**Spring**

**What is Spring?**

Spring is a dependency injection framework to make java application loosely coupled.

Spring frame makes the easy development of JavaEE application.

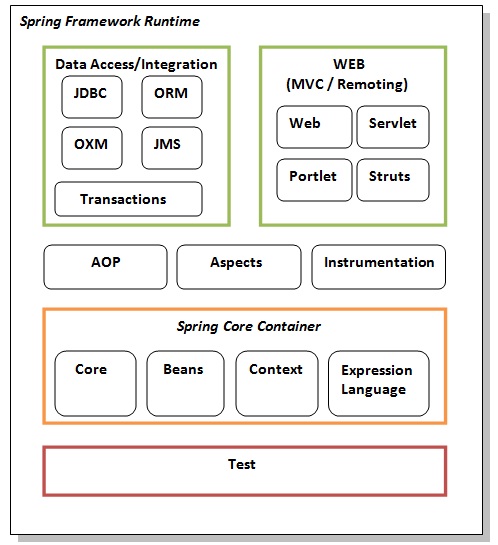
**It was developed by Rod Johnson in 2003.**

UI Layer

Business/Services Layer

Data Access Layer(DAO)

Spring Module:-



Test

This layer provides support of testing with JUnit and TestNG.

Spring Core Container

The Spring Core container contains core, beans, context and expression language (EL) modules.

Core and Beans

These modules provide IOC and Dependency Injection features.

Context

This module supports internationalization (I18N), EJB, JMS, Basic Remoting.

Expression Language

It is an extension to the EL defined in JSP. It provides support to setting and getting property values, method invocation, accessing collections and indexers, named variables, logical and arithmetic operators, retrieval of objects by name etc.

AOP, Aspects and Instrumentation

These modules support aspect oriented programming implementation where you can use Advices, Pointcuts etc. to decouple the code.

The aspects module provides support to integration with AspectJ.

The instrumentation module provides support to class instrumentation and classloader implementations.

Data Access / Integration

This group comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database.

Web

This group comprises of Web, Web-Servlet, Web-Struts and Web-Portlet. These modules provide support to create web application.

**Dependency Injection:-**

It is design pattern.

Eg:-Spring helps the two class dependent each other for some work when ever Class A own object created it is tightly coupling to resolved this problem spring framework loosely coupling.

In spring IOC (Inversion of control) container responsibility all the object created own, through annotation or web.xml

**Dependency Injection Two ways:-**

Information providing to the **configuration file** which will you use.

**Setter Injection:-**

IOC container object create at runtime it will called **setter** method to use putting the values. It is also called properties injection.

**Constructor Injection:-**

IOC container object create at by the help of constructor.

Data Types:-

1. Primitive Data Types

2. Collection Type

3. Reference Type

**Maven project create:-**

1. Create maven project( **org.apache.maven.archetypes**--------**maven-archetypes-quickstart )**
2. Adding dependencies=>spring core ,Spring context
3. Creating configure file =>config.xml

Intialixing bean:-

ApplicationContext context= new ClassPathXmlApplicationContext("config.xml");

Student student1=(Student) context.getBean("student1");

System.out.println(student1);

**Values as a element**

<bean class="com.SpringCore.Student" name="student1">

<property name="studentId">

<value>101</value>

</property>

<property name="studentName">

<value>Mirza</value>

</property>

<property name="studentAddress">

<value>Odisha</value>

</property>

</bean>

**Values as a Attributes**

<!--This is our Second beans values as a Attributes -->

<bean class="com.SpringCore.Student" name="student2">

<property name="studentId" value="102" />

<property name="studentName" value="Satya" />

<property name="studentAddress" value="Odisha" />

</bean>

**Values using p-schema**

<!--This is our Third beans values using p-schema -->

<bean class="com.SpringCore.Student" name="student3" p:studentId="103" p:studentName="Anil" p:studentAddress="Odisha" />

