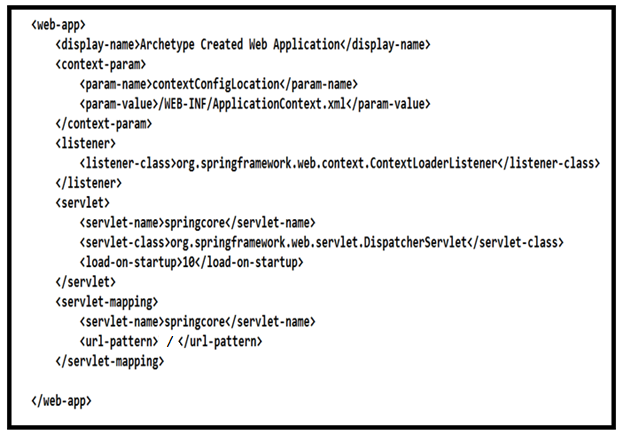


When we deploy the project at that time only the servletContext object is created then at that time event is raised, then server will check any listener Interface is listen that event or not ,if any listener is listen that event then server will create the object (here listener is contextLoaderListener) or else it will keep quite. So here contextLoaderListener is available so server will create the object and in that contextLoaderListener we have init () method, in that init () we will create the parent container object(webApplicationContextRoot) and we will transfer that parent container object into servletContext object.

Why we will transfer this parent container object into ServletContext object?

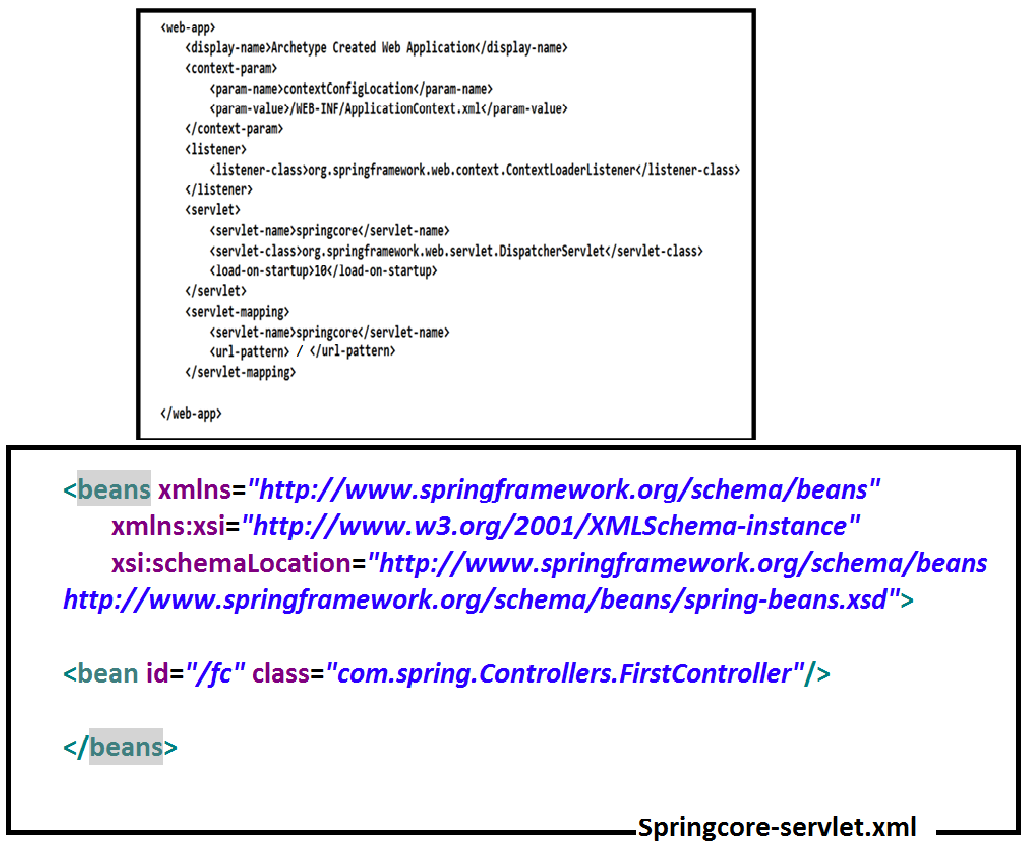
Because the servletContext can be used by total entire Application so by this child containers also used this parent container and used their beans.

And then DispatcherServlet object is created and in that dispatcher servlet we have init () method, in that init () method we will create the Child container object.



When we are configuring the dispatcherServlet SpringBeanconfiguration file, the <servlet-name> in web.xml and springBeanConfiguration file name must be same. And when we are saving name of springBeanConfiguration it should same name as <servlet-name> tag in web.xml and we should add extra thing (-servlet to that name of the springBeanConfiguration file) is like this

