Spring MVC Hello World Example

By Lokesh Gupta | Filed Under: Spring MVC

In this example, we will build a hello world web application using the **Spring MVC** framework. Spring MVC is one of the most important modules of the **Spring** framework. It builds on the powerful Spring **IoC container** and makes extensive use of the container features to simplify its configuration.

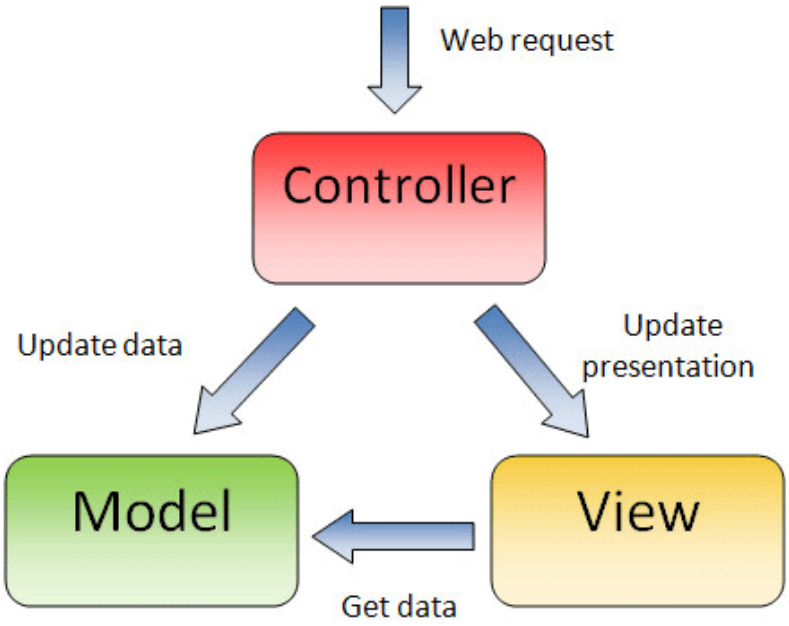
Table Of Contents

Download Sourcecode

What is MVC Framework?

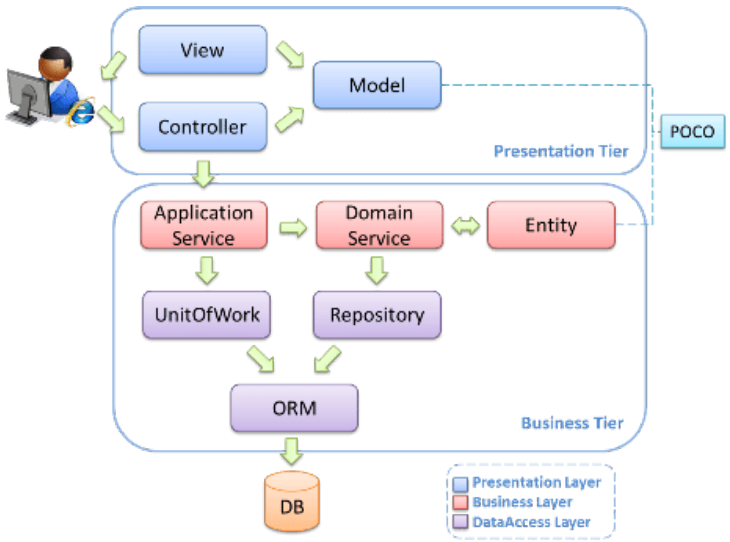
Model-view-controller (MVC) is a well known **design pattern** for designing UI based applications. It mainly decouples business logic from UIs by separating the roles of **model, view, and controller** in an application. Usually, models are responsible for encapsulating application data for views to present. Views should only present this data, without including any business logic. And controllers are responsible for receiving requests from users and invoking back-end services (manager or dao) for business logic processing. After processing, back-end services may return some data for views to present. Controllers collect this data and prepare models for views

to present. The core idea of the MVC pattern is to separate business logic from UIs to allow them to change independently without affecting each other.



In a Spring MVC application, models usually consist of POJO objects that are processed by the service layer and persisted by the persistence layer. Views are usually JSP templates written with Java Standard Tag Library (JSTL). Controller part is played by dispatcher servlet which we will learn about in this tutorial in more detail.

