Spring web MVC

Goal

- Understand some concepts of Spring Web MVC
- Use Spring Web MVC to build a web application project.

Agenda

- Spring Web MVC
- DispatcherServlet
- Implement Controller
- Resolving Views

Spring Web MVC

What is Spring Web MVC?

- Is Framework using Spring Library.
- Is designed with Model-View-Controller (MVC) pattern
- Use *DispatcherServlet that dispatches request to handler*

(*)The default handler is based on the @Controller and @RequestMapping annotations

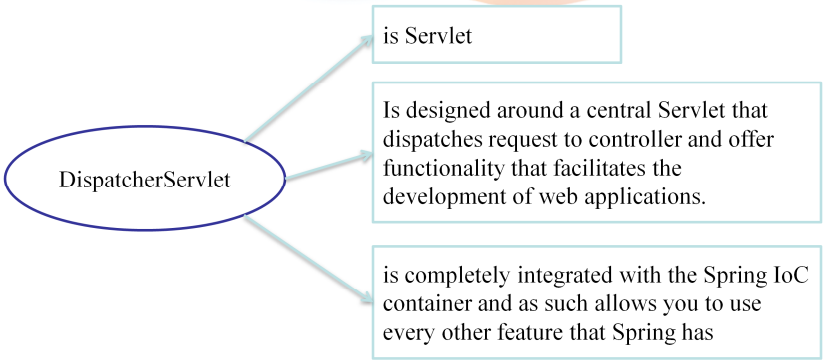
Spring Web MVC

Feature of Spring Web MVC:

- Clear separation of roles
- Powerful and straightforward configuration of both framework and application classes as JavaBeans
- Adaptability, non-intrusiveness, and flexibility.
- Reusable business code, no need for duplication
- Customizable binding and validation
- Customizable handler mapping and view resolution
- Flexible model transfer
- Customizable locale and theme resolution
- A simple yet powerful JSP tag library known as the Spring tag library

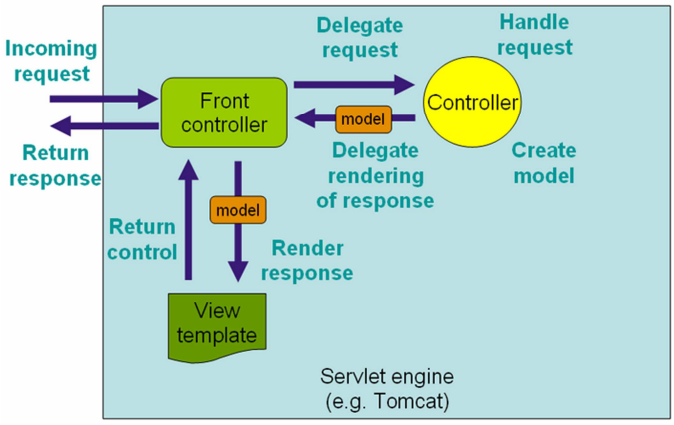
DispatcherServlet

What is DispatcherServlet:



DispatcherServlet

The request processing workflow in Spring Web MVC



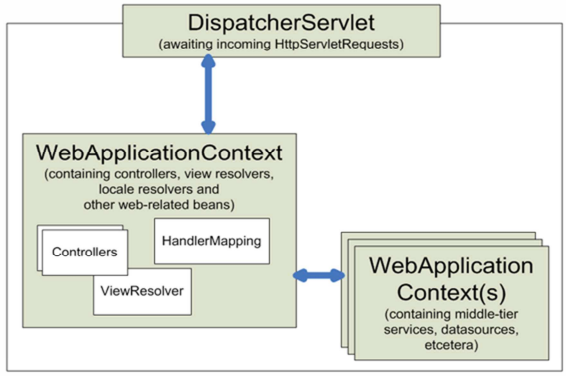
DispatcherServlet

Configuration of DispatcherServlet in web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/ja
  <display-name>SpringMVCHibernate</display-name>

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

  <servlet>
      <servlet-name>dispatcher</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
      <servlet-name>dispatcher</servlet-name>
      <url-pattern>*.do</url-pattern>
  </servlet-mapping>

  <listener>
      <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:applicationContext.xml</param-value>
  </context-param>

</web-app>
```

DispatcherServlet

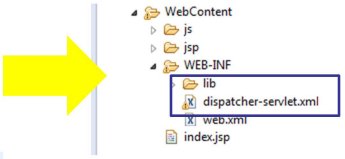
In the Web MVC framework, each **DispatcherServlet** has its own **WebApplicationContext**, which inherits all the beans already defined in the root **WebApplicationContext**.



DispatcherServlet

Upon initialization of a DispatcherServlet, Spring MVC looks for a file named ***[servlet-name]-servlet.xml*** *in the WEB-INF directory of your web application and creates the beans defined there,*overriding the definitions of any beans defined with the same name in the global scope.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/ja
  <display-name>SpringMVCHibernate</display-name>

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

  <servlet>
      <servlet-name>dispatcher</servlet-name>
      <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
      <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
      <servlet-name>dispatcher</servlet-name>
      <url-pattern>*.do</url-pattern>
  </servlet-mapping>

  <listener>
      <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>classpath:applicationContext.xml</param-value>
  </context-param>

</web-app>
```



web.xml

DispatcherServlet

Special bean types in the WebApplicationContext

Bean type	Explanation
HandlerMapping	Maps incoming requests to handlers and a list of pre- and post-processors (handler interceptors) based on some criteria the details of which vary by HandlerMapping implementation.
HandlerAdapter	Helps the DispatcherServlet to invoke a handler mapped to a request regardless of the handler is actually invoked. For example, invoking an annotated controller requires resolving various annotations
HandlerExceptionResolver	Msoalpves rexceptions to views also allowing for more complex exception handling code.
ViewResolver	Resolves logical String-based view names to actual View types.

