Lesson 3: Spring MVC framework

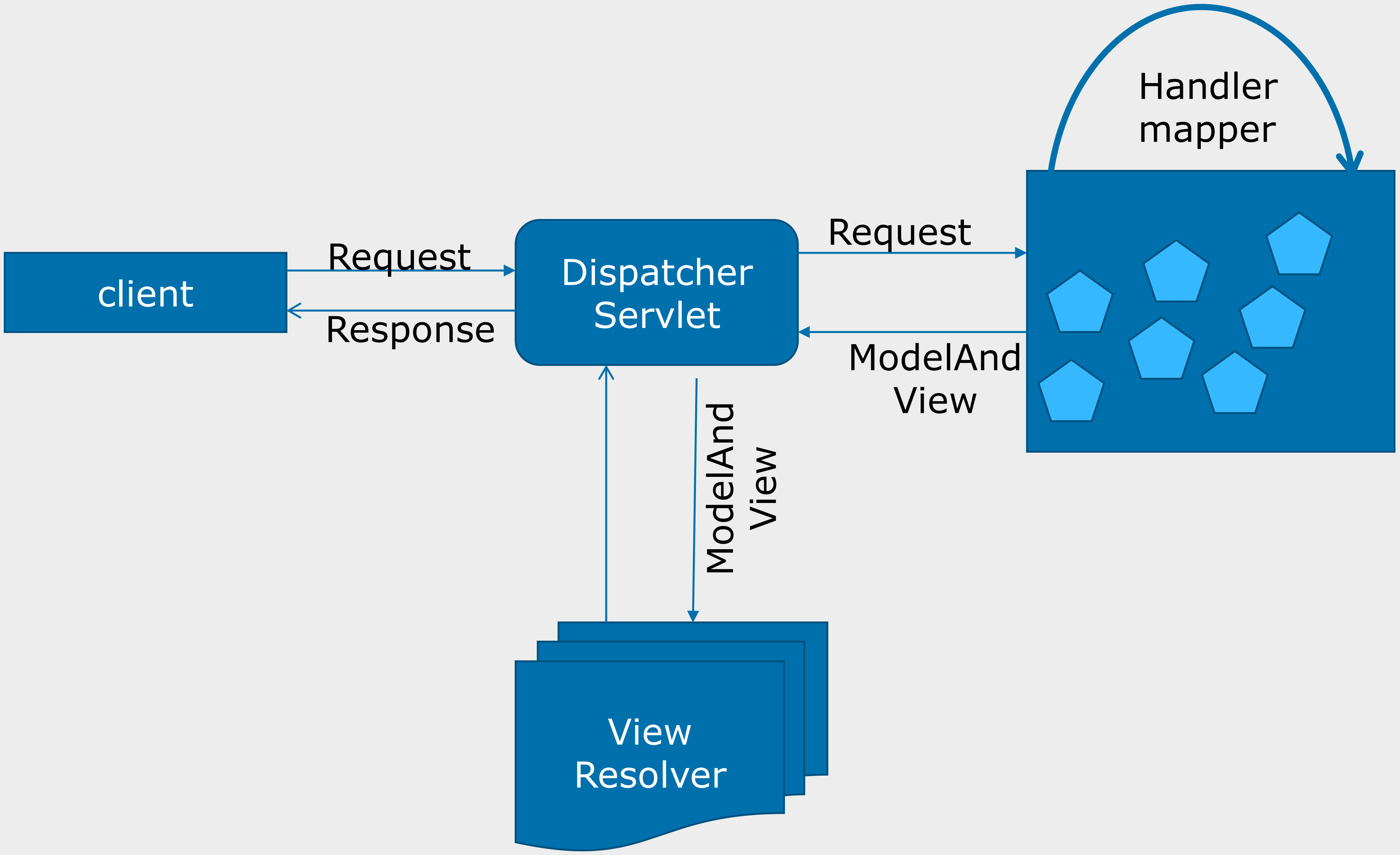
Basic Spring 5.0

Lesson Objectives

3.1:Spring MVC introduction

- MVC design pattern
- Dispatcher Servlet – Front Controller
- Controllers
- Request Handler
- ModelAndView
- ViewResolver