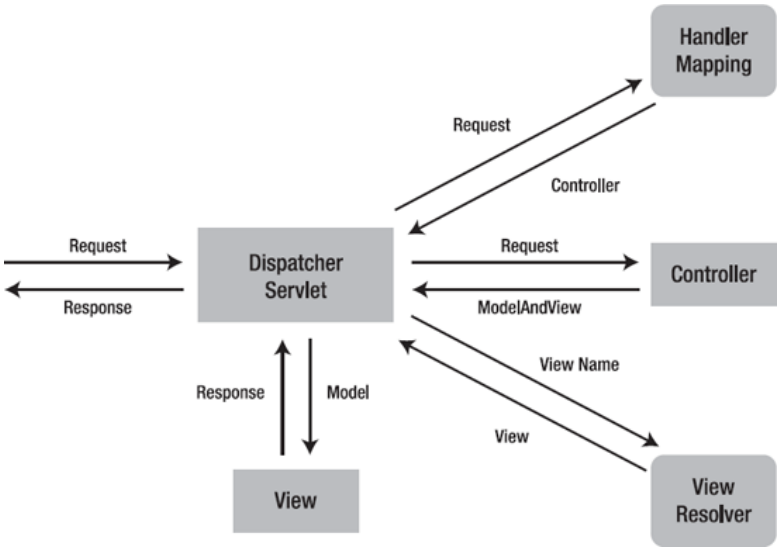
> Some developers consider the service layer and DAO layers classes as part of model component in MVC. I have a different opinion on this. I do not consider service and DAO layers classes the part of MVC framework. Usually a web application is 3-tier architecture i.e. data-service-presentation. MVC is actually part of presentation layer.



Dispatcher Servlet (Spring Controller)

In the simplest Spring MVC application, a controller is the only servlet you need to configure in a Java web deployment descriptor (i.e., the web.xml file). A Spring MVC controller—often referred to as a Dispatcher Servlet implements front controller design pattern and every web request must go through it so that it can manage the entire request life cycle.

When a web request is sent to a Spring MVC application, dispatcher servlet first receives the request. Then it organizes the different components configured in Spring's web application context (e.g. actual request handler controller and view resolvers) or annotations present in the controller itself, all needed to handle the request.



Spring dispatcher servlet

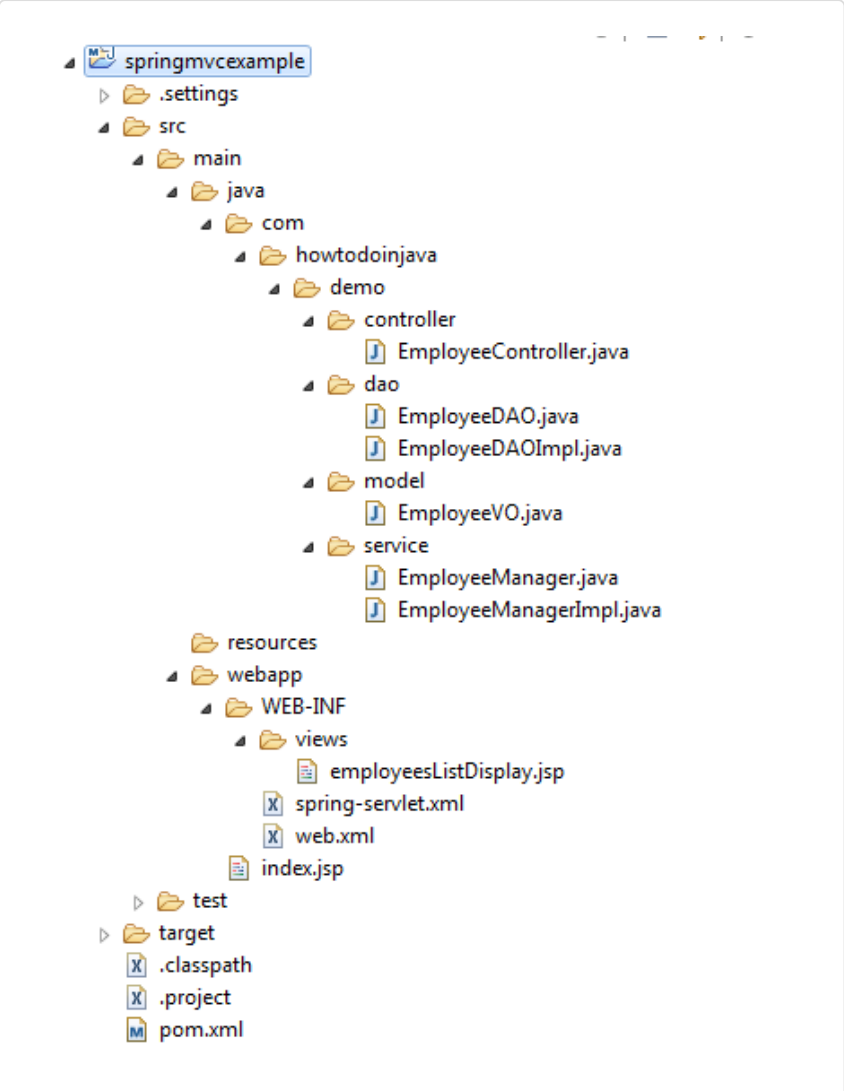
To define a controller class in Spring 3.0, a class has to be marked with the @Controller annotation. When a `@Controller` annotated controller receives a request, it looks for an appropriate handler method to handle the request. This requires that a controller class map each request to a handler method by one or more handler mappings. In order to do so, a controller class's methods are decorated with the `@RequestMapping` annotation, making them handler methods.

