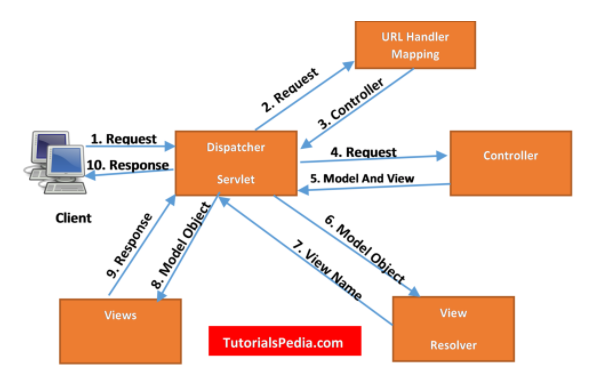
**Spring MVC Overview**

> Framework for building web applications in Java

> Based on **Model – View – Controller** design pattern

> Leverages features of the Core Spring Framework (IoC, DI)



**Explanation of Spring MVC Request Flow**

1. Client requests for a page by specifying the Web URL for the page. E.g., <https://github.com>

2. Client request is intercepted by the Dispatcher Servlet (part of the Spring framework) also known as Front Controller. **Dispatcher Servlet** is a servlet specified in **Web.XML** file (for XML based configuration).

3. Dispatcher Servlet uses **URL Mapping Handler** to find out the relevant controller class to which request should be passed for subsequent processing. For e.g. If you have a Controller defined for all requests by specifying “/” in the URL, all requests will be entertained by that **controller**.

4. Once Dispatcher Servlet has identified the controller to be considered, it passes the client request to the controller.

5. The controller class is the main class controlling the business logic flow once request has been dispatched it by dispatcher servlet. This class will implement the methods for different type of http requests (e.g., GET, POST) & all logic to call Service layer methods will reside in this controller class.

The controller class will also be responsible for returning the **ModelAndView** object back to the dispatcher servlet after getting all business logic executed & any data returned from DAO layer. **ModelAndView** object returned by the controller back to the controller specified both view & model objects.

Controller class is annotated by **@Controller** annotation.

8. After receiving **ModelAndView** object from the controller, Dispatcher Servlet now sends model object to view resolver to get the name of the view which needs to be rendered.

9. Once the view to be rendered has been identified, Dispatcher Servlet passes model object to the view. Model object contains the data which needs to be displayed in the view. View will be rendered with the model data. Views can be designed in any front – end technology.

10. This view is returned to the client & client can see the view & associated data on his browser.

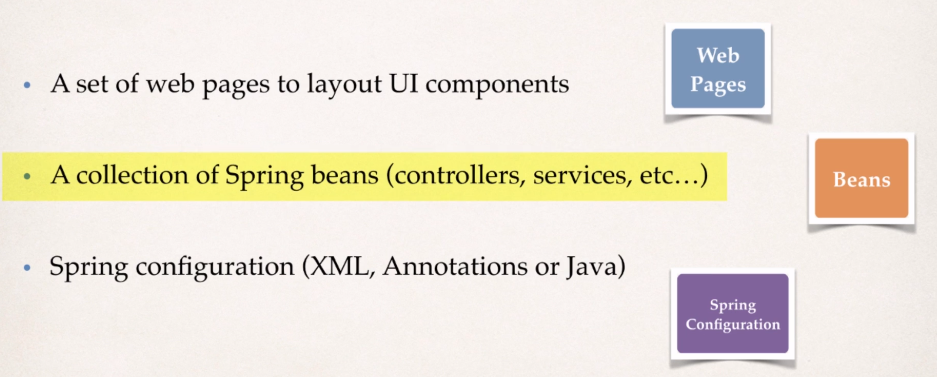
Views can be a HTML page or JSP page or Thymleaf or Velocity or FreeMarker

