**Spring Boot**

Day 16

Handson

1. Create User bean class with id, name, address, salary and inject the values using setter injection, constructor injection and print the output

J2EE - to develop web oriented appl - J2EE Components - Traditional approach

   1. JSP - Java Server Pages - used to create dynamic web pages - UI Part

   2. Servlets  - java prg used to write business logic

   3. EJB - Enterprise Java Bean - 3 beans

         1. Session bean - write business logic

         2. Entity bean - persist into db

         3. Message Driven bean - used to send sync and async messages

Problems with traditional approach

1. J2EE applications tend to contain excessive amounts of "plumbing" code -> Thers would always be a high proportion of code that doesn't do anything: JNDI lookup code, Transfer Objects, try/catch blocks to acquire and release JDBC resources. Writing and maintaining such plumbing code proves a major drain on resources that should be focused on the application's business domain.

2. J2EE applications are hard to unit test -> The J2EE APIs, especially, the EJB component model, does not take into account ease of unit testing. It is very difficult to test applications based on EJB and many other J2EE APIs outside an application server.

3. Certain J2EE technologies have failed in performance. EJB 2.x, for instance -> The main offender here is entity beans, which have proven little short of disastrous for productivity.

J2EE Frameworks

1. There are many frameworks which claim to resolve the issues mentioned earlier. For instance, Struts.

2. Struts is a web framework which works on the web tier and helps us achieve MVC and is doing pretty well in the market. However Spring takes over struts in that it is not just a web framework but an application framework.

3. Unlike single-tier frameworks such as Struts or Hibernate, Spring aims to help structure whole applications in a consistent, productive manner. It has modules that offer services for use throughout an application.

4. The essence of Spring is in providing enterprise services to Plain Old Java Objects (POJOs). This is valuable in a J2EE environment.

Spring framework

    - open source framework, used to develop both standalone(Java) as well as Web(J2EE) appl. It is a framework so we have to download all jar files and put inside appl

Features

1. Reduce glue code/plumbing work

Spring Framework takes lot of load off the programmer by providing dependencies when required and by using AOP(Aspect Oriented Programming) - define common functionality in single place(bean) and access it anywhere

2. Externalize dependencies - Inversion of Control(IOC) - Dependency Injection

Dependencies are described in a separate file (xml) rather than mixing it with the business logic code itself. This gives a better control over the application.

    Loosely coupled appl - inject the values at runtime defined in separate xml file

3. Manage dependencies at a single place

Dependencies can be managed better due to this.

4. Improve testability

Actual code can easily be replaced by a stub for testing purposes.

5. Foster good application design

Since the actual implementation sits behind the interfaces, it fosters good application design.

6. Flexibility

Spring offers integration points with several other frameworks. So, you do not have to write them yourself.

7 modules

1. Spring Core module

         - support IOC concept

         - Using BeanFactory interface

2. Spring Context module

         - supports IOC concept + support internationalization, lifecycle events, integration with EJB, emailing, remoting, scheduling, integration with FreeMarker and Velocity framework

         - Using ApplicationContext interface

3. Spring AOP(Aspect Oriented Programming)

       - define common functionality in single place(bean) and access it anywhere

eg: security, transaction mgt, logging

4. Spring DAO(Data Access Object) module

        - It provides some inbuilt interface to connect with database using Spring itself(JdbcTemplate class)

5. Spring ORM(Object Relational Mapping) module

        - Used to integrate Spring with Hibernate framework

6. Spring Webflow module

        - Used to integrate Spring with Struts framework

7. Spring MVC module

        - used to develop MVC architecture using Spring itself

IOC - Inversion of Control

Dependency Injection

    - objects are given their dependencies at runtime in separate xml file

3 types of DI

1. Setter injection - Spring

        - dependencies are configured through beans setter methods for their properties using <property> tag

2. Constructor injection - Spring

        - dependencies are configured through beans contructors for their properties using <constructor-arg> tag

3. Interface injection - Avalon

        - dependencies are configured through interface for their properties

Spring IOC container - 2 types of IOC containers are there to perform DI

1. BeanFactory interface

     - org.springframework.beans.factory.\* package

     - It is lazy loading

     - used to create and destroy the bean

     - Use XmlBeanFactory(Resource res) class to load the beans defined in xml file

     - Object getBean(String name) - used to instantiate the bean and it performs DI

2. ApplicationContext interface

      - org.springframework.context.\* package

      - It is eager loading

      - used to create and destroy the bean

      - 3 classes

1. ClassPathXmlApplicationContext(String xmlfilename) - loads the xml file from classpath(inside src folder)

2. FileSystemXmlApplicationContext(String xmlfilename) - loads the xml file  from file system (ie) c:/ or d:/ or inside project

