repository

spring-context

Types of DI

1. Setter Injection: It uses setter methods to initialize the object
2. Constructor Injection: It uses constructor arguments to initialize the object

You can configure application related data in XML file spring container can pass these data to the object via constructor or setters these are called as setter injection not only these data spring can also pass objects via constructor/setters

Setter Injection: Use <property>

Constructor Injection: Use <constructor-arg>

Try to comment all the <property> tags of DBSource bean & use constructor to initialize username, url, password using <constructor-arg>

DBSource(String username, String password, String url) { … }

Supplying an object into another object

class A {   
  
}  
class B {   
 A a1;  
}

<bean id = “b3” class = “com.A”>  
</bean>

<bean id = “b4” class = “com.B”>  
 <property name = “a1” ref = “b3” />  
</bean>

Annotation configuration

Spring can configure the beans using different types of annotations like

@Component, @Service, @Repository, @Controller, @RestController

All the above annotations are written on top of the class, any class having above annotations are instantiated spring container

@Service   
class A { }

@Repository  
class B { }

@Autowired: This is used to inject the object into another object like an alternative way of using <property ref >

@Qualifier: This is used when you want to specify which object you want spring to inject to the object

Question: If we have annotation can we completely ignore XML configuration

Answer: Still you can’t ignore because some third party classes if spring needs to instantiate then the only way is to use <bean> tag in the XML

Note: If spring needs to configure the classes with annotations then we must write <component-scan> tag in the XML file

Activity:

Create two <bean> for DBSource class and check which object the spring container is supplying to the TestDao & if not, make the spring container to supply any one of the DBSource object to the TestDao

Bean life cycle methods

These methods are automatically called when objects are created/destroyed/properties are set

There are annotations for these life-cycle methods

@PostConstruct: A method having this annotation is automatically called after bean is created by spring container

@PreDestroy: A method having this annotation is automatically called before bean is removed from the spring container

Day 2 Agenda

Spring AOP

Spring Web & REST

Spring JDBC with Derby database

AOP

* Aspect Oriented Programming
* It is a programming paradigm that lets developers to execute cross cutting concerns without calling them inside methods

Aspect: A class that will have cross cutting concerns

Advice: A method inside the Aspect which will have cross cutting concerns

Join Point: A method that will have actual application logic on which the advice is called

Pointcut: An expression for the join point written on top of the Advice

Point cut expressions

execution(public com.org.ProductService.fetch()): execute on fetch() method it could be before or after the execution

i.e., @Before(“execution(….)”)

i.e., @After(“execution(…)”)

execution(\* com.org.\*.\*(..)): All the classes and their methods inside com.org

execution(\* com.org.Product.set\*(..)): Executes only setter method of Product class inside com.org

Spring Jdbc

It gives two objects that takes care of internal operations

1. DriverManagerDataSource: It takes care of initializing the database credentials and connects to the database like username, password, url, driver class all will be initialized
2. JdbcTemplate: It uses DriverManagerDataSource and provides methods to write SQL queries like insert, update, delete and select, it takes care of handling the database exceptions internally

DriverManagerDataSource: Has four properties to initialize connection

1. username
2. password
3. url
4. driverClassName

All these 4 properties you can initialize in XML

JdbcTemplate: Has one property that is DriverManagerDataSource property

You need to tell spring container to supply DriverManagerDataSource to the JdbcTemplate, JdbcTemplate has methods like

update(SQLQuery, Object…): This method can use insert, update & delete queries, Object… will accept 0 or more arguments

ex: insert into employee values(?,?,?);

ex: update(“insert into employee values(?, ?, ?)”, value1, value2, value3);

queryForObject(SQLQuery, RowMapper, Object…): This returns single object that matches to the query

query(SQLQuery, RowMapper): This returns List of objects that matches the query

Spring container must create DMDS and JdbcTemplate object, we must use JdbcTemplate in DAO layer of our application

Flow of the Dependencies would be as below

DMDS -> JdbcTemplate -> DAO -> Service -> Controller

Dependencies required

1. Spring Context
2. Spring Jdbc
3. Derby Client

XML Configuration

1. Configure DriverManagerDataSource - username, url, password, driverClassName
2. Supply DriverManagerDataSource to JdbcTemplate

Code

1. Create DAO interface & Class, in the class Inject the JdbcTemplate
2. Create Service and Inject the DAO
3. Treat Main class itself as a controller and get the Service object (in real time service object is injected to controller)

Things to create in the project

1. Employee
2. EmployeeDao
3. EmployeeDaoImpl
4. EmployeeServiceImpl
5. EmplloyeeControllerMain

RowMapper: It is an interface that has a map method which takes care of converting ResultSet records to a Java object,

JdbcTemplate uses RowMapper to simplify converting SQL records to Java objects

Note: In Core Jdbc Developers need to write the code to navigate over the ResultSet and converting each record to an object

Spring Web & Rest

Spring uses front controller to accept all the incoming requests and map to the right controller.