3.1 : Spring MVC Architecture



3.1 : Configuring DispatcherServlet in web.xml

```
<servlet>
    <servlet-name>basicspring</servlet-name>
    <servlet-class> org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>basicspring</servlet-name>
    <url-pattern>*.obj</url-pattern>
  </servlet-mapping>
```

The servlet-name given to the servlet is significant

3.1 : WebApplicationContext

```
<listener>
   <listener-class>
               org.springframework.web.context.ContextLoaderListener
   </listener-class>
</listener>
```

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
         /WEB-INF/basicspring-service.xml
        /WEB-INF/basicspring-data.xml
    </param-value>
</context-param>
```

3.2 Annotation-based configuration – Controller

```
@Controller
 public class HelloController {
     @RequestMapping("/helloWorld")
     public String showMessage() {
                 return "hello";
}}
```

3.2:Annotation-based controller configuration

- @Controller
- @RequestMapping
- @RequestParam
- @ModelAttribute

3.2.1 Handler Mapping

- Handler mapping bean in the **WebApplicationContext** that implements the HandlerMapping interface.
- Map a request to a handler according to the request’s URL.
- Use **@RequestMapping** annotation to identify the services in controller.

```
@Controller
public class MyController{

        @RequestMapping(“/”)
        public ModelAndView sayHello(){
                return new ModelAndView("hello",’msg","Hello World");
        }

}
```

3.2.1 ModelAndView

```
new ModelAndView(“viewName",“modelObjectName",”modelObject”);
```

```
Map myModel = new HashMap();
myModel.put("now",now);
myModel.put("products",getProductManager().getProducts());
return new ModelAndView("product","model",myModel);
```

3.2.1 Building a basic Spring MVC application - ViewResolver

- DispatcherServlet receives a model and a view name, it will resolve the logical view name into a view object for rendering.
- DispatcherServlet resolves views from one or more view resolvers.
- A view resolver is a bean configured in the **WebApplicationContext** that implements the **ViewResolver** interface.
- Its responsibility is to return a view object for a logical view name.

3.2.1 Building a basic Spring MVC application - Resolving Views: The ViewResolver

- **InternalResourceViewresolver**:

  Resolves logical view names into View objects that are rendered using template file resources

- **BeanNameViewResolver**:

  Looks up implementations of the View interface as beans in the Spring context, assuming that the bean name is the logical view name

- **ResourceBundleViewResolver**

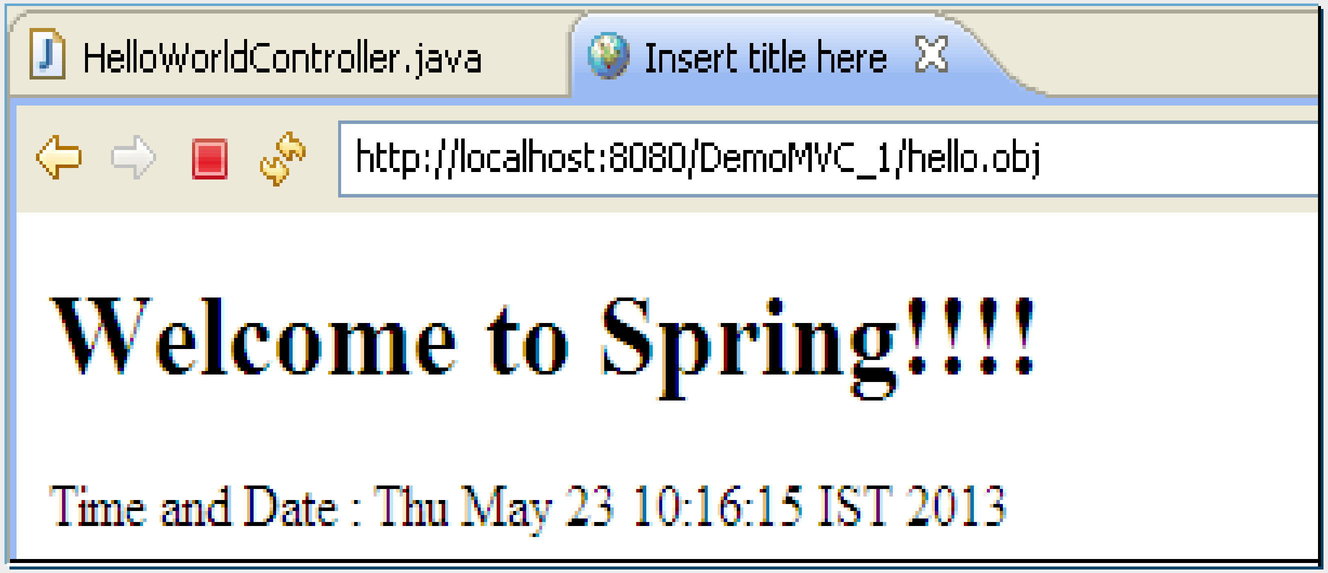
  Uses a resource bundle that maps logical view names to implementations of the View interface

- **XmlViewResolver**

  Resolves View beans from an XML file that is defined separately from the application context definition files

Demo

- Refer DemoMVC_1


HelloWorldController.java
Insert title here
http://localhost:8080/DemoMVC_1/hello.obj

Welcome to Spring!!!!

Time and Date : Thu May 23 10:16:15 IST 2013

Demo

- Refer DemoMVC_2 application

3.2.2: Spring MVC annotations -Validating input with Bean Validation

- Bean Validation (JSR – 303) Annotations:

Annotation Name	Description
Annotations for validation	
@Valid	To trigger validation of a @Controller input
@Size	Validates that the fields meet criteria on their length.
@NotNull	Validates that the fields contains value.
@Pattern	@Pattern annotation along with a regular expression ensures that the entered value is valid
@Email	Validates that the field value is a valid emailid.
@DateTimeFormat	In Spring New Date & Time API can be used in Controllers for Form Binding

3.2.2: Spring MVC annotations -Validating input : declaring validation rules