3. WebXmlApplicationContext(String xmlfilename) - loads the xml file from web.xml

public class Main {

              public static void main(String[] args) {

                             //Resource res=new ClassPathResource("hello.xml");

                             //Resource res=new FileSystemResource("hello.xml");

                             //BeanFactory factory=new XmlBeanFactory(res);  //load beans present in xml file

                             //Student st=(Student)factory.getBean("stud");//create an object for bean which has id="stud" and perform DI

                             //System.out.println(st.getName()+" "+st.getAge());

                             //ApplicationContext context=new ClassPathXmlApplicationContext("hello.xml");

                             ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");

                             Student st=(Student)context.getBean("stud");

                             System.out.println(st.getName()+" "+st.getAge());

              }

}

public class Student {

     private Integer id;

     private String name;

     private Integer age;

              public Integer getId() {

                             return id;

              }

              public void setId(Integer id) {

                             this.id = id;

              }

              public String getName() {

                             return name;

              }

              public void setName(String name) {

                             this.name = name;

              }

              public Integer getAge() {

                             return age;

              }

              public void setAge(Integer age) {

                             this.age = age;

              }

}

public class Employee {

   private Integer id;

   private String name;

   private Double salary;

   private int age;

public Employee(Integer id, String name, Double salary, int age) {

              super();

              this.id = id;

              this.name = name;

              this.salary = salary;

              this.age = age;

}

public Employee() {

              super();

              // TODO Auto-generated constructor stub

}

@Override

public String toString() {

              return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + ", age=" + age + "]";

}

}

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns=[http://www.springframework.org/schema/beans](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.springframework.org%2Fschema%2Fbeans&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822117416%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=JtJ3R9z5UIHK5TFCQ7sVSK4KU95XvfIx392v2i7d7ng%3D&reserved=0)

    xmlns:xsi=[http://www.w3.org/2001/XMLSchema-instance](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.w3.org%2F2001%2FXMLSchema-instance&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822117416%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=yJan%2BEHfupcVBXVc1PNXxXkdfSpZyUDyO3LP0O7gQSE%3D&reserved=0)

    xmlns:context=[http://www.springframework.org/schema/context](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.springframework.org%2Fschema%2Fcontext&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822117416%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=1RpnTE7v6zXFrtI8JCNAjJLgqvyPP37siWTNj464Lpw%3D&reserved=0)

    xsi:schemaLocation="http://www.springframework.org/schema/beans

[http://www.springframework.org/schema/beans/spring-beans-3.0.xsd](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.springframework.org%2Fschema%2Fbeans%2Fspring-beans-3.0.xsd&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822117416%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=3Tzm2vtWto8ggY0pNQHzJ%2FgZQqEhn3nzjs4VxGN4tKA%3D&reserved=0)

[http://www.springframework.org/schema/context](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.springframework.org%2Fschema%2Fcontext&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822117416%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=1RpnTE7v6zXFrtI8JCNAjJLgqvyPP37siWTNj464Lpw%3D&reserved=0)

[http://www.springframework.org/schema/context/spring-context-3.0.xsd](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.springframework.org%2Fschema%2Fcontext%2Fspring-context-3.0.xsd&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822117416%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=vlOZNMPlk%2BEaOVY3Hzmohyc13cly88U1zu6A%2B%2FehopU%3D&reserved=0)">

    <bean id="stud"  class="com.pack.Student">

        <property name="id" value="1000"/>

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

        <property name="age" value="23"/>

    </bean>

   <!--  <bean id="emp" class="com.pack.Employee">

        <constructor-arg  name="id" type="java.lang.Integer" value="2000"/>

        <constructor-arg  name="name" type="java.lang.String" value="Sam"/>

        <constructor-arg  name="salary" type="java.lang.Double" value="20000.0"/>

        <constructor-arg  name="age" type="int" value="20"/>

    </bean> -->

     <bean id="emp" class="com.pack.Employee">

        <constructor-arg  index="0" type="java.lang.Integer" value="2000"/>

        <constructor-arg  index="1" type="java.lang.String" value="Sam"/>

        <constructor-arg  index="2" type="java.lang.Double" value="20000.0"/>

        <constructor-arg  index="3" type="int" value="20"/>

    </bean>

</beans>

public class Main {

              public static void main(String[] args) {

                             //Resource res=new ClassPathResource("hello.xml");

                             //Resource res=new FileSystemResource("hello.xml");

