Spring Framework

It helps you to create various types of applications like Desktop, Web applications, Enterprise applications, Cloud based applications

Spring Modules

1. Spring IoC
2. Spring Web MVC
3. Spring JDBC
4. Spring Boot
5. Spring Cloud

Spring IoC:

Inversion of Control, because object creation process is inverted, it helps spring container to create the dependencies and supply to other objects.

Spring Libraries for IOC

1. Spring Context: This gives you all the necessary jar files to get the Spring IOC benefits like DI, Design Patterns, Container etc.

Dependency Injection:

It is an approach of supplying an object into another object to make loosely coupled application.

There are two types of Dependency Injection

1. Setter Injection: It uses setter method to initialize the object
2. Constructor Injection: It uses constructor argument to initialize the object

For setter injection you will use <property> tag

For constructor injection you will use <constructor-arg> tag

Configuring complex dependencies

Whenever an object depends on another object we may need to supply these complex dependencies we can configure a <bean> and supply that to another <bean> using ref attribute.

class MySQLImpl implements DBOperations {   
  
 Datasource ds;

}

<property name = “ds” ref = “beanId” />

Bean Scopes

By default beans are singleton, you can change the scope to prototype when you need multiple instance of the same bean id.

<bean id = “b1” class = “com.A”>

The object of class A is singleton

<bean id = “b2” class = “com.B” scope = “prototype”>

The object of class B is prototype i.e., on each getBean(“”) call you get a new object

Annotation based configuration

Spring gives you various annotations to register the beans in the container so that you can avoid configuring XML as much as possible

Spring has given one base annotation called @Component which registers the object in the spring container there are other component type annotations that would do the same job which are:-

* @Repository
* @Service
* @Controller
* @RestController

@Component  
class A {   
}  
@Service   
class B {   
}  
@Repository   
class C {   
}

XYZ >> class Testing { }

Advantages of annotations

1. Simple to use
2. Most of the complex dependencies can be supplied to objects using @Autowired annotation & it doesn’t need any setter or constructor to initialize

@Repository  
class A {   
}  
@Service  
class B {   
 @Autowired  
 private A obj;  
}

Note:

Annotation configuration will work only when you use <component-scan> tag in your XML file that let spring know from where it should scan the classes

AOP

Aspect Oriented Programming, which helps you to call crossing cutting concerns without actually invoking it, this is achieved using some of the AOP features

1. Aspect: These are the classes that will have cross-cutting concerns
2. Advice: These are the cross-cutting logics which are executed before or after the actual method (join point)
3. Join Point: These are the actual methods of the application where you want the advice to be run around it.
4. Point Cut Expressions: This is an expression written on the advice that tells AOP on which Join point the advice must be run

Pointcut expressions

execution(“public void com.TestService.test()”)

execution(“\* com.TestService.\*(..)”): Selects all the methods of TestService

execution(“\* com.\*.set\*(..)”): Selects setter methods of all the classes inside the com package

execution(“\* com.TestService.login(String, \*)”): Selects login method of TestService 1st parameter must be String & 2nd parameter could be anything

Spring MVC

It is used to develop web applications using MVC architecture, where you can abstract all the layers which are dependencies of another layer, so that you can have a loosely coupled application

i.e., Service layer depends on DAO layer, Controller layer depends on Service layer, here these dependency can be abstracted so that changes are limited

How Spring MVC helps you to create web application

It gives you a front-controller called DispatcherServlet who takes care of accepting all the requests and routing to appropriate controller

Tasks of Front Controller

1. Accepts all the incoming requests & route to appropriate controller
2. Configures the beans and initializes the spring container i.e, ApplicationContext
3. Uses a bean to resolve the JSP views

Configure web application in spring

1. We need an external server
2. We need to create a dynamic web project
3. Add spring WebMvc dependency
4. web.xml must have front-controller configuration which is DispatcherServlet
5. Spring Bean Configuration file

Note: DispatcherServlet will take care of loading the spring bean configuration file, it will know about the spring configuration xml file from web.xml

Front Controller: It is used to take requests and map to the controller

Controller: It is a class with a method that can handle the request & generate the response, controller depends on service layer

@Controller: It is the annotation used to create controller, this must be marked on top of the class, these will have one or more methods to handle the request & show the views

ReSTful webservices: These are online services/API’s which helps heterogenous applications to exchange the data, more commonly used data-format is JSON

@RestController: It is the annotation used to create controller that can handle rest calls and generate data instead of response.

