Spring MVC XML configuration

Create web application

```
mvn archetype:generate -DgroupId=com.kgisl.springmvcxml -DartifactId=springmvcxml -
DarchetypeArtifactId=maven-archetype-webapp -DinteractiveMode=false
```

Java version

```
<properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
</properties>
```

Spring mvc dependency

```
<dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>6.1.0</version>
</dependency>
```

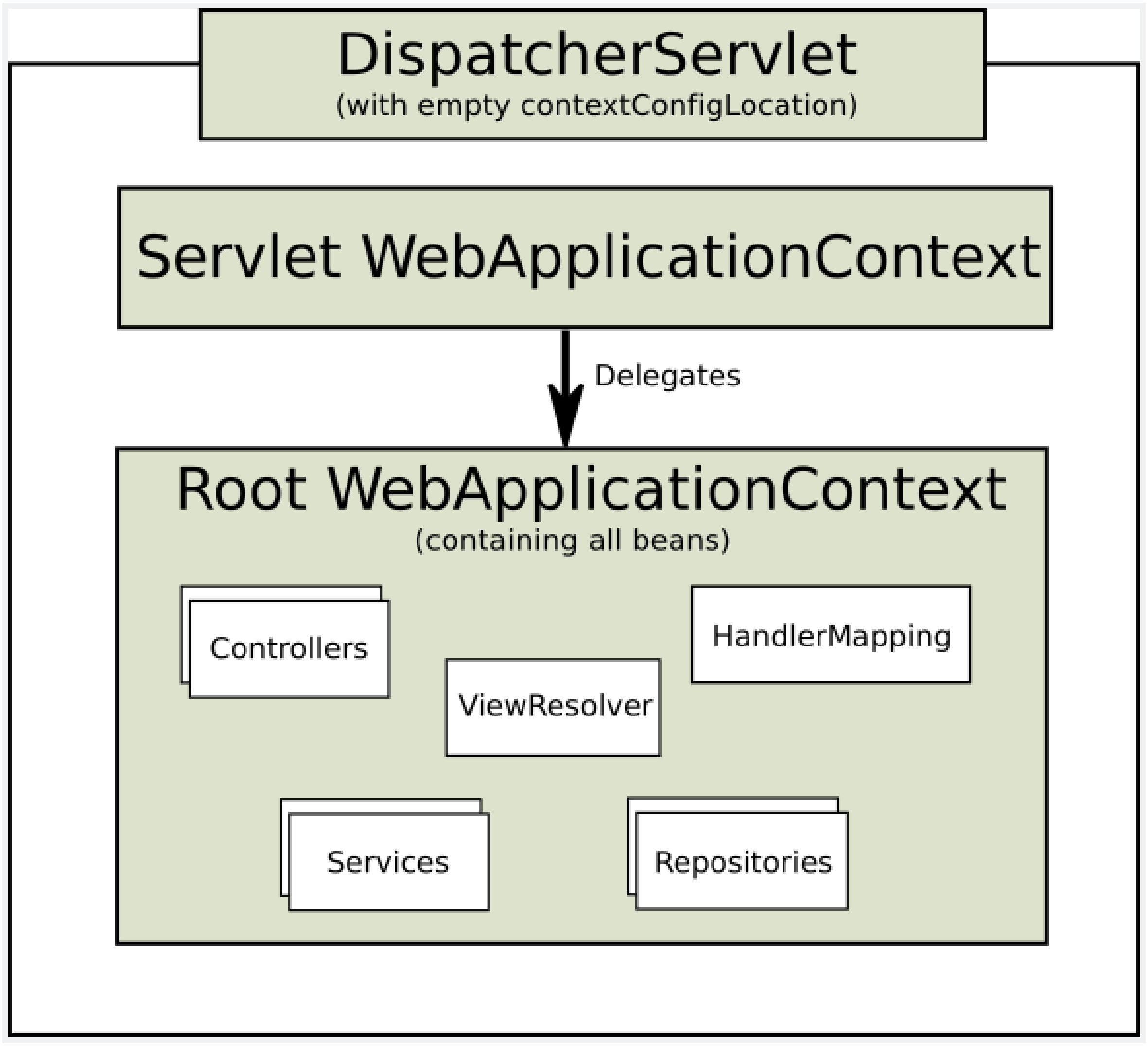
Plugins

```
<plugin>
        <groupId>org.eclipse.jetty</groupId>
        <artifactId>jetty-maven-plugin</artifactId>
        <version>11.0.11</version>
        <configuration>
          <webApp>
            <contextPath>/app</contextPath>
          </webApp>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-shade-plugin</artifactId>
        <version>3.4.1</version>
        <configuration>
```

```
            <!-- put your configurations here -->
          </configuration>
          <executions>
            <execution>
              <phase>package</phase>
              <goals>
                <goal>shade</goal>
              </goals>
            </execution>
          </executions>
        </plugin>
```

