**SPRING BOOT**

**SPRING BOOT**

- open source java based framework to create faster application

- developed by Pivotal team

**Advantage**

1. easy to understand and develop spring application

2. increase productivity

3. reduce your development time

4. To avoid xml configuration

5. In Spring Boot everything is automatic, no manual configuration

6. Spring boot comes with embedded server like tomcat(default), jetty, undertow

7. Spring boot comes with in memory database called H2 database

**Differences between Spring and Springboot**

|  |  |
| --- | --- |
| **SPRING** | **SPRING BOOT** |
| 1. Used to develop Java EE framework for building application both client and server side | 1. used to develop REST API endpoints (server side) |
| 2. It aims to simplify Java EE development using JSP and servlet that makes development more productive | 2. It aims to shorten the code length and provide easiest way to develop web application |
| 3. main features is IOC which uses Dependency Injection | 3. main features is auto-configuration |
| 4. used to develop loosely coupled application - since injection done at runtime in xml file | 4. create a standalone application with less configuration |
| 5. write lots of boilerplate code | 5. it reduces boilerplate code |
| 6. In Spring, we have to configure the server explicitly | 6. offers with embedded servers |
| 7. Spring doesn’t provide any in memory database | 7. Spring Boot provides in memory database called H2 database |
| 8. developers manually configure dependency in pom.xml | 8. Spring boot comes with concept of parent starter in pom.xml that internally takes care of downloading JAR dependencies in spring boot application |
| 9. deployment descriptor/web.xml is must in case of Spring MVC | 9. There is no need for web.xml/deployment descriptor |
| 10. It requires bean configuration in xml manually | 10. There is no need for beans xml configuration |

**Spring Boot Dependency management**

- manages the dependency and configuration automatically

- uses starter parent which internally download all the related jar files.

In short, starters are the compilation of multiple dependencies, which are resolved easily and reduces the task of including each dependency manually by specifying their names

- No need to specify the version of the dependencies

- It provides the centralize of dependency info by specifying spring boot version

- spring-boot-starter-parent is a project starter, provides all default configuration for your application

Spring boot comes with built in mechanism for application auto configuration inside application.properties/application.yml

Properties is class which contain key value pairs and both should be string

**Develop Spring Boot Application - 3 ways**

1. Spring initializer (i.e.) from spring.starter.io site

2. Spring Tool Suite(STS) ide + develop boot project

3. Spring command line interface

**Spring Boot Annotation**

1. Core Annotation of Spring and Spring MVC

- @Autowired,

- @Qualifier - if same bean is configured multiple times, which bean has to be injected and used along with @Autowired

- @Required

- @ComponentScan

- @Controller, @RequestMapping, @RequestParam, @PathVariable, @ModelAttribute

- @GetMapping, @PostMapping, @PutMapping, @DeleteMapping

- @Service

- @Component

- @Repository

- @Transactional

2. SpringBoot annotation

@SpringBootApplication = @EnableAutoConfiguration + @ComponentScan + @ Configuration

**Apache Maven**

- automatic project build tool

- Maven will download all jar files from repository

- Remote repository - maven download all jars from internet and keep into local repo

- local repository - repo present in our system (i.e.) .m2 folder

**Path for Apache Maven**

1. JAVA\_HOME- C:\Program Files\Java\jdk1.8.0\_181

2. Path - C:\Program Files\Java\jdk1.8.0\_181\bin

3. M2\_HOME - C:\Softwares\apache-maven-3.6.3-bin\apache-maven-3.6.3

4. Path - C:\Softwares\apache-maven-3.6.3-bin\apache-maven-3.6.3\bin

**Steps**

1. Go to Spring Initializer, Spring boot project can be maven project(pom.xml) or gradle project(build.gradle)

2. Extract project into workspace

3. Import the project in eclipse

4. In command prompt, we go into the project location

>mvn clean install

- it will download all dependency and put in .m2

5. Right click project - Maven - Update project - Check Force Update on snapshot Release - click ok

6. If still errors occur, go to Problems tab - delete all errors

7. In SpringBoot we always have to run main program

To develop any web application - Spring Web dependency

com.pack.SpringBootHelloWorld - main package

In spring to define the class as controller we use @Controller

@Controller

@ResponseBody

public class MyController

{

}

In Spring boot used to develop REST endpoints with @RestController = @Controller + @ResponseBody

