Spring Framework

Name : Ameya Joshi
Email : ameya.joshi_official@outlook.com
Cell No. : +91 98 506 76160

1

1

What is Spring?

* Lightweight container framework
  * Lightweight - minimally invasive
  * Container - manages app component lifecycle
  * Framework - basis for enterprise Java apps
  * Open source
* Apache licensed

2

2

Spring framework

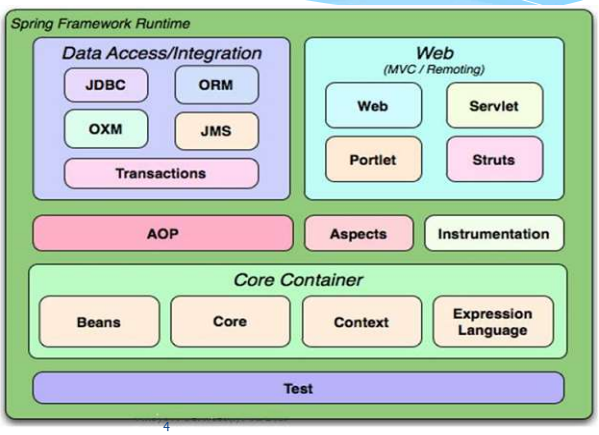
- Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications.
- Spring handles the infrastructure so you can focus on your application.
- Spring enables you to build applications from
  "plain old Java objects" (POJOs) and to apply enterprise services non-invasively to POJOs.

3

3

Spring Framework modules

- Spring IOC container(core container)
- Spring AOP
- Spring DAO
- Spring ORM
- JEE Module
- Spring web MVC



4

4

Spring Core Container

* The Spring core container provides an implementation for IOC (Inversion of control) supporting dependency injection.
* IOC is an architectural pattern that describes to have an external entity to perform Dependency injection (DI) at creation time, that is, object creation time.
* Dependency Injection is a process of injecting/pushing the dependencies into an object.

5

5

DAO

* Data Access Object(DAO) is a design pattern the describes to separate the persistence logic from the business logic.
* The Spring DAO includes a support for the common infrastructures required in creating the DAO.
* Spring includes DAO support classes to take this responsibility such as:
  * Jdbc DAO support
  * Hibernate DAO support
  * JPA DAO support

6

6

ORM

* The Object/Relation Mapping(ORM) module of Spring framework provides a high level abstraction for well accepted object-relational mapping API's such as Hibernate, JPA, OJB and iBatis.
* The Spring ORM module is not a replacement or a competition for any of the existing ORM's, instead it is designed to reduce the complexity by avoiding the boilerplate code from the application in using ORMs.

7

7

JEE

* The JEE module of Spring framework is build on the solid base provided by the core package.
* This provides a support for using the remoting services in a simplified manner.
* This supports to build POJOs and expose them as remote objects without worrying about the specific remoting technology given rules.

8

8

Web

* This part of Spring framework implements the infrastructure that is required for creating web based MVC application in java.
* The Spring Web MVC infrastructure is built on top of the Servlet API, so that it can be integrated into any Java Web Application server.
* This uses the Spring IOC container to access the various framework and application objects.

9

9

Configuration Metadata

* XML-based configuration metadata.
* Java (Annotation) -based configuration

10

10

XML-based configuration

```
<beans>
        <bean id="msgBean" class="com.ameya.Message">
                <property name="message"
                value=" Hello World ! "/>
        </bean>
</beans>
```

Message msgBean=new Message();

msgBean.setMessage("Hello World");

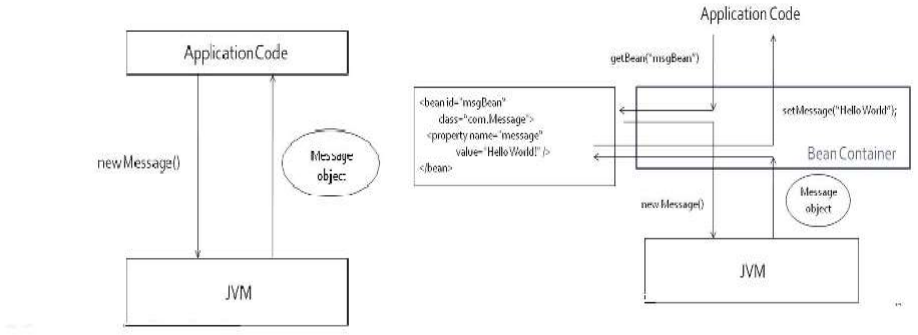
* The string " Hello World " is injected to the bean msgBean

11

11

Understanding Bean Container

- Without a bean container
- With a bean container



12

12

Dependency Injection Types

* Setter Injection
* Constructor Injection
* Method Injection
* Interface Injections