**Spring MVC Benefits**

* The Spring way of building web app UIs in Java
* Leverage a set of reusable UI components
* Help manage application state for web requests
* Process form data: validation, conversion etc.
* Flexible configuration for the view layer

**Spring MVC Documentation**: - <https://docs.spring.io/spring-framework/docs/current/reference/html/web.html>

**Components of a Spring MVC Application**



**a) Controller**

> Code created by developer

> Contains your business logic

* Handle the request
* Store/ retrieve data (db, web service…)
* Place data in model

> Send to appropriate view template

**b) Model**

> Model: contains your data

> Store/ retrieve data via backend systems

* Database, web service, etc…
* Use a Spring bean if you like

> Place your data in the model

* Data can be any Java object/ collection

**c) View Template**

> Spring MVC is flexible

* Supports many view templates

> Most common is JSP (Java Server Pages) + JSTL (JSP Standard Tag Library)

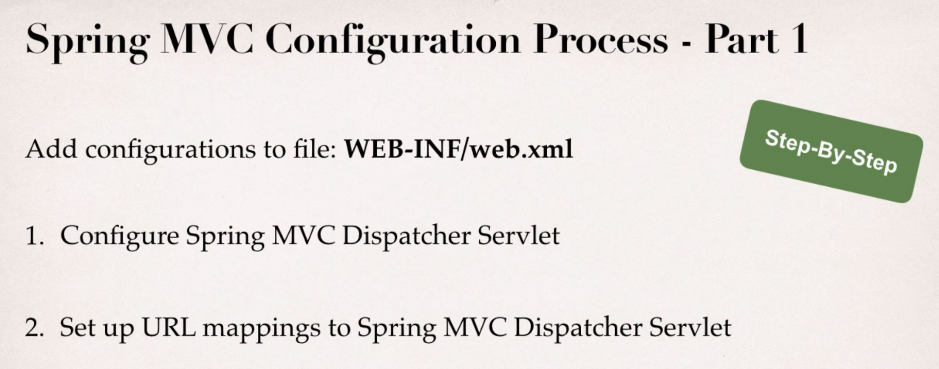
> Developer creates a page

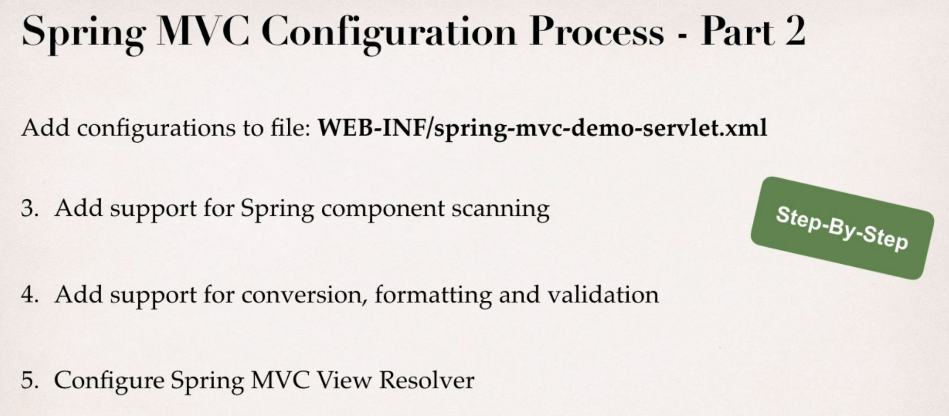
* Displays data