We can have separate applications for front-end & backend because of webservices

ex: ATM machine of different banks can exchange data with different banking services

How to create API’s/design with ReST webservices

* We must create controller with specific URL
* We must map the controller with HTTP methods so that clients can use the same HTTP method to access the API
* We must specify what datastructure it can consume & produce

How to configure webservices in spring

1. We need to create controller using @RestController
2. We need to add a library that takes care of converting Java objects to JSON & Vice versa the library is Jackson-databind
3. We need to create API’s with right HTTP methods based on the type of operations
4. GET - Fetch
5. POST - Create
6. PUT - Update
7. DELETE - Remove

Note: We need Postman application to test the webservices

What are the ways Java can represent its data in JSON format

1. A java object can be represented in JSON format, because java objects will have properties
2. A key value pair data structure like Map

Spring JDBC

It is used to interact with the database, it provides you to configure the

1. DriverManagerDatasource: which will have properties like username, password, url, driverClassName
2. JdbcTemplate: which has property DataSource that helps JdbcTemplate to connect to the Database & it also gives you methods to perform crud operations like
   1. update(SQLquery, arguments): insert, update, delete queries
   2. query(SQLquery, arguments): select command for more than one records
   3. queryForObject(SQLquery, arguments): select command for one record

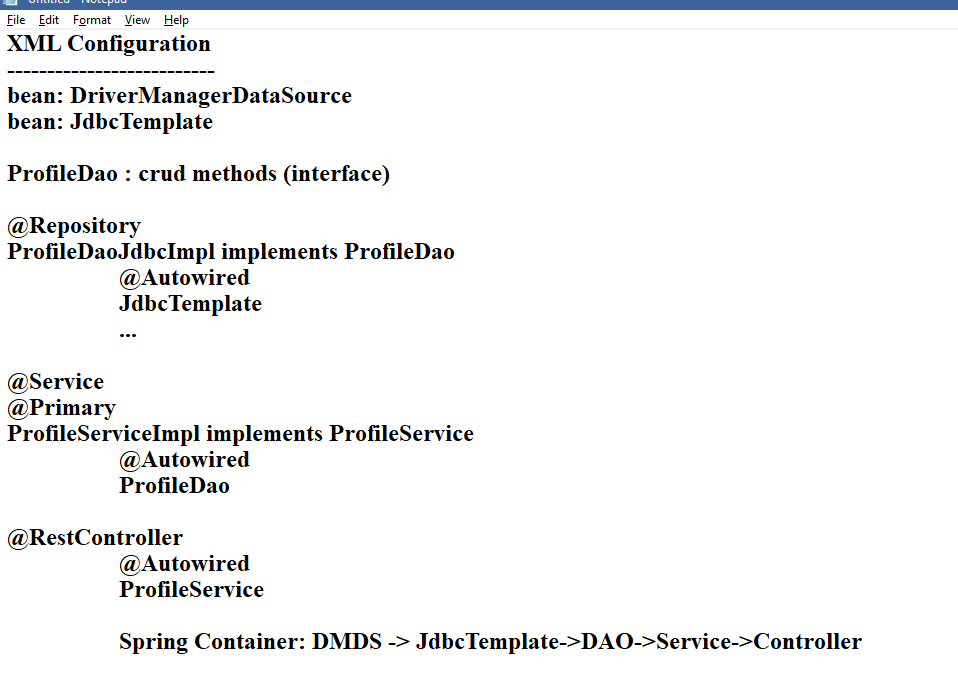
Limitations of JDBC

* You need to take care of writing SQL queries
* You need to take care of converting Java types to SQL types
* Writing complex join queries would be difficult when you want to get a result from multiple tables



Libraries we need are:

1. Spring Jdbc: spring-jdbc
2. Database driver library : derby-client



Spring ORM

It helps you to directly map java objects to the tables

ORM takes care of following things

1. Type conversion like java types to sql types
2. Mapping results to Java objects or Collections
3. No need to write queries
4. ORM generates the queries depending on the database the application is connected to
5. Complex joining queries are achieved with annotations

Spring ORM provides HibernateTemplate which performs all the CRUD operations for you, it will automatically generate SQL queries

HibernateTemplate depends on SessionFactory which depends on DriverManagerDataSource

Libraries required

1. Spring ORM
2. Hibernate Core
3. Derby Client

XML configuration

Configure tx namespace,

Configure DriverManagerDataSource

Configure LocalSessionFactoryBean

Configure HibernateTemplate

Configure TransactionManager

Spring Boot:

It simplifies developing spring applications by auto-configuring lot of generic setup, it does that using many starter libraries

* You don’t have to write any XML file
* You don’t need to setup server
* You don’t need to configure Front-Controller i.e., DispatcherServer
* You don’t need component scanning
* You don’t need to configure bean dependencies like configuring DataSource & supplying it to JdbcTemplate or HibernateTemplate because Spring Boot will take care of these things using the starter libraries

Spring Boot will perform auto-configuration based on the starter libraries

1. Spring Boot Starter Web: If you use this library then it gives you an Embedded Server i.e., (Tomcat) but you can change this if you want,
   1. Performs Front Controller configuration
   2. Performs component scanning
2. Spring Boot Starter Data JPA: If you use this library it takes care create all the beans required for your application like DriverManagerDataSource, JdbcTemplate, HibernateTemplate etc all these are auto-configured for you and it gives you some Repository interfaces so that without implementing DAO layer you can perform Database operations

All these datasource configurations is done by looking at application.properties file which will have database information’s

1. Spring Boot Starter Actuator: This library gives end points to monitor your application in production like status, health,

Spring Boot uses one annotation to configure all the setup automatically based on the starter library

@SpringBootApplication: This annotation takes care of performing auto-configuration like component scanning, front controller configuration, data source configuration

Spring Initializr website gives you spring boot project which you can download in your machine & use it in eclipse/sts/or any IDE

application.properties

You can configure application related properties like server, datasource, security, circuit breaker and many more

server.port = 9090 # this runs server in 9090

spring.datasource.username = admin # database username to login

spring.datasource.password = admin # database password to login

spring.datasource.url = jdbc:derby://localhost:1527/mydb

spring.datasource.driverClassName = org.apache.derby.jdbc.ClientDriver

Note: If spring boot has to configure DriverManagerDatasource, HibernateTemplate, JdbcTemplate you need another library called spring boot starter data jpa

You can visit spring boot common application properties to see all the properties

<https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

ResponseEntity: it is a response that can carry HTTP response with status code & content so that you can have different HTTP status code in the response as well as different content

ResponseEntity.status(200).body( object ): This will create a response with 200 status means OK / Success and Response body will have a Java object that will be converted to JSON data automatically.

ResponseEntity.status(404).body( object ): This will create a response with 404 status means Not Found & response body will have a java object which represents some error messages that will be converted to JSON automatically.

Spring Data Jpa

Takes care of implementing DAO layer for the Repository interfaces

Provides two repository interfaces that has all the CRUD operations for the particular entity class

1. CrudRepository<T, ID>: CRUD methods are present
2. JpaRepository<T, ID>: extends CrudRepository & provides methods of sorting & pagination

You must have an entity class that maps to the table

You need to create an interface that extends either CrudRepository or JpaRepository

Your interface will be auto-implemented by Spring Boot, it means you don’t have to implement DAO class at all

Spring boot takes care of implementing the methods of Repository interfaces that works on your entity

ex:

save(T t): it is implemented for your entity class so that when you call this method the entity will be saved in the mapped table

Our interface must be as below:

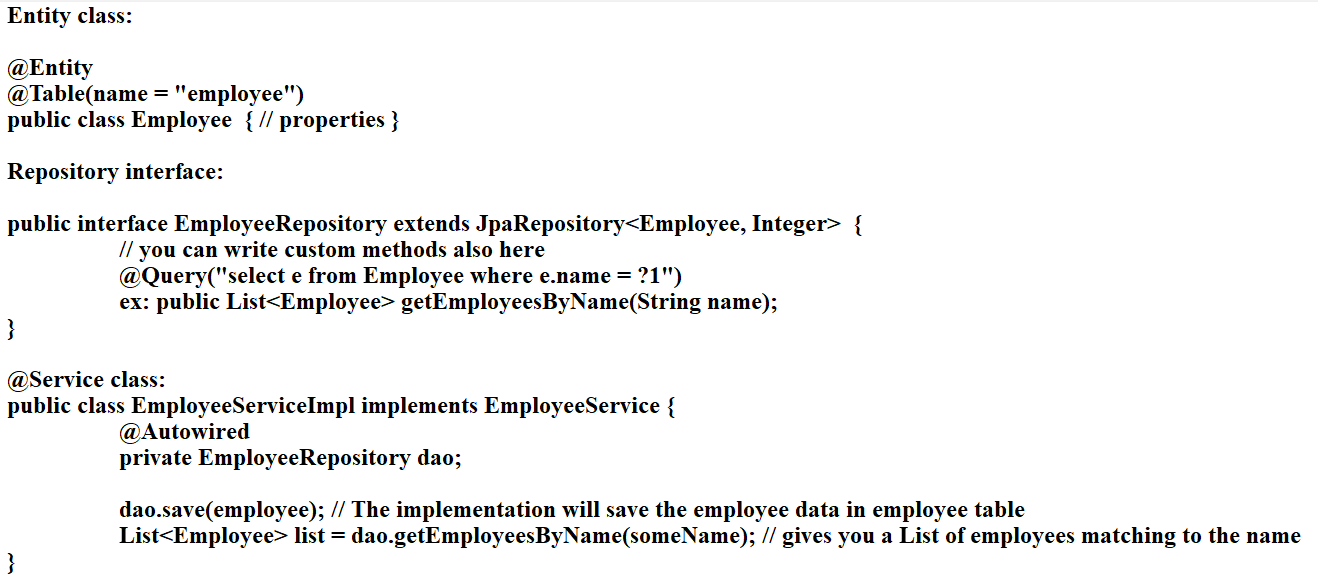
public interface EmployeeRepository extends JpaRepository<Employee, Integer> { }

Now this EmployeeRepository will have all the methods of JpaRepository like save(), findById(ID), findAll(), deleteById() and so on

Spring boot implements EmployeeRepository in such a way that all the methods would work on the table the Employee entity is mapped

We need to just auto-wired EmployeeRepository in @Service and call save(entity), delete(entity), findAll() methods from the interface so that they will be called on the auto-implemented object of EmployeeRepository

You must configure application.properties that will have datasource information as per the datasource information spring boot establishes connection



Libraries required

1. Spring boot data jpa
2. h2 database: in memory database

Configuring the beans in spring boot

@Configuration // detected by spring container while scanning  
class Config {   
 @Bean // registers the returned object in the spring container  
 public A objA() { return obj; }  
   
 @Bean  
 public B objB() { return obj; }  
}

application.properties  
app.url = <http://app.com>

@Component // or You can configure with @Bean  
class C {   
 @Value(“${app.url}”)  
 private String url;   
 setters & getters  
 }

Activity

application.properties

server.port = 9090

In EmployeeController create one variable as below

@Value(“${server.port}”)  
private String portNo;  
  
@GetMapping(“/port”)  
public String greet() {   
 return portNo;  
}

Create Jars/Wars in production

Jar or War are deployable artifacts which you can use to launch your application in production machines

Jar : It is used when you want to run the application using embedded server

War: It is used when you want to run the application using external server

Different servers that we can configure other than tomcat

1. Jetty: From eclipse
2. Undertow: From Jboss

We need to exclude tomcat and add the server dependency

Design pattern

A repeatable solution to a commonly occurring problems

There are 3 types of patterns

1. Creational patterns: Deals with various mechanism in object creation
2. Structural patterns: Deals with assembling objects to make larger structure
3. Behavioural patterns: Deals with algorithms & assignments of responsibilities between the objects