                             //BeanFactory factory=new XmlBeanFactory(res);  //load beans present in xml file

                             //Student st=(Student)factory.getBean("stud");//create an object for bean which has id="stud" and perform DI

                             //System.out.println(st.getName()+" "+st.getAge());

                            //ApplicationContext context=new ClassPathXmlApplicationContext("hello.xml");

                             //ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");

                             //Student st=(Student)context.getBean("stud");

                             //System.out.println(st.getName()+" "+st.getAge());

                             ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");

                             Employee e=(Employee)context.getBean("emp");

                             System.out.println(e);

              }

}

3 programs

1. Bean prg - simple pojo class contains getter and setter methods

2. xml file - used to perform dependency injection

3. main class

Apache Maven

     - It is project development tool, instead of downloading the jar files manually, maven will download all the jar files on behalf of the user from internet

     - pom.xml(project object model) - we have to provide dependency

     - maven will contains 2 repo

1. Remote repository - for first time maven will download all jar files remotely present in internet and put in local repo

2. Local repository - maven will internally create a folder .m2(local repo)

Installation

1. Download maven

[https://maven.apache.org/download.cgi](https://apc01.safelinks.protection.outlook.com/?url=https%3A%2F%2Fmaven.apache.org%2Fdownload.cgi&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822117416%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=BnTfT0QSbwjifRQCIyPSmn8IrYAJqIHq7G2mf4pxaKA%3D&reserved=0)

Binary zip archive            apache-maven-3.9.5-bin.zip

2. Extract apache maven

3. configure maven in env variable

In Search - Type env - Edit env variables for ur account

1. Under user variable - click New

Variable name: MAVEN\_HOME

Variable value: C:\Softwares\apache-maven-3.6.3

2. Select Path - Click Edit - Click New  - Paste C:\Softwares\apache-maven-3.6.3\bin

4. Open cmd prompt

>mvn -version

5. >mvn clean

      - It will say build failure, used to create local repo .m2

goto c:/users/urname/.m2

Spring project

1. Click File - New - Create Maven Project - Next

Check Create a simple project - Click Next

Group id: com.pack (package name)

ArtifactId: Spring1 (projectname)

Versioning: jar

Click Finish

2. Configure dependency in pom.xml

<dependencies>

                             <dependency>

                                           <groupId>org.springframework</groupId>

                                           <artifactId>spring-core</artifactId>

                                           <version>4.2.4.RELEASE</version>

                             </dependency>

                             <dependency>

                                           <groupId>org.springframework</groupId>

                                           <artifactId>spring-context</artifactId>

                                           <version>4.2.4.RELEASE</version>

                             </dependency>

              </dependencies>

3. Right click project - click properties - Copy path from location - click apply and close

4. Open cmd prompt

C:\Users\senthil.kumart>cd C:\Spring\Spring1

C:\Spring\Spring1>mvn clean install

     - It will download all jar files and keep inside local repo

5. Right click project - Maven - Update project - check Force update on snapshot - click ok

6. Create bean program inside src/main/java

public class Student {

     private Integer id;

     private String name;

     private Integer age;

              public Integer getId() {

                             return id;

              }

              public void setId(Integer id) {

                             this.id = id;

              }

              public String getName() {

                             return name;

              }

              public void setName(String name) {

                             this.name = name;

              }

              public Integer getAge() {

                             return age;

              }

              public void setAge(Integer age) {

                             this.age = age;

              }

}

public class Employee {

   private Integer id;

   private String name;

   private Double salary;

   private int age;

public Employee(Integer id, String name, Double salary, int age) {

              super();

              this.id = id;

              this.name = name;

              this.salary = salary;

              this.age = age;

}

public Employee() {

              super();

              // TODO Auto-generated constructor stub

}

@Override

public String toString() {

              return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + ", age=" + age + "]";

}

}

7. Create xml file in any name inside src or inside project

       - used to configure bean prg inside xml file to perform dependency injection

       - contain  root elt called <beans> and each bean class is configured by <bean> tag

       1. id - any name which acts as the object of bean class

       2. class - fully qualified path of bean prg

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns=[http://www.springframework.org/schema/beans](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.springframework.org%2Fschema%2Fbeans&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822117416%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=JtJ3R9z5UIHK5TFCQ7sVSK4KU95XvfIx392v2i7d7ng%3D&reserved=0)

    xmlns:xsi=[http://www.w3.org/2001/XMLSchema-instance](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.w3.org%2F2001%2FXMLSchema-instance&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822117416%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=yJan%2BEHfupcVBXVc1PNXxXkdfSpZyUDyO3LP0O7gQSE%3D&reserved=0)

    xmlns:context=[http://www.springframework.org/schema/context](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.springframework.org%2Fschema%2Fcontext&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822117416%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=1RpnTE7v6zXFrtI8JCNAjJLgqvyPP37siWTNj464Lpw%3D&reserved=0)

    xsi:schemaLocation="http://www.springframework.org/schema/beans

[http://www.springframework.org/schema/beans/spring-beans-3.0.xsd](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.springframework.org%2Fschema%2Fbeans%2Fspring-beans-3.0.xsd&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822273664%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=p4j2Dm4fkLKGRx6f5oIakkWVHnVhsPtxtZ8VWxta0CI%3D&reserved=0)