13

13

Create the Account.java:

```
package com.ameya.models;
public class Account {
private String firstName, lastName;
private double balance;
private Address address;
public Account(){
address = new Address();
}
public Account(String firstName, String lastName, double balance) {
this.firstName = firstName;
this.lastName = lastName;
this.balance = balance;
this.address = new Address();
}
public Account(String firstName, String lastName, double balance, Address address)
{this.firstName=firstName;
this.lastName=lastName;
this.balance = balance;
this.address = address;
}
//Setter & Getter
}
```

14

14

Create the Address.java:

```
package com.ameya.models;
public class Address {
private String building, street, city, state, country;
public Address() {}
public Address(String building, String street, String city, String state,
String country) {this.building = building;
this.street = street;
this.city = city;
this.state = state;
this.country = country;
}
//Setter & Getter
}
```

15

15

Constructor-based dependency Injection

```
<beans…>
        <bean id="account5"
        class="com.ameya.models.Account">
                <constructor-arg name="firstName"
                value="Ameya"/>
                <constructor-arg name="lastName" value=
                "Joshi"/>
                <constructor-arg name="balance" value=
        "lOOOO"/>
        </bean>
</beans>
```

16

16

Setter-based dependency Injection

```
<bean id="account6" class="com.ameya.models.Account">
        <property name="firstName" value="Ameya"/>
        <property name="lastName" value="Joshi"/>
        <property name="balance" value="10000"/>
        <property name="address" ref=address/>

</bean>
<bean id="address" class=com.ameya.models.Address>
        <property name="building" value="Champions"/>
        <property name="street" value= "JM Road">
        <property name="city" value="Pune"/>
        <property name="country" value="India"/>
        <property name="state" value="MH"/>
</bean>
```

17

17

Autowiring

* The Spring container can autowire relationships between collaborating beans.
* You can allow Spring to resolve collaborators (other beans) automatically for your bean by inspecting the contents of the ApplicationContext.

18

18

@Autowired

* The @Autowired annotation provides more fine-grained control over where and how autowiring should be accomplished.
* The @Autowired annotation can be used to auto wire bean on the setter method just like @Required annotation, constructor, a property or methods with arbitrary names and/or multiple arguments.

19

19

@Autowired on Setter Methods

* You can use @Autowired annotation on setter methods to get rid of the <property> element in XML configuration file.
* When Spring finds an @Autowired annotation used with setter methods, it tries to perform byType autowiring on the method.

```
import org.springframework.beans.factory.annotation.Autowired;
public class TextEditor {
        private SpellChecker spellChecker;
        @Autowired
        public void setSpellChecker( SpellChecker spellChecker ){
                this.spellChecker = spellChecker;
        }
.....
}
```

20

20

@Autowired on Properties

* You can use @Autowired annotation on properties to get rid of the setter methods.
* When you will pass values of auto wired properties using @Autowired Spring will automatically assign those properties with the passed values or references.

```
import org.springframework.beans.factory.annotation.Autowired;
public class TextEditor {
        @Autowired
        private SpellChecker spellChecker;
....
}
```

21

21

Bean Scopes

* When you create a bean definition, you create a recipe for creating actual instances of the class defined by that bean definition.
* The idea that a bean definition is a recipe is important, because it means that, as with a class, you can create many object instances from a single recipe.

22

22

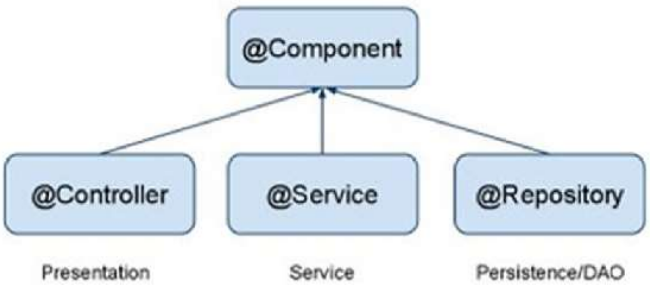
Bean Scopes

* Singleton
* Prototype
* Request
* Session

23

23

StereoType Annotations



24

24

Annotation-based container configuration

* Starting from Spring 2.5 it became possible to configure the dependency injection using annotations. So instead of using XML to describe a bean wiring, you can move the bean configuration into the component class itself by using annotations on the relevant class, method, or field declaration.
* Annotation injection is performed before XML injection, thus the later configuration will override the former for properties wired through both approaches.