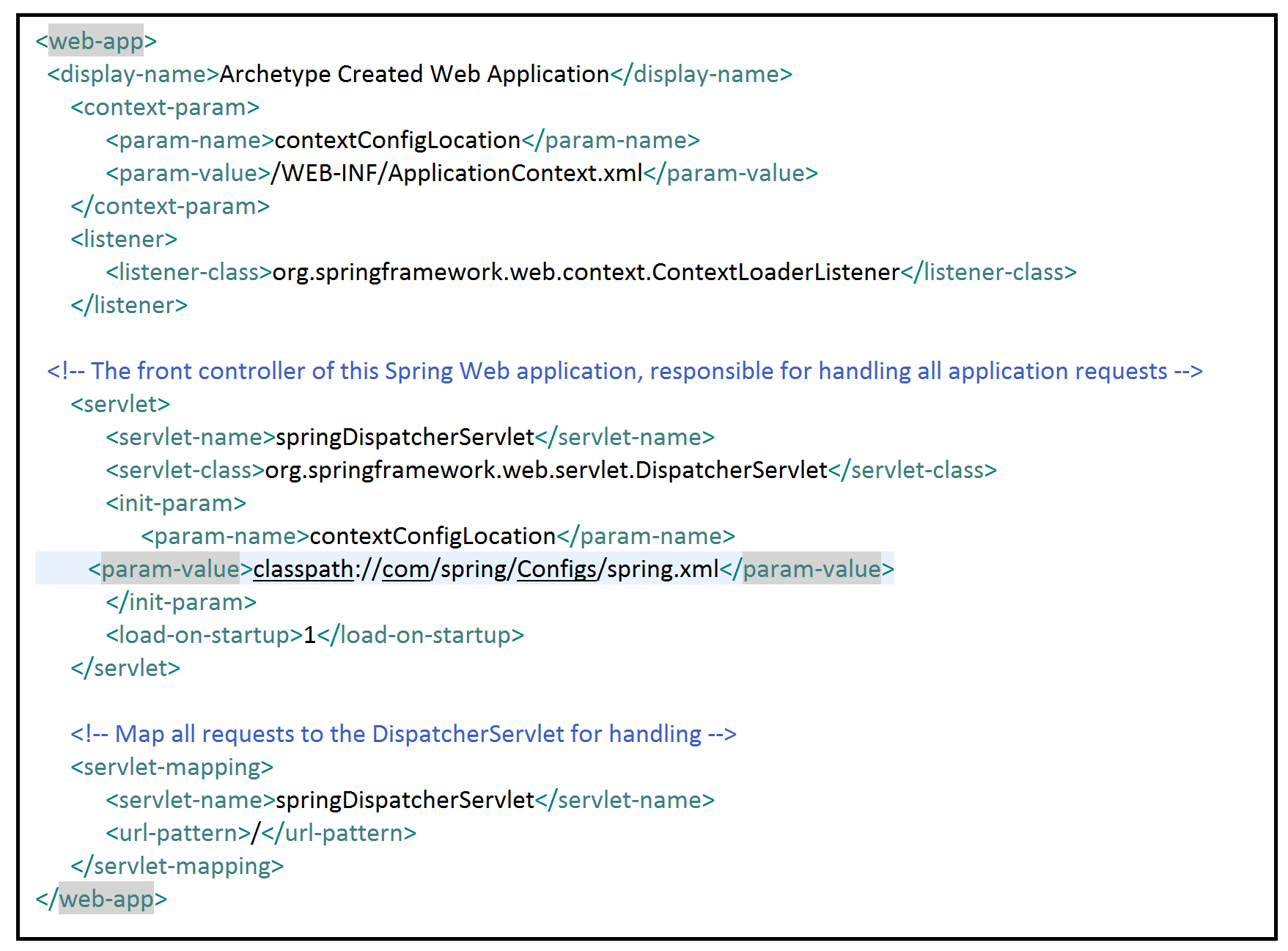
Springcore-servlet.xml. by default Container will search the SpringBeanConfiguration file in Web-INF directory with the name of the <servlet-name> with –servlet (springcore-servlet.xml). If you not specify –servlet the container unable to find your springBeanConfiguration file. Then you get the exception.



And if you want to change the Servlet name or location, you should must specify location of that springBeanConfiguration file.

If you giving the location of springBeanConfiguration file you don’t need to specify –servlet to the springBeanConfiguration file name. Directly u can give any name, then it is not guarantee for matching with the name of <servlet-name> tag in web.xml.

How to configure the SpringBeanConfiguration File now.



When it is resources directory



How we create the child container object in

DispatcherServlet init () method

In the init () method internally Servlet container will find the name of the dispatcherServlet (as u specified in <servlet-name> tag and generate complete xml name and store in a variable where file like (springcore-servlet.xml)

Example:



All the servlet objects and init () methods are executed whenever we deployed the project into the server. Because we added <load-on-startup> in the web.xml file. So without user sending a request all the servlet objects and init () will executed(in the springconfiguration files we configured dao,service,contollers classes ,all the beans will be executed at the deployment time only…if we have any object is dependent on another object at deployment time we will get exceptions if dependent object not found.)

When we deployed the project with <load-on-startup> into the server.

Then server container will open the web.xml file and read the total web.xml file using parsers and loads into the JVM Memory.

And then it will create ServletContext object and Listeners objects (in the ContextLoaderListener class we have init () method.in that init () method, we will create parent container object and transfer that container object and stored into the ServletContext class.)

So contextLoaderListener object is created and init () method also executed.

It will create DispatcherServlet Object and next it will executed init () method.in that init () method it will create child container object.

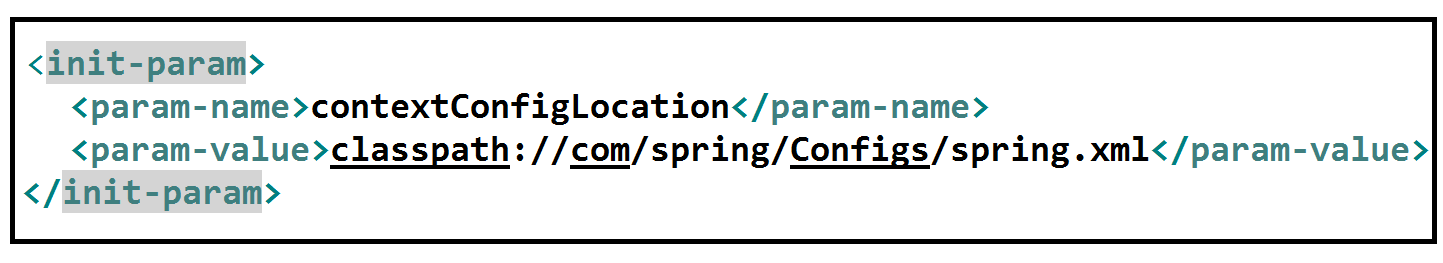
By default, server container will checks for DispactcherServlet SpringBeanConfiguration file in WEB-INF directory with name we specified in <servlet-name> tag in web.xml file (springcore-servlet.xml).