DispatcherServlet

The `DispatcherServlet` is an actual `Servlet` (it inherits from the `HttpServlet` base class), and as such is declared in the `web.xml` of your web application. You need to map requests that you want the `DispatcherServlet` to handle, by using a URL mapping in the same `web.xml` file. This is standard Java EE Servlet configuration; the following example shows such a `DispatcherServlet` declaration and mapping:



Configure DispatcherServlet in web.xml
Upon initialization of a DispatcherServlet, Spring MVC looks for a file named *[servlet-name]-servlet.xml* in the WEB-INF directory of your web application and creates the beans defined there, overriding the definitions of any beans defined with the same name in the global scope.

```
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
    version="6.0">
    <servlet>
        <servlet-name>spring</servlet-name>
        <servlet-class>
            org.springframework.web.servlet.DispatcherServlet
        </servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>spring</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>

</web-app>
```

contextConfigLocation in web.xml

This can be configured by setting contextConfigLocation servlet init parameter, as shown below:

```
<web-app xmlns="https://jakarta.ee/xml/ns/jakartaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
https://jakarta.ee/xml/ns/jakartaee/web-app_6_0.xsd"
    version="6.0">
    <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/config.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>
```

config.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:context="http://www.springframework.org/schema/context"
    xmlns:mvc="http://www.springframework.org/schema/mvc"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
```

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

    <!-- Add support for component scanning -->
    <context:component-scan base-package="com.kgisl.springmvcxml" />

    <!-- Add support for conversion, formatting and validation support -->
    <mvc:annotation-driven/>

    <!-- Define Spring MVC view resolver -->
    <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/view/" />
        <property name="suffix" value=".jsp" />
    </bean>
</beans>
```

DispatcherServlet Processing Sequence

1. The **WebApplicationContext** associated with the DispatcherServlet is searched for and bound to the request as an attribute (under the key `DispatcherServlet.WEB_APPLICATION_CONTEXT_ATTRIBUTE`). This allows controllers and other elements in the process to access the application context.
2. The **locale resolver** is optionally bound to the request. It enables elements to resolve the locale for processing (e.g., rendering views, preparing data). If you don't need locale resolution, you can skip this step.
3. The **theme resolver** is optionally bound to the request. It lets elements (such as views) determine which theme to use. If your application doesn't use themes, you can ignore this step.
4. If you specify a **multipart file resolver**, the request is inspected for multipart data. If multipart data (e.g., file uploads) is found, the request is wrapped in a `MultipartHttpServletRequest` for further processing by other elements.
5. An appropriate **handler** (controller or endpoint) is searched for based on the request. If a handler is found, the execution chain associated with that handler (including preprocessors, postprocessors, and controllers) is executed. This prepares the model or rendering for the response.
6. If a **model** is returned from the handler, the view is rendered. If no model is returned (possibly due to a preprocessor or postprocessor intercepting the request, such as for security reasons), no view is rendered because the request may have already been fulfilled.
7. **Handler exception resolvers** declared in the WebApplicationContext handle exceptions thrown during request processing. These resolvers allow you to define custom behaviors

for handling exceptions.