After a handler method has finished processing the request, it delegates control to a view, which is represented as handler method's return value. To provide a flexible approach, a handler method's return value doesn't represent a view's implementation but rather a logical view i.e. without any file extension. You can map these logical views to right implementation into applicationContext file so that you can easily change your view layer code without even touching request handler class code.

To resolve the correct file for a logical name is the responsibility of view resolvers. Once the controller class has resolved a view name into a view implementation, per the view implementation's design, it renders the objects.

Spring MVC Hello World Example

In this application, I am creating most simple **employee management application** demo having only one feature i.e. list all available employees in system. Let's note down the directory structure of this application.



Spring mvc hello world directory structure

Now let's write all the files involved into this hello world application.

pom.xml

Below `pom.xml` file contains dependencies for spring mvc and taglibs support for writing jsp files.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xs
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 h
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.howtodoinjava.demo</groupId>
    <artifactId>springmvcexample</artifactId>
    <packaging>war</packaging>
    <version>1.0-SNAPSHOT</version>
    <name>springmvcexample Maven Webapp</name>
    <url>http://maven.apache.org</url>
    <dependencies>

        <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>4.12</version>
            <scope>test</scope>
        </dependency>

        <!-- Spring MVC support -->

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-webmvc</artifactId>
            <version>4.1.4.RELEASE</version>
        </dependency>

        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-web</artifactId>
            <version>4.1.4.RELEASE</version>
        </dependency>

        <!-- Tag libs support for view layer -->

        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>jstl</artifactId>
            <version>1.2</version>
            <scope>runtime</scope>
        </dependency>

        <dependency>
            <groupId>taglibs</groupId>
            <artifactId>standard</artifactId>
            <version>1.1.2</version>
```

```
            <scope>runtime</scope>
        </dependency>

    </dependencies>

    <build>
        <finalName>springmvcexample</finalName>
    </build>
</project>
```

web.xml

This minimum `web.xml` file declares one servlet (i.e. dispatcher servlet) to receive all kind of requests. Dispatcher servlet here acts as front controller.

```
<web-app id="WebApp_ID" version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <display-name>Spring Web MVC Hello World Application</d

    <servlet>
        <servlet-name>spring</servlet-name>
            <servlet-class>
                org.springframework.web.servlet.DispatcherS
            </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>
```

spring-servlet.xml (You can have applicationContext.xml as well)

We are using annotated classes at request handler, service and dao layer so I have enabled annotation processing for all class files in base package " `com.howtodoinjava.demo` ".

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/co
    xsi:schemaLocation="http://www.springframework.org/sche
        http://www.springframework.org/schema/beans/spring-
        http://www.springframework.org/schema/context/
```

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

    <context:component-scan base-package="com.howtodoinjava

    <bean class="org.springframework.web.servlet.mvc.annota
    <bean class="org.springframework.web.servlet.mvc.annota

    <bean class="org.springframework.web.servlet.view.Inter
        <property name="prefix" value="/WEB-INF/views/" />
        <property name="suffix" value=".jsp" />
    </bean>

</beans>
```

EmployeeController.java

Annotation `@RequestMapping` at class level and method level determine the URL at which method will be invoked.