25

25

* Annotation wiring is not turned on in the Spring container by default. So, before we can use annotation-based wiring, we will need to enable it in our Spring configuration file.
* To do this You need to add the context schema and the below tag to xml file
  * **<context: annotation-config />**

26

26

Annotations for Configuration

* Java/annotation based configuration option enables you to write most of your Spring configuration without XML but with the help of few Java-based annotations.
* @Configuration
* @Bean
* @Import
* @Scope

27

Spring MVC

* The Spring web MVC framework provides model-view-controller architecture and ready components that can be used to develop flexible and loosely coupled web applications.
* The Model encapsulates the application data and in general they will consist of POJO.
* The View is responsible for rendering the model data and in general it generates HTML output that the client's browser can interpret.
* The Controller is responsible for processing user requests and building appropriate model and passes it to the view for rendering.
* Thus MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.

28

Request Mappings

* RequestMappings are really flexible
* You can define a @RequestMapping on a class and all methods
* @RequestMappings will be relative to it.
* There are a number of ways to define them:
  * URI Patterns
  * HTTP Methods (GET, POST, etc)
  * Request Parameters
  * Header values

29

29

RequestMapping - Class Level

```
package com.ameya.controllers;

@RequestMapping ("/portfolio")
@RestController
public class PortfolioController {
        @RequestMapping("/create")
        public String create() {
                return "create";
        }
}
```

* The URL for this (relative to your context root)would be:
  /portfolio/create

30

30

RequestMapping - HTTP Methods

```
package com.ameya.controllers;

@RequestMapping("/portfolio")
@RestController
public class PortfolioController {
        @RequestMapping(value="/create",method=RequestMethod.POST)
        public String save() {
                return "view";
        }
}
```

* Same URL as the previous example, but responds to POSTs

31

31

RequestMapping - Request Params

```
package com.ameya.controllers;

@RequestMapping("/portfolio")
@RestController
public class PortfolioController {
@RequestMapping(value="/view",params="details=all")
public String viewAll() {
return "viewAll";
}
}
```

* This will respond to
  /portfolio/view?details=all

32

32

RequestMapping - URI Templates

```
package com.ameya.controllers;

@RequestMapping("/portfolio")
@RestController
public class PortfolioController {
        @RequestMapping ("/viewProject/{projectId}")
        public String viewProject(@PathVariable("projectId") long
projectId) {
                return "viewProject";
        }
}
```

* The URL for this (relative to your context root) would be:
  /portfolio/viewProject/10

33

33

Spring REST

34

34

Introduction to REST

REST stands for Representational State Transfer

* It is an architectural ***pattern*** for developing web services as opposed to a ***specification***.
* REST web services communicate over the HTTP specification, using HTTP vocabulary:
  * Methods (GET, POST, etc.)
  * HTTP URI syntax (paths, parameters, etc.)
  * Media types (xml, json, html, plain text, etc)
  * HTTP Response codes.

35

Introduction to REST

* Representational
  * Clients possess the information necessary to identify, modify, and/or delete a web resource.
* State
  * All resource state information is stored on the client.
* Transfer
  * Client state is passed from the client to the service through HTTP.

36

Introduction to REST

The six characteristics of REST:

1. Uniform interface
2. Decoupled client-server interaction
3. Stateless
4. Cacheable
5. Layered
6. Extensible through code on demand (optional)

* Services that do not conform to the above required contstraints are not strictly RESTful web services.

37

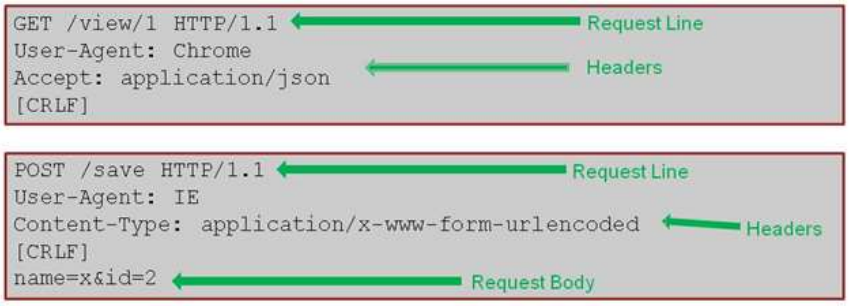
HTTP-REST Request Basics

* The **HTTP request** is sent *from the client*.
  * Identifies the location of a **resource**.
  * Specifies the **verb**, or HTTP **method** to use when accessing the resource.
  * Supplies optional **request headers** (name-value pairs) that provide additional information the server may need when processing the request.
  * Supplies an optional **request body** that identifies additional data to be uploaded to the server (e.g. form parameters, attachments, etc.)