```
public class User {
     @Size(min = 3, max = 20, message = "Username must be between 3 and 20
characters long.")
     @Pattern(regexp = "^[a-zA-Z0-9]+$", message = "Username must be
alphanumeric with no spaces")
     private String username;

     @Size(min = 6, max = 20, message = "The password must be at least 6
characters long.")
      private String password;

      @Pattern(regexp = "[A-Za-z0-9]+@[A-Za-z0-9.-]+[.][A-Za-z]{2,4}",
message = "Invalid email address.")
      private String email;

//getter and setter methods for all these properties
}
```

3.2.2 : Spring MVC annotations -Processing forms : The JSP

addUser.jsp

```
<%@ taglib prefix="sf" uri="http://www.springframework.org/tags/form"%>
<sf:form method="POST" modelAttribute="user" >
<table cellspacing="0">
<tr>
<th><sf:label path="username">Username:</sf:label></th>
<td><sf:input path="username" size="15" maxlength="15" />
       <small id="username_msg">No spaces, please.</small><br />
       <sf:errors path="username" /></td>
</tr>
<tr>
        <th><sf:label path="password">Password:</sf:label></th>
        <td><sf:password path="password" size="30" showPassword="true"/>
            <small>6 characters or more (be tricky!)</small><br/>
            <sf:errors path="password" />
           </td>
     </tr>
<tr><th></th>
<td><input name="commit" type="submit" value="Save User" /></td></tr>
</sf:form></div>
```

3.2.2 : Spring MVC annotations

- Displaying validation errors

jsp

```
<td>
    <sf:password path="password" size="30" 
showPassword="true"/>
            <small>6 characters or more (be 
tricky!)</small><br/>
            <sf:errors path="password" />
</td>
```

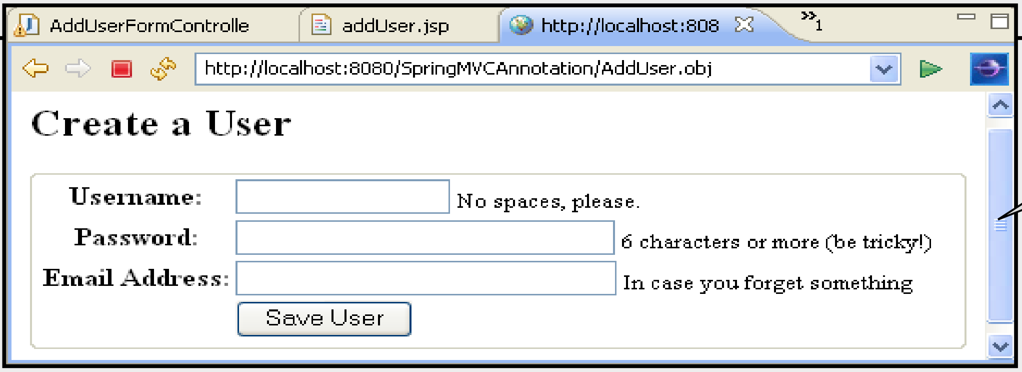
controller

