Before move ahead let’s look the below code and revise what it does

*@Controller*

*@RequestMapping*("mother")

public class MomController {

*@ResponseBody*

*@RequestMapping*("/sugar")

public String giveSugar() {

return "Here is your sugar";

}

}

Do you remember, what is the use of **@ResponseBody** here in this code ?

The use of **@ResponseBody** is that whatever returned by the **giveSugar()** method will be add into the response body and will be display over the webpage.

But this is not a professional way to respond a request. We cannot write the complete response data inside a function and when our application will grow up so it will definitely need to return data in html format which will bad practice to merge html code with java code.

So it’s time to eliminate **@ResponseBody** annotation and learn how to generate **views** and respond through that **views.**

**What is Views ?**

The views are just the files where we put all the html code which is then going to rendered on a browser webpage. These files can be .jsp, .html etc.

**Eliminate @ResponseBody :**

*@Controller*

*@RequestMapping*("mother")

public class MomController {

*@RequestMapping*("/sugar")

public String giveSugar() {

return "Here is your sugar";

}

}

If you try to access the “/sugar” url you will get **404 error code** now.

But why **404 error code ?**

When we were using @ResponseBody annotation at that time whatever method used to return was add inside the response body. But now @ResponseBody annotation not present, in this situation spring is supposing that the method is returning **the path of a View**.

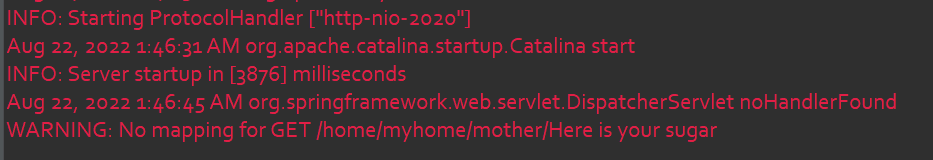
*@RequestMapping*("/sugar")

public String giveSugar() {

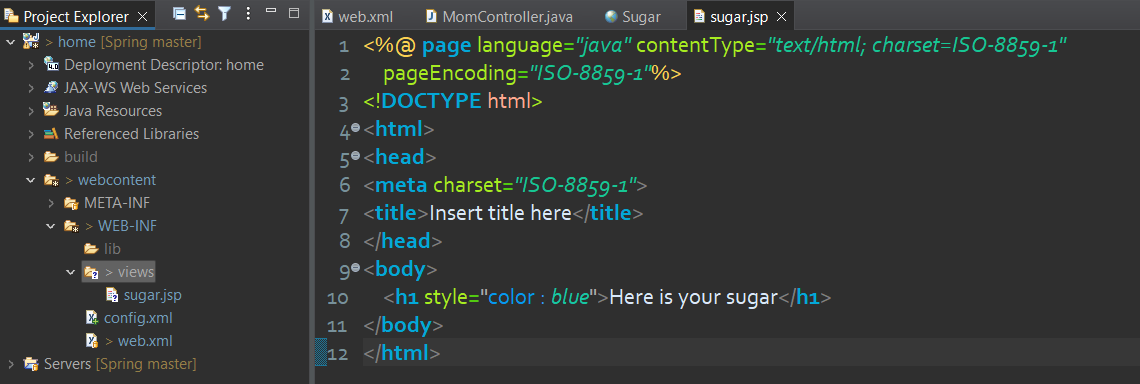
return "Here is your sugar";

}

So, spring is thinking that “Here is your sugar” is now a path for a view but spring is not able find such path in the project that why we are getting **404 error code.**