> Other view templates supported

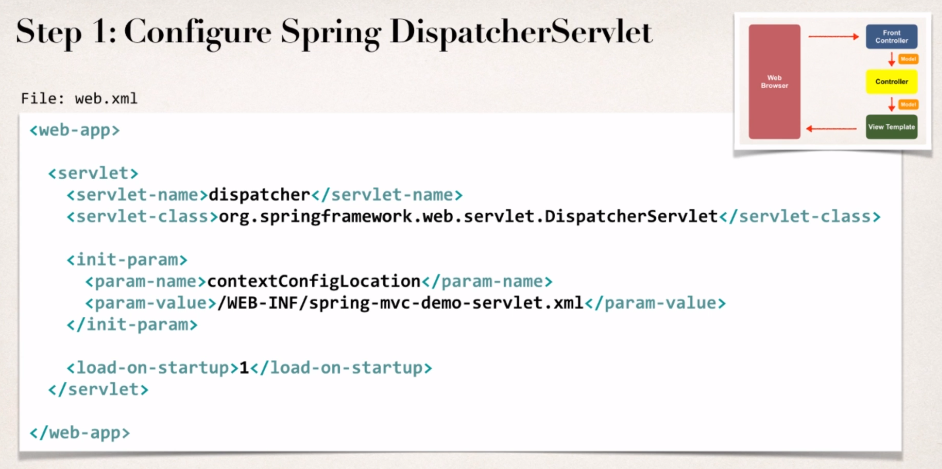
* Thymeleaf, Groovy
* Velocity, Freemarker etc…

**Spring MVC Configuration Process**





**Step 1:** Configure Spring DispatcherServlet

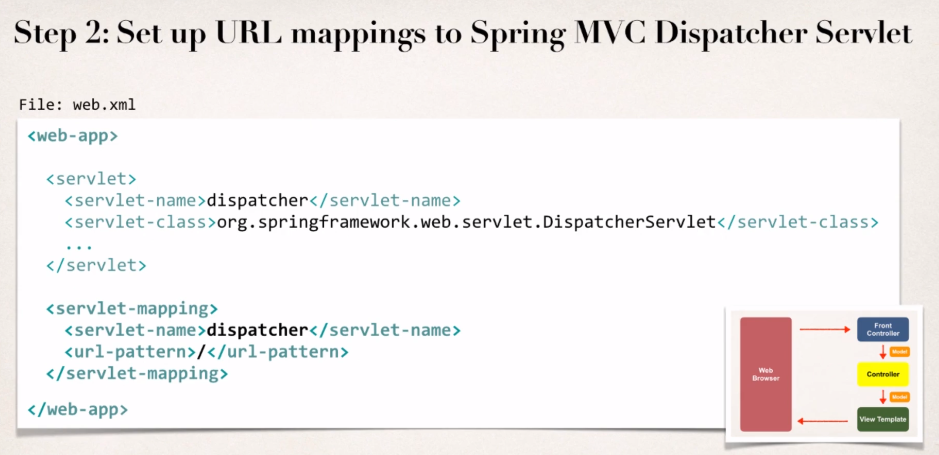


> In web.xml file, we need to add an entry for the Spring DispatcherServlet, or the Front controller.

> DispatcherServlet is part of the core Spring framework.

> Once you set up the servlet reference, then you set up initial parameter that tells where your spring context configuration file is located.

**Step 2:** Set up URL mappings to Spring MVC Dispatcher Servlet



> Here basically we want to tell the system that for any URL pattern coming in, pass it off to the DispatcherServlet i.e., all incoming web requests should be handled by the DispatcherServlet.

> servlet name in servlet mapping must be matching with the servlet definition.

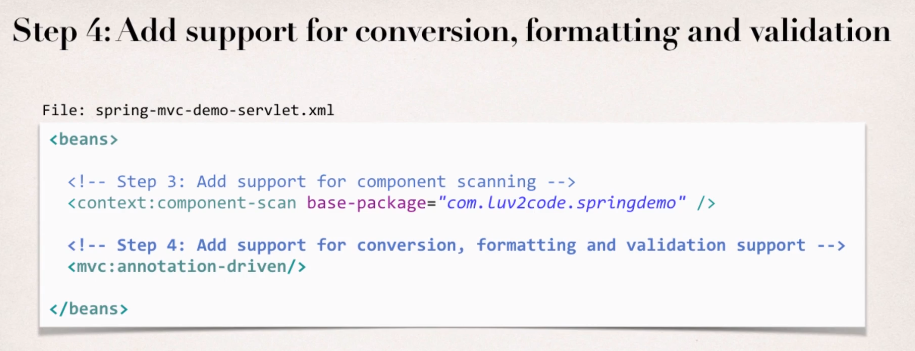
**Step 3:** Add support for Spring component scanning



> It’ll basically scan the base – package for any special Spring beans (i.e., any @Component items) & make them available & put them into the Spring context.