And in that DispatcherServlet Init () method it will find the name of the springBeanConfigurationFile

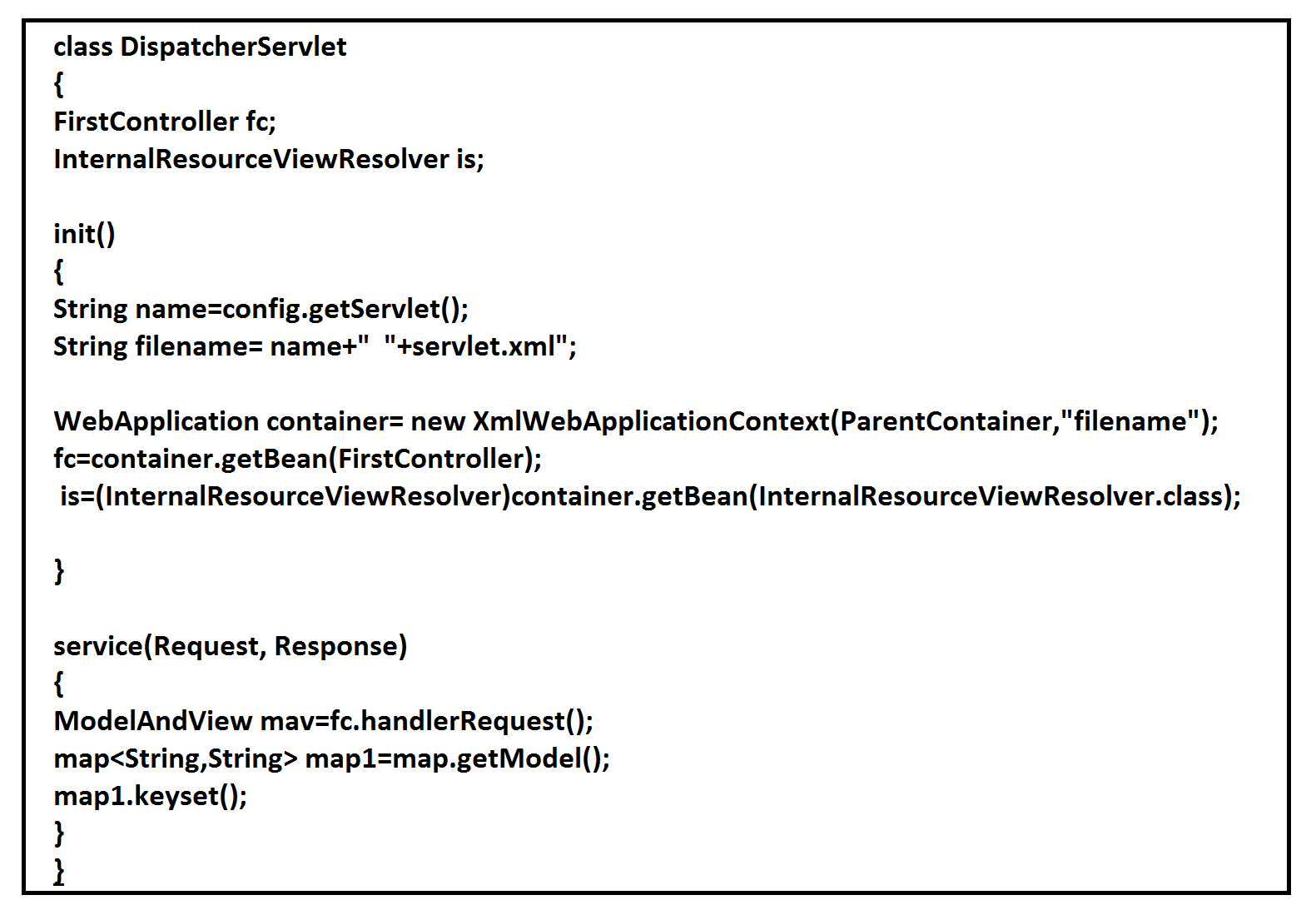
(Or)

By using <init-param> directly it will take the springBeanConfiguration file name like below example.

And then it will check for the location of the springBeanConfiguration file with in the <init-param> tag in web.xml file.



Then it find the name of the servlet and add –servlet at the ending and find the child container name and then it will take the parent container and pass to the child Container WebApplicationContext like this.



