Spring Boot

Contents

Introduction to Spring Framework

- Refer – https://github.com/sameerbhilare/spring-boot-master-class/tree/master/09.Spring-Introduction-In-10-Steps

Using Spring to Manage Dependencies

- In order for Spring to manage dependencies, we need to answer 3 questions –
    - What are the beans?
    - What are the dependencies of the bean?
    - Where to search for the beans?
- Spring application context is the one which would maintain all the beans.
- `SpringApplication.run()` returns the spring application context (`ApplicationContext`).
- To access/get the spring managed bean, we can use the Spring application context as e.g. `BinarySearchImpl bean = applicationContext.getBean(BinarySearchImpl.class);`
-

What are the beans?

- We annotate a class with **@Component** annotation to let Spring know that the class is a bean.
- Classes annotated with @Component will be managed by Spring.

What are the dependencies of the bean?

- Use **@Autowired** annotation for the properties of a class so that Spring will manage those dependencies.
- Three options for using autowiring - constructor, setter and neither setter nor constructor. Setter and "neither setter nor constructor" autowiring is the same.
- With Earlier versions of Spring, the recommendation was if you have mandatory dependencies then use constructor injection. For all other dependencies, use setter injection.

Where to search for the beans?

- Basically we need to tell Spring where (in which package) our beans (components) are.
- In order to do so, we use another annotation called **@ComponentScan**.
- However what **@SpringBootApplication** annotation does is, it by default scans the package and the sub packages of the package where this class (annotated with @SpringBootApplication) is. So we may not need to use @ComponentScan.