Controller: Takes the request and returns the response, it internally calls service layer methods

FrontController: It is like an interceptor that accepts all the incoming requests, it will also take care of initializing spring container, i.e., ApplicationContext ctx = new ClassPathXmlApplicationContext(..) is done by Front controller, In Spring DispatcherServlet is the inbuilt class that is used as a FrontController.

REST stands for REpresentational State Transfer where you can create services online that can be accessed by any client applications written in any technologies like Angular, React, Vue.js, Python and so on.

@RestController annotation helps you to create RESTful webservices, this acts like a controller also

Rules for RESTful webservices

1. It must use HTTP to communicate
2. Webservices must have some URL’s
3. Not to break the HTTP standard

Steps to create webservices in Spring

1. Add Spring Web MVC library
2. Add Jackson library (takes care of converting java to json and vice versa)
3. Configure Front Controller in web.xml
4. Configure beans in the XML file as per the name given in the web.xml

Making application interact with the database

1. Start the database
2. Add spring-jdbc & derby-client library
3. Configure beans for DriverManagerDataSource & JdbcTemplate
4. Copy the existing Employee, DAO, Service in your web service application
5. From controller access the Service by autowiring the service

Spring Boot

It simplifies developing spring applications by taking care of all the generic configurations required for the spring application.

Spring Boot doesn’t need any XML files, because it can auto-configure the beans as per the starter library you add in the project

i.e.,

* You don’t have to setup server in your project, because spring boot embeds the server in the application
* You don’t have to write component scan tag to scan the classes, spring boot will automatically do that
* You don’t have to configure the front controller, it is taken care again by spring boot
* Dependency Injection of many external beans are automatically done by spring boot with the help of starter libraries

ex: If you add Spring Boot Starter Data JDBC library, then spring boot will automatically configure DriverManagerDatasource & supplies that to the JdbcTemplate

Note: Spring boot uses application.properties file to configure the application specific configurations like db credentials, server ports, context paths, and etc

* Spring boot also takes care of downloading the right version spring libraries without explicitly mentioning version for each module

Spring vs Spring Boot

|  |  |
| --- | --- |
| Spring | Spring Boot |
| You need XML file | XML is not required |
| Component Scan configuration is required | Component Scan is automatic |
| XML configuration is complex | Spring Boot uses property file which is simple to write |
| Bean configurations & their dependencies you must write in XML | Bean configurations & their dependencies are automatic as much as possible, because in application code you would be using @Autowired wherever necessary |
| Server, DispatcherServlet everything you need to configure | Server, DispatcherServlet are automatically configured |
| Spring module versions you need to track to make it compatible | Spring module versions are handled by spring boot so that you don’t have to track |

Spring Initializr: This website helps you to download spring boot project template

How can spring boot automatically configure the application

It uses an annotation called @SpringBootApplication which takes care of doing everything for the application like

* Autoconfiguration
* Creating dependencies for the beans
* Configuring servers and front controller.

Note: @SpringBootApplication must be written on top of the class & it must appear in the root package, because the auto-configuration starts from the class having @SpringBootApplication

It means all your application classes must be in the sub-package of the above class or in the same package

Executable jars

You can create these jars and provide it to the Operation team so that they can deploy it on the server

Ops team would run the jar using java -jar filename.jar command

Overriding the properties

While running the jars you can override the properties written in the application.properties

--server.port=9092

--spring.datasource.username=admin

These options you can give while using java -jar command

ex: java -jar filename.jar --server.port=9092

Spring Boot allows you to change the embedded servers, it gives two more embedded servers

1. Undertow - JBoss
2. Jetty - Eclipse

Since the default server is tomcat, you must exclude that in pom.xml and add the embedded server you want in the pom.xml

Registering beans we create in spring container

@Bean must be used on a method that returns an object which is maintained by spring container.

How to use teXML configuration in spring boot

Note: Spring Boot main goal is to eliminate XML configuration, but if you still need to use XML, then you need to follow these steps.