[http://www.springframework.org/schema/context](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.springframework.org%2Fschema%2Fcontext&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822273664%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=eGAGylUDVcGjIsLe6bh0YtveIf9VliCl7gI%2BSyodMi0%3D&reserved=0)

[http://www.springframework.org/schema/context/spring-context-3.0.xsd](https://apc01.safelinks.protection.outlook.com/?url=http%3A%2F%2Fwww.springframework.org%2Fschema%2Fcontext%2Fspring-context-3.0.xsd&data=05%7C01%7Ckajavevaibhav.mahdev%40hcl.com%7C0e665818b02749e2226908dbcbd2d250%7C189de737c93a4f5a8b686f4ca9941912%7C0%7C0%7C638327878822273664%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C3000%7C%7C%7C&sdata=wiSqdIr6i7sEqdHA2%2Bchkouq39YtnEft64jWqeMJAO8%3D&reserved=0)">

    <bean id="stud"  class="com.pack.Student">

        <property name="id" value="1000"/>

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

        <property name="age" value="23"/>

    </bean>

   <!--  <bean id="emp" class="com.pack.Employee">

        <constructor-arg  name="id" type="java.lang.Integer" value="2000"/>

        <constructor-arg  name="name" type="java.lang.String" value="Sam"/>

        <constructor-arg  name="salary" type="java.lang.Double" value="20000.0"/>

        <constructor-arg  name="age" type="int" value="20"/>

    </bean> -->

     <bean id="emp" class="com.pack.Employee">

        <constructor-arg  index="0" type="java.lang.Integer" value="2000"/>

        <constructor-arg  index="1" type="java.lang.String" value="Sam"/>

        <constructor-arg  index="2" type="java.lang.Double" value="20000.0"/>

        <constructor-arg  index="3" type="int" value="20"/>

    </bean>

</beans>

8. Main class

Resource interface is used to locate the xml file - 2 classes

   1. ClassPathResource class - if xml file present in src folder

   2. FileSystemResource class - if xml file present in file system or project

public class Main {

              public static void main(String[] args) {

                             //Resource res=new ClassPathResource("hello.xml");

                             //Resource res=new FileSystemResource("hello.xml");

                             //BeanFactory factory=new XmlBeanFactory(res);  //load beans present in xml file

                             //Student st=(Student)factory.getBean("stud");//create an object for bean which has id="stud" and perform DI

                             //System.out.println(st.getName()+" "+st.getAge());

                             //ApplicationContext context=new ClassPathXmlApplicationContext("hello.xml");

                             //ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");

                             //Student st=(Student)context.getBean("stud");

                             //System.out.println(st.getName()+" "+st.getAge());

                             ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");

                             Employee e=(Employee)context.getBean("emp");

                             System.out.println(e);

              }

}

Day 17

Scope

    - life time of bean object - 5 types

1. singleton - default - how many bean object we are creating it will have same object

2. prototype - how many bean object we are creating it will have different reference

3. session

4. request

5. global-session

  <bean id="emp" class="com.pack.Employee" scope="prototype">

        <constructor-arg  index="0" type="java.lang.Integer" value="2000"/>

        <constructor-arg  index="1" type="java.lang.String" value="Sam"/>

        <constructor-arg  index="2" type="java.lang.Double" value="20000.0"/>

        <constructor-arg  index="3" type="int" value="20"/>

    </bean>

ApplicationContext context=new FileSystemXmlApplicationContext("hello.xml");

                             Employee e=(Employee)context.getBean("emp");

                             System.out.println(e);

                             Employee e1=(Employee)context.getBean("emp");

                             System.out.println(e1);

                             Employee e2=(Employee)context.getBean("emp");

                             System.out.println(e2);

Injection Collections

     - apart from primitive datatype we can also inject collections like List,Set, Map,Properties

public class SampleCollection {

     private List addrList;

     private Set addrSet;

     private Map addrMap;

     private Properties addrProp;

public List getAddrList() {

                             return addrList;