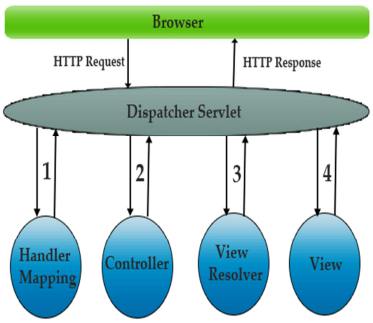
DispatcherServlet

Special bean types in the WebApplicationContext

Bean type	Explanation
LocaleResolver	Resolves the locale a client is using, in order to be able to offer internationalized Views
ThemeResolver	Resolves themes your web application can use, for example, to offer personalized Layouts
MultipartResolver	Parses multi-part requests for example to support processing file uploads from HTML forms.
FlashMapManager	Stores and retrieves the "input" and the "output" FlashMap that can be used to pass attributes from one request to another, usually across a redirect.

DispatcherServlet

DispatcherServlet Processing Sequence



- After receiving an HTTP request, ***DispatcherServlet*** consults the ***HandlerMapping*** to call the appropriate ***Controller***.
- The ***Controller*** takes the request and calls the appropriate service methods based on used GET or POST method. The service method will set model data based on defined business logic and returns view name to the ***DispatcherServlet***.
- The ***DispatcherServlet*** will take help from ***ViewResolver*** to pickup the defined view for the request.
- Once view is finalized, The ***DispatcherServlet*** passes the model data to the view which is finally rendered on the browser.

Implementing Controllers

Defining a controller with @Controller

- The @Controller annotation indicates that a particular class serves the role of a *controller.*
- The @Controller annotation acts as a stereotype for the annotated class, indicating its role. The dispatcher scans such annotated classes for mapped methods and detects @RequestMapping annotations

```
@Controller
public class ContactsControllers
{
    @Autowired
    private ContactsService contactsService;

    @Autowired
    private ContactFormValidator validator;

    @RequestMapping("/home")
    public String home()
    {
        return "home";
    }

    @InitBinder
    public void initBinder(WebDataBinder binder)
    {
        SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy");
        dateFormat.setLenient(false);
        binder.registerCustomEditor(Date.class, new CustomDateEditor(dateFormat, true));
    }
```


14

Implementing Controllers

Mapping Requests With @RequestMapping

- Use the @RequestMapping annotation to map URLs such as /appointments onto an entire class or a particular handler method

```
@RequestMapping("/searchContacts")
public ModelAndView searchContacts(@RequestParam(required= false, defaultValue="") String name)
{
    ModelAndView mav = new ModelAndView("showContacts");
    List<Contact> contacts = contactsService.searchContacts(name.trim());
    mav.addObject("SEARCH_CONTACTS_RESULTS_KEY", contacts);
    return mav;
}
```


15

Resolving views

There are 2 interfaces that are important to the way Spring handle views: *ViewResolver and View*

- The ViewResolver provides a mapping between view names and actual views
- The View interface addresses the preparation of the request and hands the request over to one of the view technologies

Resolving views

Resolving views with the ViewResolver interface

Spring Web MVC controllers resolves logical view name either explicitly (e.g., by returning a String, View, or ModelAndView) or implicitly (i.e., based on conventions). In Spring, 'logical view name' represents Views and it is resolved by a view resolver

Resolving views

Resolving views with the ViewResolver interface

There are a few view resolver in Spring.Given below the list of view resolvers in Spring:

ViewResolver	Description
AbstractCachingViewResolver	This view resolver caches views.
XmlViewResolver	This view resolver uses configuration file written in XML for view resolution.
ResourceBundleViewResolver	This view resolver use ResourceBundle, represented by bundle base name, to resolve view. Generally bundle is defined in a properties file, situated in the classpath.
UrlBaseViewResolver	This view resolver uses "logical view name" returned to find actual view.
InternalResourceViewResolver	This view resolver is the subclass of UrlBasedViewResolver and also support InternalResourceView and also subclass such as JstlView and TilesView.

18

Resolving views

Resolving views with the ViewResolver interface

ViewResolver	Description
VelocityViewResolver/ FreeMarkerViewResolver	This view resolver is the subclass of UrlBasedViewResolver which supports VelocityView, FreeMarkerView and its custom subclass.
ContentNegotiatingViewResolver	This view resolver is the implementation of the ViewResolver interface which resolves view on the basis of request file name or Accept header

19

Resolving views

Resolving views with the ViewResolver interface

Some Example Configurations:

(*) Configuration for UrlBaseViewResolver

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.UrlBasedViewResolver">
    <property name="viewClass" value="org.springframework.web.servlet.view.JstlView"/>
    <property name="prefix" value="/WEB-INF/jsp/"/>
    <property name="suffix" value=".jsp"/>
</bean
```

This view resolver add the prefix, suffix to the return “logical view name” to find actual view.

Example: if “loginpage” is returned then the actual view will be *"/WEB-INF/jsp/loginpage.jsp”*

Resolving views

Resolving views with the ViewResolver interface

Some Example Configurations:

(*) Configuration for ResourceBundleViewResolver

```
<bean id="viewResolver" class="org.springframework.web.servlet.view.
ResourceBundleViewResolver">
    <property name="basename" value="views"/>
    <property name="defaultParentView" value="parentView"/>
</bean
```

The *ResourceBundle* represented by the basename is inspected by the *ResourceBundleViewResolver* to resolve view and 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.

21

Reference

- http://www.springsource.org/documentation

Q&A

23