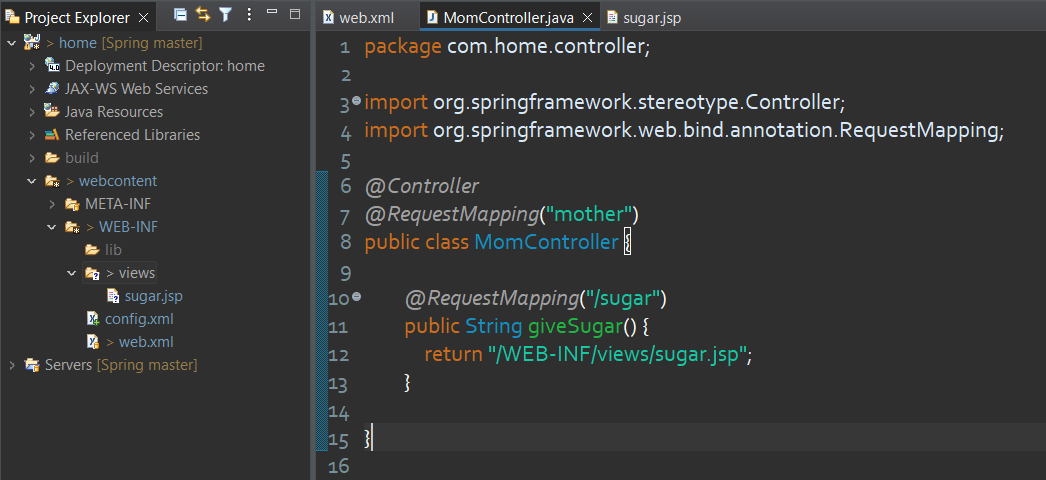


Let’s create a view and try to return exact path of the view from our method and see what happen.



I have first created a folder **views** inside **WEB-INF** folder and then I have created a view called **sugar.jsp**

Now just return the exact path of this view from your method as shown below :



As expected, **sugar.jsp** view should be rendered on webpage.

But now, there is problem in this approach. What if in future we need to change location of **sugar.jsp** ? What if we need to change the view from **sugar.jsp** to **sugar.html** or other front end technology.

**Changes that can occur :**

* Location of view can change
* Extension of view can change

**What will not change :**

* Name of view

If any above changes come in the application we need to manipulate our source code i.e. we need to change the path returning by the **giveSugar()** method.

This is not the way we code in the Spring Environment. To make our application loosely coupled while working with views, spring provide something called **View Resolver.**

Let’s understand what actually **View Resolver** is ?

**View Resolver** is nothing but a class named as InternalResourceViewResolver

This class has two following data members and their setter/getter:

private String prefix = "";

private String suffix = "";

These data members is not directly present in this class rather they are coming from its parent class.

public class InternalResourceViewResolver extends UrlBasedViewResolver

So, in reality the prefix and suffix with their getter/setter is present in UrlBasedViewResolver class.

**What these prefix and suffix is ?**

So far, we were returning the complete path from our Controller as below.

*@Controller*

*@RequestMapping*("mother")

public class MomController {

*@RequestMapping*("/sugar")

public String giveSugar() {

return "/WEB-INF/views/sugar.jsp";

}

}

Now, we are going to break this view location path in 3 different part.

Prefix + View Name + Suffix

Where,

* Prefix : /WEB-INF/views/
* View Name : sugar
* Suffix : .jsp

Spring said, you as a programmer just return View Name from your controller.

*@Controller*

*@RequestMapping*("mother")

public class MomController {

*@RequestMapping*("/sugar")

public String giveSugar() {

return "sugar";

}

}

But this is not the mean that we don’t need to tell about actual location and actual extension of the view.

We need to create **InternalResouceViewResolver** class object by setting up the prefix containing view location and suffix containing view extension. At last, we also have to put this object inside IOC container so that spring can use that object for resolving the view.

Creating object and putting them into IOC container, we have already learnt to do this kind stuff in spring core.

Now, it’s time to make an entry of bean tag into Front Controller Configuration File i.e. **config.xml**

<**bean** class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

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

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

</**bean**>

This is the only thing we need to do, now whenever spring found that we are returning a view name but not the complete path. It will always refer to **ViewResovler** which is kept inside IOC container and ask for suffix and prefix.

Now spring has all three parts of the path i.e.

**Prefix + View Name + Suffix**

If you attempt all the steps carefully, your application should run successfully.

**Question :**

Now, a valid question can come in your mind that why we studying about **InternalResouceViewResolver** class while **UrlBasedViewResolver** has **prefix** or **suffix.** Why don’t we create object of **UrlBasedViewResolver**.

Answer is very simple, it is because **InternalResouceViewResolver** has some extra feature with all the feature of **UrlBasedViewResolver.** Another reason is that, apart from **prefix** and **suffix** **UrlBasedViewResolver** has some extra required data members that need to be unnecessarily set through <property> tag for making object of **UrlBasedViewResolver.**

**Point To Be Remember :**

* We can make only one ViewResovler object, that’s why it is mandatory to have all the view with same extension and same directory.