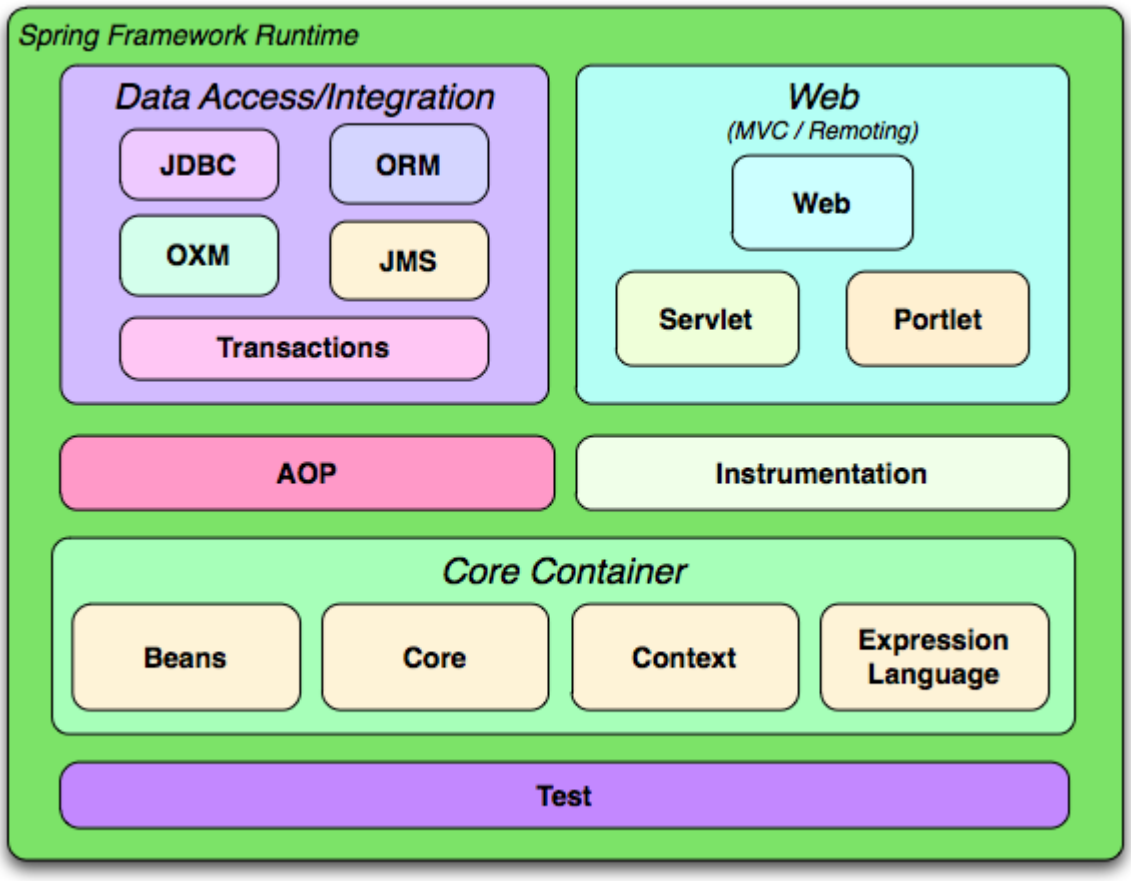
How it works in the background

- Once you run your Spring boot application the class (annotated with @SpringBootApplication) with main method runs.
- Then spring does the Component Scan as spring needs to know where the components are. So it searches for the packages mentioned in @ComponentScan or package/subpackages where the class annotated with @SpringBootApplication is.
- In the component scan, spring searches for the classes annotated with @Component.
- One Spring has figured out what components it needs to manage, then it would start creating the beans and trying to identify the dependencies.
- Once spring identifies dependencies for a component, it will create a bean i.e. it will create an instance of the class by setting the dependencies.

Dynamic Autowiring

- For instance there is QuickSort and BubbleSort classes which implement Sort interface and the Sort interface is used as a dependency in some other class like BinarySearch.
- So what if you have two components on classpath? What does Spring do to resolve that?
- By default, in this case, the Spring boot application will fail to start as it can see the conflicting classes (QuickSort and BubbleSort) and it does not know which one to inject into BinarySearch class/bean.
- **Solution** – use @Primary annotation for one the conflicting classes.
- If you have more than one component matching a specific type, you can use **@Primary** to give more importance to one of those components.

Spring Modules



- One of the important things about Spring is that it's not one big framework. So we have lot of small jars with dedicated purposes.
- This enables you to use specific modules without using the other models of Spring.
- All the Spring modules have the same release version as the Spring Framework.

Spring Projects

- Refer – https://spring.io/projects
- There are other things Spring does other then the Spring Framework and it's modules. These are called Spring projects. E.g. Spring boot, Spring batch, Spring Security, Spring Data, Spring Cloud, etc.
- These Spring projects provide solutions for different problems faced by enterprise applications
- Spring Boot has become the most popular framework used for developing microservices.
- Spring Boot makes it very very easy to develop applications quickly. With features like start up projects, auto configuration, actuator Spring Boot makes developing applications especially micro service a cakewalk.

- Spring Cloud can be used to develop Cloud native applications. We would want to be able to dynamically configure applications, we would be able to dynamically connect them. We would want to be able to dynamically deploy applications.
- Spring data provides a consistent data access through different types of databases (SQL, NoSQL).
- Spring integration addresses problems with application integration. Spring integration helps us in connecting enterprise applications very easily.
- Spring Batch helps in developing batch processing applications.
- Spring Security provides solutions for securing your applications whether it's a web application or whether it's a REST service. Spring Security has support for multiple security options like basic authentication, OAuth authentication, OAuth2 for example.
- Spring HATEOAS enables you to develop HATEOAS compatible services.

Why Spring is so popular

- Spring has survived for more than 15 years (as of March 2021).
- The most important reasons Spring is so popular are
    - It enables writing testable code.
    - There is no plumbing/boilerplate code at all. E.g. Spring makes all its exceptions unchecked so you don't need to handle those
    - Spring brings in the architecture flexibility. Spring is very modular. There are Spring modules and Spring projects for very specific purposes. And I can use a specific Spring module without using all others. If I use Spring in my project my options are not really restricted.
    - It is able to stay with the trend.