@RestController

public class MyController

{

}

To change the port no in application.properties

server.port=2000

To provide project name after the port we have to configure in application.properties

server.servlet.context-path=/ClientApp

Whenever we change anything in the program, we have to restart the server manually, but if we want to restart the server automatically then we have to use a dependency called Spring devtools

If we want to automatically restart the browser also we have to go for Livereload option

We can exclude the tomcat and we can run jetty server, for that

<dependency>

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

<artifactId>spring-boot-starter-web</artifactId>

<exclusions>

<exclusion>

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

<artifactId>spring-boot-starter-tomcat</artifactId>

</exclusion>

</exclusions>

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-jetty</artifactId>

</dependency>

**CommandLineRunner interface**

- used to indicate that the bean should run when it is contained within the spring application

- if we want to execute any code after the springboot application is started

- public void run (ApplicationArguments arg)

- @Order used to execute the command line runner in order

**SpringBootServletInitializer**

- It lets a spring boot application packaged as war file and deployed in an external tomcat server or any other server

- convert an existing web application to a spring boot application by writing main class extending SpringBootServletInitializer

Spring Boot does not know what is JSP, in order to make spring boot to understand jsp we have to configure tomcat-embed-jasper dependency

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

</dependency>

<dependency>

<groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

<scope>provided</scope>

</dependency>

Configure JSP info in application.properties file

spring.mvc.view.prefix=/WEB-INF/view/

spring.mvc.view.suffix=.jsp

We want to transfer some information from controller class to jsp using

1. Map object

2. Model object

3. ModelMap object

**Spring Boot Profiling**

- The development process of an application has different stages like development, testing, production etc. In each stage we have different set of configuration properties

- Spring Boot by default loads application.properties

- Profile is a set of configuration settings. Spring boot allows to develop different profile property in the form of application-profilename.properties

spring.profiles.active=profilename in ur application.properties

@Value - used to take the value from application.properties Ex:@Value("${myProperty}")

**SPRING DATA JPA**

- Used for persistence purpose - used to store into the database

Create repositories that uses the Java Persistence API is a tedious process that it takes lot of time and requires lot of boilerplate code

Spring Data JPA - it is a library/framework that adds an extra layer of abstraction on the top of the JPA provider

spring-boot-starter-data-jpa - it is dependency used for spring data jpa

**3 interfaces - Spring Data common Projects**

1. Repository<Entityclass, Primarykey> interface - marker interface

2. CrudRepository<Entityclass, Primarykey> extends Repository - provides CRUD operations for entity

3. PagingAndSortingRepository<Entityclass, Primarykey> extends CrudRepository - declare the methods used to sort and paginate the entity data

**2 interface - Spring Data JPA project**

1. JpaRepository<Entityclass,Primarykey>

2. JpaSpecificationExecutor<Entityclass>

Spring Boot communicates with database using JPA, Hibernate (i.e.) JPA provider, on the top of JPA provider we use spring data jpa so that we can communicate with database faster and with less number of codes.

Spring Data JPA is the abstraction that makes it easier your work with JPA provider. It is a library that adds additional wrapper code on top of the jpa provider to cut through additional boilerplate code that requires to interact with database

**Features**

1. Spring Data JPA generates automatic implementation (i.e.) we don’t need to write any code related to CRUD operations - dao layer is completely removed

2. reduces boilerplate code to interact with db, so we can able to create persistent layer very faster

**Steps**

1. Configure spring-boot-starter-data-jpa, database dependency in pom.xml

2. no need for xml configuration, instead all database configuration are given in application.properties

3. create entity class

4. we create separate interface that extends any one of repository interface

**JPA Specification**

1. @Entity

2. @Table

3. @Id

4. @GeneratedValue

**CrudRepository interface**

1. Object save(Object) - used to save single object into the database, if primary key already present it performs updation, if primary key not present it performs insertion

2. Iterable<Object> saveAll(Iterable<Object>) - used to save multiple object

3. Optional findById(int id) - to fetch single object

4. Iterable findAll() - to fetch all object

5. boolean existsById(int id)

6. void delete(Object)

**Derived Query**

- name of the query method has to start with findBy, readBy, queryBy, getBy

- If we want to limit the number of returned query result then we have to add First, Top keyword

- If we want to count the number of rows from the query we have to use countBy

- If we want to select unique result then we use distinct keyword

**Different ways of writing queries in Spring Data JPA**

1. Using predefined methods - save(), saveAll(), findById(), findAll() etc

2. We created our own simple query, where query methods have to start with findBy, readBy, getBy, queryBy, countBy, First, Top operators

