**What is Spring MVC framework?**

The Spring web MVC framework provides [model-view-controller](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) architecture and ready components that can be used to develop flexible and loosely coupled web applications. The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between model, view and controller parts of application. Spring framework provides lots of advantages over other MVC frameworks e.g.

1. Clear separation of roles – controller, validator, command object, form object, model object, DispatcherServlet, handler mapping, view resolver, etc. Each role can be fulfilled by a specialized object.
2. Powerful and straightforward configuration of both framework and application classes as JavaBeans.
3. Reusable business code – no need for duplication. You can use existing business objects as command or form objects instead of mirroring them in order to extend a particular framework base class.
4. Customizable binding and validation
5. Customizable handler mapping and view resolution
6. Customizable locale and theme resolution
7. A JSP form tag library, introduced in Spring 2.0, that makes writing forms in JSP pages much easier. etc.

**What is DispatcherServlet and ContextLoaderListener?**

Spring’s web MVC framework is, like many other web MVC frameworks, request-driven, designed around a central Servlet that handles all the HTTP requests and responses. Spring’s DispatcherServlet however, does more than just that. It is completely integrated with the Spring IoC container so it allows you to use every feature that Spring has.

After receiving an HTTP request, DispatcherServlet consults the HandlerMapping (configuration files) to call the appropriate Controller. The Controller takes the request and calls the appropriate service methods and set model data and then returns view name to the DispatcherServlet. The DispatcherServlet will take help from ViewResolver to pickup the defined view for the request. Once view is finalized, The DispatcherServlet passes the model data to the view which is finally rendered on the browser.

|  |
| --- |
| <web-app>    <display-name>Archetype Created Web Application</display-name>      <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> |

By default, DispatcherServlet loads its configuration file using <servlet\_name>-servlet.xml. E.g. with above web.xml file, DispatcherServlet will try to find spring-servlet.xml file in classpath.

ContextLoaderListener reads the spring configuration file (with value given against “**contextConfigLocation**” in web.xml), parse it and loads the beans defined in that config file. e.g.

|  |
| --- |
| <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/applicationContext.xml</param-value>      </init-param>        <load-on-startup>1</load-on-startup>  </servlet> |

**What is the front controller class of Spring MVC?**

A front controller is defined as “a controller which handles all requests for a Web Application.” **DispatcherServlet (actually a servlet) is the front controller in Spring MVC that intercepts every request and then dispatches/forwards requests to an appropriate controller.**

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.

**How to use Java based configuration?**

To configure java based MVC application, first add required dependencies.

|  |
| --- |
| <!-- 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> |

Now add DispatcherServlet entry in web.xml file so that all incoming requests come though DispatcherServlet only.

|  |
| --- |
| <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> |

Now add below entries in spring configuration file.

|  |
| --- |
| <beans>      <!-- Scan all classes in this path for spring specific annotations -->      <context:component-scan base-package="com.howtodoinjava.demo" />        <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping" />      <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />       <!-- Vierw resolver configuration -->      <bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">          <property name="prefix" value="/WEB-INF/views/" />          <property name="suffix" value=".jsp" />      </bean>    </beans> |

Add controller code.

|  |
| --- |
| @Controller  @RequestMapping("/employee-module")  public class EmployeeController  {      @Autowired      EmployeeManager manager;        @RequestMapping(value = "/getAllEmployees", method = RequestMethod.GET)      public String getAllEmployees(Model model)      {          model.addAttribute("employees", manager.getAllEmployees());          return "employeesListDisplay";      }  } |

Additionally you should add manager and dao layer classes as well. Finally you add the jsp file to display the view.

**What is MVC pattern?**

MVC is a design pattern called Model-View-Controller. It decouples data access logic from business logic.

**Model:**

The Model contains the core of the application's functionality. It encapsulates the state of the application. Sometimes the only functionality it contains is state. It knows nothing about the view or controller.

**View:**

The view provides the presentation of the model. It is the look and feel of the application. The view can access the model getters, but it has no knowledge of the setters. In addition, it knows nothing about the controller. The view should be notified when changes to the model occur.

Controller:

The controller reacts to the user input. It creates and sets the model and helps to identify which view should be part of response.

**What is Spring?**

Spring is a framework for developing enterprise JavaTM applications. The benefit of using Spring over other frameworks is that it’s open source. This means developers can build reusable code without any vendor lock-in. Another major advantage of Spring is the layered architecture that enables you to select only the components you need while offering a seamless J2EE application development framework.

**List the advantages of the Spring framework.**

Spring has the following advantages:

Layered architecture that allows you to use what you need while leaving what you don’t need.

Spring allows developers to focus on Plain Old Java Object (POJO) Programming. This allows for continuous testing and integration.

Being an open source, there is no vendor lock-in.

Dependency injection and inversion of control makes Java Database Connectivity (JDBC) simpler.

List some features of Spring.

**Container:**

Spring manages and contains the configuration and life cycle of application objects.

**Lightweight:**

When it comes to transparency and size, Spring is a lightweight application framework. The lightest version of the Spring framework only takes up 1MB. Additionally, the overhead in terms of processing is similarly minuscule.

**MVC Framework:**

Spring utilizes the model-view-controller (MVC) web application framework which is built on the core Spring functionality. This framework accommodates several view technologies such as JSP, Tiles, Velocity, POI, and iText. Additionally, it is highly configurable with the use of strategy interfaces. However, several other frameworks could be easily used in place of the Spring MVC Framework. You can learn more about other MVC frameworks from this course on ASP.NET MVC 4.

**Inversion Of Control (IOC):**

Spring achieves loose coupling through the use of Inversion of Control. Objects provide their dependencies rather than looking for or creating dependent objects.

**Transaction Management:**

A generic abstraction layer is supplied by the Spring framework for the purpose of transaction management. This allows developers to include pluggable transaction managers while making it very easy to separate transactions while avoiding low-level issues. This transaction support is not connected with the J2REE environment. Also, it can be utilized in containerless environments.

**Aspect Oriented Programming (AOP):**

By separating system services from application business logic, Spring supports aspect oriented programming. This also allows for cohesive deployment of applications.

**JDBC Exception Handling:**

The Java Database Connectivity (JDBC) abstraction layer of Spring provides a useful exception hierarchy. This makes error handling strategy much easier to develop. Additionally, Spring offers great integration services with JDO, iBATIS, and Hibernate.

**What is Dependency Injection (AKA IOC)?**

Dependency Injection or Inversion of Control (IOC), at the most basic level, allows you to describe how objects should be created rather than creating them directly. That is, you describe the services that are needed by different components using a configuration file rather than directly connecting these components and services in code. In the case of the Spring framework, these services and components are then connected by the IOC container.

For example, objects are given their dependencies when they are created by an external process that manages each object within a system. In other words, the dependencies are inserted into objects. This is contrary from the way dependencies are handled in other frameworks, which is why the term inversion of control was coined. This signifies an inversion of responsibility for creating references to dependencies within objects. Here’s a course that can teach you more about the basics of Spring 3.2 Framework.

List some different types of Dependency Injection (IOC)

The three (3) types of dependency injection, or IOC, are:

**Setter Injection** (used by Spring): JavaBeans properties are used to assign dependencies.

**Interface Injection** (used by Avalon): An interface is used for injection.

**Constructor Injection** (used by Spring, Pico container, and others): Constructor parameters are used to provide dependencies.