              }

              public void setAddrList(List addrList) {

                             this.addrList = addrList;

              }

              public Set getAddrSet() {

                             return addrSet;

              }

              public void setAddrSet(Set addrSet) {

                             this.addrSet = addrSet;

              }

              public Map getAddrMap() {

                             return addrMap;

              }

              public void setAddrMap(Map addrMap) {

                             this.addrMap = addrMap;

              }

              public Properties getAddrProp() {

                             return addrProp;

              }

              public void setAddrProp(Properties addrProp) {

                             this.addrProp = addrProp;

              }

}

<bean id="collection" class="com.pack.SampleCollection">

          <property name="addrList">

              <list>

                  <value>Ram</value>

                  <value>Sam</value>

                  <value>Raj</value>

                  <value>Ram</value>

              </list>

          </property>

          <property name="addrSet">

              <set>

                 <value>Ram</value>

                  <value>Sam</value>

                  <value>Raj</value>

                  <value>Ram</value>

              </set>

          </property>

          <property name="addrMap">

               <map>

                   <entry key="1" value="Ram"/>

                   <entry key="2" value="Sam"/>

                   <entry key="3" value="Tam"/>

                   <entry key="4" value="Raj"/>

               </map>

          </property>

          <property name="addrProp">

               <props>

                   <prop key="one">Ram</prop>

                   <prop key="two">Sam</prop>

                   <prop key="three">Raj</prop>

               </props>

          </property>

    </bean>

public class Main1 {

              public static void main(String[] args) {

                             ApplicationContext ctx=new ClassPathXmlApplicationContext("bean.xml");

                             SampleCollection s=(SampleCollection)ctx.getBean("collection");

                             System.out.println(s.getAddrList());

                             System.out.println(s.getAddrSet());

                             System.out.println(s.getAddrMap());

                             System.out.println(s.getAddrProp());

              }

}

ref attribute

     - used to refer one bean into another bean

     - ref="id of the bean"

public class Address {

     private String city;

     private String state;

     private String pincode;

              public String getCity() {

                             return city;

              }

              public void setCity(String city) {

                             this.city = city;

              }

              public String getState() {

                             return state;

              }

              public void setState(String state) {

                             this.state = state;

              }

              public String getPincode() {

                             return pincode;

              }

              public void setPincode(String pincode) {

                             this.pincode = pincode;

              }

}

public class Student {

     private Integer id;

     private String name;

     private Address address;  //referred another bean

              public Integer getId() {

                             return id;

              }

              public void setId(Integer id) {

                             this.id = id;

              }

              public String getName() {

                             return name;

              }

              public void setName(String name) {

                             this.name = name;

              }

              public Address getAddress() {

                             return address;

              }

              public void setAddress(Address address) {

                             this.address = address;

              }

}

<bean id="addr" class="com.pack.Address">

        <property name="city" value="Chennai"/>

        <property name="state" value="Tamilnadu"/>

        <property name="pincode" value="600011"/>

    </bean>

    <bean id="student" class="com.pack.Student">

         <property name="id" value="1000"/>

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

         <property name="address" ref="addr"/>

    </bean>

public class Main1 {

              public static void main(String[] args) {

                             ApplicationContext ctx=new ClassPathXmlApplicationContext("bean.xml");

                             Student st=(Student)ctx.getBean("student");

                             System.out.println(st.getId()+" "+st.getName()+" "+st.getAddress().getCity());

              }

}

wiring - refering one bean into another bean

Autowiring

   - used only in case of referring one bean into another bean

   - Instead of using ref attribute we can go for autowiring to inject one bean into another bean automatically

Types

1. autowire=none - default - where we have to do injection using ref attribute

public class Match {

   private String location;

   private String distance;

public String getLocation() {

              return location;

}

public void setLocation(String location) {

              this.location = location;

}

public String getDistance() {

              return distance;

}

public void setDistance(String distance) {

              this.distance = distance;

}

}

public class Player {

   private Integer playerId;

   private String name;

   private Match match;

public Integer getPlayerId() {

              return playerId;

}

public void setPlayerId(Integer playerId) {

              this.playerId = playerId;

}

public String getName() {

              return name;

}

public void setName(String name) {

              this.name = name;

}

public Match getMatch() {

              return match;

}

public void setMatch(Match match) {

              this.match = match;

}

}

<bean id="match" class="com.pack.Match">

        <property name="location" value="Chennai"/>

        <property name="distance" value="1000"/>

    </bean>

    <bean id="player" class="com.pack.Player">

        <property name="playerId" value="100"/>

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

        <property name="match" ref="match"/>

    </bean>

public class Main1 {

              public static void main(String[] args) {

                             ApplicationContext ctx=new ClassPathXmlApplicationContext("bean.xml");

