HCL TALENTCARE

HCL TALENTCARE

Spring - MVC

Context Setting: Overview

HCL TALENTCARE

- Spring's web module includes many unique web support features.
- Spring provides a very clean division between controllers, JavaBean models and views.
- Spring provides interceptors as well as controllers, making it easy to factor out behavior common to the handling of many requests.
- Spring MVC is truly view-agnostic. You do not get pushed to use JSP of you don't want to; you can use Velocity, XLST, or other view technologies.
- Spring Controllers are configured via loC like any other objects. This makes them easy to test, and beautifully integrated with other objects managed by Spring.
- The web tier becomes a thin layer on top of a business object layer. This encourages good practice.
- Spring provides an integrated framework for all tiers of your application

HCL

Module Information

Module Description	In this module you will cover the following: • Spring 3 MVC Introduction • Controller • Handler Mapping and Interceptor • View Resolvers • Tags • Validator and Exception Handling
Level	Practitioner
Prerequisites	Core Java, Core Spring, XML

Course Objective

HCL TALENTCARE

After completing this module, you will be able to:

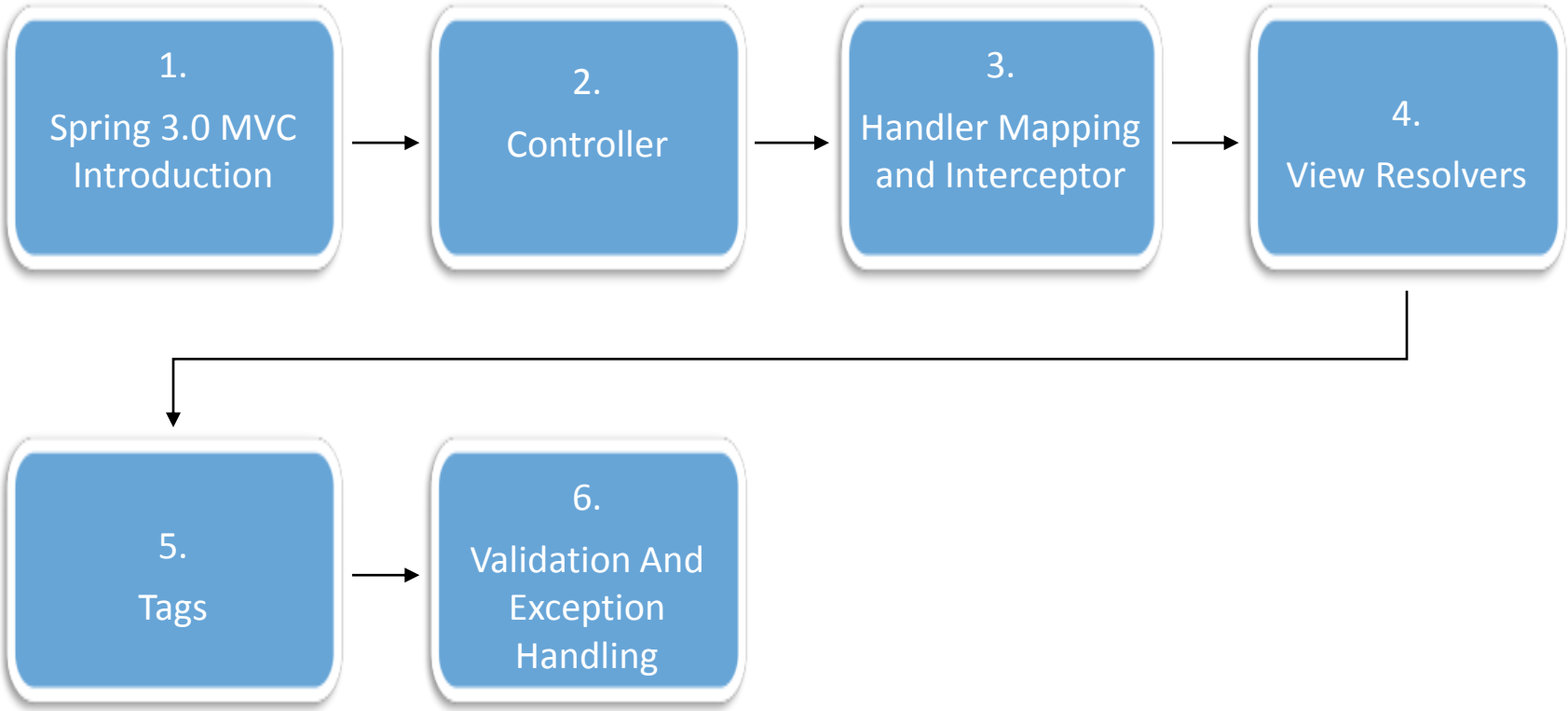
- Configure Spring in Web Application
- Explain the request flow in Spring 3 MVC application
- Explain Design Controllers
- Illustrate how to map a request to a controller
- Write handler interceptors
- Explain various view resolvers
- Describe the various Spring MVC form tags
- Explain how to validate user data

HCL

Course Outline

HCL TALENTCARE

Course Flow:



Session Plan

RIO Name	Duration
RIO1_Spring_3_MVC_Introduction	120 mins
RIO2_Spring_3_MVC_Controller	180 mins
RIO3_Spring_3_MVC_Handler_Mapping_And_Interceptor	240 mins
RIO4_Spring_3_MVC_View_Resolvers	120 mins
RIO5_Spring_3_MVC_Tags	120 mins
RIO6_Spring_3_MVC_Validator_And_Exception_Handling	180 mins

HCL TALENTCARE

HCL TALENTCARE

Spring 3.0 MVC Introduction

Spring 3 MVC Introduction: Overview

HCL TALENTCARE

- What is MVC?

Model-View-Controller (MVC) architecture separates core business model functionality from the presentation and control logic that uses this functionality.

Such separation allows multiple views to share the same enterprise data model, which makes supporting multiple clients easier to implement, test, and maintain.

Objectives

- After completing this chapter, you will be able to:

  - Explain the Spring MVC architecture.
  - Illustrate how to configure Dispatcher Servlet in web.xml.

Do You Know

HCL TALENTCARE

What is MVC?

Learn How – Tools Resources

- The resources required for this training are:

SDE 6

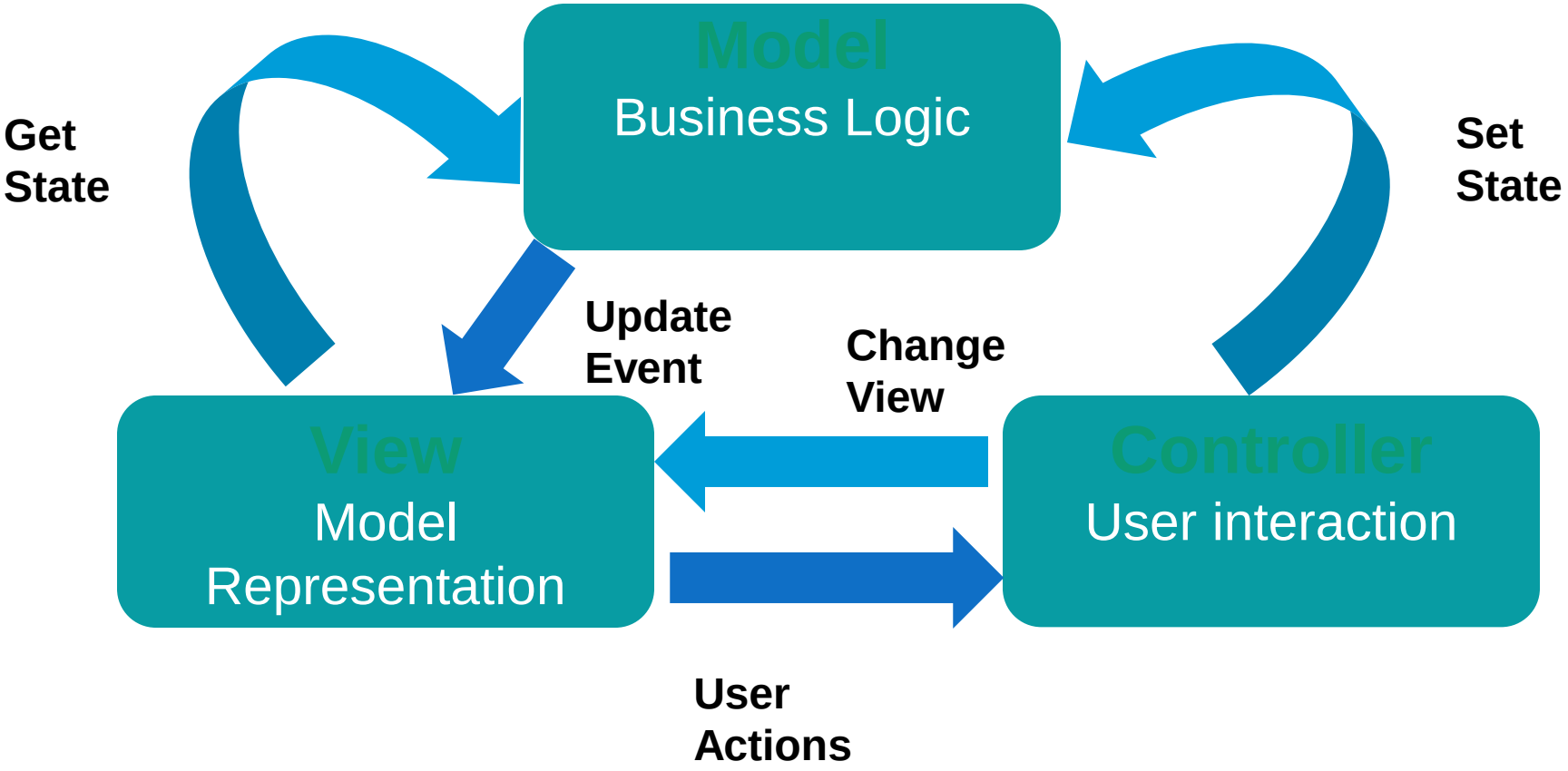
Java 1.5/1.6

Spring Framework 3.0

Apache Tomcat 6.0

MVC Architecture Diagram

- The MVC Architecture is as follows:



Example of MVC Framework

HCL TALENTCARE

- Some famous MVC Framework are:

Struts

- Struts is an open source web application development framework. It is based on the Model-View-Controller (MVC) design paradigm.

Java Server Faces (JSF)

- Java Server Faces (JSF) is a server side user interface component framework for Java™ technology-based web applications. It is based on the Model-View-Controller (MVC) design paradigm.

Webwork

- Webwork is a Java web-application development framework. It is provides robust support for building reusable UI templates, such as form controls, UI themes, internationalization, dynamic form parameter mapping to JavaBeans, robust client and server side validation, and much more.

HCL

Life before MVC: Servlets

HCL TALENTCARE

Servlets ruled the world before MVC pattern came

They were faster than CGI programs

They were more scalable with built-in threading capability

Its value-added features include Request/Response and Sessions

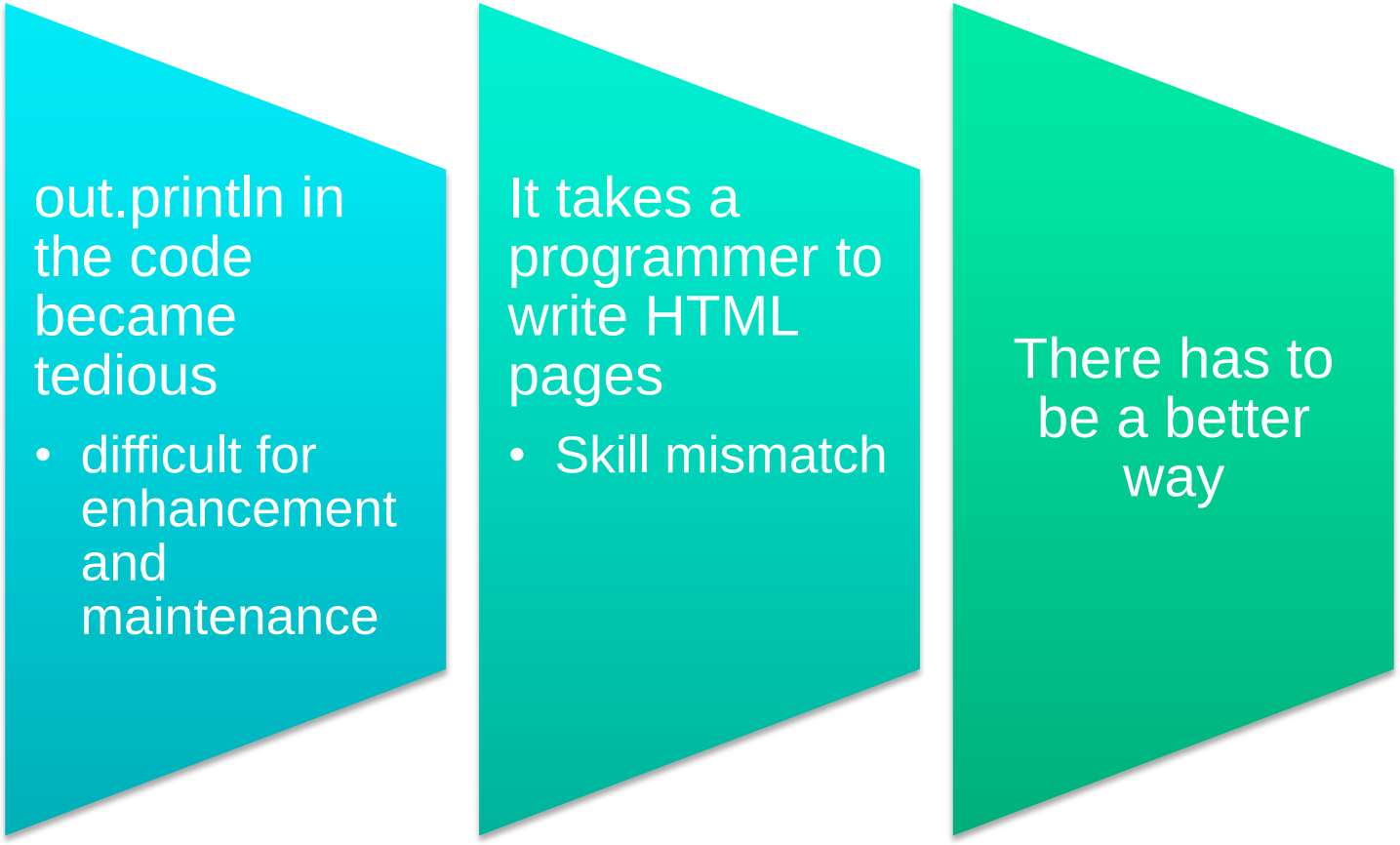
Full Java API access:

JDBC, JMS, EJB, Java Mail, and so on

Problems with Servlets

HCL TALENTCARE

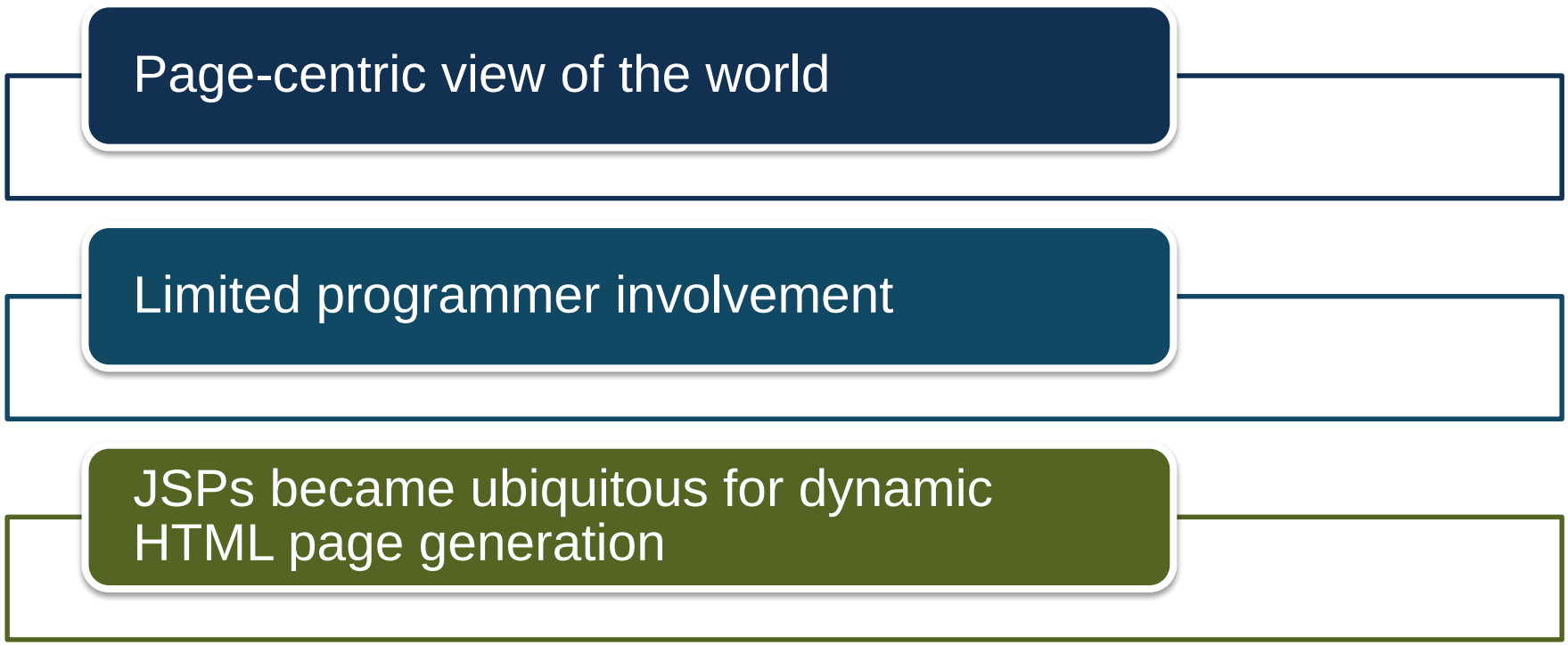
- Some of the problems with Servlets are:



Then Came the JSPs

HCL TALENTCARE

- Some issues with JSPs are:

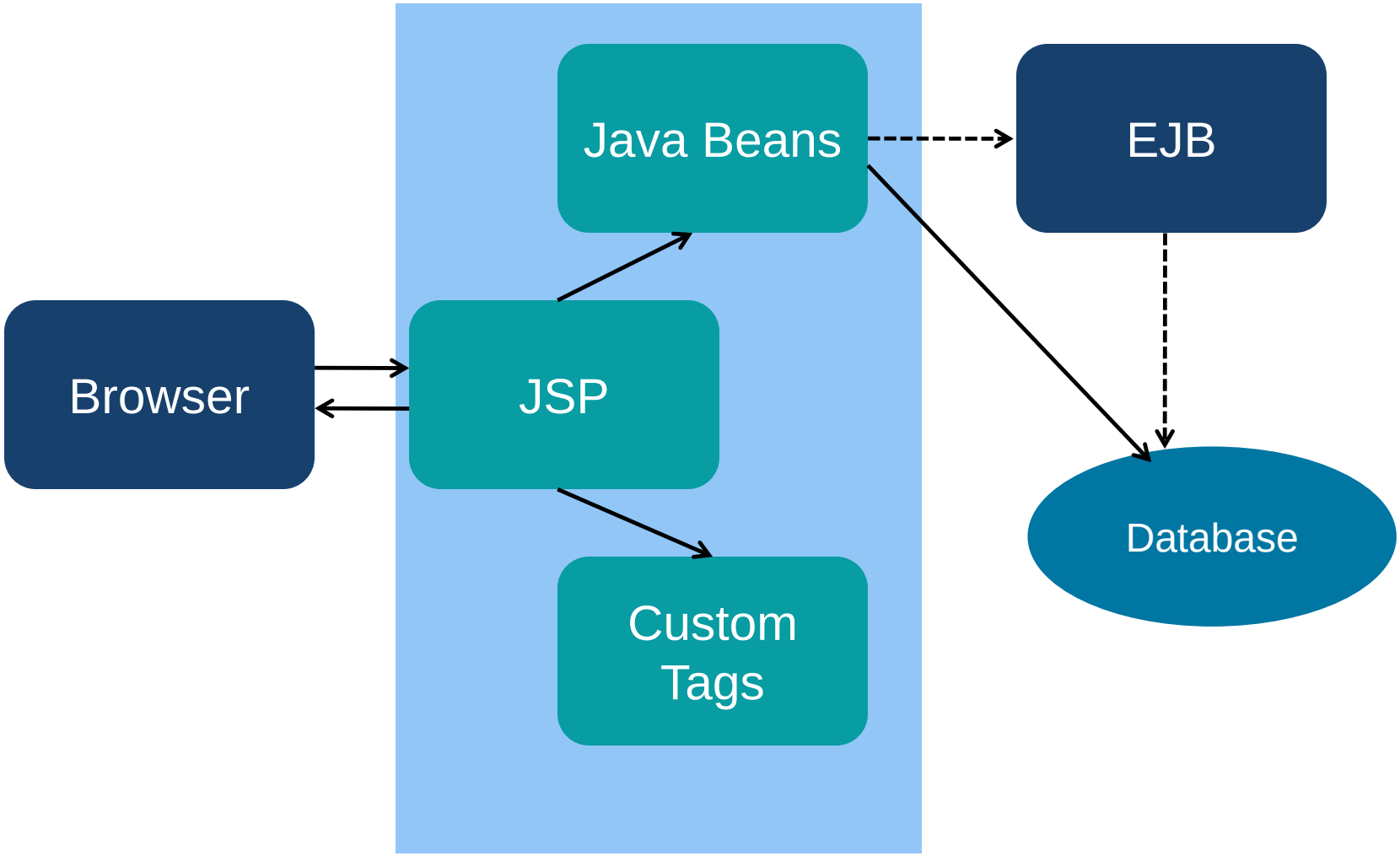


Activity

- State whether the following statement is True or False:

  Servlet never provides separation between or clarity between Presentation Logic and Business Logic.

MVC Model 1 Architecture



MVC Model 1 Architecture (Contd.)

- The feature of MVC Model 1 Architecture are:
  - It is JSP centric
  - It is suitable for small systems
  - It is susceptible to abuse due to the following reasons:
    - Excessive Java code in JSPs
    - Flow control issues

Activity

- In MVC 1 the controller is _____ .

Select from the following options:

Servlet

JSP

Custom Component

None of the above

Questions

MVC Model 2 Architecture

The feature of MVC Model 2 Architecture are as follows:

- It uses servlet for flow control and JSPs for HTML pages and dynamic content.
- It allows programmer or Web designer specialization.
- MVC comprises three components. They are:
  - Controller : Servlet + Helper classes
  - Model: Application data or Operation
  - View: Output rendered to user

MVC Control Flow

HCL TALENTCARE

- Though MVC comes in different flavors, the control flow generally works as follows:

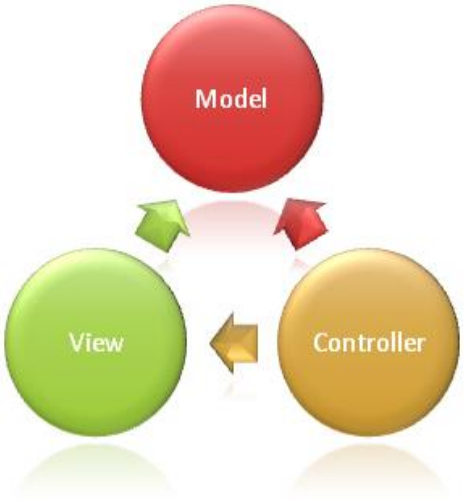
The user interacts with the user interface in some way (For example, user presses a button).

A controller handles the input event from the user interface, often via a registered handler or callback.

The controller accesses the model, possibly updating it in a way appropriate to the user's action (For example, controller updates user's shopping cart). Complex controllers are often structured using the command pattern to encapsulate actions and simplify extension.

A view uses the model to generate an appropriate user interface. For example, view produces a screen listing the shopping cart contents. The view gets its own data from the model. The model has no direct knowledge of the view.

The user interface waits for further user interactions, which begins the cycle anew.



Spring MVC Introduction

HCL TALENTCARE

Spring's web MVC framework is request-driven; designed around a central servlet that dispatches requests to controllers and offers other functionality that facilitates the development of web applications.

There is clear separation of roles: controller, validator, command object, form object, model object.

Simplification of testing through Dependency Injection.

Reusable business code, no need for duplication; use existing business objects as command or form objects instead of mirroring them to extend a particular framework base class.

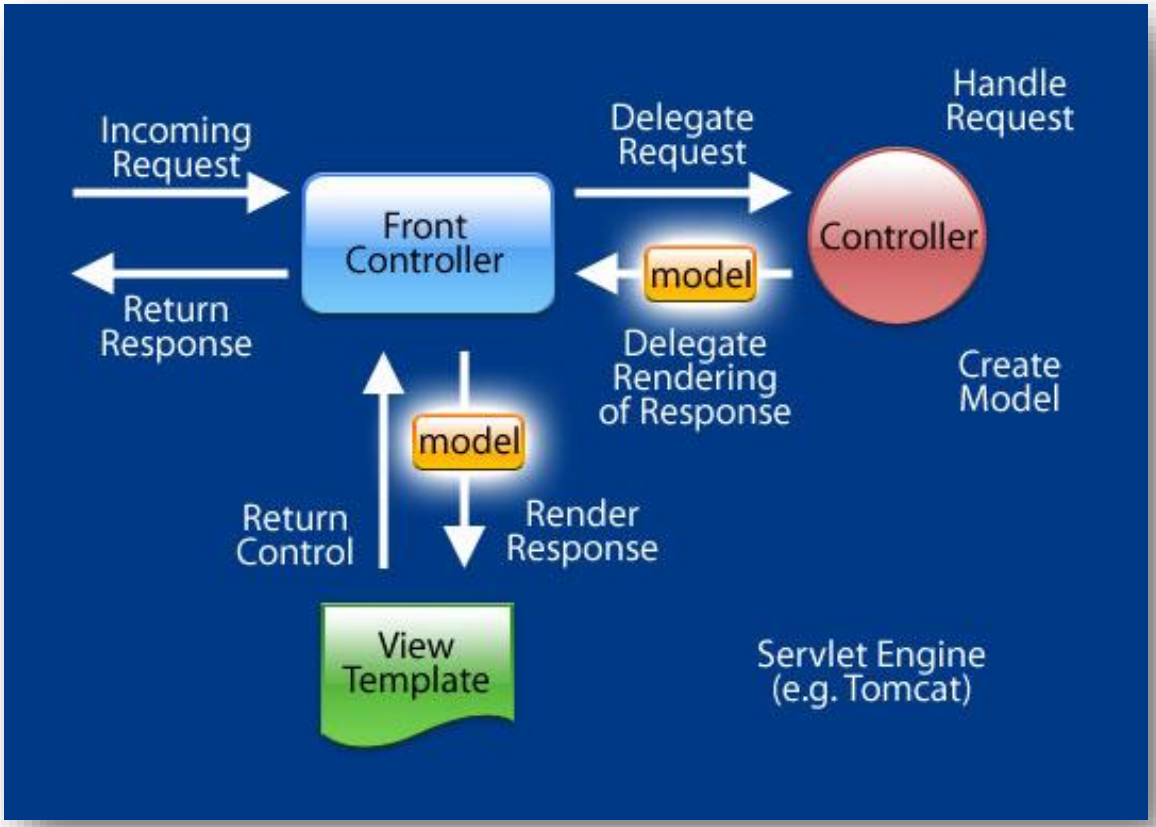
Spring MVC supports JSP, FreeMarker, Velocity, XSLT, JasperReports, Excel, and PDF views.