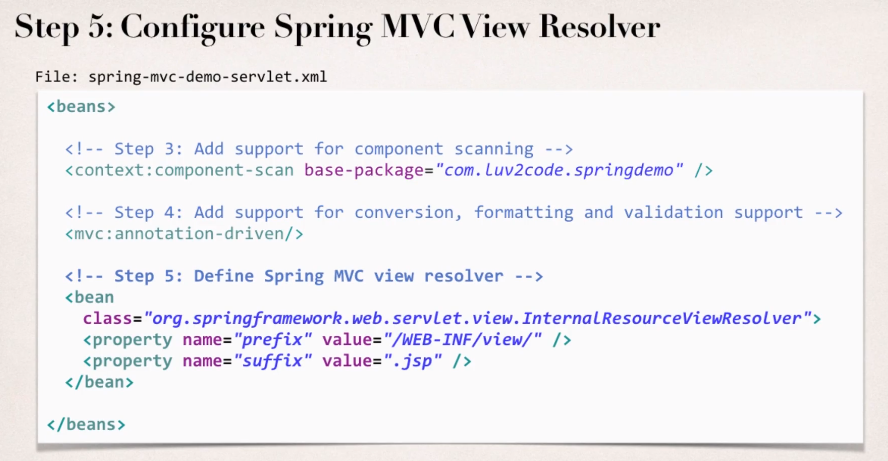
**> Spring will recursively scan for components starting at the base package & scan all sub packages.**

**Step 4:** Add support for conversion, formatting & validation



> It could perform conversions of form data, format form data & perform form validation.

**Step 5:** Configure Spring MVC View Resolver



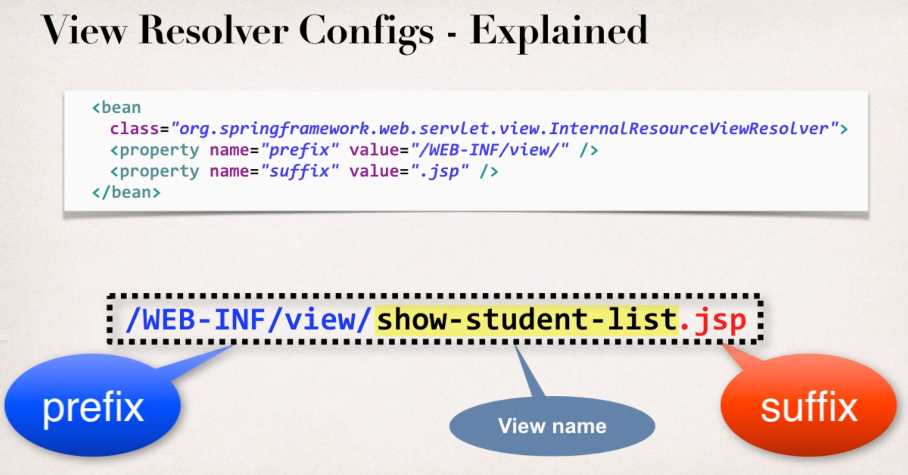
> It tells us how the pages are displayed, where the pages are located.

> So, we make use of this **InternalResourceViewResolver** & we give a prefix & then a suffix.