                             Player p=(Player)ctx.getBean("player");

                             System.out.println(p.getName()+" plays at "+p.getMatch().getLocation());

              }

}

2. autowire=byName - name of the property = id of the bean in xml file, then it will do autowire by name

<bean id="match" class="com.pack.Match">

        <property name="location" value="Chennai"/>

        <property name="distance" value="1000"/>

    </bean>

    <bean id="player" class="com.pack.Player" autowire="byName">

        <property name="playerId" value="100"/>

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

    </bean>

3. autowire=byType - if datatype of the property should be somewhere configured in xml file, then it will do autowire by type

<bean id="match" class="com.pack.Match">

        <property name="location" value="Chennai"/>

        <property name="distance" value="1000"/>

    </bean>

    <bean id="player" class="com.pack.Player" autowire="byType">

        <property name="playerId" value="100"/>

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

    </bean>

4. autowire=constructor - similar to byType or byname but it will do injection through constructor

public class Match {

   private String location;

   private String distance;

public Match(String location, String distance) {

              super();

              this.location = location;

              this.distance = distance;

}

public Match() {

              super();

              // TODO Auto-generated constructor stub

}

@Override

public String toString() {

              return "Match [location=" + location + ", distance=" + distance + "]";

}

}

public class Player {

   private Integer playerId;

   private String name;

   private Match match1;

public Player(Integer playerId, String name, Match match1) {

              super();

              this.playerId = playerId;

              this.name = name;

              this.match1 = match1;

}

public Player() {

              super();

              // TODO Auto-generated constructor stub

}

@Override

public String toString() {

              return "Player [playerId=" + playerId + ", name=" + name + ", match1=" + match1 + "]";

}

}

<bean id="match" class="com.pack.Match">

       <constructor-arg index="0" type="java.lang.String" value="Mumbai"/>

       <constructor-arg index="1" type="java.lang.String" value="1000"/>

    </bean>

    <bean id="player" class="com.pack.Player" autowire="constructor">

       <constructor-arg index="0" type="java.lang.Integer" value="1001"/>

       <constructor-arg index="1" type="java.lang.String" value="Kohli"/>

    </bean>

public class Main1 {

              public static void main(String[] args) {

                             ApplicationContext ctx=new ClassPathXmlApplicationContext("bean.xml");

                             Player p=(Player)ctx.getBean("player");

                             System.out.println(p);

              }

}

autowire-candidate=false

    - If same bean is configured multiple times in xml file, which bean class has to be injected, it is defined using autowire-candidate property

    - So which bean we dont want to use we have to define autowire-candidate=false

<bean id="match" class="com.pack.Match" autowire-candidate="false">

       <constructor-arg index="0" type="java.lang.String" value="Mumbai"/>

       <constructor-arg index="1" type="java.lang.String" value="1000"/>

    </bean>

    <bean id="match1" class="com.pack.Match">

       <constructor-arg index="0" type="java.lang.String" value="Delhi"/>

       <constructor-arg index="1" type="java.lang.String" value="2000"/>

    </bean>

    <bean id="player" class="com.pack.Player" autowire="constructor">

       <constructor-arg index="0" type="java.lang.Integer" value="1001"/>

       <constructor-arg index="1" type="java.lang.String" value="Kohli"/>

    </bean>

primary="true"

    - If same bean is configured multiple times in xml file, which bean class has to be injected, it is defined using primary property

    - So which bean we want to use we have to define primary="true"

<bean id="match" class="com.pack.Match" primary="true">

       <constructor-arg index="0" type="java.lang.String" value="Mumbai"/>

       <constructor-arg index="1" type="java.lang.String" value="1000"/>

    </bean>

    <bean id="match1" class="com.pack.Match">

       <constructor-arg index="0" type="java.lang.String" value="Delhi"/>

       <constructor-arg index="1" type="java.lang.String" value="2000"/>

    </bean>

    <bean id="player" class="com.pack.Player" autowire="constructor">

       <constructor-arg index="0" type="java.lang.Integer" value="1001"/>

       <constructor-arg index="1" type="java.lang.String" value="Kohli"/>

    </bean>

Spring using Annotation

     - used instead of configuring bean in xml file

@Configuration - class level annotation, something like xml file, where we configure all bean classes (ie) collection of bean classes

@Bean - used to configure single bean class

public class Example1 {

    private String message1;

              public String getMessage1() {

                             return message1;

              }

              public void setMessage1(String message1) {

                             this.message1 = message1;