HCL

Request Life Cycle in Spring MVC

HCL TALENTCARE

- Requesting processing workflow in Spring Web MVC:



Request Life Cycle in Spring MVC (Contd.)

HCL TALENTCARE

- When a request is sent to the Spring MVC Framework, the following sequence of events happen:

1. The Dispatcher Servlet first receives the request.
2. The Dispatcher Servlet consults the HandlerMapping and invokes the Controller associated with the request.
3. The Controller process the request by calling the appropriate service methods and returns a ModeAndView object to the DispatcherServlet. The ModeAndView object contains the model data and the view name.
4. The DispatcherServlet sends the view name to a ViewResolver to find the actual View to invoke.
5. Now the DispatcherServlet will pass the model object to the View to render the result.
6. The View with the help of the model data will render the result back to the user.

HCL

Comparison with Struts 1.x

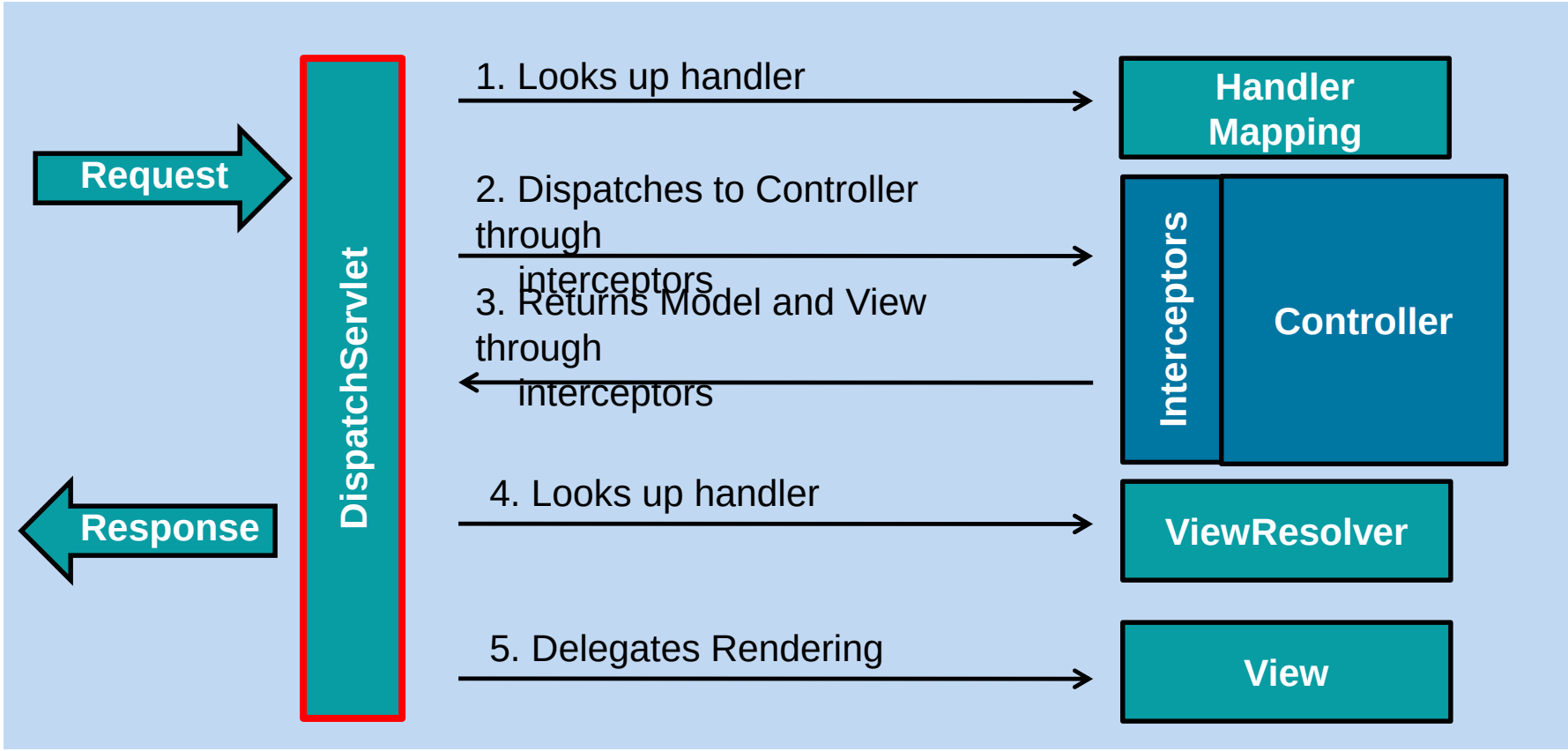
HCL TALENTCARE

In Spring Web MVC, any object can be used as a command or form-backing object. A framework-specific interface or base class need not be implemented, unlike Struts, which forces the Action and Form objects into concrete inheritance.

Spring's data binding is highly flexible. For example, it treats type mismatches as validation errors that can be evaluated by the application, not as system errors. Thus, business objects' properties need not be duplicated as simple, un-typed strings in the form objects simply to handle invalid submissions, or to convert the Strings properly. Instead, it is often preferable to bind directly to the business objects or domain objects.

HCL

Introduction to Dispatcher Servlet



Dispatcher Servlet

The Spring Web Model-View-Controller (MVC) framework is designed around a Dispatcher Servlet. It dispatches requests to handlers, with configurable handler mappings, view resolution, locale, and theme resolution as well as gives support for uploading files.

Example of *Front Controller* pattern is entry point for all Spring MVC requests.

Dispatcher Servlet inherits from the HttpServlet base class.

Dispatcher Servlet has its own WebApplicationContext, which inherits all the beans already defined in the root WebApplicationContext.

Web.xml

HCL TALENTCARE

- Dispatcher Servlet configuration in web.xml:

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

Dispatcher Servlet

HCL TALENTCARE

Note that the *.do url pattern has been mapped with example DispatcherServlet. Thus any url with *.do pattern calls Spring MVC Front controller.

Upon initialization of a Dispatcher Servlet, the framework looks for a file named [servlet-name]-servlet.xml in the WEB-INF directory of your web application and creates the beans defined there.

As per previous configuration of dispatcher servlet ,framework looks for example-servlet.xml in the WEB-INF directory. This file contains all Spring MVC specific beans.

Web.xml

HCL TALENTCARE

- Application context configuration in web.xml:

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

Web.xml (Contd.)

HCL TALENTCARE

Closing associated term with the Dispatcher Servlet is the Application Context. An Application Context usually represents a set of Configuration Files that are used to provide Configuration Information to the Application.

Application Context can be loaded declaratively within the context of an application server. It is created via ContextLoaderListener.

By default, this Context Listener will try to look for the Configuration File by name 'applicationContext.xml' in the '/WEB-INF' directory. But with the help of the parameter 'contextConfigLocation' the default location can be overridden. Even multiple Configuration Files each containing separate piece of Information is also possible.

Two Applications Context

HCL TALENTCARE

The parent context or Root ApplicationContext contains all of the non-web specific beans such as the services, DAOs, and so on. It is created via ContextLoaderListener.

The WebApplicationContext is an extension of the plain Application Context. It has some extra features necessary for web applications.

The WebApplicationContext is bound in the ServletContext, and by using static methods on the RequestContextUtils class the WebApplicationContext can always be looked up.

The WebApplicationContext is nested inside the root ApplicationContext so that the web components can easily find their dependencies.

Activity

- Front Controller component in Spring 3 MVC application is _____ .

Select from the following options:

ActionServlet

DispatcherServlet

DispatcherActionServlet

None of the above

Web Application Context

HCL TALENTCARE

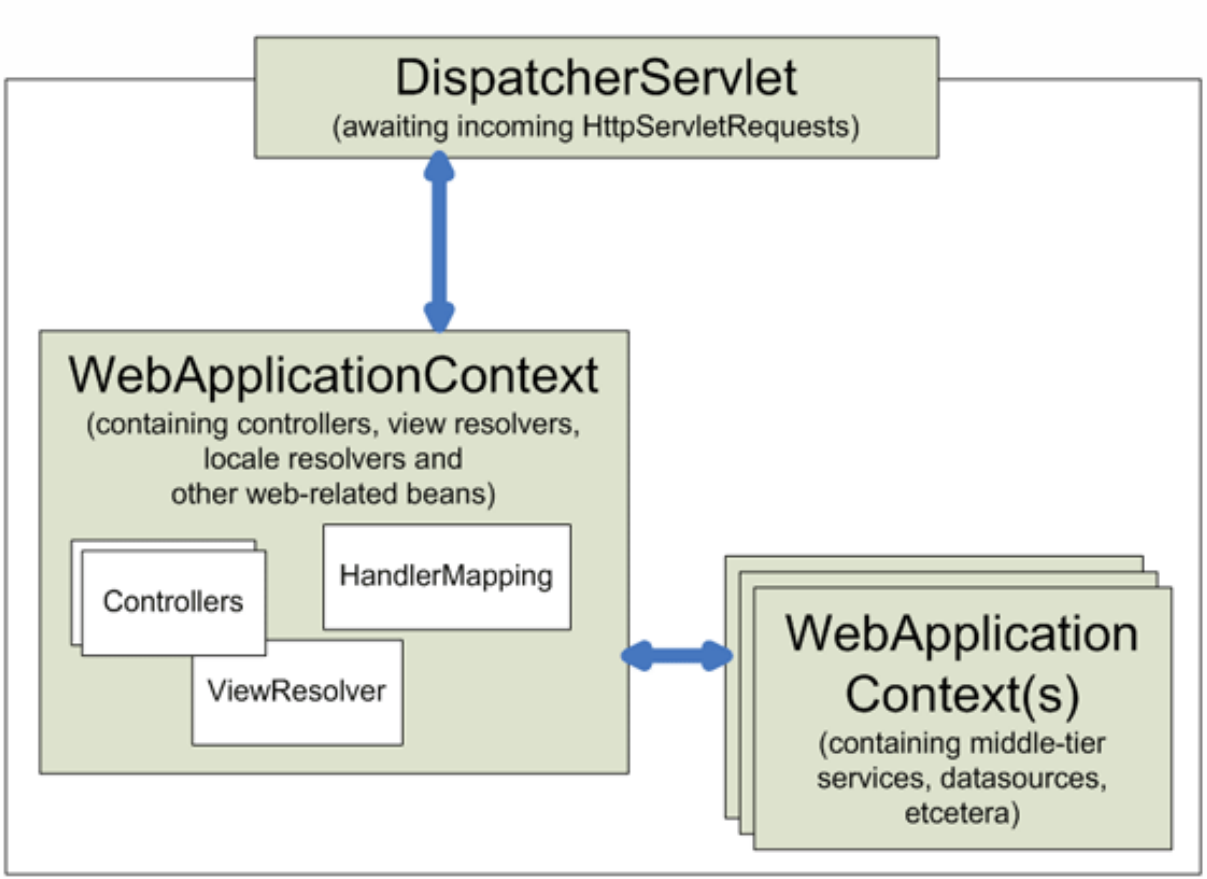
- Take a look at the Special beans in the WebApplicationContext.

Bean Type	Explanation
Controllers	Form the C part of the MVC.
Handlers mappings	Handle the execution of a list of pre-processors and post-processors and controllers that will be executed if they match certain criteria (for example, a matching URL specified with the controller).
Validator	Spring 3 provides support for declarative validation with JSR-303. This support is enabled automatically if a JSR-303 provider, such as Hibernate Validator, is present on your classpath. When enabled, you can trigger validation simply by annotating a Controller method parameter with the @Valid annotation
View Resolvers	Resolves view Names to views
Handler Exception resolvers	Contains functionality to map exceptions to views or implement other more complex exception handling code.
Locale Resolver	Most parts of Spring's architecture support internationalization, just as the Spring web MVC framework does. DispatcherServlet enables automatically resolve messages using the client's locale. This is done with LocaleResolver objects.
Theme Resolver	You can apply Spring Web MVC framework themes to set the overall look-and-feel of your application. A theme is a collection of static resources, typically style sheets and images, that affect the visual style of the application. To use themes in your web application, you must set up an implementation of the org.springframework.ui.context.ThemeSource interface.

Context Hierarchy in Spring Web MVC

HCL TALENTCARE

- This diagram shows the context hierarchy in Spring Web MVC.



What is New in Spring 3 MVC

HCL TALENTCARE

- The things that are new in Spring 3 MVC are as follows:
  - Spring 3 introduces a MVC namespace that greatly simplifies Spring MVC setup.
  - Using the Spring 3 Type Conversion Service as a simpler and more robust alternative to JavaBeans PropertyEditors.
  - Support for formatting Number fields with @NumberFormat.
  - Support for formatting Date, Calendar, and Joda Time fields with @DateTimeFormat, if Joda Time is on the class path.
  - Support for validating @Controller inputs with @Valid, if a JSR-303 Provider is on the classpath.
  - Support for reading and writing XML, if JAXB is on the classpath.
  - Support for reading and writing JSON, if jackson is on the classpath.
  - Spring MVC 3 provides support Ajax remoting with JSON.

HCL

Questions

Test Your Understanding

HCL TALENTCARE

1. A special servlet that has some extra features necessary for web applications. Is Spring MVC a request-driven framework?

True | False

2. Which design pattern does Spring MVC follow?

Front Controller Pattern | Back Controller Pattern

Middle Controller Pattern | None of the above

Test Your Understanding

HCL TALENTCARE

1. A special servlet that has some extra features necessary for web applications. Is Spring MVC a request-driven framework?

