Spring MVC

Agenda

- What is and Why Spring MVC?
- Request life-cycle
- DispatcherServlet
- URL Handler mapping
- Controllers
- View & View Resolvers
- Validation

What is and Why Spring MVC?

What is Spring MVC?

- Web application framework that takes advantage of Spring design principles
  - Dependency Injection
  - Interface-driven design
  - POJO without being tied up with a framework

Why Spring MVC?

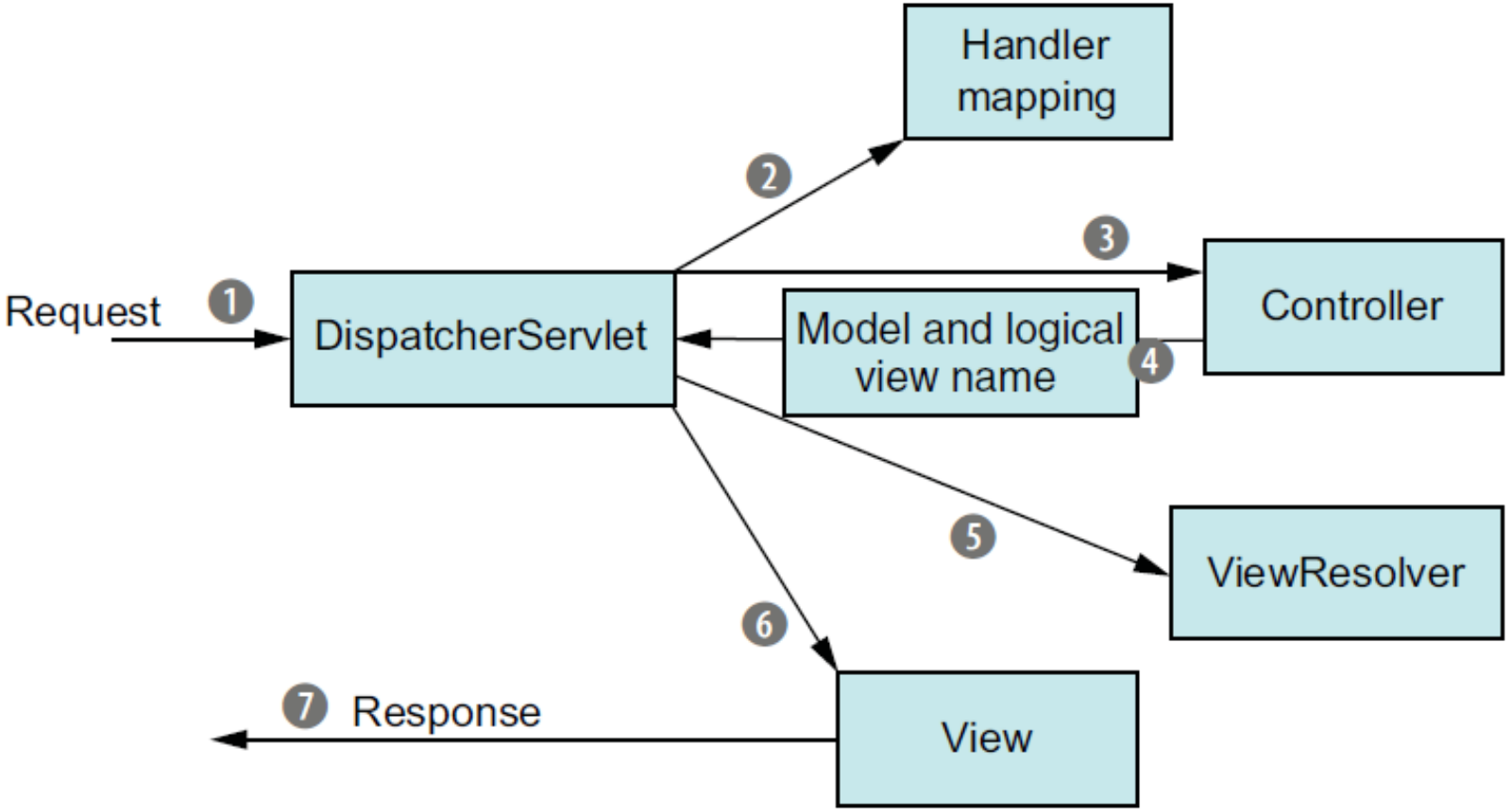
- Testing through Dependency Injection
- Binding of request data to domain objects
- Form validation
- Error handling
- Multiple view technologies
  - JSP, Velocity, Excel, PDF
- Page workflow

Request Life-cycle

Request Life-cycle

- *DispatchServlet* receives the HTTP request
  - The Front controller of Spring MVC
- URL Handler mapping
  - Controller is invoked
  - Controller returns *ModelAndView* object
- *ViewResolver* selects a view
  - Maps logical views to physical path for a view to be rendered.