1. Keep your xml in the src/main/resources folder
2. In @Configuration class use @ImportResource(locations = {xmlFileLocation}) to load all the bean configuration

Spring Boot Best Practices

1. Use Build Systems like Maven
2. Structure your code
   1. Avoid default packages
   2. Have @SpringBootApplication in root package
   3. categorize the classes with different packages which must be in subpackage of root package
3. Use @Configuration class instead of XML
4. Use Dev tools - auto-reload the server while the code is changed
5. Packaging the application that can work in various production environment
6. Reference the migration guide of spring boot document

Spring Data Jpa

It helps you to provide DAO layer implementation automatically based on the Repository you extend, you just need to create entity class and provide database properties in the application.properties, spring data jpa takes care of implementing DAO layer & registering its object in the spring container, so that you can use @Autowired on your Repository interface

Repository interfaces provided by spring data jpa

1. CrudRepository<T, ID>
2. JpaRepository<T, ID>

You need to extend any one of the above repositories

interface EmployeeRepository extends JpaRepository<Employee, ID> { }

Here Employee is the entity class that maps to some table

ID is the primary key type

Libraries required

1. spring boot starter web
2. spring boot dev tools
3. spring boot starter data jpa
4. derby client

Steps to interact with the Database using Spring Boot & Data Jpa

1. Start the database
2. Configure application.properties
3. Entity class
4. Repository interface that extends Jpa[or]CrudRepository - No need to implement this
5. Service interface & Implementations
6. Exception class that is generated on certain conditions
7. REST Controller to call Service methods

@ControllerAdvice

To register your class to handle exceptions

@ExceptionHandler

To specify what kind of exceptions you want to handle

Custom methods in the Repository

You can create methods that spring boot data jpa can still implement, but you must write JPQL on top that method

JPQL: Java Persistence Query Language, this is written for the entity and behind the scene SQL is generated

SQL to select all the records

select \* from employee

JPQL to select all the entities

select emp from Employee emp

SQL to select a single record is

select \* from employee where id = ?

JPQL to select a single entity is

select emp from Employee emp where emp.id = ?1

Now how to select all the entities having the same name

select emp from Employee emp where emp.name = ?1

CORS

Cross Origin Resource Sharing will be disabled in every webservice, we must enable it so that only certain applications must able to access the data, in spring we can use @CrossOrigin(origins = {“…”, “…”})

1. Try to generate the id for employee automatically without you passing id value

Hint: You need to use few annotation on the id property of Employee class

1. Use 2 tables Employee & Address, when you are storing employee object it must have address data also that will be stored in Address table, when you fetch employee you must also have address of that employee

Hint: You must have a foreign key of employee id in the address table

Ex:

{  
 “name” : “Ajay”, “salary”:42000, “dob”:”2001-10-15”,   
 address : {“state”:”KA”, “city”:”BLR”, “pin”:”560001”}  
}

Above JSON data should initialize Employee object which will have address property and when you store employee object it must store employee data in Employee table & address data in Address table

Microservices

These are small independent services which can be developed, test, build independently from other services

Benefits

1. Services are loosely coupled with other services
2. You can scale up or scale down only the particular services you want
3. If any one service goes down other services would be still working
4. You can use different technologies for different features

Limitations of Microservice

1. Expensive
2. Skilled resources are mandatory
3. Competing with global market is not easy

Important design principles every microservice should follow

* Service Discovery
* Discovery Client
* Load Balancer
* Circuit Breaker
* Security
* External Configuration

Spring provides a library called spring cloud that implements all the above design patterns, so that you don’t have to worry about implementing it

Spring Microservices uses 2 projects of spring

1. Spring Boot: To quickly create spring applications & get the benefits of auto-configurations
2. Spring Cloud: This gives all the necessary tools & design patterns to create microservices in a simpler way using annotations ex: @EnableEurekaServer, @EnableEurekaClient, @LoadBalanced

Note: Because spring boot & spring cloud are different projects we need to use spring boot release train to have compatible versions of both the projects

Spring Cloud gives annotations to simplify development process and add the necessary tools & design patterns to your application

@EnableEurekaServer: This creates service discovery and takes care of registering all the microservices and abstracts the microservice location with the instance name, it will refresh the registry automatically whenever any updates need to be done

@EnableEurekaClient: This helps in registering the microservice in service discovery and sending heart beats every 30s about its status, registers its physical address & instance name as well

@LoadBalanced: This is a client side load balancer which is going to route the requests to microservice and also takes care of equally distributing the loads on multiple instances

Creating Service Discovery

Eureka Server

Dev tools (Optional)

Note: Every microservice registers its instance into the service discovery by searching the service discovery in 8761 port by default.

Hence its better to run our service discovery in 8761 port

Creating Microservices / Eureka Clients

Eureka Client

Web

Dev tools (Optional)

Communication between Microservices

RestTemplate can be used to call a microservice, but we must configure it to use Load balancer

Note: We must use instance-id to communicate with the microservice

RestTemplate has http methods like get, post, put, delete and returns the response content

In Spring Boot you need to create RestTemplate object & register in the spring container so that you can auto-wire it wherever you need it.

/account/1234

{ “balance”, “port” }

YAML: It can be used as a replacement to the properties, it avoids you to use repeated properties

spring.profiles.active = dev

The above code will load the dev.properties

Feign Client: It helps you to create reusable methods that can easily call the microservices in a declarative way through interface, it is an alternate form of RestTemplate, FeignClient internally uses LoadBalancer

Open Feign is the library which you need to add in your project