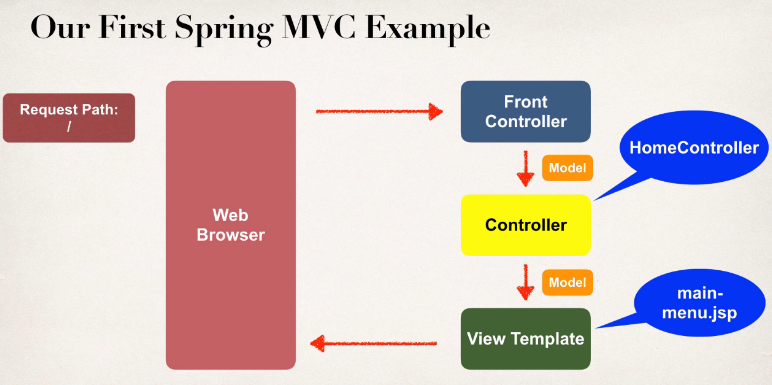
> When your app provides a “view” name, Spring MVC will

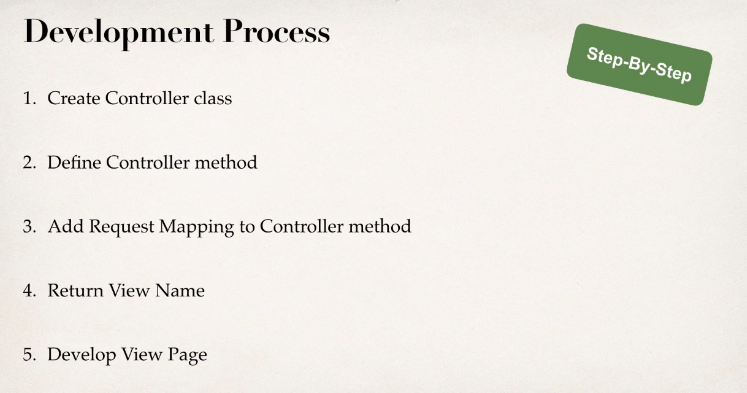
* Prepand the prefix
* Append the suffix

> So, it’s basically telling where to look for files, to actually render the view for the application.

