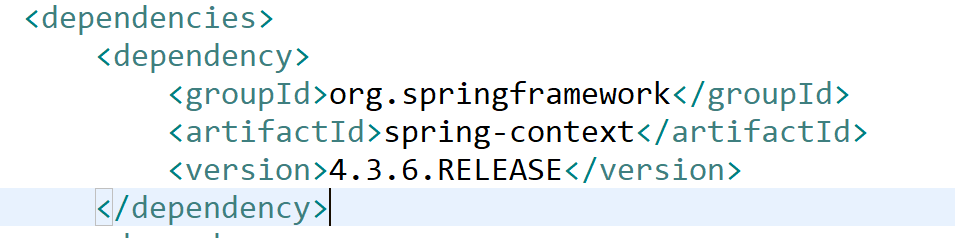
Autowiring:

Is a feature where, a bean can identify the eligible dependent bean and inject it automatically.

Example:

Book has an author

1. Create a maven project (archetype: quickstart)
2. pom.xml



1. create the beans: Author and Book

package com.ust.spring;

public class Author {

private Integer id;

private String firstName;

private String lastName;

public Author() {}

public Author(Integer id, String firstName, String lastName) {

super();

this.id = id;

this.firstName = firstName;

this.lastName = lastName;

}

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

@Override

public String toString() {

return "Author [id=" + id + ", firstName=" + firstName + ", lastName=" + lastName + "]";

}

}

//-----------------------

package com.ust.spring.\_may\_autowire;

import java.util.ArrayList;

import java.util.List;

public class Book {

private String title;

private List<Author> authors=new ArrayList<>();

public Book() {}

public Book(String title, List<Author> authors) {

super();

this.title = title;

this.authors = authors;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public List<Author> getAuthors() {

return authors;

}

public void setAuthors(List<Author> authors) {

this.authors = authors;

}

@Override

public String toString() {

return "Book [title=" + title + ", authors=" + authors + "]";

}

}

1. Configure these classes as beans in spring.xml

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:p="http://www.springframework.org/schema/p"

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

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="abcd" class="com.ust.spring.\_may\_autowire.Author">

<property name="id" value="1" />

<property name="firstName" value="Shiv" />

<property name="lastName" value="Khera" />

</bean>

<bean id="book" class="com.ust.spring.\_may\_autowire.Book" autowire="byType">

<property name="isbn" value="123" />

<property name="title" value="You can win" />

</bean>

</beans>

1. App.java

Lets get Bean of Book now.

package com.ust.spring.\_may\_autowire;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

/\*\*

\* Hello world!

\*

\*/

public class App

{

public static void main( String[] args )

{

System.out.println( "Hello World!" );

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

Book book=(Book) ctx.getBean("book");

System.out.println(book);

}

}

Now, the autowiring will be done in annotations.

MVC

Model View Controller

Model

Data definition

Data access

View

Input/ output UI front end

Controller

If controller is not there, then view and model should be present locally.

By having a controller, we are able to allow views to access model from anywhere.

It can be a middleware also. It can take care of business logic

Any application has

Presentation services (view)

Business logic (controller)

Data services (model)

We are going to create a spring MVC project now.

1. Create a maven project (archetype: webapp last archetype)

30-may-mvc-1

1. In pom.xml

<dependency>

<groupId>com.fasterxml.jackson.core</groupId>

<artifactId>jackson-databind</artifactId>

<version>2.9.6</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>4.3.9.RELEASE</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-web</artifactId>

<version>4.3.9.RELEASE</version>

</dependency>

<!-- Servlet -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>servlet-api</artifactId>

<version>2.5</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet.jsp</groupId>

<artifactId>jsp-api</artifactId>

<version>2.1</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

1. A web application will have a deployment descriptor file (web.xml)

It is present in “src/main/webapp/WEB-INF/web.xml

We need to configure DispatcherServlet

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

<web-app xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance" xmlns="https://java.sun.com/xml/ns/javaee" xsi:schemaLocation="https://java.sun.com/xml/ns/javaee https://java.sun.com/xml/ns/javaee/web-app\_3\_0.xsd" id="WebApp\_ID" version="3.0">