Web Application with Spring Boot

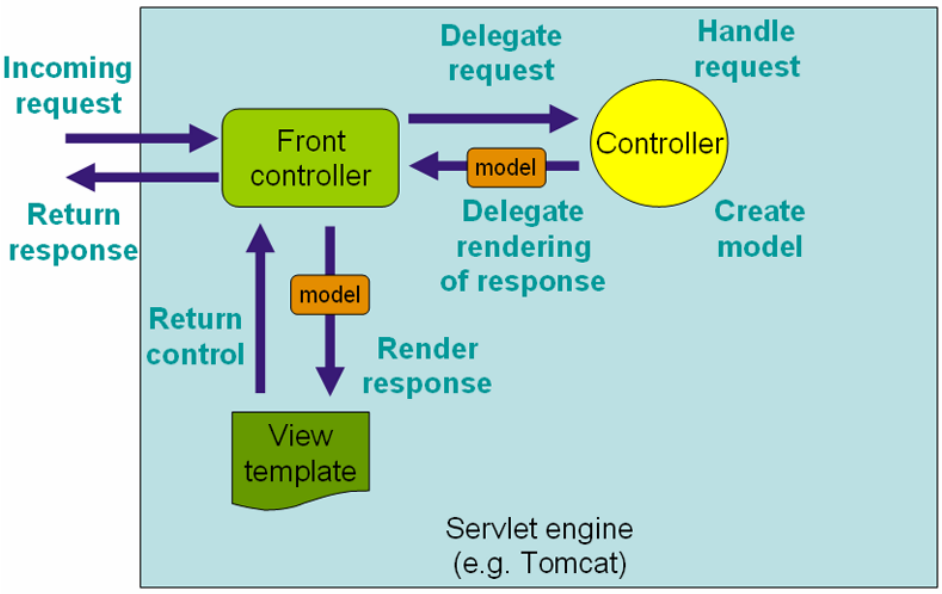
- Refer – https://github.com/sameerbhilare/spring-boot-master-class/tree/master/02.Spring-Boot-Web-Application

Notes

- @RequestMapping
- @Controller
- @ResponseBody – to return value returned from a method directly as response body
- The **spring-boot-starter-parent** specifies the versions of the dependencies. It provides the default plugins and it specifies the default Java versions among a lot of other things.
- **spring-boot-starter-web** basically provides two things – All the dependencies that we need to run web application (spring mvc, spring core, validator api, hibernate validator, logging, Jackson bind, embedded tomcat, etc.)
- Embedded server means the server is part of our application.
- @RequestParam

Spring MVC Request Flow

- DispatcherServlet receives HTTP Request.
- DispatcherServlet identifies the right Controller based on the URL.
- Controller executes Business Logic.
- Controller returns a) Model b) View Name Back to DispatcherServlet.
- DispatcherServlet identifies the correct view (ViewResolver).
- DispatcherServlet makes the model available to view and executes it.
- DispatcherServlet returns HTTP Response Back.



The Magic of Spring

- @Component – generic way to ask spring to manage a bean.
- @Autowired
- @ComponentScan – packages and subpackages which has "beans" to be managed by spring. Spring scans everything in given package and subpackages.
- @Service – is specialization of @Component. It can be use it on anything which is a business service so anything that has business logic.
- @Controller - is specialization of @Component. It is typically used in terms of the MVC.
- @Repository is typically used in terms of a data store.

Session vs Model vs Request

- All the requests parameters have the scope of that specific request.
- Values in Model are by default request scoped. However using @SessionAttributes we can store some specific attributes in session scope.
- To use values across multiple requests, we need to use session scope.
- @SessionAttributes – we need to give the name of the Model attributes which we want to store in session. Used as a "conversational storage".

Web Jars

- Web Jars are basically your static things that typically are your .css, .js files. These are zipped up, created as a jar and made available in the maven repository.
- We can version then similar to normal jars.
- Spring boot MVC autoconfigures the included webjars and serves them via path /webjars/*
- Then we can include them in or JSPs like this –

```
<link href="webjars/bootstrap/3.3.6/css/bootstrap.min.css" rel="stylesheet">

<script src="webjars/jquery/1.9.1/jquery.min.js"></script>
<script src="webjars/bootstrap/3.3.6/js/bootstrap.min.js"></script>
```

Validation API

- Hibernate validator is the implementation of the Java Validation API.
- Steps –
    - Command Bean or Form Backing bean. So we would map values directly from the form to the bean and vice versa.
    - Once we setup a command bean, we can add validations on the command bean using the bean validation API.