True | False

2. Which design pattern does Spring MVC follow?

Front Controller Pattern | Back Controller Pattern

Middle Controller Pattern | None of the above

Test Your Understanding

HCL TALENTCARE

3. What is WebApplicationContext?

- A special servlet that has some extra features necessary for web applications.
- An extension of the plain Application Context that has some extra features necessary for web applications.
- None of the above
- Both the definitions are relevant

Test Your Understanding

HCL TALENTCARE

3. What is WebApplicationContext?

- A special servlet that has some extra features necessary for web applications.
- An extension of the plain Application Context that has some extra features necessary for web applications.
- None of the above
- Both the definitions are relevant

Summary

- Let us take a quick look at the key points covered in this chapter:
  - The Spring MVC is a web development framework based on the Model View Controller (MVC) design pattern.
  - The features of Spring MVC Framework are the Pluggable View technology and Injection of services into controllers.
  - The Spring Web MVC framework is designed around a Dispatcher Servlet.
  - Spring MVC provides a very clean division between controllers, JavaBean models, and views.
  - Spring MVC supports JSP, FreeMarker, Velocity, XSLT, JasperReports, Excel and PDF views.

Source

- http://www.springsource.org
- http://blog.codebeach.com/2008/06/spring-mvc-application-architecture.html
- "Expert Spring Web MVC and Web Flow" by Seth Ladd and others (published by Apress) is an excellent hard copy source of Spring Web MVC goodness.

HCL TALENTCARE

You have successfully completed -
Spring 3 MVC Introduction

HCL TALENTCARE

Thank You

HCL

Activity

- Do you Agree or Disagree with these statement:

1. The Spring Web MVC framework is designed around a Dispatcher Servlet.

2. Spring MVC provides a very clean division between controllers, JavaBean models, and views.

Spring 3 MVC Controller: Overview

HCL TALENTCARE

The notion of a controller is part of the MVC design pattern ,more specifically it is the *'C'* in MVC.

Controllers provide access to the application behavior that you typically define through a service interface.

Controllers interpret user input and transform it into a model that is represented to the user by the view.

Spring 3.0 MVC used an annotation-based programming model for MVC controllers that uses annotations such as @RequestMapping, @RequestParam, @ModelAttribute, and so on.

Controllers implemented in this style do not have to extend specific base classes or implement specific interfaces.

HCL

Objectives

After completing this chapter you will be able to:

- Define a controller with @Controller.
- Explain the use of annotations such as @RequestMapping, @RequestParam, @ModelAttribute , @RequestBody etc.
- Illustrate the Mock Testing on Spring MVC controller.

Do You Know

- Defining a simple controller with @Controller:

```
Package com.cts.spring.controllers;
import org.springframework.stereotype.Controller;
import
org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class HomeController {

@RequestMapping(value = "/home.do")
public String getHome() {
System.out.print("Calling getHome method of HomeController");
return "WEB-INF/views/home.jsp";

}
}
```

Controller

- Let us do a walk through the key aspects of this class:

The class has been annotated with the @Controller annotation, indicating that this is a Spring MVC Controller capable of handling web requests.

The class will automatically be detected by the Spring container as part of the *container's component scanning process.* This will create a bean definition and allowing instances to be dependency injected like any other Spring-managed component.

The getHome method has been annotated with a @RequestMapping annotation, specifying that this method should handle web requests to the path *"/home.do",* which is, the *home* path for the application.

The getHome method simply logs a message to system out, and then returns WEB-INF/views/home.jsp, indicating the view which should handle the response, in this case, a JSP page (If hardcoding the entire view path including WEB-INF prefix, and the fact that it's a JSP, seems wrong to you, you are right. We will deal with this later).

View

- Now, a view needs to be created. This JSP page will simply print a greeting.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core"
prefix="c" %>
<html>
  <head>
     <title>Home</title>
  </head>
  <body>
            <h1>Hello world!</h1>
  </body>
</html>
```

Spring 3 MVC Configuration

HCL TALENTCARE

- Spring 3 introduces a mvc XML configuration namespace that simplifies the setup of Spring MVC inside web application.

```
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:mvc="http://www.springframework.org/schema/mvc"
        xmlns:p="http://www.springframework.org/schema/p"
        xmlns:context="http://www.springframework.org/schema/context"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-3.0.xsd
        http://www.springframework.org/schema/mvc
        http://www.springframework.org/schema/mvc/spring-mvc-3.0.xsd">
          <context:component-scan base-package="com.cts.spring.controllers"/>
        <mvc:annotation-driven />
</beans>
```


HCL

Spring 3 MVC Configuration (Contd.)

HCL TALENTCARE

- Let us walk through the key aspects of this configuration:

This configuration details needs to be added in <servlet-name>-servlet.xml file.

This is added to example-servlet.xml.

A few different Spring XML namespaces are used: *context*, *mvc*, and the default *beans.*

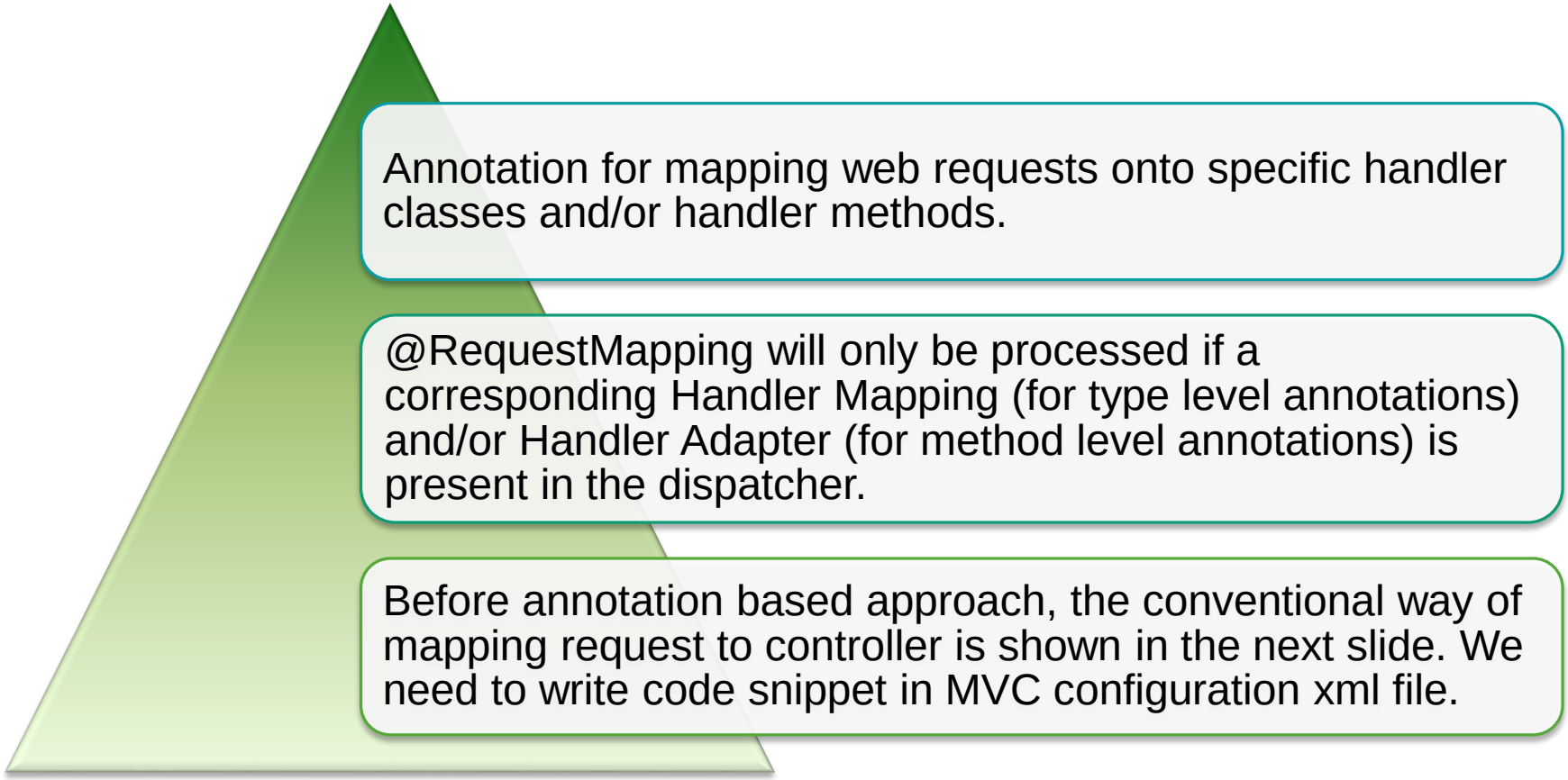
The <context:component-scan> declaration ensures the Spring container does component scanning, so that any code annotated with @Component subtypes such as @Controller is automatically discovered. You'll note that for efficiency, we limit (to com.cts.spring.controllers in this case) what part of the package space Spring should scan in the class path.

HCL

Mapping Request

HCL TALENTCARE

- Mapping requests with @RequestMapping:



HCL

Request Mapping XML-Based Approach

HCL TALENTCARE

- Defining a Request mapping

```
<bean id="userController" class="com.cts.tejas.controller.UserController" >
            <property name="commandName" value="user" />
            <property    name="commandClass"value="com.cts.model.user.User" />
            <property name="formView"   value="simpleUrlMapping" />
            <property name="userService" ref="userService" />
</bean>

<bean id="simpleUrlMapping"
class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
   <property name="mappings">
      <props>
            <prop  key="/userController.do">userController</prop>
       </props>
    </property>
     <property name="order" value="0"></property>
</bean>
```


HCL

Mapping Request: Options

HCL TALENTCARE

- @RequestMapping with different options:

By path	@RequestMapping(“path”)
By HTTP method	@RequestMapping(“path”, method=RequestMethod .GET)

  - POST, PUT, DELETE, OPTIONS, and TRACE are also supported

By presence of query parameter	@RequestMapping(“path”, method=RequestMethod.GET, params=”foo”)

  - Negation also supported: params={ “foo”, “!bar” })

By presence of request header	@RequestMapping(“path”, header=”content-type=text/*”)

HCL

Mapping Request: Revisited

HCL TALENTCARE

- Simplest possible @Controller revisited:

```
@Controller
public class HomeController {
@RequestMapping(“/”, method=RequestMethod.GET,
headers=”Accept=text/plain”)
Public  String home() {
return “hello world”;
}
}
```

Mapping Request: At Class Level

HCL TALENTCARE

- Request mapping at the class level:
  - Concise way to map all requests *within a path to a @Controller*

```
@Controller
@RequestMapping(“/accounts/*)”
public class AccountsController {

        @RequestMapping(“active”)
         public String active() { + }

        @RequestMapping(“inactive”)
        public String inactive() { + }
}
```

Mapping Request

- The same rules expressed with method-level mapping only:

```
@Controller
public class AccountsController {

        @RequestMapping("/accounts/active")
         public String active() { + }

        @RequestMapping("/accounts/inactive")
        public String inactive() { + }
}
```

Activity

1. Identify the optional element of @RequestMapping from the following.

Method

Headers

Value

All the above

Obtaining Request Data: @RequestParam

HCL TALENTCARE

- Binding request parameters to method parameters with @RequestParam:
  - Use the @RequestParam annotation to bind request parameters to a method parameter in your controller.

```
@Controller
@RequestMapping("/pets")

public class EditPetFormController {

@Autowired
Private IPetService service;

@RequestMapping(method = RequestMethod.GET)
public String setupForm(@RequestParam("petId") int petId,
ModelMap model)
{
      Pet pet = service.loadPet(petId);
      model.addAttribute("pet", pet);
      return "petForm";
}
```


HCL

Obtaining Request Data: @RequestBody

HCL TALENTCARE

- Mapping the request body with the @RequestBody annotation:
  - The @RequestBody method parameter annotation indicates that a method parameter should be bound to the value of the HTTP request body.
    - For example:

```
@RequestMapping(value = "/something", method =
RequestMethod.PUT)

public void handle(@RequestBody String body, Writer writer)
throws IOException
{
 writer.write(body);
}
```

  - HttpMessageConverter is responsible for converting from the HTTP request message to an object and converting from an object to the HTTP response body.

HCL

Obtaining Request Data: @PathVariable

HCL TALENTCARE

- A path element value:
  - @PathVariable method parameter annotation to indicate that a method parameter should be bound to the value of a URI template variable.
    - The usage of a single @PathVariable in a controller method:

```
@RequestMapping(value="/users/{userId}",
method=RequestMethod.GET)
public String findUser(@PathVariable String userId, Model model)
{
        Owner owner = ownerService.findOwner(userId);
         model.addAttribute("owner", owner); return
"displayUser";

}
```

  - The URI Template "/users/{userId}" specifies the variable name userId. When the controller handles this request, the value of userId is set to the value in the request URI. For example, when a request comes in for /users/fred, the value fred is bound to the method parameter String userId.

HCL

Obtaining Request Data: @CookieValue

HCL TALENTCARE

- Mapping cookie values with the @CookieValue annotation:
  - The @CookieValue annotation allows a method parameter to be bound to the value of an HTTP cookie.
    - Let us consider that the following cookie has been received with an http request:

```
JSESSIONID=415A4AC178C59DACE0B2C9CA727CDD84
```

  - The following code sample demonstrates how to get the value of the JSESSIONID cookie:

```
@RequestMapping("/displayHeaderInfo.do")
public void displayHeaderInfo(@CookieValue("JSESSIONID") String
cookie) {
//Implementation goes here
}
```


HCL

Obtaining Request Data: @RequestHeader

HCL TALENTCARE

- Mapping request header attributes with the @RequestHeader annotation:
  - The @RequestHeader annotation allows a method parameter to be bound to a request header.

