Page No.
Date: 26 / 8 / 2024

Introduction To spring

spring Architecture & Modules

Spring framework

* Java based open source framework
* Enterprise Application Development
  Stand alone Application - MS office, Adobe
  Web application - sites
  Distributed Application - App to App communication.

Ex: Amazon - connected to payment services

* Modularized framework & It is independent modules, one module is not depend on another module.
* Simplicity POJOS &
* Dependency Injection & Inverse of control

  Spring creating objects and it inject one class to another class.

* Lightweight framework

Date / /

Data Access/Integration
JDBC ORM
OXM JMS
Transactions

Web
Websocket Servlet
Web portlet

AOP ASPECTS INSTRUMENTATION MESSAGING

CORE CONTAINER
Beans core context SpEL
Test

course outline

Spring core → Spring Boot → Spring Data JPA → SpringBoot
WebMVC

→ Spring Rest → Micro-services.
Spring security
Messaging queues
Redis Cache
Tools
Deployment
Backend Mini project

Spring framework → Rod Jhonson
October-2002

Page No.

Date 03/09/2024

The first version was written by Rod Jhonson. the framework with publication of his book Expert One-on-One J2EE Design and Developments in October 2002.

The framework first released under the Apache 2.0 licence in June 2003.

The first production release 1.0, was released in march 2004.

Spring framework 6.0 has been released on 16 Nov 2022. and came with Java 17.

Spring Boot Developped by Pivotal Team

Page No.

Date / /

① What is Spring boot?

Spring boot is a java-based spring framework used for rapid Application Development (to build stand-alone microservices).

It has extra support of auto-configuration and embeded application severs like tomcat, jetty.

Create standalone spring application with minimal configuration needed.

It has embeded tomcat, jetty which makes it just code and run the application.

Provide production-ready features such as metrics, health checks, and externalized configuration.

Absolutely no requirement for XML configuration.

Spring Boot Feaduries :-

① Spring Application
② Lazy Initialization
③ Admin Features
④ Security
⑤ Logging
⑥ Caching
⑦ Kotlin Support
⑧ Validation
⑨ JSON
⑩ Testing
⑪ Task execution and Scheduling

Page No.
Date / /

② Spring Boot key components?

Spring Boot auto-configuration
Spring Boot CLI
Spring Boot Starter POMs
Spring Boot Actuators.

Spring Boot Auto-Configuration:

Spring Boot auto-configuration attempts to automatically configure your spring application based on jar dependencies you have added.
It reduces the need of manual configuration by providing sensible defaults.

Example:
if you include spring-boot-starter-web in your project. Spring Boot will automatically configure a web server, such as Tomcat and such as Spring MVC.

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

Spring Boot CLI :-
Spring Boot CLI (Command Line Interface) is a command line tool that allows you to quickly prototype with Spring.
It supports Groovy scripts, which means you can write Spring application with less boilerplate code.

```
//app.groovy
@Rest controller
class HelloController {
    @RequestMapping("/")
    String Home() {
        "Hi"
    }
}
```

$ spring run app.groovy

Spring Boot Starter POMs :-

Spring Boot starters are set of convenient dependency descriptors you can include in your application.

They simplify the process of setting up a new Spring Application by providing a curated set of dependencies

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

Spring Boot Actuators :-

Spring Boot Actuators production-ready features to help you monitor and manage you application.

It includes endpoints for health checks, metrics, info and more.

Example: To enable Actuator in you Spring Boot application. add the following dependency.

```
<!-- Maven dependency -->
<dependency>
    <groupId> org.springframework.boot</groupId>
    <artifactId> spring-boot-starter-actuator</artifactId>
</dependency>
```

You can access various endpoints, such as:

- /actuator/health for health status
- /actuator/metrics for application metrics

Configurations you can customize Actuator in application.properties

```
mangement.endpoints.web.exposure.include=health,info,metrics
mangement.endpoints.health.show-details=always.
```

Page No.
Date / /

③ What is the starter dependency of the spring boot module. Here are some most commonly used dependency.

Data JPA starter
Test starter
Security starter
Web starter
Mail starter
Thymeleaf starter.