And server container will load that DispatcherServlet SpringBeanConfiguartion file using parsers and it will create beans

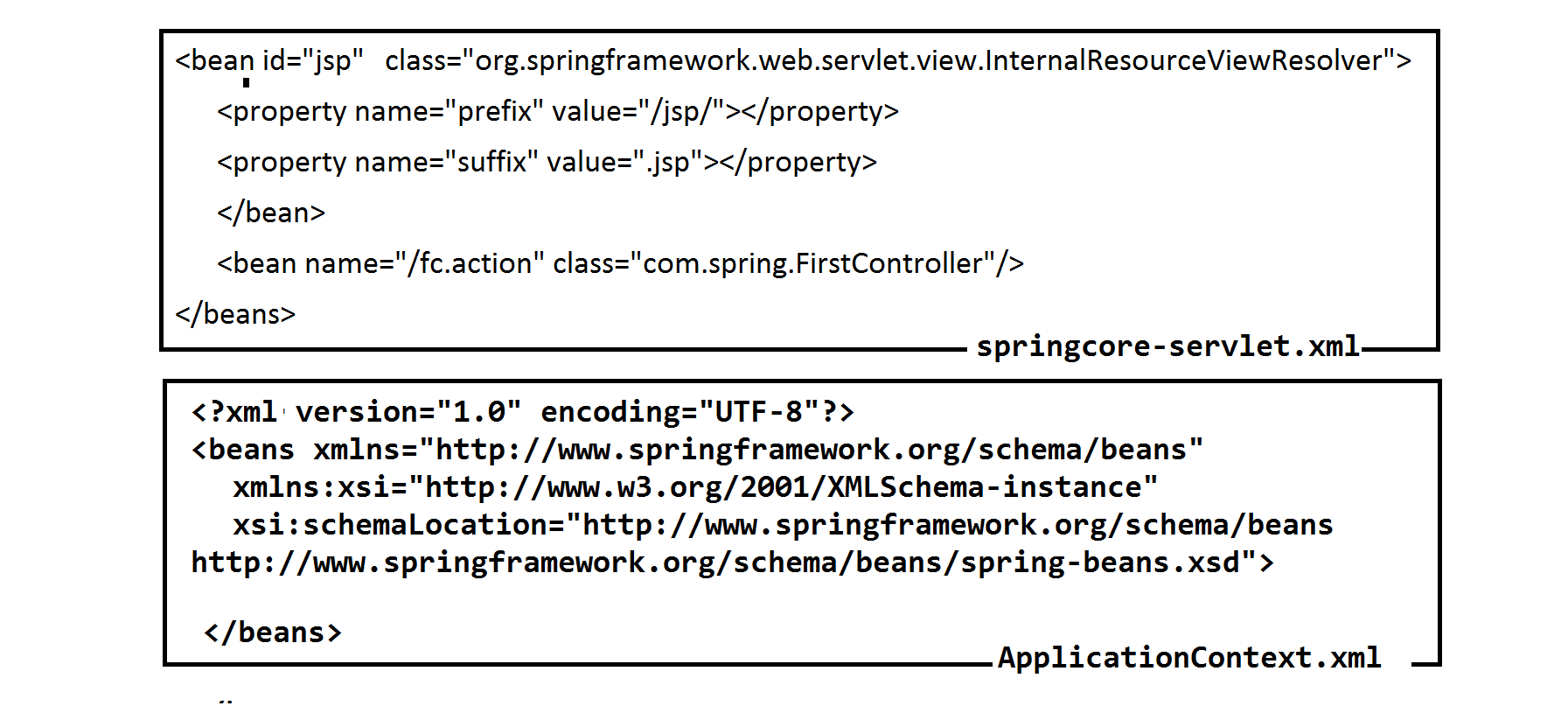
Available in dispatcherServlet springBeanConfiguration file.

And at the time of creating beans (which scope is singleton) objects and also execute life cycle methods of these beans.

Note: suppose if the bean scope is “prototype” container will rejected to create the objects of that beans.

Because WebApplicationContext is eager container because it is directly extends ApplicationContext Interface, When the Container object is created all the Singleton Scope Object will created without calling getBean().

And Now all the Servlet Objects created and all init () method are executed.

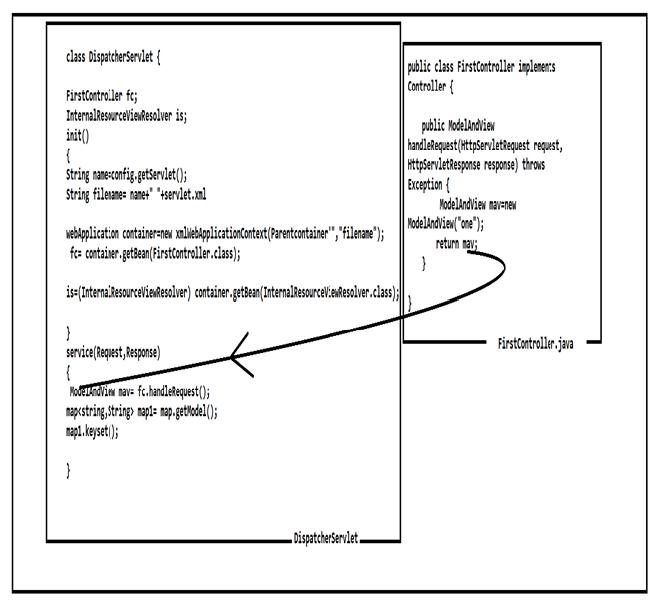


When the user send a request from the browser the url like (http://localhost:2015/SpringWebMvcXml/fc) to server ,the server will receive the request and request and response objects are created and then server container will open the web.xml file and read the total web.xml file using parsers and loads into the JVM Memory.

It will check for Url pattern ‘/fc.action’ , if it is available it will create appropriate servlet name object and (or) If it is not exact match to that url it will take the appropriate match url ‘\*.action ‘

So, we are using <load-on -startup>, already DispatcherServlet init() is executed. So, that the url is available in the request object given to the service () of DispatcherServlet, By using this request object it will create object of controller using the url available in the request object.

So, by this DispatcherServlet container will create the controller object and invoke the method handleRequest() available in the FirstContrller class is executed. This handleRequest() return the ModelAndView object and passed to service () available in the DispatcherServlet class.