```
Host                    localhost:8080
Accept
text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Language         fr,en-gb;q=0.7,en;q=0.3
Accept-Encoding     gzip,deflate
Accept-Charset      ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive           300
```

Obtaining Request Data: @RequestHeader

HCL TALENTCARE

- The following code sample demonstrates how to get the value of the Accept Encoding and Keep-Alive headers in controller method:

```
@RequestMapping("/displayHeaderInfo.do")

public void   displayHeaderInfo(@RequestHeader("Accept-
Encoding") String encoding, @RequestHeader("Keep-Alive")
long keepAlive) {

//Implementation goes here
}
```

Generating Responses: @ResponseBody

HCL TALENTCARE

- Mapping the response body with the @ResponseBody annotation:
  - The @ResponseBody annotation is similar to @RequestBody. This annotation can be put on a method and indicates that the return type should be written straight to the HTTP response body (and not placed in a Model, or interpreted as a view name).
    - For example:

```
@RequestMapping(value = "/something", method =
RequestMethod.PUT)
@ResponseBody
public String helloWorld() {
return "Hello World";
}
```

    - The above example will result in the text Hello World being written to the HTTP response stream.

HCL

Getting Form Data: @ModelAttribute

HCL TALENTCARE

- Providing a link to data from the model with @ModelAttribute:
  - When place @ModelAttribute on a method parameter, @ModelAttribute maps a model attribute to the specific, annotated method parameter (see the processSubmit() method below). This is how the controller gets a reference to the object holding the data entered in the form.

```
@RequestMapping(method = RequestMethod.POST)
public String processSubmit( @ModelAttribute(“user") User user,
BindingResult result)
{
 new UserValidator().validate(user, result);
if (result.hasErrors())
 {
 return “UserForm";
} else
{ userService.storeUser(user);
return "redirect:profile.do? Id=" + user..getId();
}
```


HCL

Activity

- Identify the annotation that is used to bind request parameter to a method parameter in the controller.

@RequestMapping

@RequestParam

@RequestBody

@RequestParameter

Storing Data in a Session

HCL TALENTCARE

- Specifying attributes to store in a session with @SessionAttributes:
  - The type-level @SessionAttributes annotation declares session attributes used by a specific handler.
    - The following code snippet shows the usage of this annotation:

```
@Controller
@RequestMapping("/editUser.do")
@SessionAttributes(“user")
public class EditForm {
// ...
}
```

Activity

HCL TALENTCARE

- Identify the way to define multiple value in @SessionAttribues

@SessionAttributes("attributeA", "attributeB", "attributeC")

@SessionAttributes(["attributeA", "attributeB", "attributeC"])

@SessionAttributes({"attributeA", "attributeB", "attributeC"})

None of the above

Unit Testing of Controller

HCL TALENTCARE

Unit Testing Spring MVC Controllers:

The main challenge of unit testing Spring MVC controllers, as well as web controllers in other web application frameworks, is simulating HTTP request objects and response objects in a unit testing environment.

Spring supports web controller testing by providing a set of mock objects for the Servlet API (including MockHttpServletRequest, MockHttpServletResponse, and MockHttpSession).

Spring provides convenient support for JUnit 3, JUnit 4, and TestNG5.

TestNG

HCL

Unit Testing of Controller (Contd.)

HCL TALENTCARE

- Let us create a controller:

```
@Controller
public class DepositController {

@Autowired
private AccountService accountService;

@RequestMapping("/deposit.do")
Public String deposit(
@RequestParam("accountNo") String accountNo,
@RequestParam("amount") double amount,
ModelMap model, HttpServletRequest req, HttpServletResponse res)
{
accountService.deposit(accountNo, amount);
model.addAttribute("accountNo", accountNo);
model.addAttribute("balance", accountService.getBalance(accountNo));
return "success";
}
}
```


HCL

Unit Testing of Controller (Contd.)

HCL TALENTCARE

- Test class for controller using Junit 4:

```
public class DepositControllerTest
{
   private static final String TEST_ACCOUNT_NO = "1234";
   private static final double TEST_AMOUNT = 50;
    private MockHttpServletRequest request;
    private MockHttpServletResponse response;
    private DepositController depositController;
    private MockControl mockControl;
    private AccountService accountService;
@Before
    public void setUp() throws Exception {
        this.request = new MockHttpServletRequest();
        this.response = new MockHttpServletResponse();
       mockControl =
MockControl.createControl(AccountService.class);
       accountService = (AccountService) mockControl.getMock();
      depositController = new DepositController(accountService);
    }
```


HCL

Unit Testing of Controller (Contd.)

HCL TALENTCARE

- Test class continued:

```
 @Test
public void testController(){
            request.setRequestURI("/deposit.do");
            String viewName= handlerAdapter.handle(request, response,
controller);
            ModelMap model = new ModelMap();
            String viewName =
            depositController.deposit(TEST_ACCOUNT_NO,
TEST_AMOUNT,   model,request,response);
            mockControl.verify();
            assertEquals(viewName, "success");
            assertEquals(model.get("accountNo"), TEST_ACCOUNT_NO);
            assertEquals(model.get("balance"), 150.0);
}
```

Questions

Test Your Understanding

1. Which of the following annotations allow a method parameter to be bound to the value of an HTTP cookie?

- @CookieValue
- @CookiesValue
- @Cookies
- @CookieBind

Test Your Understanding

1. Which of the following annotations allow a method parameter to be bound to the value of an HTTP cookie?

- @CookieValue
- @CookiesValue
- @Cookies
- @CookieBind

Test Your Understanding

- Spring 3 MVC needs special beans to act like a backing bean for the input forms.

True

False

Test Your Understanding

- Spring 3 MVC needs special beans to act like a backing bean for the input forms.

True

False

Summary

- Let us take a quick look at the key points covered in this chapter.
  - Controllers interpret user input and transform it into a model that is represented to the user by the view.
  - @RequestMapping: Annotation for mapping web requests onto specific handler classes and/or handler methods.
  - @RequestParam: Annotation to bind request parameters to a method parameter in your controller.
  - @RequestBody: Method parameter annotation indicates that a method parameter should be bound to the value of the HTTP request body.
  - @PathVariable: Method parameter annotation to indicate that a method parameter should be bound to the value of a URI template variable.

Summary (Contd.)

- @CookieValue: Annotation allows a method parameter to be bound to the value of an HTTP cookie.
- @RequestHeader: Annotation allows a method parameter to be bound to a request header.
- @ModelAttribute: Maps a model attribute to the specific, annotated method parameter.
- @SessionAttributes: Annotation declares session attributes used by a specific handler.

Source

- “Spring Recipes : A problem - solution approach” by Gary Mak, Josh Long, and Daniel Rubio

- www.springframework.org

Disclaimer: Parts of the content of this course is based on the materials available from the Web sites and books listed above. The materials that can be accessed from linked sites are not maintained by Cognizant Academy and we are not responsible for the contents thereof. All trademarks, service marks, and trade names in this course are the marks of the respective owner(s).

HCL TALENTCARE

Thank You

HCL

HCL TALENTCARE

HCL TALENTCARE

Handler Mapping and Interceptor

Activity

- Match the following:

1> @Session Attribute	A> annotation allows a method parameter to be bound to the value of an HTTP cookie
2>@CookieValue	B>annotation for mapping web requests onto specific handler classes and/or handler methods
3>@RequestMapping	C>annotation declares session attributes used by a specific handler

Handler Mapping and Interceptor: Overview

HCL TALENTCARE

The web request need to be mapped to appropriate handlers, so that the request can be processed.

In Spring 3.0, the Dispatcher Servlet dispatches the request to appropriate Controller classes which is annotated with @Controller annotation.

The request mapping can be achieved by the @RequestMapping Annotation.

Objectives

After completing this chapter, you will be able to:

- Define a controller with @Controller.
- Explain the use of annotations such as @RequestMapping.
- Explain the mapping of each request.
- Illustrate the use of Handler interceptor.

Handler Mapping

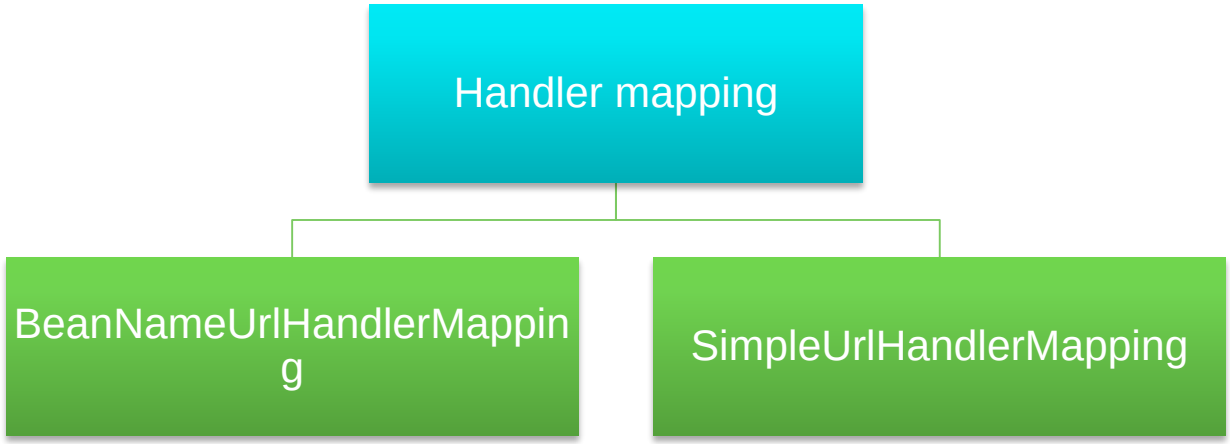
HCL TALENTCARE

HandlerMapping provides the delivering of a HandlerExecutionChain to process the request.

HandlerExecutionChain must contain the handler that matches the incoming request and may also contain a list of handler interceptors that are applied to the request.

When a request comes in, the DispatcherServlet will hand it over to the handler mapping to let it inspect the request and come up with an appropriate HandlerExecutionChain. Then, the DispatcherServlet will execute the handler and interceptors in the chain.

Handler Mapping: Types



- They both extend the AbstractHandlerMapping and share the following properties:
  - Interceptors
  - defaultHandler
  - Order
  - alwaysUseFullPath
  - urlDecode
  - lazyInitHandlers

Handler Mapping Terminologies

HCL TALENTCARE

Interceptors	•Interceptors located in the handler mapping must implement HandlerInterceptor from the org.springframework.web.servlet package. •This interface defines three methods, one that will be called *before* the actual handler will be executed, one that will be called *after* the handler is executed, and one that is called *after the complete request has finished*. •These three methods should provide enough flexibility to do all kinds of pre- and post-processing.
defaultHandler	•The default handler to use, when this handler mapping does not result in a matching handler.
Order	•Based on the value of the order property (org.springframework.core.Ordered interface), Spring will sort all handler mappings available in the context and apply the first matching handler.
alwaysUseFullPath	•If this property is set to true, Spring will use the full path within the current servlet context to find an appropriate handler. •If this property is set to false (the default), the path within the current servlet mapping will be used. •For example, if a servlet is mapped using /testing/* and the alwaysUseFullPath property is set to true, /testing/viewPage.html would be used, whereas if the property is set to false, /viewPage.html would be used.
urlDecode	•The default value for this property is true, as of Spring 2.5. If you prefer to compare encoded paths, switch this flag to false. However, note that the HttpServletRequest always exposes the servlet path in decoded form. Be aware that the servlet path will not match when compared with encoded paths.
lazyInitHandlers	•It allows for lazy initialization of singleton handlers (prototype handlers are always lazily initialized). Default value is false.

HCL

Class Level

- Defining @RequestMapping at Class level:

```
@Controller
@RequestMapping("/department/*")
public class DepartmentController {
            @RequestMapping("add")
             public String addDepartment(Model model) {
                 model.addAttribute("department", new
Department());
                 …
                 …
             }
}
```

Key Aspects of Class Level

HCL TALENTCARE

Let us walk through the key aspects of this class:

- The class has been annotated with the @Controller annotation, indicating that this is a Spring MVC Controller capable of handling web requests.
- This controller class is also annotated with @RequestMapping at class level.
- Each method inside this class will be annotated with @RequestMapping.
- So, the final request mapping String will be the addition of the values declared in the both Class level and method level @RequestMapping annotation.
- The final URL pattern will be /department/add to invoke the addDepartment Method.
- There is a simplest way to map the request just by giving the Annotation at each method.
- The request should be matched with the annotation given at the method level.

HCL

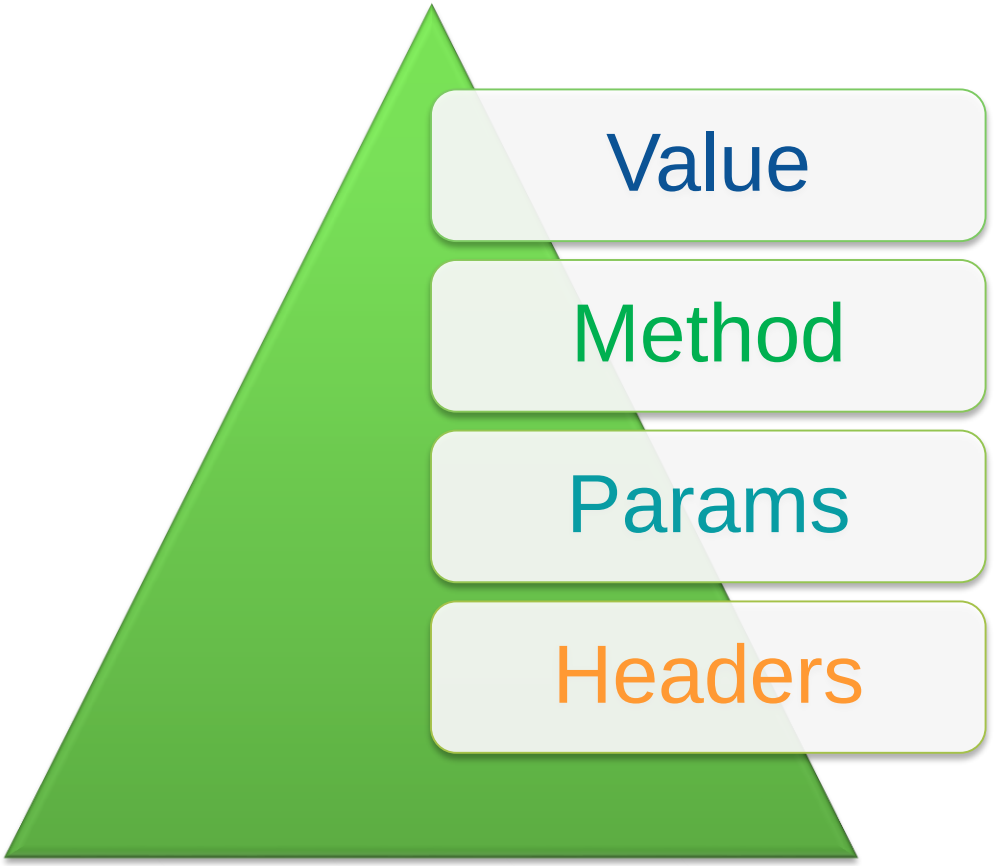
Class Level (Contd.)

HCL TALENTCARE

- Defining @RequestMapping at Class level:

```
@Controller
public class DepartmentController {
        @RequestMapping("addDepartment")
        public String addDepartment(Model model) {
                model.addAttribute("department", new
Department());
                …
                …
}
```

Request Mapping Attributes



Request Mapping Attributes: Value

HCL TALENTCARE

Request Mapping Attributes:

- Value:
  - Public abstract String[] **value** The primary mapping expressed by this annotation. In a Servlet environment: the path mapping URIs (For example, */myPath.do*). Ant-style path patterns are also supported (e.g. "/myPath/*.do"). At the method level, relative paths (For example, *edit.do*) are supported within the primary mapping expressed at the type level.
  - In a Portlet environment: the mapped portlet modes (such as, *EDIT*, *VIEW*, *HELP* or any custom modes).
  - Supported at the type level as well as at the method level: When used at the type level, all method-level mappings inherit this primary mapping, narrowing it for a specific handler method.
  - In case of Servlet-based handler methods, the method names are taken into account for narrowing if no path was specified explicitly, according to the specified MethodNameResolver (by default an InternalPathMethodNameResolver). Note that this only applies in case of ambiguous annotation mappings that do not specify a path mapping explicitly. In other words, the method name is only used for narrowing among a set of matching methods; it does not constitute a primary path mapping itself.

HCL

Request Mapping Attributes: Value (Contd.)

HCL TALENTCARE

- If you have a single default method (without explicit path mapping), then all requests without a more specific mapped method found will be dispatched to it. If you have multiple such default methods, then the method name will be taken into account for choosing between them.

- Default:{}

- Example:

```
        @RequestMapping(value= "addAssociate", method =
    RequestMethod.POST)
public String addAssociate(@ModelAttribute("asc") Associate member,
    Model model
BindingResult result) {
        …..
}
```

Request Mapping Attributes: Method

HCL TALENTCARE

Request Mapping Attributes:

- Method:
  - public abstract RequestMethod[] method The HTTP request methods to map to, narrowing the primary mapping: GET, POST, HEAD, OPTIONS, PUT, DELETE, TRACE. Supported at the type level as well as at the method level! When used at the type level, all method-level mappings inherit this HTTP method restriction (this means that the type-level restriction gets checked before the handler method is even resolved).
  - Supported for Servlet environments as well as Portlet 2.0 environments.
  - Default:{}
  - Example:

```
@RequestMapping(method = RequestMethod.POST)
public String addAssociate(@ModelAttribute("asc") Associate
member, Model model
BindingResult result) {       }
```


HCL

Request Mapping Attributes: Params

HCL TALENTCARE

Request Mapping Attributes:

- Params:
  - public abstract String[] params The parameters of the mapped request, narrowing the primary mapping.
  - Same format for any environment: a sequence of "myParam=myValue" style expressions, with a request only mapped if each such parameter is found to have the given value.
  - Expressions can be negated by using the "!=" operator, as in "myParam!=myValue". *myParam* style expressions are also supported, with such parameters having to be present in the request (allowed to have any value).
  - Finally, *!myParam* style expressions indicate that the specified parameter is not supposed to be present in the request.
  - Supported at the type level as well as at the method level: When used at the type level, all method-level mappings inherit this parameter restriction (that is, the type-level restriction gets checked before the handler method is even resolved).

.

HCL

Request Mapping Attributes: Params (Contd.)

HCL TALENTCARE

- In a Servlet environment, parameter mappings are considered as restrictions that are enforced at the type level. The primary path mapping (that is the specified URI value) still has to uniquely identify the target handler, with parameter mappings simply expressing preconditions for invoking the handler
- In a Portlet environment, parameters are taken into account as mapping differentiators, i.e. the primary portlet mode mapping plus the parameter conditions uniquely identify the target handler. Different handlers may be mapped onto the same portlet mode, as long as their parameter mappings differ.
- **Default:**{}

HCL

Request Mapping Attributes: Headers

HCL TALENTCARE

Request Mapping Attributes

- Headers:
  - public abstract String[] headers The headers of the mapped request, narrowing the primary mapping. Same format for any environment: a sequence of "My-Header=myValue" style expressions, with a request only mapped if each such header is found to have the given value. Expressions can be negated by using the "!=" operator, as in "My-Header!=myValue". "My-Header" style expressions are also supported, with such headers having to be present in the request (allowed to have any value). Finally, "!My-Header" style expressions indicate that the specified header is not supposed to be present in the request.

  - Also supports media type wildcards (*), for headers such as Accept and Content-Type. For example,

    @RequestMapping(value = "/something", headers = "content-type=text/*") will match requests with a Content-Type of "text/html", "text/plain", and so on.

HCL

Request Mapping Attributes: Headers (Contd.)

HCL TALENTCARE

- Supported at the type level as well as at the method level: When used at the type level, all method-level mappings inherit this header restriction (that is, the type-level restriction gets checked before the handler method is even resolved).
- Maps against HttpServletRequest headers in a Servlet environment, and against PortletRequest properties in a Portlet 2.0 environment.
- **Default:**{}

HCL

Activity

- Handler methods which are annotated with this annotation are NOT allowed to have very flexible signatures.

True

False

Handler Interceptors

HCL TALENTCARE

- Handler Interceptors:

Workflow interface that allows for customized handler execution chains. Applications can register any number of existing or custom interceptors for certain groups of handlers, to add common pre-processing behavior without needing to modify each handler implementation.

A HandlerInterceptor gets called before the appropriate HandlerAdapter triggers the execution of the handler itself. This mechanism can be used for a large field of preprocessing aspects, for example, for authorization checks or common handler behavior like locale or theme changes. Its main purpose is to allow for factoring out repetitive handler code.

Typically an interceptor chain is defined per HandlerMapping bean, sharing its granularity. To be able to apply a certain interceptor chain to a group of handlers, one needs to map the desired handlers via one HandlerMapping bean. The interceptors are defined as beans in the application context, referenced by the mapping bean definition via its "interceptors" property (in XML: a <list> of <ref>).

HCL

Handler Interceptors (Contd.)

HCL TALENTCARE

HandlerInterceptor is basically similar to a Servlet 2.3 Filter, but in contrast to the latter it just allows custom pre-processing with the option of prohibiting the execution of the handler itself and custom post-processing. Filters are more powerful, for example, they allow for exchanging the request and response objects that are handed down the chain. Note that a filter gets configured in web.xml while a HandlerInterceptor in the application context.

As a basic guideline, fine-grained handler-related pre-processing tasks are candidates for HandlerInterceptor implementations, especially factored-out common handler code and authorization checks. On the other hand, a Filter is well-suited for request content and view content handling, like multipart forms and GZIP compression. This typically shows when one needs to map the filter to certain content types (for example, images), or to all requests.

Preprocessing and post processing of the Requests are handled by the Handler Interceptors.

The life cycle of the Interceptor will be handled by three methods. They are:	preHandle
	postHandle
	afterCompletion

Life Cycle of the Interceptor: Methods

HCL TALENTCARE

prehandle Method

- preHandle method will be called before the action getting the request.

posthandle Method

- postHandle method will be called after the action performed the request and before the response reaches back to the client.

afterCompletion

- afterCompletion() will be called after the view has been rendered.

Overriding the Methods

- There are two ways to override the following methods:

Implementing HandlerInterceptor :

- If you want to override all the three methods of **HandlerInterceptor** interface.

Extending the abstract class HandlerInterceptor Adapter:

- Normally, all three methods are not required at all times. So, instead of implementing HandlerInterceptor interface, extend this HandlerInterceptorAdapter and override only the methods that are required.

Implementing Handler Interceptor

HCL TALENTCARE

- Implementing HandlerInterceptor:

```
import org.springframework.web.servlet.HandlerInterceptor;
import org.springframework.web.servlet.ModelAndView;

public class LoggerInterceptor implements HandlerInterceptor {
        public boolean preHandle(HttpServletRequest request,
          HttpServletResponse response, Object handler) throws Exception {
           ……
          return true;
           }
          public void postHandle(HttpServletRequest request,HttpServletResponse
response,
                                  Object handler,ModelAndView modelAndView)  throws
Exception {
          …….
           }
           public void afterCompletion(HttpServletRequest request,
        HttpServletResponse response, Object handler, Exception ex)
        throws Exception {
          ……
        }
}
```

Key Aspects of Handler Interceptors

HCL TALENTCARE

- Let us walk through the key aspects of this class:

The class is implemented with **HandlerInterceptor** Interface.

All the three methods are overidden.

Extending Handler Interceptor

HCL TALENTCARE

- Extending HandlerInterceptorAdapter:

```
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;
import org.springframework.web.servlet.ModelAndView;


public class LoggerInterceptor extends HandlerInterceptorAdapter {
        public boolean preHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler) throws Exception {
                ...
        }
        public void postHandle(HttpServletRequest request,
        HttpServletResponse response, Object handler,
        ModelAndView modelAndView) throws Exception {
                ...
        }
}
```


HCL

Key Aspects of Handler Interceptors (Contd.)

HCL TALENTCARE

- Let us walk through the key aspects of this class:

The class is implemented with HandlerInterceptor Adapter Abstract class.

Please note that all the three methods are not overridden. Only the required methods are overridden.

Registering Interceptor and Ordering

HCL TALENTCARE

- Registering Interceptor and ordering:

This interceptor will be registered to the DefaultAnnotationHandlerMapping. Once interceptor registered, all the request will be passed through this interceptor.

List of interceptors will be registered to DefaultAnnotationHandlerMapping.

The order of the interceptor will be mentioned with the Order property. The lowest order will take the higher precedence.

Handler Interceptor: Example

HCL TALENTCARE

- Example:

```
<beans ...>
      <bean id="LoggerInterceptor"
    class="com.cts.loggerInterceptor" />
     <bean id="LoggingInterceptor"
              "class="com.cts.LoggingInterceptor" />
     <bean
        class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandler
        Mapping">
     <property name="order" value="1"/>
     <property name="interceptors">
           <list>
                 <ref bean=" LoggerInterceptor " />
           </list>
     </property>
</bean>
```


HCL

Handler Interceptor: Example (Contd.)

HCL TALENTCARE

```
<bean
   class="org.springframework.web.servlet.mvc.annotation.DefaultAnnota
   tionHandlerMapping">

   <property name="order" value="1"/>

    <property name="order" value="0" />

              <property name="urls">

    <list>

    <value>/associateSummary*</value>

    </list>

    </property>
```

Handler Interceptor: Example (Contd.)

```
        <property name="interceptors">
        <list>
                <ref bean="loggingInterceptor" />
        </list>
        </property>
    </bean>
</beans>
```

Activity

- Which of the following HandlerInterceptor Callback method triggered after completion of request processing, that is, after rendering the view?

preHandle()

postHandle()

afterCompletion()

None of the above

Test Your Understanding

HCL TALENTCARE

1. Identify the life cycle methods of Handler Interceptor?

- preHandle, postHandle, afterComplete
- preRequest, postRequest, afterRequest
- preHandle, postHandle, afterCompletion
- None of the above

Test Your Understanding

1. Identify the life cycle methods of Handler Interceptor?

- preHandle, postHandle, afterComplete
- preRequest, postRequest, afterRequest
- preHandle, postHandle, afterCompletion
- None of the above

Test Your Understanding

HCL TALENTCARE

2. What property of handler mappings will sort all handler mappings available in the context and apply the first matching handler?

- priority
- order
- index
- None of the above

Test Your Understanding

2. What property of handler mappings will sort all handler mappings available in the context and apply the first matching handler?

- priority
- order
- index
- None of the above

Summary

- Let us take a quick look at the key points covered in this chapter:
  - HandlerMapping provides the delivering of a HandlerExecutionChain to process the request.
  - There are two commonly used handler mappings named *BeanNameUrlHandlerMapping* and *SimpleUrlHandlerMapping.*
  - Preprocessing and post processing of the requests are handled by the Handler Interceptors.
  - The life cycle of the Interceptor is handled by three methods. They are: preHandle, postHandle, and afterCompletion.

Source

- “Spring Recipes : A problem - solution approach” by Gary Mak, Josh Long, and Daniel Rubio

- www.springframework.org

Disclaimer: Parts of the content of this course is based on the materials available from the Web sites and books listed above. The materials that can be accessed from linked sites are not maintained by Cognizant Academy and we are not responsible for the contents thereof. All trademarks, service marks, and trade names in this course are the marks of the respective owner(s).

HCL TALENTCARE

Thank You

HCL

HCL TALENTCARE

HCL TALENTCARE

View Resolvers

Activity

HCL TALENTCARE

What is the role of a view resolver in Spring MVC architecture?

Spring MVC View Resolver: Overview

HCL TALENTCARE

- The view has to be rendered after DispatchServlet processes the request.
- You want to devise a strategy so the correct content and type is returned for all requests.
- When a request is received for a web application, it contains a series of properties that allow the processing framework, in this case Spring MVC, to determine the correct content and type to return to the requesting party. The main two properties include:
  - The URL extension provided in a request
  - The HTTP Accept header

HCL

Objectives

- After completing this chapter you will be able to:
  - Explain the view resolvers.
  - Identify the multiple view resolvers.
  - Explain XML based view resolvers.
  - Describe Resource bundle based view resolvers.
  - Explain the content negotiating view resolvers.

View Resolver Interface

HCL TALENTCARE

- ViewResolver:

ViewResolver Interface needs to be implemented by objects that can resolve views by name.

View state does not change during the running of the application, so implementations are free to cache views.

Implementations are encouraged to support internationalization, that is, localized view resolution.

All the view resolvers classes are implemented with this ViewResolver interface.

An application can have many view resolvers declared in the configuration file. While having many resolvers, it will be ordered with order property.

When chaining ViewResolvers, an InternalResourceViewResolver always needs to be last, as it will attempt to resolve any view name, no matter whether the underlying resource actually exists.

HCL

InternalResourceViewResolver

HCL TALENTCARE

Here, view Location is formed by the help of a template provided with Prefix and Suffix.

With the string returned from the controller method, the resolver forms the view by using the prefix and suffix provided in the template.

InternalResourceViewResolver (Contd.)

- InternalResourceViewResolver
  - Class Hierarchy:

java.lang.Object
  └ org.springframework.context.support.ApplicationObjectSupport
    └ org.springframework.web.context.support.WebApplicationObjectSupport
      └ org.springframework.web.servlet.view.AbstractCachingViewResolver
        └ org.springframework.web.servlet.view.UrlBasedViewResolver
          └ **org.springframework.web.servlet.view.InternalResourceViewResolver**

All Implemented Interfaces:

ApplicationContextAware, Ordered, ServletContextAware, ViewResolver

InternalResourceView Resolver (Contd.)

HCL TALENTCARE

- Declaring a view resolver bean

