**47.2. Spring MVC Java Configuration**

**Java Configuration**:

* Instead of configuring Spring MVC app using XML
  + Web.xml
  + spring-mvc-demo-servlet.xml
* Configure the Spring MVC app with Java code

**Development Process (Step-by-Step)**:

1. Add Maven dependencies for Spring MVC Web App
2. Create Spring App Configuration (@Configuration)
3. Create Spring Dispatcher Servlet Initializer
4. Develop our Spring controller
5. Develop our JSP view page

**Step 1: Add Maven dependencies for Spring MVC Web App**:

* Spring-webmvc
* Jstl
* Javax.servlet-api
* Javax.servlet.jsp-api

**Spring -webmvc**:

<!-- Spring MVC support -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>...</version>

</dependency>

This will load all supporting dependencies spring-core, spring-context, spring-bean, logging etc. …

**Add Servlet, JSP and JSTL**:

<!-- Servlet, JSP and JSTL support -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>3.1.0</version>

</dependency>

<dependency>

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

<artifactId>javax.servlet.jsp-api</artifactId>

<version>2.3.1</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>jstl</artifactId>

<version>1.2</version>

</dependency>

**Custom Maven Build**:

Now we have to customize the Maven build. Because we are not using the **web.xml**.

<!-- TO DO: Add support for Maven WAR Plugin -->

<build>

<finalName>spring-security-demo</finalName>

<pluginManagement>

<plugins>

<plugin>

<!-- Add Maven coordinates (GAV) for: maven-war-plugin -->

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-war-plugin</artifactId>

<version>3.2.3</version>

</plugin>

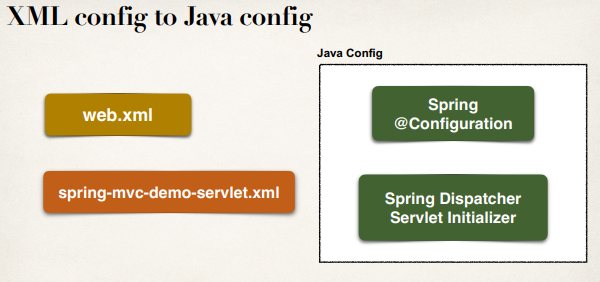
</plugins>

</pluginManagement>

</build>

Here we are adding Maven WAR plugging for customizing Maven build process.

**XML config to Java config**:



This Java configuration replace XML configuration.

**XML config (the old way)**:

<!-- Add Aspect autoproxy support for AOP -->

<aop:aspectj-autoproxy />

<!-- Add support for component scanning -->

<context:component-scan

base-package=*"com.odduu.ruhul"* />

<!-- 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>

**Enabling the MVC Java Config**:

**@EnableWebMvc**

* Provides similar support to **<mvc:annotation-driven />** in XML.
* Adds conversion, formatting and validation support
* Processing of @Controller classes and **@RequestMapping** etc ... methods

**Step 2: Create Spring App Configuration**:

@Configuration

@EnableWebMvc // provides similar support to <mvc:annotation-driven />

@ComponentScan(basePackages = "com.ruhul.springsecurity.demo")

**public** **class** DemoAppConfig {

// define a bean for ViewResolver

@Bean

**public** ViewResolver viewResolver() {

InternalResourceViewResolver viewResolver = **new**

InternalResourceViewResolver();

viewResolver.setPrefix("/WEB-INF/view/");

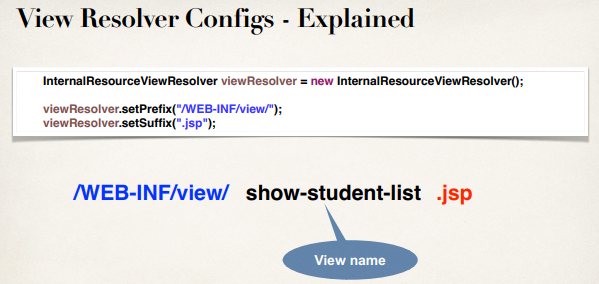
viewResolver.setSuffix(".jsp");

**return** viewResolver;

}

}

**View Resolver Configs – Explained**:



**The old way (Web.xml)**:

<web-app

...

>

<display-name>spring-mvc-crud-demo</display-name>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

<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-crud-demo-servlet.xml</param-value>

</init-param>

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

</servlet>

<servlet-mapping>

<servlet-name>dispatcher</servlet-name>

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

</servlet-mapping>

</web-app>

**Web App Initializer**:

* Spring MVC provides support for web app initialization
* Makes sure your code is automatically detected
* Your code is used to initialize the servlet container

**AbstractAnnotationConfigDispatcherServletInitializer**

**Web App Initializer (more info)**:

* Your TO DO list
  + Extend this abstract base class
  + Override required methods
  + Specify servlet mapping and location of your app config

**Step 3: Create Spring Dispatcher Servlet Initializer**:

**public** **class** MySpringMvcDispatcherServletInitializer **extends**

AbstractAnnotationConfigDispatcherServletInitializer {

// no root config classes for our project... only servlet config classes

@Override

**protected** Class<?>[] getRootConfigClasses() {

// **TODO** Auto-generated method stub

**return** **null**;

}

@Override

**protected** Class<?>[] getServletConfigClasses() {

// **TODO** Auto-generated method stub

**return** **new** Class[] { DemoAppConfig.**class** };

}

@Override

**protected** String[] getServletMappings() {

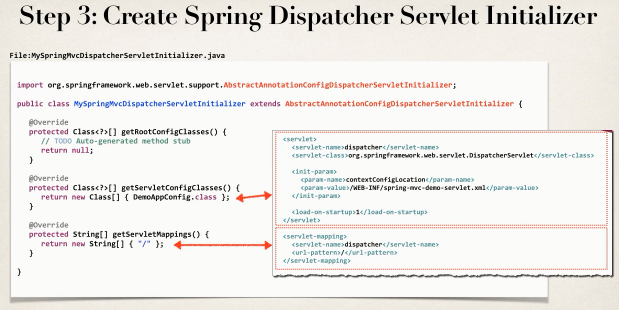
// **TODO** Auto-generated method stub

**return** **new** String[] { "/" };

}

}

**Step 3: Create Spring Dispatcher Servlet Initializer (Diagram):**



**Step 4: Develop our Spring Controller**:



**Code**:

@Controller

**public** **class** DemoController {

@GetMapping("/")

**public** String showHome() {

**return** "home";

}

}

**Step 5: Develop our JSP view page**:



47.2. Spring MVC Java Configuration