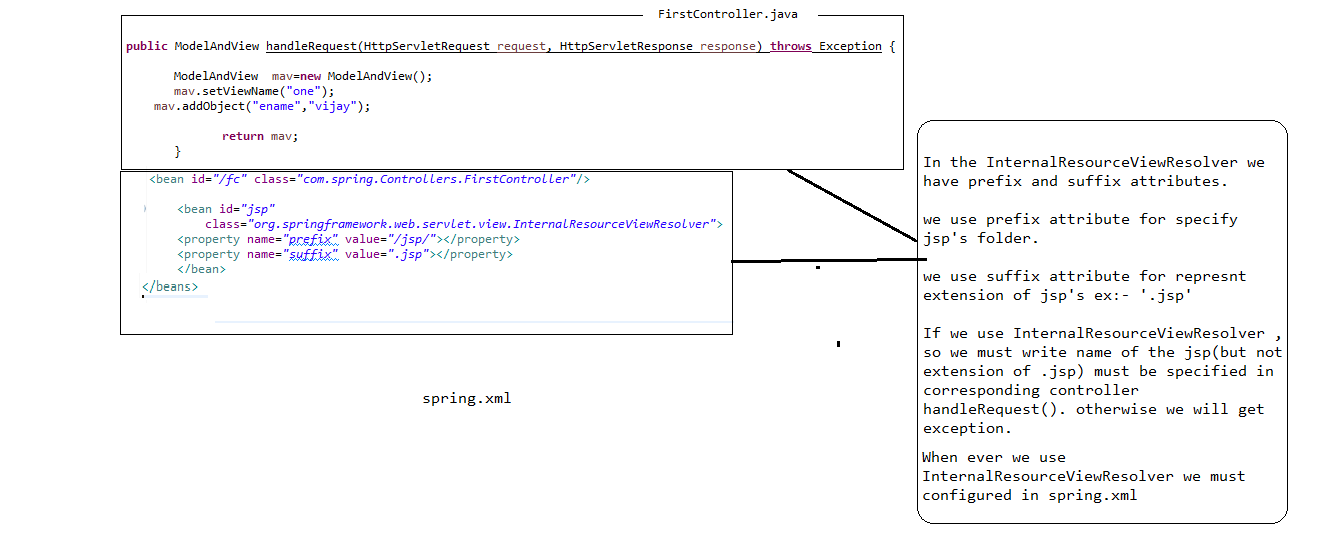


Now DispatcherServlet contains ModelAndView object.

When we store the data into ModelAndView object, it is the responsibility of the DispatcherServlet to take data from the ModelAndView object and stored into the request object.

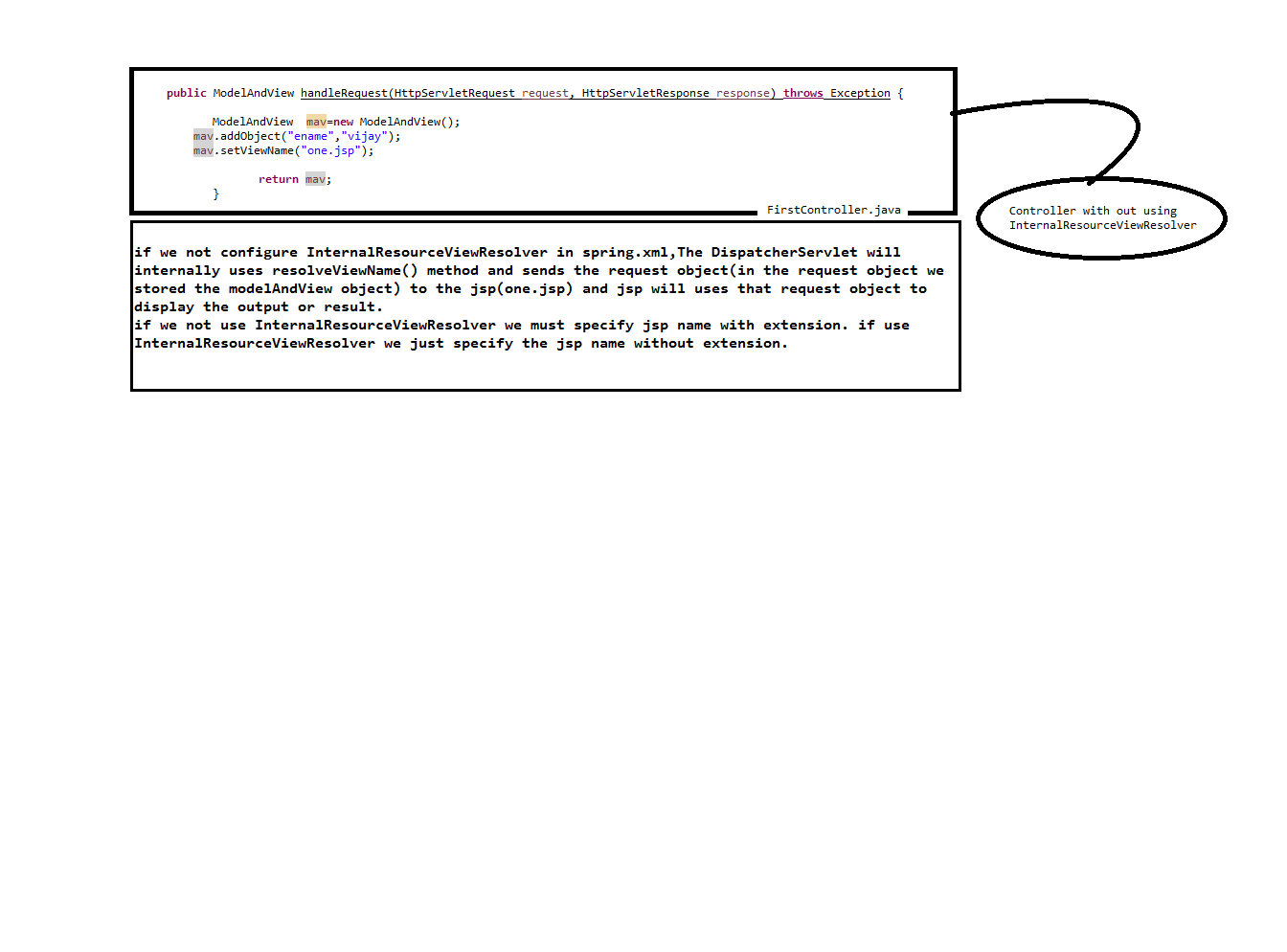
DispatcherServlet doesn’t know whom to send the ModelAndView object to which jsp so, it will take the help of InternalResourceViewResolver.