**Collection type injecting:-**

<bean class="com.CollectionTypeInjecting.Employee" name="emp1">

<property name="name" value="Mirza"/>

<property name="phones" >

<list>

<value>75404321534</value>

<value>9776331132</value>

</list>

</property>

<property name="address">

<set>

<value>Odisha</value>

<value>Banglore</value>

</set>

</property>

<property name="courses">

<map>

<entry key="Java" value="2months" />

<entry key="Python" value="1months" />

</map>

</property>

<property name="props">

<props>

<prop key="name">Mirza</prop>

<prop key="Country">India</prop>

</props>

</property>

</bean>

**Reference Type Injecting:-**

<bean class="com.ReferenceInjection.B" name="bref">

<property name="y" value="458"/>

</bean>

<bean class="com.ReferenceInjection.A" name="aref">

<property name="x" value="45568"/>

<property name="obj" >

<ref bean="bref"/>

</property>

</bean>

<bean class="com.ReferenceInjection.B" name="bref1" p:y="10" />

<bean class="com.ReferenceInjection.A" name="aref1" p:x="20" p:obj-ref="bref1"/>

public class Driver {

public static void main(String[] args) {

ApplicationContext context=new ClassPathXmlApplicationContext("com/ReferenceInjection/Referenceconfig.xml");

A a=(A)context.getBean("aref");

System.out.println(a);

System.out.println(a.getX());

System.out.println(a.getObj().getY());

System.out.println(a.getX()+a.getObj().getY());

A a1=(A)context.getBean("aref1");

System.out.println(a1);

System.out.println(a1.getX());

System.out.println(a1.getObj().getY());

System.out.println(a1.getX()+a1.getObj().getY());

}

}

**Construnctor Injection Example:-/And C-schema**

<bean class="com.consructorInjection.Certificat" name="cref">

<constructor-arg value="BPUT"/>

</bean>

<bean class="com.consructorInjection.Person" name="person1">

<constructor-arg value="Mirza"/>

<constructor-arg value="101" type="int"/>

<constructor-arg ref="cref"/>

</bean>

<!-- This is C-schema -->

<bean class="com.consructorInjection.Certificat" name="cref1" c:name="GITAM"/>

<bean class="com.consructorInjection.Person" name="person2" c:name="Satya" c:personId="102" c:cert-ref="cref1" />

**Life Cycle of Beans:-**

1. **By xml**
2. **By using interface**
3. **By Annotation**

**By XML:-**

**Public void init()->** Using Intializing,code,Loading,config,Connection db,Webservice etc…

**Public void destoy()->**Using for clean operation

You want to destroy method called first import **AbstractApplicatioContext**

Then called **registerShutdownHook() it is present in AbstractApplicatioContext.**

import org.springframework.context.support.AbstractApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Driver {

public static void main(String[] args) {

AbstractApplicationContext context=new ClassPathXmlApplicationContext("com/Lifecycle/ciconfig.xml");

Biscuit biscuit=(Biscuit) context.getBean("item1");

System.out.println(biscuit);

}

}

<bean class="com.Lifecycle.Biscuit" init-method="init" destroy-method="destroy" name="item1" p:price="15"/>

**Using Interface implements configuration bean lifecycle:-**

1.class implementing InitializingBean,DisposableBean interface ,and implementing method.

There are two type :-

* afterPropertiesSet()-It is also like init method
* destroy()

By Using Annotation Beans Life cycle:-

Adding dependencies and importing the package then use

And enable all annotation to configure inxml file using <context:annotation-config/> tag.

* @PostConstruct
* @PreDestroy

@PostConstruct

public void start() {

System.out.println("Strating method");

}

@PreDestroy

public void end() {

System.out.println("Ending method");

}

Autowiring:-

* Features of spring Framework in which spring container inject the dependencies automatically.
* Autowiring cannot be used to inject primitive and string values.It works with reference only.
* By the help of xml and annotation(@Autowiring)

**Advantage of Autowiring**

* It requires the less code because we don't need to write the code to inject the dependency explicitly.