```
<bean id="jspViewResolver"
              class="org.springframework.web.servlet.view.InternalResourceViewResolver">
               <property name="viewClass”
value="org.springframework.web.servlet.view.JstlView" />
               <property name="prefix" value="/WEB-INF/jsp/" />
               <property name="suffix" value=".jsp" />
</bean>
<bean id="viewResolver”
class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
<property name="order" value="0" />
<property name="mediaTypes">
<map>
<entry key="html" value="text/html" />
<entry key="pdf" value="application/pdf" />
<entry key="xls" value="application/vnd.ms-excel" />
<entry key="xml" value="application/xml" />
</map>
</property>
</bean>
```


HCL

View Resolver: Example

HCL TALENTCARE

- Take a look at some examples of View Resolver.

S.No	String returned from controller	Rendered view
1	addAssociate	/WEB-INF/jsp/addAssociate.jsp
2	deleteAssociate	/WEB-INF/jsp/deleteAssociate.jsp

View Resolver: XML Configuration

HCL TALENTCARE

- Resolving Views from an XML Configuration File:
  - In this, the views are declared as beans in a separate XML file and location of this XML file will be declared by XmlViewResolver in the web application context.
  - By default, the XmlViewResolver searches the beans from /WEB-INF/views.xml. However this location can be overridden by giving different location.
- Example:

```
<bean
    class="org.springframework.web.servlet.view.XmlViewResolver">
    <property name="location">
        <value>/WEB-INF/associate -views.xml</value>
    </property>
</bean>
```

View Resolver: Resource Bundle

HCL TALENTCARE

- Resource Bundle based view resolvers:
  - This resolver will be useful, if we need to render the view based on a different locale.
  - Each entry on the resource bundle will have the view resolver class and the corresponding location (URL) of the view.
- Example:

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

addAssociate.url=/WEB-INF/jsp/addAssociate.jsp

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

deleteAssociate.url=/WEB-INF/jsp/deleteAssociate.jsp

HCL

View Resolver: Content Negotiation Slide One

HCL TALENTCARE

It needs to be defined in a way to get the expected content and type is rendered on the view if multiple resolvers are present in the application.

This content negotiation will be achieved by the extension provided in request URL or by the HTTP accept headers.

Content Negotiation in Spring is based on the ContentNegotiatingViewResolver.

The resolvers are prioritized by the value of the order attribute.

The value of the order attribute also uses SpEL to have relative order for these resolvers instead of hard coding the order values.

The Lower order value view resolver takes the higher precedence.

Normally the ContentNegotiatingViewResolver takes the highest priority by the having lowest order value.

View Resolver: Content Negotiation Slide Two

HCL TALENTCARE

- ContentNegotiatingViewResolver resolves the request by the following method:

1. Verifying the request path for the extension.
2. If the extension is present but no bean matches then FileTypeMap in Java activation framework is used to determine the media type.
3. If there is no extension, then Accept header is used to determine the media type.
4. If media type to be rendered is found and the logical view named on the highest priority resolver is not matching then remaining resolvers is searched to match the media type.

View Resolver: Content Negotiation Slide Thre

HCL TALENTCARE

- Example:

```
<beans ...>
    ...
    <bean id="contentNegotiatingResolver"
    class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">
    <property name="order" value="#{T(org.springframework.core.Ordered).
    HIGHEST_PRECEDENCE}" />
    <property name="mediaTypes">
        <map>
            <entry key="html" value="text/html"/>
            <entry key="pdf" value="application/pdf"/>
```


HCL

View Resolver: Content Negotiation Slide Thre

HCL TALENTCARE

```
                <entry key="xsl" value="application/vnd.ms-excel"/>
                <entry key="xml" value="application/xml"/>
                <entry key="json" value="application/json"/>
        </map>
</property>
....
</bean>
```


HCL

View Resolver: Content Negotiation SlideFour

HCL TALENTCARE

```
<bean id="resourceBundleResolver"

class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
    <property name="order" value="#{contentNegotiatingResolver.order+1}" />
    ....
</bean>
<bean id="secondaryResourceBundleResolver"

class="org.springframework.web.servlet.view.ResourceBundleViewResolver">
```

View Resolver: Content Negotiation

```
            <property name="basename" value="secondaryviews" />

            <property name="order"  value="#{resourceBundleResolver.order+1}" />

            </bean>

                <bean id="internalResourceResolver"

            class="org.springframework.web.servlet.view.InternalResourceView
Resolver">

            <property name="order"
value="#{secondaryResourceBundleResolver.order+1}" />

            ....

        </bean>

</beans>
```

Interactive Activity

HCL TALENTCARE

- Which of the following ViewResolver should be used for rendering views based on Locale?

Select from the following options:

BeanNameViewResolver

XMLViewResolver

ResourceBunderViewResolver

None of the above

Activity

- Which property of the ViewResolver is used to set the order when more than view resolver is configured in the Web Application?

Select from the following options:

priority

order

type

None of the above

Test Your Understanding

1. Identify the view class that should be used when using JSTL.

- JspView
- JstlView
- ServletView
- None of the above

Test Your Understanding

1. Identify the view class that should be used when using JSTL.

JspView

JstlView

ServletView

None of the above

Test Your Understanding

2. Identify the simple implementation of the ViewResolver interface that effects the direct resolution of logical view names to URLs.

- XmlViewResolver
- UrlBasedViewResolver
- ContentNegotiatingViewResolver
- None of the above

Test Your Understanding

2. Identify the simple implementation of the ViewResolver interface that effects the direct resolution of logical view names to URLs.

- XmlViewResolver
- UrlBasedViewResolver
- ContentNegotiatingViewResolver
- None of the above

Summary

- Let us take a quick look at the key points covered in this course.
  - ViewResolver Interface needs to be implemented by objects that can resolve views by name.
  - View Location is formed by the help of a template provided with Prefix and Suffix.
  - XmlViewResolver views are declared as beans in a separate XML file.
  - ContentNegotiatingViewResolver content negotiation will be achieved by the extension provided in request URL or by the HTTP accept headers.

Source

- “Spring Recipes : A problem - solution approach” by Gary Mak, Josh Long, and Daniel Rubio

- www.springframework.org

Disclaimer: Parts of the content of this course is based on the materials available from the Web sites and books listed above. The materials that can be accessed from linked sites are not maintained by Cognizant Academy and we are not responsible for the contents thereof. All trademarks, service marks, and trade names in this course are the marks of the respective owner(s).

HCL TALENTCARE

Thank You

HCL

HCL TALENTCARE

HCL TALENTCARE

Tags

Activity

Let us discuss:

What are JSP tags?

Can you list a few that you might have used?

Spring Tags: Overview

HCL TALENTCARE

Spring 3.0 provides a comprehensive set of data binding-aware tags for handling form elements when using JSP and Spring Web MVC.

Each tag provides support for the set of attributes of its corresponding HTML tag counterpart, making the tags familiar and intuitive to use.

Unlike other form or input tag libraries, Spring's form tag library is integrated with Spring Web MVC. This helps to give the tags access to the command object and reference data your controller deals with.

Objectives

After completing this chapter, you will be able to:

- Explain the use Spring's form tag to develop user interface for an Web application.

Configuration

The library descriptor is called spring-form.tld.

•To use the tags from this library, add the following directive to the top of your JSP page:
<%@ taglib prefix="form" uri="http://www.springframework.org/tags/form" %> where form is the tag name prefix you want to use for the tags from this library.

The Form Tag

This tag renders an HTML 'form' tag and exposes a binding path to inner tags for binding.

It puts the command object in the PageContext so that the command object can be accessed by inner tags.

All the other tags in this library are nested tags of the form tag.

Tag Example: Form Tag

HCL TALENTCARE

- The Form tag :

```
<form:form>
  <table>
    <tr>
      <td>First Name:</td>
      <td><form:input path="firstName" /></td>
    </tr>
    <tr>
      <td>Last Name:</td>
      <td><form:input path="lastName" /></td>
    </tr>
    <tr>
      <td colspan="2">
      <input type="submit" value="Save Changes" />
      </td>
    </tr>
  </table>
</form:form>
```


HCL

Tag Example: Form Tag

HCL TALENTCARE

- This tag renders an HTML 'input' tag with type 'text' using the bound value.

```
<form:input path="firstName" />
<form:input path="lastName"  />
```

Tag Example: Form Tag (Contd.)

HCL TALENTCARE

- The generated HTML looks like a standard form.

```
<form method="POST">
<table>
<tr>
<td>First Name:</td>
<td><input name="firstName" type="text" value="John"/></td>
</tr>
<tr>
<td>Last Name:</td>
<td><input name="lastName" type="text" value="Terry"/></td>
</tr>
<tr>
<td colspan="2">
<input type="submit" value="Save Changes" />
</td>
</tr>
</table>
</form>
```

Activity

- Which attribute in the input tag binds the data entered in the input field to a property in the data backing object of the controller used for the JSP?

property

data

path

None of the above

Activity

- <form:errors/> must be within a <form:form/> .

True

False

Tag Example: Checkbox Slide One

HCL TALENTCARE

- This tag renders an HTML 'input' tag with type 'checkbox'.
  - Let us assume *Employee* has preferences such as newsletter subscription and a list of hobbies. Below is an example of the Preferences class:

```
public class Preferences {
private boolean receiveNewsletter;
String[] interests
private String favouriteWord;..add setter and getter
methods }
```

Tag Example: Checkbox Slide Two

HCL TALENTCARE

- The jsp would look like:

```
<form:form>
      <table>
         <tr> <td>Subscribe to newsletter?:</td>
        <td>
        <form:checkbox path="preferences.receiveNewsletter"/></td>
</tr>
        <tr> <td>Interests:</td>
       <td>
Hibernate: <form:checkbox path="preferences.interests"
value="Hibernate"/>     Spring: <form:checkbox
path="preferences.interests" value="Spring"/> </td> </tr> <tr>
     <td>Favourite Subject:</td>
    <td>
   JSF: <form:checkbox path="preferences.favouriteWord"
value="JSF"/>
   </td>
   </tr>
  </table>
</form:form>
```


HCL

Tag Example: Checkbox Slide Three

HCL TALENTCARE

- Below is an HTML snippet of some checkboxes:

```
<tr>
<td>Interests:</td>
<td>
Hibernate: <input name="preferences.interests" type="checkbox"
value="Hibernate"/>
<input type="hidden" value="1" name="_preferences.interests"/>
Spring: <input name="preferences.interests" type="checkbox"
value="Spring"/>
<input type="hidden" value="1" name="_preferences.interests"/>
</td>
</tr>
…
```

Tag Example: Radio Button Tag

HCL TALENTCARE

- This tag renders an HTML *input* tag with type *radio*.

- A typical usage pattern will involve multiple tag instances bound to the same property but with different values.

```
Male :<form:radiobutton path="sex" value="M"/>
<br/>
Female: <form:radiobutton path="sex" value="F"/>
```

Tag Example: Radio Button Tag (Contd.)

HCL TALENTCARE

- HTML snippet of some radio buttons is given below:

```
<td>
Male     : <input name="gender" type="radio"  value="M"/>
Female: <input name="genders" type="radio" value="F"/>
</td>
…
```

Tag Example: Password Tag

HCL TALENTCARE

- This tag renders an HTML *input* tag with type *password* using the bound value.

```
<tr>
    <td>Password:</td>
   <td> <form:password path="password" /> </td>
 </tr>
```

- Note that by default, the password value is *not* shown. To hide the password value, set the value of the *showPassword* attribute to true, as shown below:

```
<tr>
    <td>Password:</td>
   <td> <form:password path="password"
value="^76525bvHGq" showPassword="true"/> </td>
 </tr>
```

Tag Example: Select Tag

- This tag renders an HTML *select* element. It supports data binding to the selected option as well as the use of nested option and options tags.

```
<td>Skills:</td>
<td>
<form:select path="skills" items="${skills}"/>
</td>
```

Tag Example: Select Tag (Contd.)

HCL TALENTCARE

- The snapshot of the generated HTML is given below:

```
<tr>
<td>Skills:</td>
<td>
      <select name="skills" multiple="true">
       <option value="Java">Java</option>
       <option value="Flex"
selected="selected">Flex</option>
       <option value="SOA">SOA</option></select>
</td>
</tr>
…
```

Tag Example: Option Tag

HCL TALENTCARE

- This tag renders an HTML *option*. It sets *selected* as appropriate based on the bound value.