```
import org.springframework.beans.factory.annotation.Autowir
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMappi
import org.springframework.web.bind.annotation.RequestMetho

import com.howtodoinjava.demo.service.EmployeeManager;

@Controller
@RequestMapping("/employee-module")
public class EmployeeController
{
    @Autowired
    EmployeeManager manager;

    @RequestMapping(value = "/getAllEmployees", method = Re
    public String getAllEmployees(Model model)
    {
        model.addAttribute("employees", manager.getAllEmplo
        return "employeesListDisplay";
    }
}
```

Read More : How to use @Component, @Repository, @Service and @Controller Annotations?

EmployeeVO.java

This class act as model for MVC pattern.

```
package com.howtodoinjava.demo.model;

import java.io.Serializable;

public class EmployeeVO implements Serializable
{
    private static final long serialVersionUID = 1L;

    private Integer id;
    private String firstName;
    private String lastName;

    //Setters and Getters

    @Override
    public String toString() {
        return "EmployeeVO [id=" + id + ", firstName=" + fi
                + ", lastName=" + lastName + "]";
    }
}
```

EmployeeDAO.java

The classes at third tier in 3-tier architecture. Responsible for interacting with underlying DB storage.

```
import java.util.List;

import com.howtodoinjava.demo.model.EmployeeVO;

public interface EmployeeDAO
{
    public List<EmployeeVO> getAllEmployees();
}
```

EmployeeDAOImpl.java

```
import java.util.ArrayList;
import java.util.List;

import org.springframework.stereotype.Repository;

import com.howtodoinjava.demo.model.EmployeeVO;

@Repository
public class EmployeeDAOImpl implements EmployeeDAO {

    public List<EmployeeVO> getAllEmployees()
    {
        List<EmployeeVO> employees = new ArrayList<Employee

        EmployeeVO vo1 = new EmployeeVO();
        vo1.setId(1);
```

```
        vo1.setFirstName("Lokesh");
        vo1.setLastName("Gupta");
        employees.add(vo1);

        EmployeeVO vo2 = new EmployeeVO();
        vo2.setId(2);
        vo2.setFirstName("Raj");
        vo2.setLastName("Kishore");
        employees.add(vo2);

        return employees;
    }
}
```

EmployeeManager.java

The classes at second tier in 3-tier architecture. Responsible for interacting with DAO Layer.

```
import java.util.List;

import com.howtodoinjava.demo.model.EmployeeVO;

public interface EmployeeManager
{
    public List<EmployeeVO> getAllEmployees();
}
```

EmployeeManagerImpl.java

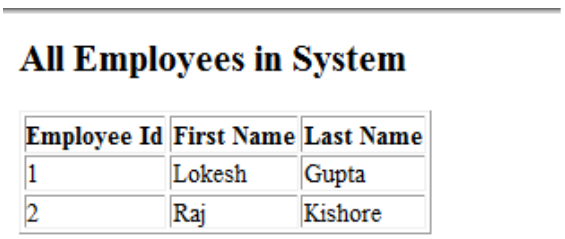
```
import java.util.List;

import org.springframework.beans.factory.annotation.Autowir
import org.springframework.stereotype.Service;

import com.howtodoinjava.demo.dao.EmployeeDAO;
import com.howtodoinjava.demo.model.EmployeeVO;

@Service
public class EmployeeManagerImpl implements EmployeeManager

    @Autowired
    EmployeeDAO dao;

    public List<EmployeeVO> getAllEmployees()
    {
        return dao.getAllEmployees();
    }
}
```

employeesListDisplay.jsp

This jsp is used to display all the employees in system. It iterates the collection of employees in loop, and print their details in a table. This fits into view layer of MVC pattern.

```
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/cor
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/f

<html>
<head>
    <title>Spring MVC Hello World</title>
</head>

<body>
    <h2>All Employees in System</h2>

    <table border="1">
        <tr>
            <th>Employee Id</th>
            <th>First Name</th>
            <th>Last Name</th>
        </tr>
        <c:forEach items="${employees}" var="employee">
            <tr>
                <td>${employee.id}</td>
                <td>${employee.firstName}</td>
                <td>${employee.lastName}</td>
            </tr>
        </c:forEach>
    </table>