**Disadvantage of Autowiring**

* No control of programmer.
* It can't be used for primitive and string values.
* There are many autowiring modes:

|  |  |  |
| --- | --- | --- |
| **No.** | **Mode** | **Description** |
| 1) | no | It is the default autowiring mode. It means no autowiring bydefault. |
| 2) | byName | The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method. |
| 3) | byType | The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method. |
| 4) | constructor | The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters. |
| 5) | autodetect | It is deprecated since Spring 3. |

@Autowired annotation:-

* @Autowired on Properties
* @Autowired on Constructor
* @Autowired on Setter method

|  |  |  |
| --- | --- | --- |
| Autowired on Properties | Autowired on Constructor | Autowired on Setter method |
| S  Y  N  T  A  X | **public** **class** Employee {  @Autowired  **private** Address address;  } | @Autowired  **public** Employee(Address address) {  **super**();  **this**.address = address;  } | @Autowired  **public** **void** setAddress(Address address) {  **this**.address = address;  } |

@Qulifier annotation:-

By using the @Qualifier annotation, we can **eliminate the issue of which bean needs to be injected**. By including the @Qualifier annotation, together with the name of the specific implementation we want to use.

**Syntax:-**

public class Employee {

@Autowired

@Qualifier("address1")

private Address address;

@Component Annotation:-

@Component is **a class-level annotation**. It is used to denote a class as a Component. We can use @Component across the application to mark the beans as Spring's managed components. A component is responsible for some operations.

**Syntax:-**

@Component("obj")

public class Student {

@Value("Mirza")

private String studentName;

@Value("Bhubaneswar")

private String city;

Value using in List object:-

In xml file create standalone list and declare some values.

**Syntax:-**

@Value("#{temp}")

**private** List<String> address;

**Bean Scope:-**

1. Singleton:- This scopes the bean definition to a single instance per Spring IOC container (default).
2. Prototype:- This scopes a single bean definition to have any number of object instances.
3. Request:- This scopes a bean definition to an HTTP request. Only valid in the context of a web-aware Spring ApplicationContext.
4. Session:- This scopes a bean definition to an HTTP session. Only valid in the context of a web-aware Spring ApplicationContext.
5. Globalsession:- This scopes a bean definition to a global HTTP session. Only valid in the context of a web-aware Spring ApplicationContext(Ported application).

**Prototype:-**

**Declaration:-**

|  |  |
| --- | --- |
| **Annotation** | **Beans** |
| @Component("obj")  @Scope("prototype")  **public** **class** Student {  @Value("Mirza")  **private** String studentName;  @Value("Bhubaneswar")  **private** String city;  @Value("#{temp}") | <bean class=*"com.stereotype.Teacher"* name=*"teacher"* scope=*"prototype"*/> |

**Spring Expression Language(SpEL):-**

It supports Parsing and executing expression with the help of @Value annotation.

Syntax:-@Value(“#{Expression}”)

Classes, Variables, Objects, Methods, Constructors and

Symbol(char, numeric, operator, Keywords and special symbols which returns a value)

Example:-

@Value(“#{11+22}”)

@Value(“#{8>7?88:66}”)

How to invoke static method and variable in SpEL?

Static method:-

T(class).method(parameter) -> @Value("#{ T(java.lang.Math).sqrt(25) }")

Variable:-

T(class).Variable -> @Value("#{ T( java.lang.Math).E }")

Object:-

new Object(Value)-> @Value("#{new java.lang.String('Mirza')}")

@Bean

It is method level annotation, It is used to explicitly declare a single bean rather than letting Spring do it automatically.

@Component

It is method level annotation,It any class is annotated with @Componenet it will be automatically detect by using class path scan.

@ComponentScan

@ComponentScan annotation is used with the @Configuration annotation to tell Spring the packages to scan for annotated components. @ComponentScan also used to specify base packages and base package classes using thebasePackageClasses or basePackages attributes of @ComponentScan.