Now DispatcheServlet will give request object(in the request object we have modelAndview object) to InternalResourceViewResolver.



The InternalResourceResolver is not configured in spring.xml, DispatcherServlet internally uses this method as follows

Protected View resolveViewName (){ }



The InternalResourceviewResolver return or given the jsp address or information like jsp name (Ex:- one.jsp) to DispatcherServlet containing service().

Now finally service() will gives or pass the result information to corresponding jsp and the jsp is responsible to display the output.

The ModelAndView object contains the jsp information.

The dispatcherServlet is transfer the jsp information to InternalResourceViewResolver

The InternalResourceViewResolver is responsible is finds the jsp responsible to generate the output which jsp and where it is available and these detais sent back to dispatcherServlet

Now the Dispatcher Servlet take the help of InternalResourceViewResolver to get the jsp information to transfer the request finally the DispatcherServlet knows which jsp responsible to generate output to call that jsp.

Controller:

Controller is a interface is available in org.springframework.web.servlet.mvc

Def:-

The Class implements Controller interface is directly or indirectly is called controller class.

 Controller interface, representing a component that receivesHttpServletRequest and HttpServletResponse instances just like a HttpServlet but is able to participate in an MVC workflow.

Any implementation of the Controller interface should be a *reusable, thread-safe* class, capable of handling multiple HTTP requests throughout the lifecycle of an application. To be able to configure a Controller easily, Controller implementations are encouraged to be (and usually are) JavaBeans.

**Workflow:**

After a DispatcherServlet has received a request and has done its work to resolve locales, themes, and suchlike, it then tries to resolve a Controller, using a [HandlerMapping](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/HandlerMapping.html). When a Controller has been found to handle the request, the [handleRequest](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/mvc/Controller.html#handleRequest-javax.servlet.http.HttpServletRequest-javax.servlet.http.HttpServletResponse-) method of the located Controller will be invoked; the located Controller is then responsible for handling the actual request and — if applicable — returning an appropriate [ModelAndView](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/ModelAndView.html). So actually, this method is the main entry point for the [DispatcherServlet](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/DispatcherServlet.html)which delegates requests to controllers.

So basically any *direct* implementation of the Controller interface just handles HttpServletRequests and should return a ModelAndView, to be further interpreted by the DispatcherServlet.

Disadvantages of Controller:

1. It will not support validation and form handling …..

To Reslove this problem we use AbsractController

**AbstractContrller:**

The AbstractController is available in org.springframework.web.servlet.mvc package

AbstractController is implements Controller interface. The AbstractController has some impotants features like support validation and form handling …..

1.Supported Methods

2.require session

3. CacheSeconds

4.Synchronize on Session

public abstract class **AbstractController**

extends [WebContentGenerator](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/support/WebContentGenerator.html)

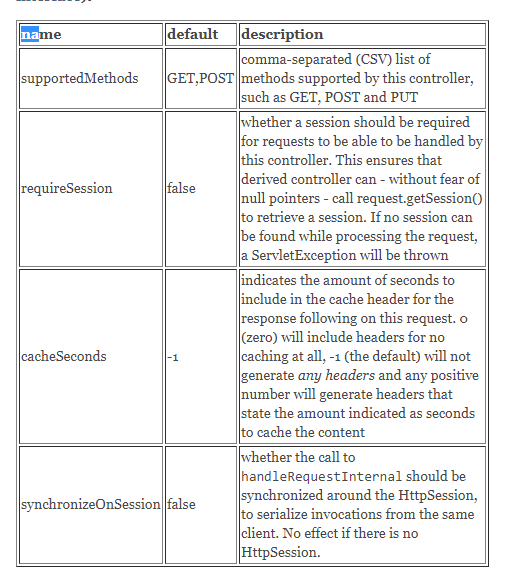
implements [Controller](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/web/servlet/mvc/Controller.html)

{

}

Advantages Of AbstractContrller:

* The Adavantages of AbstractController is as follows as…
* We no need to write the huge amount of code for checking whether the client as sent Get Request or Post Request.
* We no need to write code to check the whether session object is available or not.



Model:

The Model is object in spring web mvc and it is used for storing data or information or collection of list data.

Spring will use the Model object

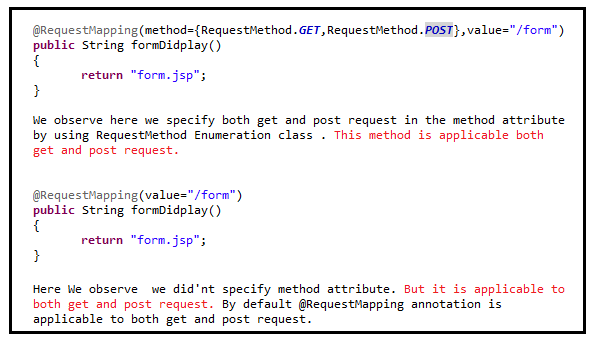
Request Mapping:

1. **@RequestMapping** is one of the most widely used **Spring MVC** annotation. org.springframework.web.bind.annotation.RequestMapping annotation is used to map web requests onto specific handler classes and/or handler methods.