              }

}

public class Example2 {

   private String message2;

public String getMessage2() {

              return message2;

}

public void setMessage2(String message2) {

              this.message2 = message2;

}

}

@Configuration

public class ExampleConfiguration {

              @Bean

              @Scope("prototype")

              public Example1 example1() {

                             return new Example1();

              }

              @Bean

              public Example2 example2() {

                             return new Example2();

              }

}

public class Main2 {

              public static void main(String[] args) {

                             ApplicationContext ctx=

                                                          new AnnotationConfigApplicationContext(ExampleConfiguration.class);

                             /\*Example1 e1=(Example1)ctx.getBean(Example1.class);

                             e1.setMessage1("Hello world");

                             System.out.println(e1.getMessage1());

                             Example2 e2=(Example2)ctx.getBean(Example2.class);

                             e2.setMessage2("Welcome world");

                             System.out.println(e2.getMessage2());\*/

                             Example1 e1=(Example1)ctx.getBean(Example1.class);

                             System.out.println(e1);

                             Example1 e2=(Example1)ctx.getBean(Example1.class);

                             System.out.println(e2);

              }

}

Day 18

Spring Annotations

   1. @Configuration

   2. @Bean

public class Example3 {

   private String message;

public Example3(String message) {

              super();

              this.message = message;

}

public Example3() {

              super();

              // TODO Auto-generated constructor stub

}

@Override

public String toString() {

              return "Example3 [message=" + message + "]";

}

}

@Configuration

public class ExampleConfiguration {

              @Bean

              @Scope("prototype")

              public Example1 example1() {

                             return new Example1();

              }

              @Bean

              public Example2 example2() {

                             return new Example2();

              }

              @Bean

              public Example3 example3() {

                             return new Example3("Spring");

              }

@Bean

              @Primary

              public Example3 example31() {

                             return new Example3("Springboot");

              }

              @Bean

              public Example3 example32() {

                             return new Example3("Struts");

              }

}

@Autowired

    - perform autowiring based on by type

    - used above only setter method or constructor or properties

    - If we use autowire by annotation, in order to make ur spring to understand the annotation we have to configure <context:annotation-config/> in xml file

public class Address {

     private String city;

     private String state;

     private String pincode;

              public String getCity() {

                             return city;

              }

              public void setCity(String city) {

                             this.city = city;

              }

              public String getState() {

                             return state;

              }

              public void setState(String state) {

                             this.state = state;

              }

              public String getPincode() {

                             return pincode;

              }

              public void setPincode(String pincode) {

                             this.pincode = pincode;

              }

}

public class Student {

     private Integer id;

     private String name;

     @Autowired

     private Address address;  //referred another bean

              public Integer getId() {

                             return id;

              }

              public void setId(Integer id) {

                             this.id = id;

              }

              public String getName() {

                             return name;

              }

              public void setName(String name) {

                             this.name = name;

              }

              public Address getAddress() {

                             return address;

              }

              //@Autowired

              public void setAddress(Address address) {

                             this.address = address;

              }

}

<context:annotation-config/>

    <bean id="addr" class="com.pack.Address">

        <property name="city" value="Chennai"/>

        <property name="state" value="Tamilnadu"/>

        <property name="pincode" value="600011"/>

    </bean>

    <bean id="student" class="com.pack.Student">

         <property name="id" value="1000"/>

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

        <!--  <property name="address" ref="addr"/> -->

    </bean>

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

                             Student st=(Student)context.getBean("student");

                             System.out.println(st.getName()+" "+st.getAddress().getCity());

@Qualifier

    - If same bean is configured multiple times in xml file which bean has to be injected while using @Autowired is decided using @Qualifier

<context:annotation-config/>

    <bean id="addr" class="com.pack.Address">

        <property name="city" value="Chennai"/>

        <property name="state" value="Tamilnadu"/>

        <property name="pincode" value="600011"/>

    </bean>

    <bean id="addr1" class="com.pack.Address">

        <property name="city" value="Pune"/>

        <property name="state" value="Maharastra"/>

        <property name="pincode" value="200011"/>

    </bean>

    <bean id="student" class="com.pack.Student">

         <property name="id" value="1000"/>

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

        <!--  <property name="address" ref="addr"/> -->

    </bean>

public class Student {

     private Integer id;

     private String name;

     @Autowired

     @Qualifier("addr1")

     private Address address;  //referred another bean

              public Integer getId() {

                             return id;

              }

              public void setId(Integer id) {

                             this.id = id;

              }

              public String getName() {

                             return name;

              }

              public void setName(String name) {

                             this.name = name;

              }

              public Address getAddress() {

                             return address;