Creational patterns

1. Factory pattern
2. Singleton pattern
3. Prototype pattern

Structural pattern

1. Adapter pattern
2. Decorator pattern

Behavioural pattern

1. Strategy pattern
2. Observer pattern

Day 5

Microservices:

Microservices are small independent services that can be developed, test, built independently from other services

Benefits

1. Services are loosely coupled
2. You can scale up or scale down only the particular services
3. Release of new feature will consume less time
4. You are free to use different technologies for different services

Limitations

1. Expensive
2. Competing with global market is not easy
3. Skilled people / resources

Design patterns

Microservices are deployed on multiple machines on cloud hence there must be certain patterns the services should follow

1. Service Discovery: It registers the microservices & helps microservices to locate other microservices
2. Discovery Client: It is a microservice who wants to register in service discovery & communicate with other microservices
3. Client Side Load Balancer: It will equally distribute the load across multiple instances of the microservice
4. External Configuration: It is used to distribute the configurations to multiple microservices
5. Circuit Breaker: It is used to avoid cascading failures when communicating between the microservices

Spring Microservices

It uses 2 projects

1. Spring Cloud: It gives all the design patterns required to create microservices program with lot of simple annotations
2. Spring Boot: It is to create a production grade application with all the auto-configuration features

Spring cloud has given simple annotations that applies the design pattern in the microservice architecture

1. @EnableEurekaServer: This creates service discovery & gives the dashboard so that you can see all the registered microservices, & it takes care of removing the service id’s if any service is down

Note: If microservices doesn’t send heart beats after 3 chances then service discovery removes the instances form its list

1. @EnableEurekaClient: This is used to register the microservice in the service discovery, it takes care of sending heart beats to the service discovery every 30s & it will have instance id which will be registered in the service-discovery
2. @LoadBalanced: This is used to create client side load balancer, this takes care of distributing the requests across multiple instances of microservice

Note:

You must always use spring cloud & spring boot compatible versions, we need to look the official document release train to use the correct version of spring cloud compatible with spring boot

Service Discovery

It is a registry to register all the microservices

Library required

1. Eureka Server
2. Dev tools

Discovery Client

It is a microservice that registers in service discovery

Library required

1. Eureka Client
2. Dev tools
3. Web

Create another microservice and register in service discovery

spring.application.name = upi-app  
server.port=9999

Communicating between the microservices

We need to use the application name / instance id of the microservice in another microservice to call.

RestTemplate: It is an object that can send http request to another microservice/webservice

@LoadBalanced: It will take care of distributing the load to multiple instances of the services & it resolves the ip & port of the microservice while calling

FeignClient:

It is an alternate way to make REST calls, it is by default configured with LoadBalancer

It is reusable as well.

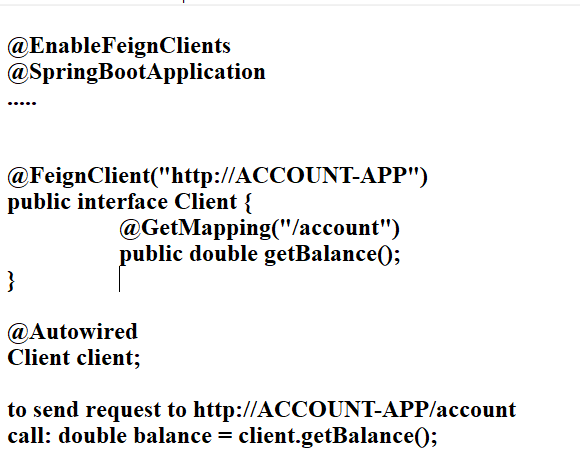
@FeignClient(“<http://ACCOUNT-APP>”)

interface Client {   
   
 @GetMapping(“/account”)  
 public double getBalance();  
}

@Autowired  
Client client;

client.getBalance(); // sends request to http://ACCOUNT-APP/account

Note: You don’t have to implement the interface, spring boot does that the only thing you need to do is you must @EnableFeignClients annotation on top of the @SpringBootApplication or in any other configuration class



Note: You need to add OpenFeign library in your project