As we have use @RequestMapping top of the method we no need to specify ‘ method element ‘

Here the method element is one of the attribute or element of @RequestMapping .In this method attribute we specify value are POST,GET,HEAD,PATCH…..

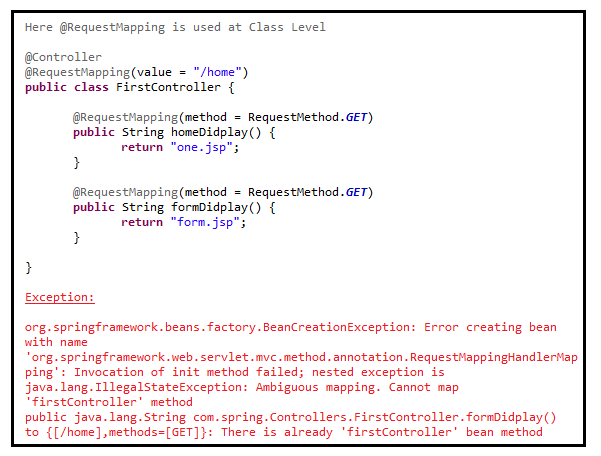
By default the @RequestMapping supports both GET and POST request.

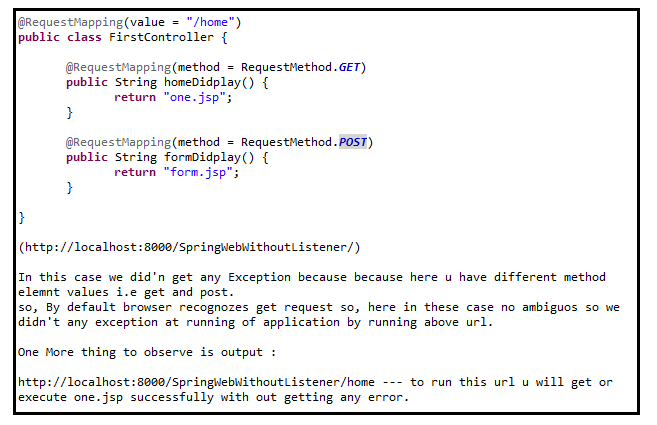


This @RequestMapping

The **@Controller** annotation is used to mark the class as the controller in Spring 3.

The **@RequestMapping** annotation is used to map the request url. It is applied on the method.





Write notes for specify get and post request separately

Difference b/w get and post methods execution difference

**class DispatcherServlet**

**{**

**FirstController fc;**

**InternalResourceViewResolver is;**

**init()**

**{**

**String name=config.getServlet();**

**String filename= name+" "+servlet.xml";**

**WebApplication container= new XmlWebApplicationContext(ParentContainer,"filename");**

**fc=container.getBean(FirstController);**

**is=(InternalResourceViewResolver)container.getBean(InternalResourceViewResolver.class);**

**}**

**service(Request, Response)**

**{**

**ModelAndView mav=fc.handlerRequest();**

**map<String,String> map1=map.getModel();**

**map1.keyset();**

**}**

**Flow of web mvc application.**

**@Controller**

When we deployed the project with <load-on-startup> into the server.

Then server container will open the web.xml file and read the total web.xml file using parsers and loads into the JVM Memory.

And then it will create ServletContext object and Listeners objects (in the ContextLoaderListener class we have init () method.in that init () method, we will create parent container object and transfer that container object and stored into the ServletContext class.)

So contextLoaderListener object is created and init () method also executed.

It will create DispatcherServlet Object and next it will executed init () method.in that init () method it will create child container object.

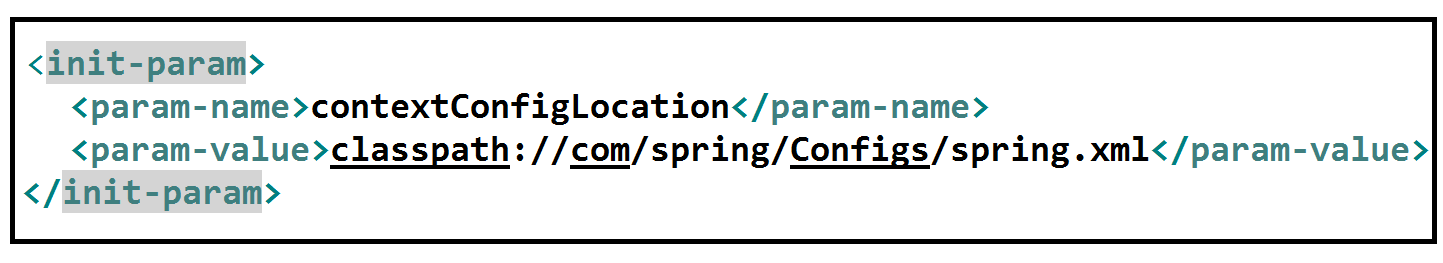
By default, server container will checks for DispactcherServlet SpringBeanConfiguration file in WEB-INF directory with name we specified in <servlet-name> tag in web.xml file (springcore-servlet.xml).