              }

              //@Autowired

              public void setAddress(Address address) {

                             this.address = address;

              }

}

@Required

    - It indicates that the property should be complusorly injected in xml file

    - always used only above setter method

public class Student {

     private Integer id;

     private String name;

     @Autowired

     @Qualifier("addr1")

     private Address address;  //referred another bean

              public Integer getId() {

                             return id;

              }

              public void setId(Integer id) {

                             this.id = id;

              }

              public String getName() {

                             return name;

              }

              @Required

              public void setName(String name) {

                             this.name = name;

              }

              public Address getAddress() {

                             return address;

              }

              //@Autowired

              public void setAddress(Address address) {

                             this.address = address;

              }

}

BeanFactory is Lazy loading - it creates the bean only when we call getBean()

ApplicationContext is eager loading - preloads all the beans at time of startup itself, so we have to convert ApplicationContext to lazy loading

   1. using lazy-init="true" in xml file

   2. Using @Lazy annotation

public class Sample1 {

              public Sample1() {

                             System.out.println("Sample1 bean is created");

              }

}

public class Sample2 {

    private Sample1 sample1;

              public Sample1 getSample1() {

                             return sample1;

              }

              public void setSample1(Sample1 sample1) {

                             this.sample1 = sample1;

              }

              public Sample2() {

                             System.out.println("Sample2 bean is created");

              }

}

<bean id="sample1" class="com.pack.Sample1" lazy-init="true"/>

    <bean id="sample2" class="com.pack.Sample2" lazy-init="true">

        <property name="sample1" ref="sample1"/>

    </bean>

public class Main3 {

              public static void main(String[] args) {

                            //Resource res=new ClassPathResource("bean.xml");

                             //BeanFactory factory=new XmlBeanFactory(res);

                             //Sample2 s=(Sample2)factory.getBean("sample2");

                             ApplicationContext ctx=new ClassPathXmlApplicationContext("bean.xml");

                             Sample2 s=(Sample2)ctx.getBean("sample2");

              }

}

Lifecycle of spring bean

1. Using InitializingBean and DisposableBean interface

InitializingBean interface- public void afterPropertiesSet() - invoked whenever a bean is instantiated when getBean() is called

DisposableBean interface - public void destroy() - invoked whenever ur bean is destroyed

ConfigurableApplicationContext interface

- abstract class AbstractApplicationContext implements ConfigurableApplicationContext interface

void registerShutDownHook() - destroy the bean

   or

void close() - destroy the bean

public class Person implements InitializingBean,DisposableBean {

   private String name;

public String getName() {

              return name;

}

public void setName(String name) {

              this.name = name;

}

public void destroy() throws Exception {

              System.out.println("Destroying person bean");

}

public void afterPropertiesSet() throws Exception {

    System.out.println("Initializing person bean "+name);

}

}

<bean id="person" class="com.pack.Person">

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

    </bean>

public class Main4 {

              public static void main(String[] args) {

                             ConfigurableApplicationContext ctx=new ClassPathXmlApplicationContext("lifecycle.xml");

                             Person p=(Person)ctx.getBean("person");

                             System.out.println(p.getName());

                             //ctx.registerShutdownHook();

                             ctx.close();

              }

}

2. custom init and destroy method

       - using init-method and destroy-method in xml file for particular bean class

public class Example {

   private String name;

public String getName() {

              return name;

}

public void setName(String name) {

              this.name = name;

}

   public void customInit() {

                 System.out.println("Initializing example bean");

   }

   public void customDestroy() {

                 System.out.println("Destroying example bean");

   }

}

<bean id="example" class="com.pack.Example" init-method="customInit" destroy-method="customDestroy">

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

    </bean>

ConfigurableApplicationContext ctx=new ClassPathXmlApplicationContext("lifecycle.xml");

                             Example e=(Example)ctx.getBean("example");

                             System.out.println(e.getName());

                             ctx.registerShutdownHook();

3. default-init-method="customInit" and default-destroy-method="customDestroy" in beans tag, so that it is applicable for all beans in xml file

4. Using @PostConstruct and @PreDestroy annotation

<dependency>

                                 <groupId>javax.annotation</groupId>

                                 <artifactId>javax.annotation-api</artifactId>

                                 <version>1.3.2</version>

                             </dependency>

Spring sterotype annotations

1. @Component - it allows spring to identify that class as custom bean so that it can be injected in other prg using @Autowired

2. @Service - it is a bean class that contains business logic, so that it can be injected in other prg using @Autowired

3. @Repository - it is a bean class that interacts with database

4. @Controller - used to define a controller class that accept the request and gives the response