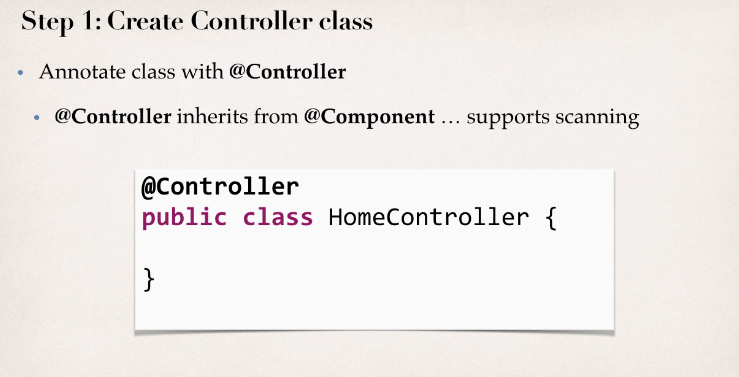


**Spring MVC Example**

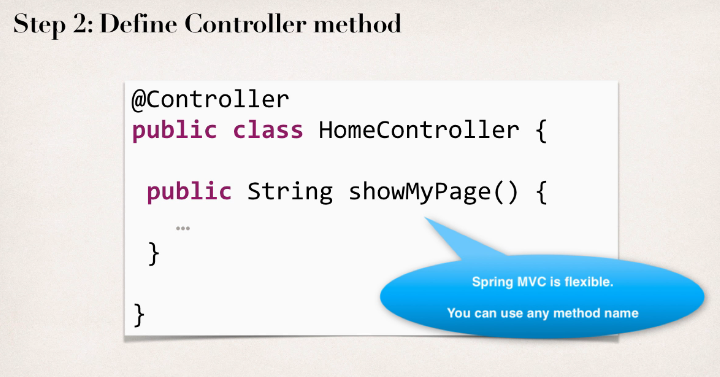




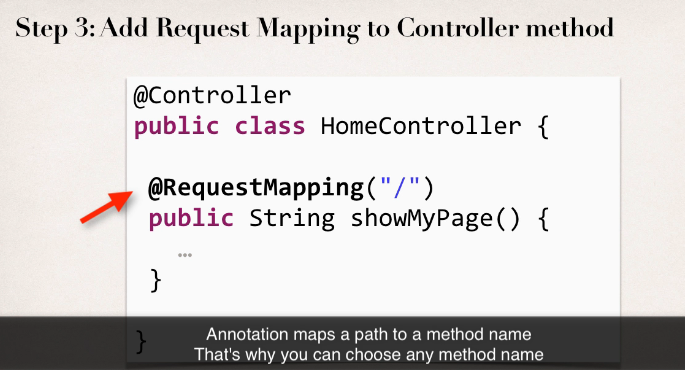
**Step 1:** Create Controller class **(@Controller inherits from @Component**, so it’s really just a specialized component that supports web MVC so when Spring does its component scanning, then it’ll also pick up @Controller)



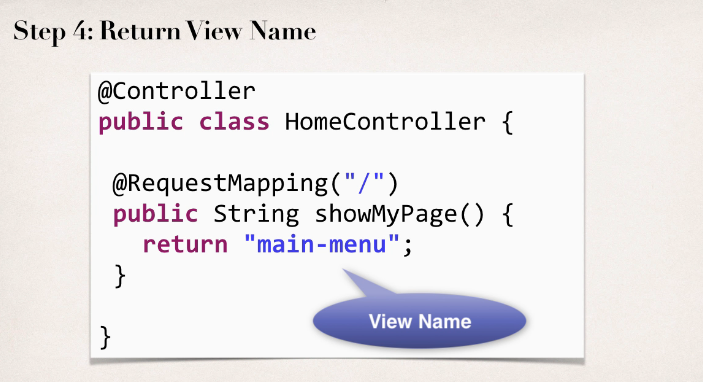
**Step 2:** Define Controller method (**return the actual view name** that I want to show)



**Step 3:** Add Request Mapping to Controller method (so we basically need to map some type of web request to this method & @RequestMapping will handle all types of requests – get, post etc.)



**Step 4:** Return View Name (Spring’s actually gonna use information from its configuration file & it’ll actually find the view page.)