    - Once you add validation on the beans, you can use them in the controller or you can enable them in the controller.
    - And if there's a validation error you can show an error in the view.
- For command bean, we need to use it in the controller and in the view we need to use Spring form tags.
- @Valid
- BindingResult

Notes

- @InitDataBinder – to initialize the WebDataBinder. E.g. to map a date form field to date filed in Command Bean. With this we can register custom editors for different types. E.g. dd.mm.yyyy for say Date class.
- JSP Fragments – similar to tiles but we can create different sections of a web page and include those in other main pages. The different sections each has its own file with extension .jspf. e.g. header.jspf, navigation.jspf, footer.jspf.
- Error controller: @Controller("error"). Also we can use @ExceptionHandler.

Spring Security

- It would secure the application so that only somebody who has logged in will be able to access the details of the application.
- Spring Security provides a default login page.
- Include spring-boot-starter-security starter project. With this starter, Spring will automatically configure security around all the URLs of the application. (Males every url protected by enabling basic authentication).
- The default userid is "user' and default password is printed in the logs. Of course, we would use our own credentials from say LDAP or other sources.
- @Configuration – allows us to add more Spring configuration. So it's basically a class where we are defining a few bean and these are to be processed by the spring container to generate bean definitions.

Unit Testing

JUnit Testing

- Imp annotations - @Test, @Before, @After, @BeforeClass,@AfterClass
- Refer – https://github.com/sameerbhilare/spring-boot-master-class/tree/master/03.JUnit-Introduction-In-5-Steps

Mockito

- Mockito is the default mocking framework which is used to test spring boot applications.
- It is already included in the spring-boot-starter project.
- Earlier we would have to create stubs for interface implementations. This had few disadvantages like if interface is changed, we also need to update stubs. Also for different scenarios, we may have to create multiple stubs. (Stub is just an implementation of interface).
- With Mocks, we don't need to create multiple versions of the stub.
- Mocks make it really easy to dynamically create different classes and make them return the data that we would want to return.
- There are lot of useful methods with org.mockito.Mockito class.
- Some Mockito Annotations –
    - @Mock
    - @InjectMocks
    - @RunWith()
- Refer – https://github.com/sameerbhilare/spring-boot-master-class/tree/master/04.Mockito-Introduction-In-5-Steps

Spring Boot Deep Dive

- Refer – https://github.com/sameerbhilare/spring-boot-master-class/tree/master/05.Spring-Boot-Advanced

Spring Boot Developer Tools

- With Spring Boot Developer Tools, by default, any entry on the classpath that points to a folder (/src/java, /src/resource) will be monitored for changes.
- These will not trigger restart - /META-INF/maven, /META-INF/resources ,/resources ,/static ,/public or /templates.
- To exclude folders: spring.devtools.restart.exclude=static/,public/
- To add new Paths : spring.devtools.restart.additional-paths

Spring Profiles

- spring.profile.action=<env>
- We would need application-<env>.properties file then.
- Use @Profile to create a bean based on environment.
- @ConfigurationProperties

Spring Data Rest

- @RepositoryRestResource

Spring Boot **Integration** Test

- With Integration Test, we need to launch entire application and test end to end functionalities.
- @RunWith
- @SpringootTest
- @LocalServerPort
- TestRestTemplate
- JSONAssert

Spring Boot **Unit** Test

- @WebMvcTest – to test a specific Rest Controller
- @MockBean
- @MockMvc

Tips and Tricks

- Spring Boot Version – SNAPSHOT, M1, M2, M3, M4 releases are typically WORK IN PROGRESS. The Spring team is still working on them. Recommend NOT using them.
- Embedded server means the server is part of our application.
- Web Jars are basically your static things that typically are your .css, .js files. These are zipped up, created as a jar and made available in the maven repository.
- For in memory H2 database, if you need some initial data, you can write insert statements in a data.sql file in src/main/resources folder. This will be picked up during server startup so that you will have some data to test with.