```
<<tr>
<td>Location:</td>
<td>
    <form:select path="location">
        <form:option value="GMR"/>
        <form:option value="ASV"/>
        <form:option value="TCO"/>
        <form:option value="MEPZ"/>
    </form:select> </td>
</tr>
```

Tag Example: Option Tag (Contd.)

HCL TALENTCARE

- The snapshot of the generated HTML is given below:

```
<tr>
<td>Location:</td>
<td>
<select name="location">
<option value="GMR" selected="selected">GMR</option>
<option value="ASV">ASV</option>
<option value="TCO">TCO</option>
<option value="MEPZ">MEPZ</option>
</select>
</td>
</tr>
```

Tag Example: Options Tag

HCL TALENTCARE

- This tag renders a list of HTML *option* tags. It sets the *selected* attribute as appropriate based on the bound value.

```
<tr>
 <td>Country:</td>
 <td> <form:select path="country">
 <form:option value="-" label="--Please Select"/>
 <form:options items="${countryList}"
itemValue="code" itemLabel="name"/> </form:select>
</td>
</tr>
```

Tag Example: Options Tag (Contd.)

HCL TALENTCARE

- The snapshot of the generated HTML is given below:

```
<tr>
<td>Country:</td>
<td>
<select name="country">
<option value="-">--Please Select</option>
<option value="AT">Austria</option>
<option value="UK" selected="selected">United
Kingdom</option>
<option value="US">United States</option>
</select>
</td>
</tr>
```

Tag Example: Textarea Tag

HCL TALENTCARE

- This tag renders an HTML *textarea*.

```
<tr>
<td>Notes:</td>
<td><form:textarea path="notes" rows="3" cols="20"
/></td>
<td><form:errors path="notes" /></td>
</tr>
```

Tag Example: Hidden Tag

HCL TALENTCARE

- This tag renders an HTML *input* tag with type *hidden* using the bound value. To submit an unbound hidden value, use the HTML input tag with type *hidden*.

```
<form:hidden path="house" />
```

Tag Example: Errors Tag

HCL TALENTCARE

- This tag renders field errors in an HTML 'span' tag.
- It provides access to the errors created in the controller or those that were created by any validators associated with the controller.

```
<form:input path="userName" />
        <%-- Show errors for userName field --%>
      <form:errors path="userName" />
```

path="*" - displays all errors
Path="userName" - displays all errors associated with the userName field

Test Your Understanding

- Which of the following is a mandatory attribute of form input tag?

path

id

Lang

title

Test Your Understanding

- Which of the following is a mandatory attribute of form input tag?

path

id

Lang

title

Summary

- Let us take a quick look at the key points covered in this chapter.
  - Spring3.0 provides the very much awaited tag library to create user interface using JSP.
  - The tag library consists of tags to generate HTML tags that are HTML 4.X compliant.
  - These tags are data-binding aware.

Source

- http://static.springframework.org/spring/docs/3.0.x/reference/mvc.html#mvc-formtaglib

Disclaimer: Parts of the content of this course is based on the materials available from the Web sites and books listed above. The materials that can be accessed from linked sites are not maintained by Cognizant Academy and we are not responsible for the contents thereof. All trademarks, service marks, and trade names in this course are the marks of the respective owner(s).

HCL TALENTCARE

Thank You

HCL

HCL TALENTCARE

HCL TALENTCARE

Validator and Exception Handling

Activity

Let us discuss:

How to validate user input data entered in the form?

Spring MVC Validator and Exception handling :Overview

HCL TALENTCARE

Spring features a Validator interface that can be used to validate objects.

The Validator interface works using an errors object. Hence, while validating, validators can report validation failures to the Errors object.

To resolve uncaught exceptions, one or more exception handler beans can be declared in the Web application context.

Objectives

- After completing this chapter, you will be able to:
  - Illustrate how to validate data.
  - Illustrate how to handle exceptions.
  - Explain the use JSR 303 annotations to validate beans.

Validator Interface

- The Validator interface is having two methods. They are:

1. ***supports(Class)****:* To verify the given class is supported by this validator or not.

2. ***validate****(****Object, org.springframework.validation.Errors):*** *To validate the given object and in case of validation errors, register those with the given Errors object.*

Validation Utils

- ValidationUtils Class:
  - The ValidationUtils class is offering some helper methods which will be used for basic validations such as empty, null and empty space.

- Example:

```
public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "ascFstName",
                "required.asc.first.Name", "Associate First name Required");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "ascId", "required.ascId", "Associate
                ID Required");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "ascLastName",
                "required.asc.last.Name", "Associate Last name Required");
}
```

Validator: Example

HCL TALENTCARE

- Validator Example:

```
@Component
public class AssociateValidator implements Validator{
        public boolean supports(Class clazz) {
                return Associate.class.equals(clazz);
}

public void validate(Object target, Errors errors) {
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "ascFstName",
                "required.asc.first.Name", "Associate First name
Required");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors, "ascId",
                "required.ascId", "Associate ID Required");
        ValidationUtils.rejectIfEmptyOrWhitespace(errors,
"ascLastName",
                "required.asc.last.Name", "Associate Last name
Required");
        }
}
```


HCL

ValidationUtils Methods

HCL TALENTCARE

- ValidationUtils class Methods:
  - **rejectIfEmpty (Errors errors, String field, String errorCode)**: Reject the given field with the given error code and message if the value is empty.
  - **rejectIfEmpty(Errors errors, String field, String errorCode, Object[] errorArgs, String defaultMessage):** Reject the given field with the given error code, error arguments and message if the value is empty.
  - **rejectIfEmpty(Errors errors, String field, String errorCode, String defaultMessage):** Reject the given field with the given error code and message if the value is empty.
  - **rejectIfEmptyOrWhitespace**(Errors errors, String field, String errorCode): Reject the given field with the given error code and message if the value is empty or just contains whitespace.

HCL

ValidationUtils Methods (Contd.)

HCL TALENTCARE

- **rejectIfEmptyOrWhitespace**(Errors errors, String field, String errorCode): Reject the given field with the given error code, error arguments and message if the value is empty or just contains whitespace.

- **rejectIfEmptyOrWhitespace** (Errors errors, String field, String errorCode, Object[] errorArgs, String defaultMessage): Reject the given field with the given error code, error arguments and message if the value is empty or just contains whitespace.

- **rejectIfEmptyOrWhitespace** (Errors errors, String field, String errorCode, String defaultMessage): Reject the given field with the given error code and message if the value is empty or just contains whitespace.

Custom Validations

HCL TALENTCARE

Custom Validations: These can be added in the Validator and can be registered with the error object.

The *Error* object stores and exposes information about data-binding and validation errors for a specific object.

Field names can be properties of the target object (For example, *name,* when binding to a customer object), or nested fields in case of subobjects (For example, *asc.address*).

Supports subtree navigation via **setNestedPath(String)**: For example, an AddressValidator validates *address*, not being aware that this is a subobject of associate.

HCL

Custom Validation: Example

HCL TALENTCARE

- Example: To verify the length of the associate ID.

```
public void validate(Object target, Errors errors) {
        Associate associateDetails = (Associate)target;
        String ascIdString = associateDetails.getAscId().toString();
        if(ascIdString.length()<6)
        {
        errors.rejectValue("ascId","asc.id.minimum.length",
         "Associate ID should be  given with 6 digits");
        }
  }
```


HCL

Validators

HCL TALENTCARE

- How to call this validator:

The validator will be registered with the Controller.

The validate method of the validator will be called from the controller.

The validation messages are registered with the error object if the validation fails.

The view rendering will be based on the return value from the validator.

Validator: Example

HCL TALENTCARE

- Example:

```
@RequestMapping("addAssociate")

public ModelAndView addAssociate(@ModelAttribute("associate")
Associate associate,

    BindingResult result) {

        associateValidator.validate(associate, result);

        ModelAndView  modelAndView = new
        ModelAndView("addAssociateSubPageContents");

        modelAndView.addObject("associate", associate);
```


HCL

Validator: Example

```
   if(result.hasErrors()){
   return modelAndView;
}
associateService.addAssociate(associate);
return new ModelAndView("successAddAsociate");
   }
```

Validations with Annotations (JSR 303): Slide

HCL TALENTCARE

Beginning with Spring 3, Spring MVC has the ability to automatically validate @Controller inputs.

JSR-303 or bean validation is a specification whose objective is to standardize the validation of Java beans through annotations.

This allows validation rules to be specified directly in the code they are intended to validate, instead of creating validation rules in separate classes.

To configure a JSR-303-backed Validator with Spring MVC, simply add a JSR-303 Provider, such as Hibernate Validator, to your classpath.

Spring MVC will detect it and automatically enable JSR-303 support across all Controllers.

Validations with Annotations (JSR 303): Slide

HCL TALENTCARE

- The Spring MVC configuration required to enable JSR-303 support is shown below:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance”
xsi:schemaLocation=“http://www.springframework.org/schema/beans
                    http://www.springframework.org/schema/beans/spring-
beans-3.0.xsd
                    http://www.springframework.org/schema/mvc
                    http://www.springframework.org/schema/mvc/spring-mvc-
3.0.xsd”>

<!-- JSR-303 support will be detected on classpath and enabled automatically -->
<mvc:annotation-driven/>

</beans>
```

With this minimal configuration, anytime a @Valid @Controller input is encountered, it will be validated by the JSR-303 provider.

HCL

Validations with Annotations (JSR 303):

HCL TALENTCARE

- The Associate class now have JSR 303 Annotations:

```
import javax.validation.constraints.Size;
import org.hibernate.validator.constraints.NotEmpty;
public class  Associate{
                @NotEmpty
                @Size(min = 1, max = 50)
                private String ascFstName;

                   @NotEmpty
               @Size(min = 1, max = 50)
                private String ascLastName;

                     //getters and setters…

            }
```

Validations with Annotations (JSR 303):

HCL TALENTCARE

- The AssociateController will be as follows:

```
import javax.validation.Valid;
…
@Controller
public class AssociateController {

@RequestMapping("addAssociate")
public ModelAndView addAssociate(@Valid Associate associate, BindingResult result) {
    ModelAndView  modelAndView = new ModelAndView("addAssociateSubPageContents");
    modelAndView.addObject("associate", associate);
    If(result.hasErrors()){
            return modelAndView;
    }
associateService.addAssociate(associate);
return new ModelAndView("successAddAsociate");
  }
}
```

- Any ConstraintViolations will automatically be exposed as errors in the BindingResult renderable by standard Spring MVC form tags.

HCL

Activity

HCL TALENTCARE

1. Identify from the following the class which provides a few helper methods to perform some common validations.

- ValidatorUtils
- ValidationUtils
- Validator
- None of the above

2. If you are configuring Spring MVC with <mvc:annotation-driven/> then it will automatically configure a JSR 303 validator, if some JSR 303 validator implementation is present in the classpath.

- True
- False

Exception Handling

HCL TALENTCARE

Exception Handling:

To resolve uncaught exceptions, one or more exception handler beans can be declared in the Web application context.	The declared beans will implement HandlerExceptionResolver to help the DispatcherServlet to detect them automatically.	To display different errors on different pages, **SimpleMappingExceptionResolver has exception mappings in** the web application context.	If any of the exception declared in the exceptionMappings property is not matching then default exception page is rendered. This is declared with *defaultErrorView property.*

Exception Handling Example: Slide 1

HCL TALENTCARE

- Example:

```
public class AssociateNotAvailableException extends
RuntimeException {
        private String ascName;
        private Date date;
        private int hour;
        …..
        …..
        …..
    }
```

Exception Handling Example: Slide 2

HCL TALENTCARE

Exception bean Declaration:

```
<bean
    class="org.springframework.web.servlet.handler.SimpleMappingExcepti
    onResolver">
        <property name="exceptionMappings">
            <props>
                <prop
    key="com.cts.AssociateNotAvaialableException">
                    associateNotAvailable
                </prop>
            </props>
        </property>
        <property name="defaultErrorView" value="error" />
</bean>
```

Exception Handling Example: Slide 3

HCL TALENTCARE

- Exception page: associateNotAvailable.jsp

```
<%@ taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
<html>
        <head>
          <title>Associate Not Available</title>
        </head>
        <body>
          ${exception.ascName} is not  available on
          <fmt:formatDate value="${exception.date}" pattern="yyyy-MM-
    dd" /> at
          ${exception.hour}:00 for the task assignment.
        </body>
</html>
```

Test Your Understanding

1. The validation messages are registered with the error object if the validation fails.

True

False

Test Your Understanding

1. The validation messages are registered with the error object if the validation fails.

True

False

Test Your Understanding

2. What is the property name of the *SimpleMappingExceptionResolver* to display the default error page?

- CustomErrorView
- DefaultErrorView
- UserErrorView
- None of the above

Test Your Understanding

HCL TALENTCARE

2. What is the property name of the *SimpleMappingExceptionResolver* to display the default error page?

- CustomErrorView
- DefaultErrorView
- UserErrorView
- None of the above

Summary

HCL TALENTCARE

- Let us take a look at the key points covered in this chapter:
  - Validator interface can be used to validate objects.
  - The ValidationUtils class is offering some helper methods which will be used for basic validations such as empty, null and empty space.
  - JSR 303 provides a convenient way to validate @controller beans.
  - To resolve uncaught exceptions, we can declare one or more exception handler beans in the Web application context.

HCL

Source

- “Spring Recipes : A problem - solution approach” by Gary Mak, Josh Long, and Daniel Rubio

- www.springframework.org

Disclaimer: Parts of the content of this course is based on the materials available from the Web sites and books listed above. The materials that can be accessed from linked sites are not maintained by Cognizant Academy and we are not responsible for the contents thereof. All trademarks, service marks, and trade names in this course are the marks of the respective owner(s).

HCL TALENTCARE

Thank You

HCL