38

HTTP Message

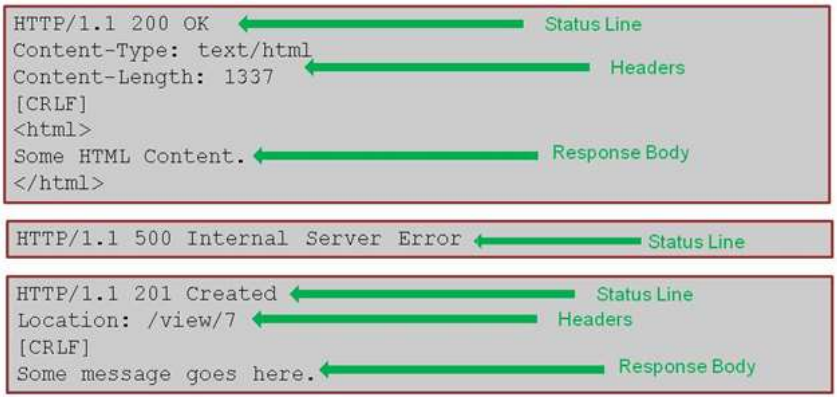
* What does an HTTP message look like?



39

39

HTTP Message - Responses



40

40

RequestBody

* Annotating a handler method parameter with @RequestBody will bind that parameter to the request body

```
@RequestMapping("/echo/string")
public void writeString(@RequestBody String input) { }

@RequestMapping("/echo/json")
public void writeJson(@RequestBody SomeObject input) { }
```

41

41

ResponseBody

* Annotating a return type with @ResponseBody tells Spring MVC that the object returned should be treated as the response body

```
@RequestMapping("/echo/str±ng")
public @ResponseBody String readString() { }

@RequestMapping("/echo/json")
public @ResponseBody SomeObject readJson() { }
```

42

42

Other parts of the HttpMessage

* What if you need to get/set headers?
* Or set the status code?

```
@RequestMapping("/echo/string")
public String echoString(
                    @RequestBody String input,
                     HttpServletRequest request,
                     HttpServletResponse response) {
          String requestType = request.getHeader("Content-Type");
          response.setHeader("Content-Type", "text/plain");
          response.setStatus(200);
          return input
}
```

43

43

@ResponseStatus

* There is a convenient way to set what the default status for a particular handler should be

```
@RequestMapping("/create")
@ResponseStatus(HttpStatus.CREATED) // CREATED = 201
public void echoString(String input) {
}
```

44

44

SPRING BOOT

45

Spring Framework Limitations

* Huge framework
* Multiple setup steps
* Multiple configuration steps
* Multiple Build and Deploy steps
* **Can We abstract these all steps?**

46

What is Spring Boot?

* Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can "just run".

47

What is Spring Boot?

* Opinionated (It makes certain assumptions)
* Convention Over Configuration
* Stand alone
* Production ready
* Spring module which provides RAD (Rapid Application Development) feature to Spring framework.

48

Features

* Create stand-alone Spring applications
* Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
* Provide opinionated 'starter' POMs to simplify your Maven configuration
* Automatically configure Spring whenever possible
* Provide production-ready features such as metrics, health checks and externalized configuration
* Absolutely **no code generation** and **no requirement for XML** configuration

49

Setup Spring Boot

* Pre-requisites
  * Hardware
    * Core i5 machine
    * 8 gb ram
  * Software
    * 64 bit Windows 7/10
    * Java 1.8 or higher
    * Spring Tools Suite (We use sts 4.x)

50

Setup Spring Boot

* Install Maven
  * Set MAVEN_HOME
  * Add it to PATH Environment Variable
  * Run mvn –version from command prompt to ensure maven is installed

51

Setup Spring Boot

* Start STS
* Create new Maven Project
  * In the STS UI
    * Check Create a simple project
    * Click on next
    * Enter Group Id (com.ameya)
    * Enter Artifact Id (course-api)
    * Enter Version (Keep default)
    * Enter Name (Ameya Joshi Course Api)

52

Setup Spring Boot

* Add following in pom.xml ,
* save the file and update the Maven Project

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.3.0.RELEASE</version>
</parent>
 <dependencies>
        <dependency>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
 </dependencies>
 <properties>
        <java.version>1.8</java.version>
 </properties>
```

53

Few more dependencies.

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

<dependency>
    <groupId>org.apache.derby</groupId>
     <artifactId>derby</artifactId>
      <scope>runtime</scope>
</dependency>
```

54

Writing First App

* **Type following code and run as java application**

```
package com.ameya;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CourseApiApp {

        public static void main(String[] args) {
                SpringApplication.run(CourseApiApp.class, args);

        }

}
```

55

Behind the Scenes

SpringApplication.*run(CourseApiApp.**class, args**);*

This runs the CourseApiApp class

This class is annotated with @SpringBootApplication

The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration, and @ComponentScan

56

Behind the Scenes

* As a result Spring Boot :
* Sets up the default configuration
* Starts Spring application context
* Performs classpath scan
* Starts tomcat server

57