</body>
</html>
```

Now deploy the application in your application server (i am using tomcat 7). And hit the URL "**http://localhost:8080/springmvcexample/employee-module/getAllEmployees**". You will see below screen if you have configured everything correctly.



All Employees in System

Employee Id	First Name	Last Name
1	Lokesh	Gupta
2	Raj	Kishore

Application front UI

Download Sourcecode

Drop me comments if something not working for you or is not clear in this tutorial.

Happy Learning !!

About Lokesh Gupta

A family guy with fun loving nature. Love computers, programming and solving everyday problems. Find me on Facebook and Twitter.

20 Leave a Reply

Join the discussion...

11 9 0 18

Subscribe newest oldest most voted

Yong Liang

How does Spring know where to find your Bean config file (spring-servlet.xml)? Other examples that I've seen specifies that location of that file with with in web.xml, or by AnnotationConfigWebApplicationContext class using pure Java configuration.

+ 0 − Reply 2 months ago

Ishan

Ideally you should use contextConfigLocation as below.

spring
org.springframework.web.servlet.DispatcherServlet

contextConfigLocation
/WEB-INF/default-servlet.xml

1

If you do not specify, then spring will automatically look for +"-"+servlet.xml in the WEB-INF folder.

+ 0 − Reply 2 months ago

Charanraj M G

I really struggled very hard to get this :

It would be more useful if you specify the development environment.

I had to change the spring-servlet.xml as below to get the output.

+ 0 − Reply 7 months ago

Lokesh Gupta

Thanks for the feedback. Added in my TODO list.

+ 0 − Reply 7 months ago

MdSardar

What did you change in spring-servlet.xml ? Can you put a note on that ? I'm continue to get
"The origin server did not find a current representation for the target resource or is not willing to disclose that one exists."

+ 2 − Reply 2 months ago

yan

I meet same problem

+ 0 − Reply 8 days ago

Neelavathi k

I tried the above example in my own way. in that I did not know how to configure the xml file for above program it shows the error repeatedly(File not found like).
Will you please explain how to configure and what are the files need to be configured

+ 0 − Reply 7 months ago

aditya

The corresponding table in the jsp page is arriving but not displaying any data.

+ 0 − Reply 11 months ago

Robin

there is no data displayed in the .jsp because the list was not set to page/request/session/application attribute.
you need to set it in the controller
ex: request.setAttribute("employees", employees); or
session.setAttribute("employees", employees);

+ -1 − Reply 6 months ago

worash

hey i am start to use spring mvc 5.0.9 and use java based configuration , when try to access my jsp page
i get the folowing error message
Type Status Report

Message /Contactapp/WEB-INF/view/ hello.jsp

Description The origin server did not find a current representation for the target resource or is not willing to disclose that one exists.
Apache Tomcat/9.0.8

+ 0 − Reply 1 year ago

Gopi

Did you checked the file name and without .jsp

+ 0 − Reply 10 months ago

czs_bugKiller

the first one is to show different parts' name,like employee or tourist
then url will be "http://localhost:8080/projectName/tourist/
the second one is to show one part's method name. like login or logout
then url will be
"http://localhost:8080/projectName/employee/login
hope to fix your problem.

+ 0 − Reply 1 year ago

murthi

Hi Lokesh,
your articles are simple to read and understand.
I have a question – what is the need of EmployeeManager and EmployeeManagerImpl classes here
Why you call 'manager.getAllEmployees()', instead we can call directly 'dao.getAllEmployees()' by using EmployeeDAO class.
Please clarify us.

+ 0 − Reply 3 years ago

Lokesh Gupta

Read my comments on this question : Service layer vs DAO — Why both?

+ 0 − Reply 3 years ago

Kamal

How view page(jsp) gets to know about model attribute “employees” and gets iterated.?
Are we can manually configure all the request maping to handlermapping ??? Can you please elaborate this

+ 0 − Reply 4 years ago

Lokesh Gupta

@ModelAttribute binds a method parameter or method return value to a named model attribute, exposed to a web view. It is essentially form backing object. After controller method execution, it’s data is copied to to both HttpServletRequest and HttpSession and thus accessible to JSP.

+ 0 − Reply 4 years ago

sandhya

why thr is two @requestmapping there at class level and method level?

+ 0 − Reply 4 years ago

Jamie

The class level maps to a url, and the second within the class maps to an even further extension. So class level starts it, and you can keep digging further into different pages.

So class level takes you too localhost:8080/employee-module

The method within the class takes you to localhost8080/employee-module/getAllEmployees

+ 0 − Reply 9 months ago

springmvc newbie

Hi – In the above example, can you pls add exception handling using @ExceptionHandler or @ControllerAdvice and illustrate how to handle exceptions thrown from DAO layer, via a common exception handling class in the controller layer?

+ 4 − Reply 4 years ago

Charles

Lokesh: What a great article you posted there! Thank you for sharing your knowledge!

+ 0 − Reply 5 years ago

/