Spring Framework

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

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

Spring framework

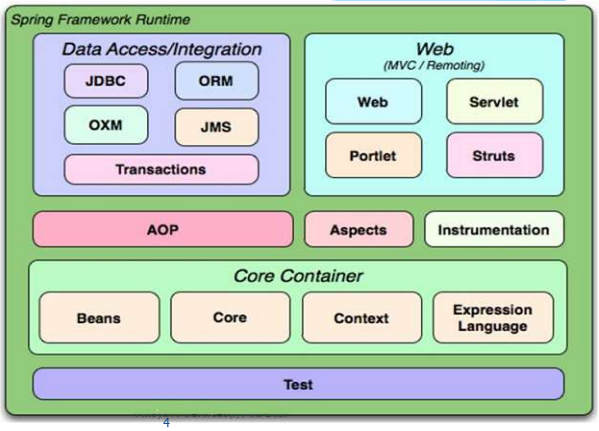
- Spring Framework is a Java platform that provides comprehensive infrastructure support for developing Java applications.
- Spring handles the infrastructure so you can focus on your application.
- Spring enables you to build applications from

  "plain old Java objects" (POJOs) and to apply enterprise services non-invasively to POJOs.

3

Spring Framework modules

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



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

Spring AOP

* Spring AOP module provides an implementation of AOP (Aspect Oriented Programming).
* Spring is a proxy-based framework implemented in Java.
* AOP integrates the concerns dynamically into the system.
* Concerns: A part of the system divided based on the functionality (transaction, security, logging, caching).

6

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

7

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.

8

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.

9

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.

10

What is IOC ?

* IOC is also known as dependency injection (DI).
* Dependency Injection is the act of injecting dependencies into an Object.
* Inversion of Control is the general style of using Dependency injection to wire together application layers.
* Hence Spring is an Inversion of Control container. That is, it is a container that handles Dependency Injection for you.

11

Configuration Metadata

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

12

XML-based configuration

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

Message msgBean=new Message();

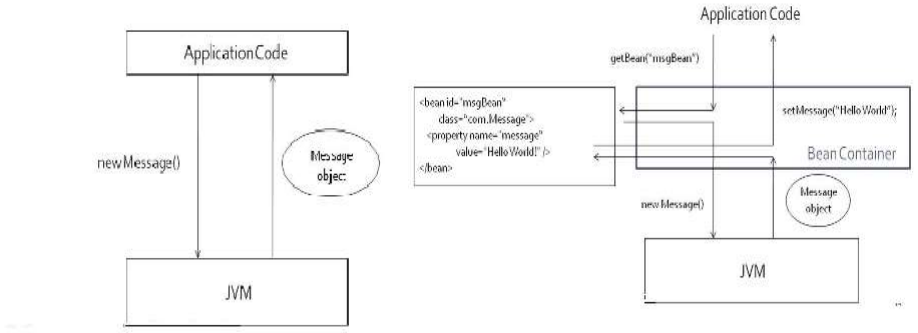
msgBean.setMessage("Hello World");

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

13

Understanding Bean Container

- Without a bean container
- With a bean container



14

Bootstrapping the IOC container

* To start an app using IOC :
  * Create an ApplicationContext object and tell it where configuration beans.xml is.
  * ApplicationContext appContext = newClassPathXmlApplicationContext("beans.xml");
  * This just has to be done once on startup, and can be done in the main method or whatever code bootstraps the application.

15

Dependency Injection Types

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

16

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
}
```

17

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
}
```

18

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>
```

19

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>
```

20

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.

21

@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.

22

@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;
        }
.....
}
```

23

@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;
....
}
```

24

@Autowired on Constructor

* You can apply @Autowired to constructors as well.
* A constructor @Autowired annotation indicates that the constructor should be autowired when creating the bean, even if no <constructor-arg> elements are used while configuring the bean in XML file.

```
import org. springframework.beans.factory.annotation.Autowired;
public class TextEditor {
        private SpellChecker spellChecker;
        @Autowired
        public TextEditor(SpellChecker spellChecker){
                System.out.println("Inside TextEditor constructor.");
                this.spellChecker = spellChecker;
        }
....
}
```

25

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.

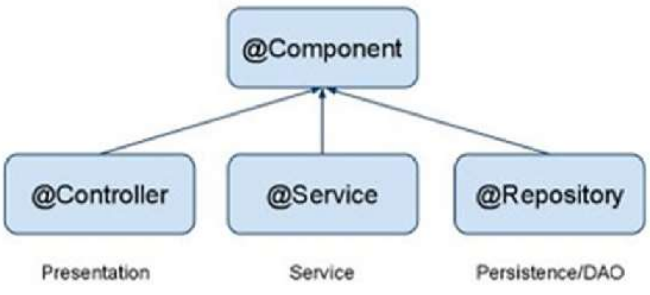
26

Bean Scopes

* Singleton
* Prototype
* Request
* Session

27

StereoType Annotations



28