And in that DispatcherServlet Init () method it will find the name of the springBeanConfigurationFile

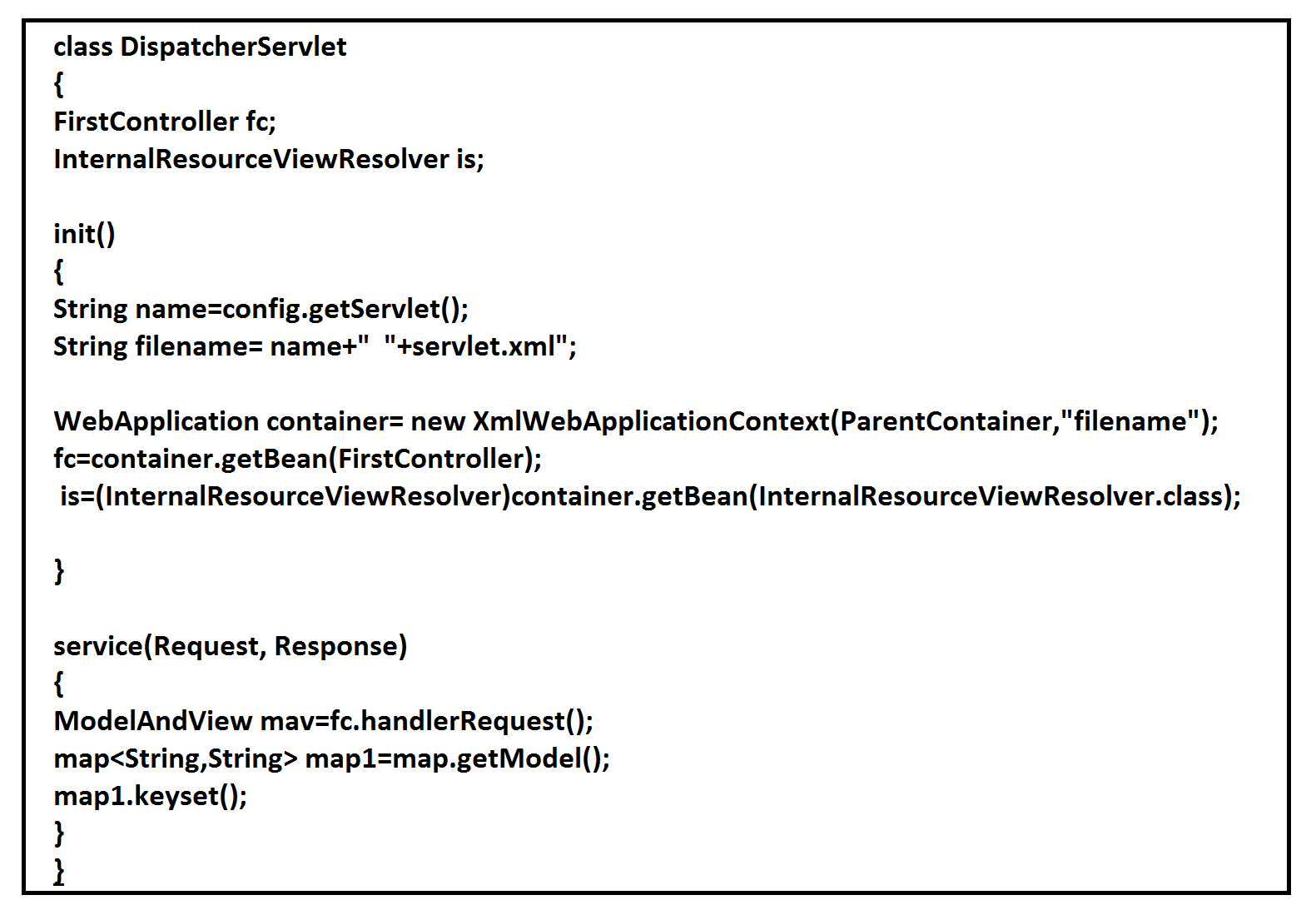
(Or)

By using <init-param> directly it will take the springBeanConfiguration file name like below example.

And then it will check for the location of the springBeanConfiguration file with in the <init-param> tag in web.xml file.



Then it find the name of the servlet and find the child container name and then it will take the parent container and pass to the child Container WebApplicationContext like this.



And server container will load that DispatcherServlet SpringBeanConfiguartion file using parsers and it will create beans

Available in dispatcherServlet springBeanConfiguration file.

And at the time of creating beans (which scope is singleton) objects and also execute life cycle methods of these beans.

Note: suppose if the bean scope is “prototype” container will rejected to create the objects of that beans.

Because WebApplicationContext is eager container because it is directly extends ApplicationContext Interface, When the Container object is created all the Singleton Scope Object will created without calling getBean().

And Now all the Servlet Objects created and all init () method are executed.

When the user send a request from the browser the url like (http://localhost:2015/SpringWebMvcXml/fc) to server ,the server will receive the request and request and response objects are created and then server container will open the web.xml file and read the total web.xml file using parsers and loads into the JVM Memory.

It will check for Url pattern ‘/fc.action’ , if it is available it will create appropriate servlet name object and (or) If it is not exact match to that url it will take the appropriate match url ‘\*.action ‘

So, we are using <load-on -startup>, already DispatcherServlet init() is executed. So, that the url is available in the request object given to the service () of DispatcherServlet, By using this request object it will create object of controller using the url available in the request object.

User send a Request(<http://192.168.1.125:8026/springwebxml/home>) to spring web mvc application then it will display Home.jsp.

How it will display:

Using RequestMapping Annoation it will hit to getHome() available in Controller take url and display appropriate jsp.

@**Bean:**

@Bean is annotation is available in org.springframework.context.annotation.

This annotation can be applied at method level and Annotation level. This annotation at executed at Runtime.