3. In case we use complex queries (i.e.) joins, then we have to create our own queries using @Query

- SQL Query - query the table

- HQL Query - query the entity class

If we want to pass parameters to query using ?1 or ?2 like that which is called positional parameter

Named parameter - :id and for this parameter we can pass the value using @Param

4. Instead of writing the query in Repository, we are going to write the hql query in entity class by providing the name which is called as Namedqueries using @NamedQuery

5. Even we can use @Query for simple queries

6. We use like operator in query to check the content, instead we have containing in spring data jpa

7. We can write sql query which queries the table using nativeQuery=true which is called as native query

8. If we write an sql query in entity class, it is identified by a name called as @NamedNativeQuery

9. #entityname - used to fetch the entity name automatically

10. In Spring data JPA, @Query is used to write only select statement. But if we want to write any DML statements(update,delete,insert) then we have to use @Modifying, we have to also include @Transactional

11. Sort class is used to display the output in sorted manner

**SPRING SECURITY**

- Spring Security basically do 2 important jobs **Authentication and Authorization**

- Authentication basically collect the username and password of the user and if the credentials match, it will fetch the role and based on the role we can decide what privileges that particular user have in the application.

- Authorization means which page the user is going to access based on the role

**- To get the authentication details, 3 ways**

1. In-memory authentication - while writing the program itself we statically specify the username, password and role

2. Database authentication - we are going to fetch all credentials from database

3. LDAP

- Apart from above spring security provides 2 more concept

1. OAUTH - Open Authentication - in order to login into the application we use third party credentials like facebook, google etc

2. JWT - JSON Web token

Steps

1. create tables - User and Role

create table role(role\_id int primary key,name varchar(20));

mysql> insert into role values(1,'USER');

mysql> insert into role values(2,'ADMIN');

CREATE TABLE user (user\_id int(11) primary key,username varchar(45) NOT NULL, password varchar(64) NOT NULL,

enabled tinyint(4) DEFAULT 1,role\_id int references role(role\_id));

2. Declare dependency for Spring Security like spring-boot-starter-security

1 User has 1 Role - One to One

1 User has n Role - One to Many

@OneToMany

private List<Role> role;

3. Configure security features

- Since we are going to security concept, it won’t allow to access the request directly, instead it will ask for username and password

- Class has to extends WebSecurityConfigurerAdapter - 2 methods

1. configure(authorization) and configureGlobal(authentication)

2. configure(authorization) and configure(authentication)

HttpSecurity responsible for Authorization

AuthenticationManagerBuilder responsible for Authentication

@Override

protected void configure(HttpSecurity http) throws Exception

{

http.authorizeRequests()

.antMatchers("/registration").permitAll()

.antMatchers("/").hasAnyAuthority("USER", "ADMIN")

.antMatchers("/new").hasAnyAuthority("ADMIN", "USER")

.antMatchers("/editProduct/\*\*").hasAnyAuthority("ADMIN")

.antMatchers("/deleteProduct/\*\*").hasAuthority("ADMIN")

.anyRequest().authenticated()

.and()

.formLogin()

.loginPage("/login")

.permitAll()

.and()

.logout()

.permitAll()

.and()

.exceptionHandling().accessDeniedPage("/403");

}

@Autowired

public void configureGlobal(AuthenticationManagerBuilder auth) throws Exception

{

}

- In order to do db authentication Spring Security provides an inbuild interface called UserDetailsService, it takes username and password from db and check it is correct or not.

-To validate db authentication we use auth.userDetailsService() and takes an object of class as argument

- We create a separate class which implements UserDetailsService interface

- UserDetailsService is called as functional interface contains one abstract method called loadUserByUsername()- this method takes the username and check with database, if it is valid then it returns that particular user and if it is not valid it throws an exception called UsernameNotFoundException

- By default, Spring boot uses some encoding to encode the password - BCRYPT Password encoder, SHA, MD5

-Bcrypt password encoder uses adaptive hash algorithm, it generates different encoder for the same string and always of length 60

- UserDetails interface which contains username, password, list of roles

- GrantedAuthority interface - represents the role info( authority granted to authenicated user)

- class SimpleGrantedAuthority implements GrantedAuthority - concrete implementation of GrantedAuthority

- class User implements UserDetails - core user information retrieve from UserDetailsService

CSRF - Cross Site Request Forgery - unique, secret value that is generated by server side application and transmitted to client to avoid hacking