LAB- Spring Mvc - Using Java Configuration

In this lab you will be understanding how to create a spring mvc application using Java Configuration

STEP -1

In this step, you will be working on **05mvc-javaconfiguration-start** project

1) The code provided to you is same as the one in **03mvc-basic-way2learn-solution** project .

Deploy the Application and make sure it is working normally,

2) Now Replace db-config.xml with **RootConfig.java** inside **com.way2learnpackage** ,

Luckily, to save time, you have been already provided with RootConfig.java . Just annotatte it with following :

```
@EnableTransactionManagement
@ComponentScan(basePackages={"com.way2learnonline.service","com.way2learnonline.dao"})
@Configuration
```

3) Now delete db-config.xml.
Open web.xml and see that we are configuring contextConfigLocationas below :

```
<context-param>
	<param-name>contextConfigLocation</param-name>
	<param-value>/WEB-INF/db-config.xml</param-value>
</context-param>
```

Now Replace the above with

```
<context-param>
	<param-name>contextConfigLocation</param-name>
	<param-value>com.way2learnonline.RootConfig</param-value>
</context-param>
```

We have to tell which ApplicationContext implementation has to be used.

By default if we dont specify anything, it will use XmlWebApplicationContext.

Since we are using java configuration , we have to specify AnnotationConfigWebApplicationContext as shown below :

```
<context-param>
      <param-name>contextClass</param-name>
      <param-value>org.springframework.web.context.support.AnnotationConfigWebApplicat
                                                ionContext</param-value>
</context-param>
```

Now deploy the Application and make sure it works like earlier.

4) Similarly Replace mvc-config.xml with MvcConfig.java
Fortunately, you have been given with MvcConfig.java already. Just annotate it with the followong :

```
@EnableWebMvc
@ComponentScan(basePackages={"com.way2learnonline.controllers"})
@Configuration
```

Delete mvc-config.xml.

Now modify DispatcherServlet configuration in web.xml as follows :

```
  <servlet>
    <servlet-name>dispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>com.way2learnonline.MvcConfig</param-value>
    </init-param>
     <init-param>
      <param-name>contextClass</param-name>

<param-value>org.springframework.web.context.support.AnnotationConfigWebApplicationCont
ext</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
```

Deploy the application and check it. You will see that css is not applied to your pages (If css is applied and page is displayed properly, it may be because of cache. Just refresh the page. U should see that css is not applied).

Can you find the reason ? Think for a while before reading further.

Since we mapped dispatcher servlet to / , static resource request cannot be handled properly. If you remember, in xml we declared defaultServlet handler in mvc-config.xml as shown below :

```
<mvc:default-servlet-handler/>
```

So, How to define equivalent of above tag in mvc namespace ?

If you want to define equivalent of tags defined in mvc namespace, your MvcConfig.java must extend **WebMvcConfigurerAdapter .**

So, modify MvcConfig.java to extend WebMvcConfigurerAdapter and overrride a method as shown below :

```
@Override
	public void configureDefaultServletHandling(DefaultServletHandlerConfigurer
configurer) {
		configurer.enable();
	}
```

Now deploy and test. It should work as usual

5) Now the resources folder is outside of WEB-INF . move it inside of WEB-INF .

Now deploy and test it. Again Css is not applied .

How to map requests for urls /resources/** to /WEB-INF/resources ?

Do you remember the following configuration in xml ?

```
<mvc:resources location="/WEB-INF/resources/" mapping="/resources/**"
cache-period="600" />
```

What is the equivalent of above tag in java configuration ?

Just override a method in MvcConfig as shown below:

```
@Override
	public void addResourceHandlers(ResourceHandlerRegistry registry) {
		registry.addResourceHandler("/resources/**")
			.addResourceLocations("/WEB-INF/resources/");
	}
```

Now deploy and test. It should work as usual

6) Now we want to eliminate web.xml also. Delete it.

In servlet 3.x , what is the equivalent to web.xml ?

WAY2LEARN

You have to write an implementation of WebApplicationInitializer as shown below (Dont Write it now . Just read next.):

```
public class MyServletContainerInitializer implements ServletContainerInitializer {

	@Override
	public void onStartup(Set<Class<?>> classes, ServletContext context) throws
ServletException {

		ServletRegistration.Dynamic
dispatcherServlet=context.addServlet("dispatcherServlet", DispatcherServlet.class);
		dispatcherServlet.setLoadOnStartup(1);
		dispatcherServlet.addMapping("/");

	}

}
```

Then you have to create a file META-INF/services/javax.servlet.ServletContainerInitializer . Inside that file, you have to write the name of this implementation class.

This implementation class will be the starting point for our webapplication incase web.xml is not present.

But fortunately, and implementation is already provided in spring and the class name is configured in Spring-web-someversion.jar/ META-INF/services/javax.servlet.ServletContainerInitializer

So, SpringServletContainerInitializer is implementing ServletContainerInitializer and it scans the classpath for any file implementing WebApplicationInitializer and call methods on it.

So, If you write any implementaion of WebApplicationInitalizer, that will be invoked by SpringServletContainerInitializer .

You could write a class like below (Dont Write it immediately. There is a shortcut. Keep Reading.)

```
	public class MyWebAppInitializer implements WebApplicationInitializer
{
		@Override
		public void onStartup(ServletContext servletContext)
				throws ServletException {

			AnnotationConfigWebApplicationContext rootContext= new
AnnotationConfigWebApplicationContext();
			rootContext.register(AppConfig.class);

			servletContext.addListener(new
ContextLoaderListener(rootContext));
```

```
                AnnotationConfigWebApplicationContext dispatcherContext=
new AnnotationConfigWebApplicationContext();
                dispatcherContext.register(CustomMvcConfig.class);

                ServletRegistration.Dynamic dispatcherServlet=

      servletContext.addServlet("dispatcherServlet", new
DispatcherServlet(dispatcherContext));
                dispatcherServlet.setLoadOnStartup(1);
                dispatcherServlet.addMapping("/");
            }
      }
```

But Instead of we writing all these logic to register dispatcher servlet and ContextLoaderListener, there is already a class `AbstractAnnotationConfigDispatcherServletInitializer` .

you can just extend this as shown below :

```
public class WebInitializer extends AbstractAnnotationConfigDispatcherServletInitializer
{

      @Override
      protected Class<?>[] getRootConfigClasses() {
            return new Class[]{RootConfig.class};
      }

      @Override
      protected Class<?>[] getServletConfigClasses() {
            return  new Class[]{MvcConfig.class};
      }

      @Override
      protected String[] getServletMappings() {
            return new String[]{"/"};
      }

}
```

So, the above class is a replacement of web.xml
delete web.xml
Write the above class inside com.way2learnonline package.

Deploy the application and observe that everything works as usual.

Congratulations !! You know how to create mvc application using Java Configuration!!

WAY2LEARN