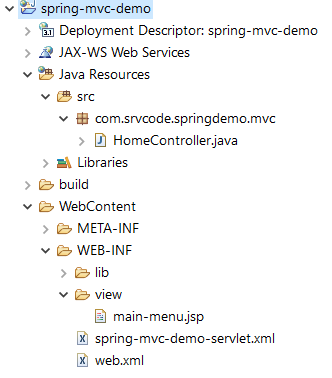


**Step 5:** Develop View page



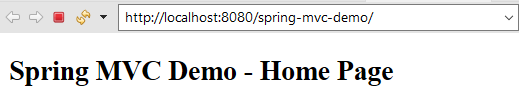
**Code:**

Create a dynamic web project in eclipse

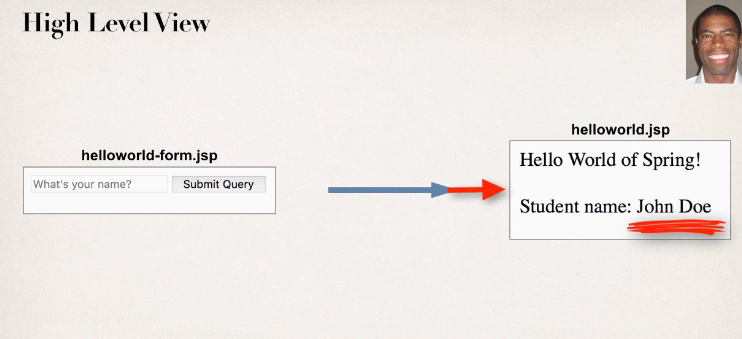


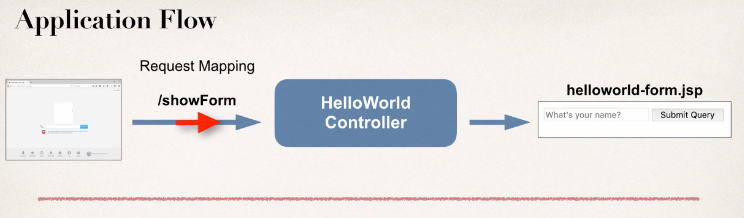
|  |
| --- |
| **web.xml**  <?xml version=*"1.0"* encoding=*"UTF-8"*?>  <web-app xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*  xmlns=*"http://xmlns.jcp.org/xml/ns/javaee"*  xsi:schemaLocation=*"http://xmlns.jcp.org/xml/ns/javaee http://xmlns.jcp.org/xml/ns/javaee/web-app\_3\_1.xsd"*  id=*"WebApp\_ID"* version=*"3.1"*>  <display-name>spring-mvc-demo</display-name>  <absolute-ordering />  <!-- Spring MVC Configs -->  <!-- Step 1: Configure Spring MVC Dispatcher Servlet -->  <servlet>  <servlet-name>dispatcher</servlet-name>  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>  <init-param>  <param-name>contextConfigLocation</param-name>  <param-value>/WEB-INF/spring-mvc-demo-servlet.xml</param-value>  </init-param>  <load-on-startup>1</load-on-startup>  </servlet>  <!-- Step 2: Set up URL mapping for Spring MVC Dispatcher Servlet -->  <servlet-mapping>  <servlet-name>dispatcher</servlet-name>  <url-pattern>/</url-pattern>  </servlet-mapping>    </web-app>  **spring-mvc-demo-servlet.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"*>  <!-- Step 3: Add support for component scanning -->  <context:component-scan base-package=*"com.srvcode.springdemo"* />  <!-- Step 4: Add support for conversion, formatting and validation support -->  <mvc:annotation-driven/>  <!-- Step 5: 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>  **Step 1, 2, 3, 4**  @Controller  **public** **class** HomeController {  @RequestMapping("/")  **public** String showPage() {  **return** "main-menu";  }  }  **Step 5**  <!DOCTYPE html>  <html>  <body>  <h2>Spring MVC Demo - Home Page</h2>  </body>  </html> |

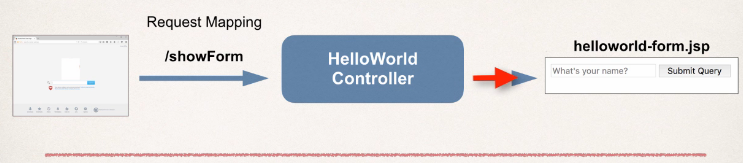
**Output:**

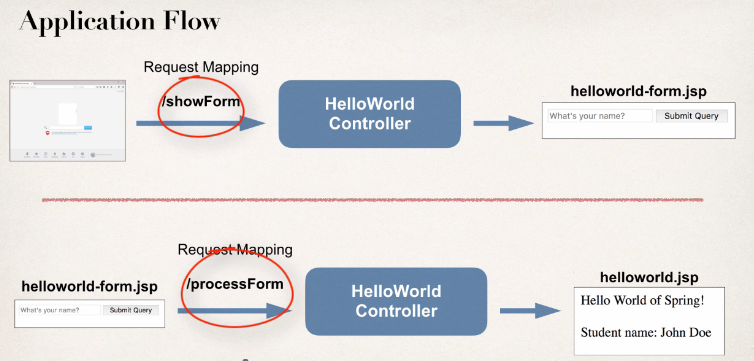


**Reading HTML Form Data**









**Note**: The key point here is that we’re using one HelloWorld Controller that’s going to have two request mappings.