Data JPA starter:
Provide support for the Java Persistance API using spring Data JPA.

```
<dependency>
    <groupId> org.springframework.book </groupId>
    <artifactId> spring-boot-starter-data-jpa </artifactId>
</dependency>
```

Test starter
Includes all dependencies for testing Spring Boot Applications with JUnit, Hamcrest and Mockito.

```
<artifactId> spring-boot-starter-test </artifactId>
<scope> test </scope>
```

Security starter: Provides supports for securing application.

```
<artifactId> spring-boot-starter-security </artifactId>
```

Web starter: Includes all dependencies for building web app. including spring MVC

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

Mail starter: Provide support for sending emails using Java

```
<artifactId> spring-boot-starter-mail </artifactId>
```

Thymeleaf starter: Thymeleaf template engine with spring Boot

```
<artifactId> spring-boot-starter-thymeleaf </artifactId>
```

Page No.
Date / /

@SpringBootApplications
Annotation is a convenience of that is used to enable a host of features in a Spring Boot application.
it is a combination of three annotations:
① @EnableAutoConfiguration → configure dependencies in class path
② @ComponentScan → Package and subpackages, configuration and
③ @Configuration → define beans

① @EnableAutoConfiguration
It ~~Indicates~~ Enables Spring Boot's auto configuration mechanism

This annotation tells spring boot to start adding beans based on classpath settings, other beans, and various property settings.
It attempts to automatically configure your Spring application based on jar dependencies you have added

② @ComponentScan:
Enables component scanning so that the web controller classes and other components you create will be automatically discovered and registered as bean in the Spring application context.
→ By default, it scans the package of the class that declares the annotation.

③ @Configuration
Indicates that the class can be used by the Spring IoC container as source of a bean definitions

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.SpringBootApplication;
@SpringBootApplication
Public class MyClass {
    P S VM (String[] args){
        SpringApplication.run(MyClass.class,args);}}
```

Page No.
Date / /

Customization:

You can customize the behaviour of @SpringBootApplication
* exclude & Exclude specific auto-configuration classes
* excludeName & Exclude specific auto-configuration classes by name

```
@SpringBootApplication(exclude = {DataSource.class})
public class MyClass {
    psvm(String[] args) {
    SpringApplication.run(MyClass.class, args);
    }
}
```

@RestController and @Controller

@Controller:

@Controller annotation is used to define a traditional spring MVC controller. It is typical used when the controller needs to return HTML views as response, rather than JSON or XML data

Ex: @Controller

```
public class MyController {
  @GetMapping("/home")
  public String homepage() {
    return "home";
  }
  @GetMapping("/api/data")
  @ResponseBody
  public Data getData() {
    return new Data("sample data");
  }
}
```

Date / /

@RestController &

The @RestController annotation is specialized version of @Controller.
It is used to create RESTful web services and automatically serializes return objects into JSON or XML, eliminating the need for the @ResponseBody annotation.

→ The @RestController annotation is essentially a combination of @Controller and @ResponseBody.

Ex: @RestController
@RequestMapping("/api")
public class MyRestController {
@GetMapping("/data")
public Data getData() {
return new Data("sample Data");
}
@PostMapping("/save")
public ResponseEntity<String> saveData(@Request
@RequestBody Data data) {
return ResponseEntity.ok("Data saved sucessfully");
}
}

Inversion of Control

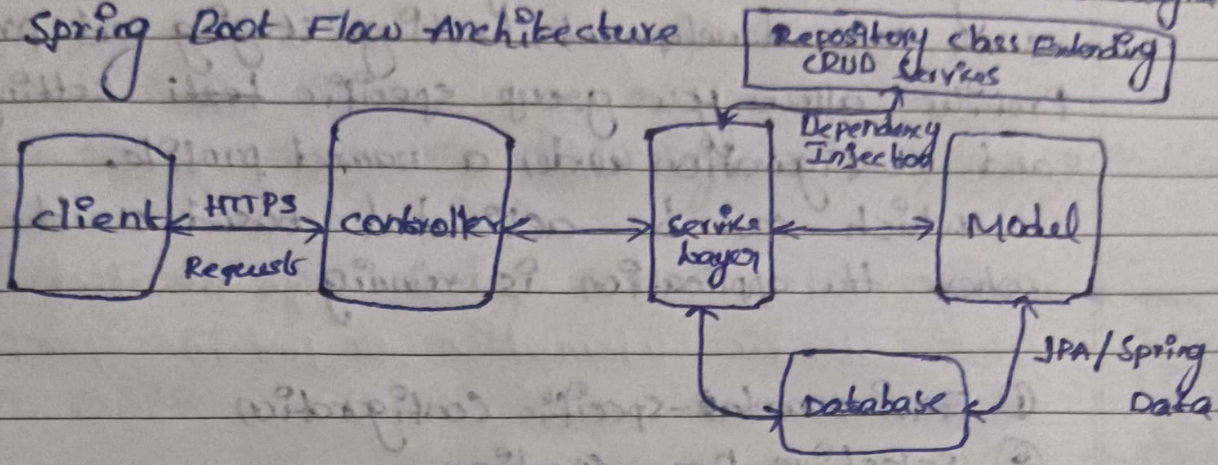
container is a core container
concept in frameworks like Spring, which is responsible for managing the lifecycle, configuration, and dependencies of objects (also known as beans) in an application. The term "Inversion of Control" refers to the way the control of creating and managing objects is inverted, or delegated from the application code to the container.

Page No.
Date / /

FLOW of HTTPS requests through the Spring Boot App?

On a high-level spring boot application follows the MVC pattern which is depicted in the below flow diagram

Spring Boot Flow Architecture



In Spring Framework @RequestMapping and @GetMapping are both annotations used to map HTTP requests to specific handler methods in a controller.
However, they have different scopes and purposes.

@RequestMapping:
The annotation is a versatile and generic annotation that can be used to map HTTP requests to handler methods of MVC and REST controllers.
It can handle different types of HTTP methods (GET, POST, PUT, DELETE, etc).

@GetMapping:
The GetMapping annotation is a shortcut for @RequestMapping that is specially designed for mapping HTTP GET requests to handler methods.

→ To eliminates the "method" coreption in GetMapping.

Page No.
Date / /

Profiles in Spring Boot:

Profiles are a powerful feature used to manage application configurations for different environments such as development, testing, staging and production. Profiles allow you group specific ~~testi~~ settings, beans and configuration under a named profile.
Which can be activated based on the environment where the application is running.

1. Environmental-specific Configuration
2. Selective Bean creation
3. Configuration File Management
4. Simplified Deployment Process

using Environmental variables:

```
export SPRING_PROFILES_ACTIVE=dev
```

Application Properties:

```
spring.profiles.active=dev
```

@Profile Annotation:

```
@Profile("dev")
public class DevConfig {
}
```

creating profile Specific configuration:

application.properties (default configuration)
application-dev.properties
application-prod.properties
application-test.properties

Page No.

Date 04/09/2024

Spring Actuator:

Spring Actuator is a powerful feature in the Spring Boot framework that helps monitor and deploy manage Spring Boot applications. It provides production ready features such as metrics, health checks, and monitoring, making it easier to understand what's happening inside your application.

Key Features of Spring Actuator

1. Health check Endpoints:

The health endpoint provides basic application health information, such as whether the application is up or down. It can be extended to include custom health indicator that can check the status of components such as databases, message brokers, or any other external dependencies.

2. Metrics:

Actuator provide various metrics that can be used to monitor the performance of your application. These include JVM metrics, memory usage, garbage collections information and more.

The metrics endpoint allows you to view the values of various metrics in your application.

3. Application Info

The info endpoint can be used to expose information about the application, such as build details. (e.g: version, time, etc.).

4. Auditing: Spring Actuator also supports auditing, which allows you to track changes and actions performed within the application, such as security-related events.

Page No.
Date / /

⑤ HTTP Tracing:
The httptrace endpoints provides details about HTTP request/response exchanges, which can be useful for diagnosing issues related to HTTP traffic in your application.

⑥ Custom Endpoints:
Spring Actuator allows developers to create custom endpoints that can expose specific information or perform specific actions.

Examples:
1. Enable Spring Actuator in your Spring Boot Application:
spring-boot-starter-actuator in your pom.xml.

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

2. Basic Usage of Health Endpoint:
Once you've added Actuator, a simple HTTP GET request to /actuator/health

```
{
  "status": "UP"
}
```

3. Custom Health Indicator: You can create a custom health indicator by implementing the HealthIndicator interface

```
public class customHealthIndicator implements HealthIndicator
{
}
```

Page No.
Date / /

4. Securing Actuator Endpoints:

To secure specific Actuator endpoints like health and info and allow only authenticated users to access them, you can configure this in the application.properties

```
#Expose only specific Actuator endpoints (health and info)
mangement.endpoints.web.exposure.include=health,info

#Require authentication for all Actuator endpoints
mangement.endpoint.health.show-details=when-authorized
mangement.endpoint.health.roles=ACTUATOR,ADMIN
management.endpoints.web.base-path=/actuator

#Configure Spring Security for basic authentication
spring.security.user.name=admin
spring.security.user.password=adminappspassword
spring.security.user.roles=ACTUATOR
```

5. Exposing custom Application:

You can expose custom information such as the application's name, description and version in the info

application.properties

```
#custom application info
info.app.name=MyApp
info.app.description=custom my app
info.app.version=1.0.0
info.company.name=Mycompany
info.company.department=Engineering
```

hit the -> /actuator/info endpoin in postman.

```
{ "app": {
    "name": "MyApp",
    "description": custom myapp
    "version": "1.0.0"
}
```