<display-name>spring-mvc-example</display-name>

<!-- Add Spring MVC DispatcherServlet as front controller -->

<servlet>

<servlet-name>spring</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>/WEB-INF/spring-servlet.xml</param-value>

</init-param>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>spring</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

</web-app>

1. Create a xml file named as “spring-servlet.xml”

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:p="http://www.springframework.org/schema/p"

xmlns:tx="http://www.springframework.org/schema/tx"

xmlns:mvc="http://www.springframework.org/schema/mvc"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-4.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-4.0.xsd

http://www.springframework.org/schema/tx

http://www.springframework.org/schema/tx/spring-tx-4.0.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop-4.0.xsd

http://www.springframework.org/schema/mvc

http://www.springframework.org/schema/mvc/spring-mvc-4.0.xsd">

<!-- DispatcherServlet Context: defines this servlet's request-processing

infrastructure -->

<!-- Enables the Spring MVC @Controller programming model -->

<mvc:annotation-driven />

<context:component-scan base-package="com.upskillit.spring" />

<!-- Resolves views selected for rendering by @Controllers to .jsp resources

in the /WEB-INF/views directory -->

<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix">

<value>/WEB-INF/views/</value>

</property>

<property name="suffix">

<value>.jsp</value>

</property>

</bean>

</beans>

1. Create the jsp files in the location specified in the prefix

WEB-INF/views (this is the prefix)

Lets create “views” folder under “WEB-INF”

There is an index.jsp file already, move it to this location

1. Create the controller classes in the location specified in “com.ust.spring”

If “src/main/java” folder is not found, create it.

Inside that folder, create “MyController” class under package “com.ust.spring”

**package** com.ust.spring;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.web.bind.annotation.GetMapping;

@Controller

**public** **class** MyController {

@GetMapping("/home")

**public** String home()

{

**return** "index"; //is prefixed and suffixed /WEB-INF/views/index.jsp

}

}

1. Run the application

Right click the project -> Run As -> Run on Server

Choose Apache tomcat (version as you have)

Error:

Port 8080 required by Tomcat… is already in use.

Solution:

Change the port in which our server will run.

Window->show view-> servers

Double click the server (you will see a config file)

Change the port from 8080 to 8081

Run again

In the browser, enter url as

<http://localhost:8081/30-may-mvc-1/home>

(because, in our controller class, “home” is the url for index.jsp

In pom.xml, if dependencies tag is not found, create it above <build> tag

Spring Boot MVC

Advantages of Spring Boot

Download spring boot project from spring initializr

We don’t need to download apache tomcat server (because, spring boot has inbuilt apache tomcat server)

We have a main method to run

Convention over Configuration

Starter dependencies:

Lets see how to create spring boot mvc project:

1. Go to <https://start.spring.io/>

Type the group id, artifact id, choose maven, java, version etc

In the dependencies, add “Spring Web”

Generate button to download as zip

1. Extract and move the folder to your eclipse workspace
2. Import this into eclipse:

File -> Import -> Existing maven projects

Browse to location where pom.xml is present

Import.

1. Go to pom.xml and add apache tomcat dependency

<dependency> <groupId>org.apache.tomcat.embed</groupId>

<artifactId>tomcat-embed-jasper</artifactId>

</dependency>

1. src/main/resource folder and open “application.properties”

spring.mvc.view.prefix=/WEB-INF/views/

spring.mvc.view.suffix=.jsp

server.port=8083

1. right click base package (a package where Application.java is present)

and create a class “MyController” class

@Controller

public class MyController

{

@GetMapping

public String home()

{

return “index”;

}

}

1. right click “src/main/webapp” folder and create folder

WEB-INF/views

1. inside this folder, create index.jsp
2. Run the project: Go to Application.java and ctrl+F11

Steps:

1. <https://start.spring.io/>