```
public String processForm(@Valid User user, BindingResult 
bindingResult) {
   if (bindingResult.hasErrors()) {
       return "failure";
    }
.....
```

3.2.2 : Spring MVC annotations - Processing forms :

The controller class

```
@Controller
public class AddUserFormController {
   @RequestMapping(value = "/AddUser", method = RequestMethod.GET)
     public String showForm(Model model) {
            model.addAttribute(new User());
             return "addUser";
      }
    @RequestMapping(method = RequestMethod.POST)
     public String processForm(@Valid User user, BindingResult bindingResult) {
             if (bindingResult.hasErrors())     return "failure";
             else {
                     // some logic to persist user
                     return "success";
    }}
```



http://localhost:8080/SpringMVCAnnotation/AddUser.obj

Create a User

Username:		No spaces, please.
Password:		6 characters or more (be tricky!)
Email Address:		In case you forget something

addUser.jsp

3.2.3 Dispatcher Servlet Java Based Configuration

DispatcherServlet can be configured programmatically by implementing or extending either of these three support classes provided by Spring –

- WebAppInitializer interface
- AbstractDispatcherServletInitializer abstract class
- AbstractAnnotationConfigDispatcherServletInitializer abstract class

- WebApplicationInitializer is a perfect fit for use with Spring's code-based @Configuration classes.

3.2.3 Dispatcher Servlet Java Based Configuration

```
public class MyWebAppInitializer implements WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext container) {
      // Create the 'root' Spring application context
      AnnotationConfigWebApplicationContext rootContext =
        new AnnotationConfigWebApplicationContext();
      rootContext.register(AppConfig.class);

      // Manage the lifecycle of the root application context
      container.addListener(new ContextLoaderListener(rootContext));
}
```

3.2.3 Dispatcher Servlet Java Based Configuration

```
// Create the dispatcher servlet's Spring application context
   AnnotationConfigWebApplicationContext dispatcherContext =
     new AnnotationConfigWebApplicationContext();
   dispatcherContext.register(DispatcherConfig.class);

   // Register and map the dispatcher servlet
   ServletRegistration.Dynamic dispatcher =
     container.addServlet("dispatcher", new    DispatcherServlet(dispatcherContext));
   dispatcher.setLoadOnStartup(1);
   dispatcher.addMapping("/");
  }
```

3.2.4 Spring 5 MVC Annotations

@Controller :

It will make class as a request handler.

@GetMapping :

It is specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.GET). @GetMapping annotated methods handle the HTTP GET requests matched with given URI expression

@PostMapping :

It is specialized version of @RequestMapping annotation that acts as a shortcut for @RequestMapping(method = RequestMethod.POST). @PostMapping annotated methods handle the HTTP POST requests matched with given URI expression.

@EnableWebMvc

Other annotations like **@PutMapping, @DeleteMapping, @PatchMapping** can be used in similar way.

Demo

- Refer MVCJavaBasedExample application

Demo

- Refer the following Demos:
  - DemoMVC_3
  - DemoMVC_4
  - DemoMVC_5
  - DemoMVC_6
  - DemoMVC_Complete
  - MVCJavaBasedExample

Summary

- How to use Spring MVC architecture to build flexible and powerful web applications.
- Components like handler mappings, ViewResolvers and controllers
- MVC Annotations like @Controller, @RestController, @RequestMapping , @RequestParam, @PathVariable
- Spring 5 MVC Annotations like @GetMapping, @PostMapping,@EnableWebMvc
- Spring 5 MVC Java based Web Application

Review Questions

- Question 1: If multiple handler mappings have been declared in an application, select the property that indicates which handler mapping has precedence?
  - Option 1: Order
  - Option 2: Sequence
  - Option 3: Index
  - Option 4: An application cant have multiple handler mappings
- Question 2: To figure out which controller should handle the request, DispatcherServlet queries _______
  - Option 1: HandlerMappings
  - Option 2: ModelAndView
  - Option 3: ViewResolver
  - Option 4: HomeController