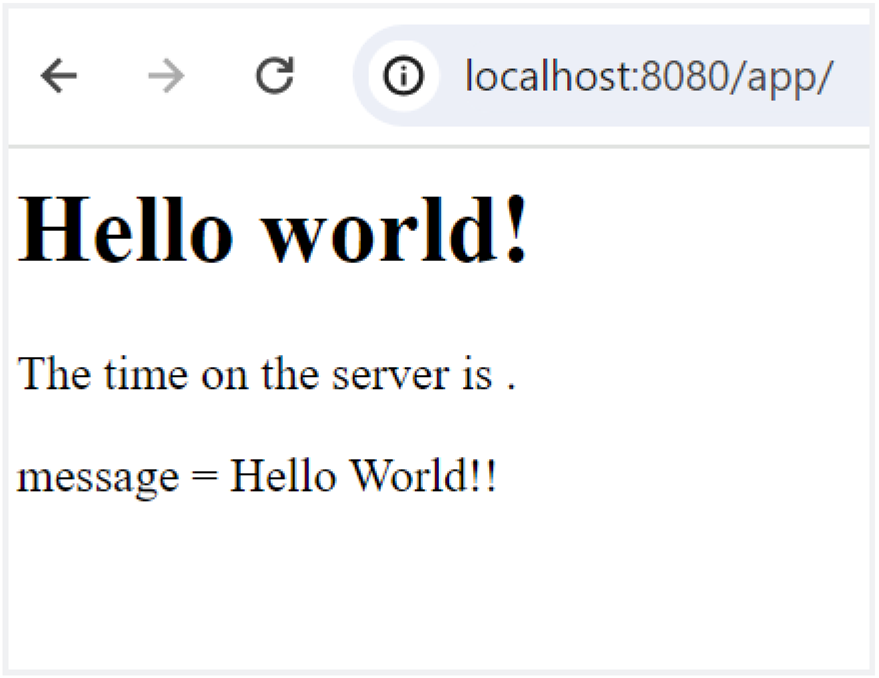
HelloController.java

```
package com.kgisl.springmvcxml;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HelloController {
    @RequestMapping("/")
    public String showPage(Model model) {
          model.addAttribute("message", "Hello World!!");
        return "home";
    }
}
```

home.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<html>
<head>
<title>Home</title>
</head>
<body>
    <h1>Hello world!</h1>
    <P>The time on the server is ${serverTime}.</p>
     message =  ${message}
</body>
</html>
```

Output


localhost:8080/app/

Hello world!

The time on the server is .

message = Hello World!!

Differences between Model, ModelMap, and ModelAndView

Model:
It is an Interface. It defines a holder for model attributes and primarily designed for adding attributes to the model.

Example:

```
@RequestMapping(method = RequestMethod.GET)
    public String printHello(Model model) {
          model.addAttribute("message", "Hello World!!");
          return "hello";
       }
```

ModelMap:
Implementation of Map for use when building model data for use with UI tools.Supports chained calls and generation of model attribute names.

Example:

```
@RequestMapping("/helloworld")
public String hello(ModelMap map) {
    String helloWorldMessage = "Hello world!";
    String welcomeMessage = "Welcome!";
    map.addAttribute("helloMessage", helloWorldMessage);
```

```
        map.addAttribute("welcomeMessage", welcomeMessage);
        return "hello";
    }
```

ModelAndView:
This class merely holds both to make it possible for a controller to return both model and view in a single return value.

Example:

```
@RequestMapping("/welcome")
public ModelAndView helloWorld() {
        String message = "Hello World!";
        return new ModelAndView("welcome", "message", message);
    }
```

Reference
https://docs.spring.io/spring-framework/docs/4.3.5.RELEASE/spring-framework-reference/html/mvc.html