Spring MVC View

The view is a component of MVC architecture that is used to return a user interface output to the user in response to the user request. A View page can be any HTML or JSP file.

Spring MVC internally uses a view resolver to fetch the requested view to the user. There are several ways to configure the view in the Spring application. We can use XML, Java code, and property file to configure the view.

Spring MVC defines the ViewResolver and View interfaces that let you render models in a browser without tying you to specific view technology. ViewResolver provides a mapping between view names and actual views. View addresses the preparation of data before handing over to the specific view of technology.

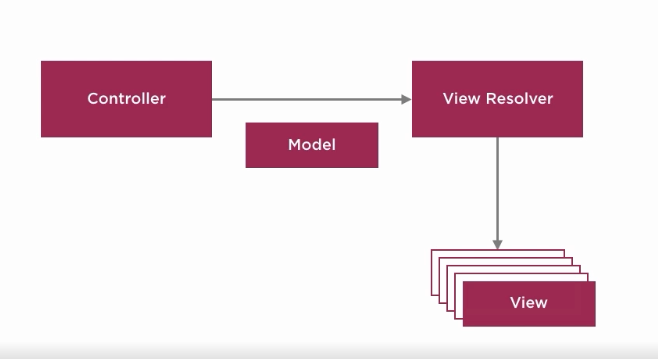
ViewResolver Implementations

The following are the implementation classes of the ViewResolver interface.

| **ViewResolver** | **Description** |
| --- | --- |
| AbstractCachingViewResolver | This class is used to make a cache of the view instance. Caching improves the performance of certain view technologies. Although we can turn off the cache by setting the cache property to false. |
| XmlViewResolver | This class is used to handle the XML view files. It accepts a configuration file written in XML with the same DTD as Spring’s XML bean factories. |
| ResourceBundleViewResolver | It uses bean definitions in a ResourceBundle, specified by the bundle base name. For each view it is supposed to resolve, it uses the value of the property [viewname].(class) as the view class and the value of the property [viewname].url as the view URL. |
| UrlBasedViewResolver | It used to affect the direct resolution of logical view names to URLs without an explicit mapping definition. |
| InternalResourceViewResolver | It is a convenient subclass of UrlBasedViewResolver that supports InternalResourceView and subclasses such as JstlView and TilesView. |
| FreeMarkerViewResolver | This class is a subclass of UrlBasedViewResolver that supports FreeMarkerView. |
| ContentNegotiatingViewResolver | It is used to resolve a view based on the request file name. |

How Spring ViewResolver Works?

When a user requests a view then the controller interacts with the ViewResolver. The ViewResolver returns the requested view back to the controller and then the controller back to the user.



Configure views in Spring

There are three ways to configure views in the Spring application. We can use any of these ways to configure view:

* XML
* Java Code
* Property file

Configure View using XML Tags

We can use <bean> tag to set class InternalResourceViewResolver and <property> tag to set prefix and suffix for the views page. For example, prefix tells the package location where views files are stored and suffix tells the extension of the view file. See the below code.

<!-- Define Spring MVC view resolver -->

<bean

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix" value="/WEB-INF/views/" />

<property name="suffix" value=".jsp" />

</bean>

Configure View using Java Code

We can configure view handling in Java by adding some code to the configuration class. A class that is marked with @Configuration annotation is treated as a configuration class. You can see, we used setPrefix() and setSuffix() methods as substitutes for <property> tag of the XML file.

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.web.servlet.ViewResolver;

import org.springframework.web.servlet.view.InternalResourceViewResolver;

@Configuration

public class AppConfig {

@Bean

public ViewResolver viewResolver() {

InternalResourceViewResolver irvr = new InternalResourceViewResolver();

irvr.setPrefix("WEB-INF/views");

irvr.setSuffix(".jsp");

irvr.setOrder(0);

return irvr;

}

}

Configure View using Properties File

We can use property file for setting view configuration such as prefix and postfix.

spring.mvc.view.prefix = /WEB-INF/views

spring.mvc.view.suffix = .jsp

After creating this, we can read it into our application using @PropertySource annotation. For example,

@PropertySource("classpath:views.properties")

Here, views is a name of the property file. In our next topic, we will see a Simple Hello Spring application.