Request Life-cycle



DispatcherServlet

The DispatcherServlet

- The front controller of Spring MVC
- Responsible to initialize Spring WebApplicationContext
- An application can use any number of DispatcherServlet(s)
- Each DispatcherServlet will create its own Context either from XML or Annotation Driven Configuration.

Spring MVC at Glance

1 Write the controller class that performs the logic behind the homepage.

2 Configure the controller in the DispatcherServlet’s context configuration file

3 Configure a view resolver to tie the controller to the JSP.

4 Write the JSP that will render the homepage to the user

Controllers

Controller

- A Spring component which processes request.
- Invoked by the DispatcherServlet.
- DispatcherServlet consults the HandlerMapping configuration to locate the Controller.
- The controller after processing the request sends the DispatcherServlet some information in the form of Model.
- It also sends the DispatcherServlet the logical view name.

A Typical Controller

```
@Controller
public class LoginController {

        @RequestMapping(value = "/login.do", method = RequestMethod.POST)
        public   String login(@RequestParam("username") String user,
                              @RequestParam("password") String password) {

          if (user.equals(password)) {
                    return "success";
             } else {
                     return "failure";
          }
      }

}
```

Spring MVC

Setup

Configuring DispatcherServlet

- Conventionally the DispatcherServlet is configured in web.xml (till Servlet Specification 2.5)
- With Servlet 3.0, the DispatcherServlet can now be configured using Java Configuration.
- In Servlet 3.0 based Web Containers, Spring Provides org.springframework.web.servlet.support.

AbstractAnnotationConfigDispatcherServletInitializer.

- We need to write an Initializer class which must extend the above class.

Configuring DispatcherServlet (web.xml)

```
<servlet>
    <servlet-name>springapp</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet
                                                    </servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
……..
<servlet-mapping>
        <servlet-name>springapp</servlet-name>
            <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

Loding More than One Context File

- By default Dispatcher servlet will load only one context configuration file
- For additional context configuration file you need to configure 'Context Loader

```
<listener>
    <listener-class>
        org.springframework.web.context.ContextLoaderListener
    </listener-class>
</listener>
<context-param>
<param-name>contextConfigLocation</param-name>
<param-value> /WEB-INF/app-data.xml
              /WEB-INF/app-security.xml
</param-value>
</context-param>
```

Configuring DispatcherServlet (Java Config)

```
public class MyWebAppInitializer
extends
AbstractAnnotationConfigDispatcherServletInitializer {
@Override
protected String[] getServletMappings() {
return new String[] { "/" };
}
@Override
protected Class<?>[] getRootConfigClasses() {
return new Class<?>[] { RootConfig.class };
}
@Override
protected Class<?>[] getServletConfigClasses() {
return new Class<?>[] { WebConfig.class };
}
};
```

DispatcherServlet Url patterns

Root Config classes

DispatcherServlet Config classes

Enable MVC in Spring

- MVC can be enabled in two ways
- XML Configuration

<mvc:annotation-driven>

OR Java Configuration

```
@Configuration
@EnableWebMvc
public class WebConfig {

//Other Bean
configurations

}
```

View & View Resolvers

View

- Renders the output of the request to the client
- Implements the *View* interface
- Built-in support for
  - JSP, XSLT, Velocity, Freemaker
  - Excel, PDF, JasperReports

View Resolvers

- Resolves logical view names returned from controllers into *View* objects
- Implements *ViewResolver* interface
  - *View resolveViewName(String viewName, Locale locale) throws Exception*
- Spring provides several implementations
  - *InternalResourceViewResolver*
  - *BeanNameViewResolver*
  - *ResourceBundleViewResolver*
  - *XmlViewResolver*

ResourceBundleViewResolver

- The View definitions are kept in a separate configuration file
  - You do not have to configure view beans in the application context file
- Supports internationalization (I18N)

ResourceBundleViewResolver

```
<!-- This bean provides explicit View mappings in a resource bundle
instead of the default InternalResourceViewResolver. It fetches
the view mappings from localized "views_xx" classpath files, i.e.
"/WEB-INF/classes/views.properties" or "/WEBINF/
classes/views_de.properties". Symbolic view names returned by
Controllers will be resolved by this bean using the respective
properties file, which defines arbitrary mappings between view
names and resources. -->
<bean id="viewResolver"
      class="org.springframework.web.servlet.view.ResourceBundleViewRes
olver">
      <property name="basename" value="views"/>
</bean>
```

Example: views.properties

```
# This is from petclinic sample application

welcomeView.(class)=org.springframework.web.servlet.view
    .JstlView

welcomeView.url=/WEB-INF/jsp/welcome.jsp

vetsView.(class)=org.springframework.web.servlet.view.Jstl
    View

vetsView.url=/WEB-INF/jsp/vets.jsp

# A lot more are defined
```

Validation