Basic Spring 4.0

Lesson 4: Spring MVC framework

Lesson Objectives

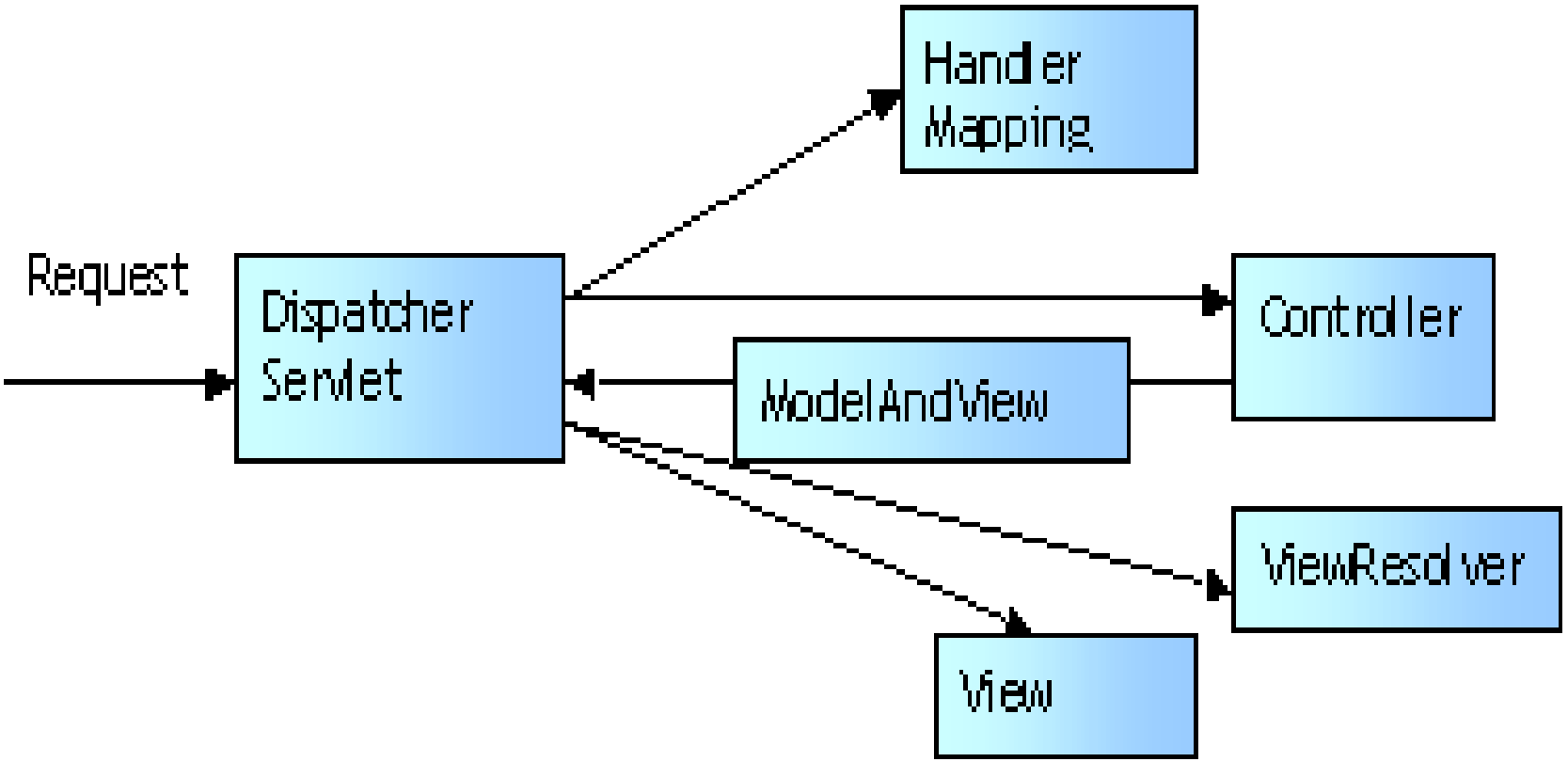
- Introduction to Spring MVC framework
  - Learn how to develop web applications using Spring
  - Understand the Spring MVC architecture and the request cycle of Spring web applications
  - Understand components like handler mappings, ViewResolvers and controllers
  - Use MVC Annotations like @Controller, @RequestMapping and @RequestParam
  - Introduction to REST web Services
  - REST Controllers on the top of MVC

Spring MVC Framework Features

- Provides you with an out-of-the-box implementations of workflow typical to web applications
- Allows you to use a variety of different view technologies
- Enables you to fully integrate with your Spring based, middle-tier logic through the use of dependency injection
- Displays modular framework, with each set of components having specific roles and completely decoupled from the rest of the framework

Spring MVC lifecycle

- Life cycle of a Request in Spring MVC



Dispatcher Servlet

- The central component of Spring MVC is DispatcherServlet .
- It acts as the front controller of the Spring MVC framework
- Every web request must go through it so that it can manage the entire request-handling process.

Configuring the 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

- Steps to build a homepage in Spring MVC:
  - Write the controller class that performs the logic behind the homepage
  - Configure controller in the DispatcherServlet's context configuration file
  - Configure a view resolver to tie the controller to the JSP
  - Write the JSP that will render the homepage to the user

Breaking Up the Application Context

- Configuring the ContextLoaderListener in web.xml

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

- Setting ContextConfigLocation Parameter

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

Controller

- The most typical handler used in Spring MVC for handling web requests is a controller.

- From Spring2.5 onwards @Controller annotation is used to design a controller class

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

Implementing Controllers

Annotation Name	Description
@Controller	Indicates that an annotated class is a "Controller"
@RequestMapping	Map web requests onto specific handler classes and/or handler methods.
@RequestParam	@RequestParam annotation is used to retrieve the URL parameter and map it to the method argument.
@ModelAttribute	Annotation that binds a method parameter or method return value to a named model attribute, exposed to a web view.

Handler Mapping

- A handler mapping is a bean configured in the web application context that implements the HandlerMapping interface. It is responsible for returning an appropriate handler for a request.
- Handler mappings usually map a request to a handler according to the request's URL. It is an arbitrary Java object that can handle web requests.
- The most typical handler used in Spring MVC for handling web requests is a controller.

ModelAndView Class

- This class fully encapsulates the view and model data that is to be displayed by the view. Eg:

```
ModelAndView("hello","now",now);
```

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

- Every controller returns a ModelAndView
- Views in Spring are addressed by a view name and are resolved by a view resolver

ModelAndView

- After a controller has finished handling the request, it returns a model and a view name, or sometimes a view object, to DispatcherServlet.
- The model contains the attributes that the controller wants to pass to the view for display.
- If a view name is returned, it will be resolved into a view object for rendering. The basic class that binds a model and a view is ModelAndView.

```
@Controller
 public class HelloController {
    @RequestMapping("/helloWorld")
    public String handleMyRequest(
            Map<String, Object> model) {
     String now = new java.util.Date().toString();
      model.put("now", now);
      return "hello";
}}
```

```
@Controller
  public class HelloController {
     @RequestMapping("/helloWorld")
public ModelAndView handleRequest(
        HttpServletRequest request,
        HttpServletResponse response)
         throws ….. {
   String now = new java.util.Date().toString();
 return new ModelAndView(
                              "hello", "now", now  );
    }}
```

ViewResolver

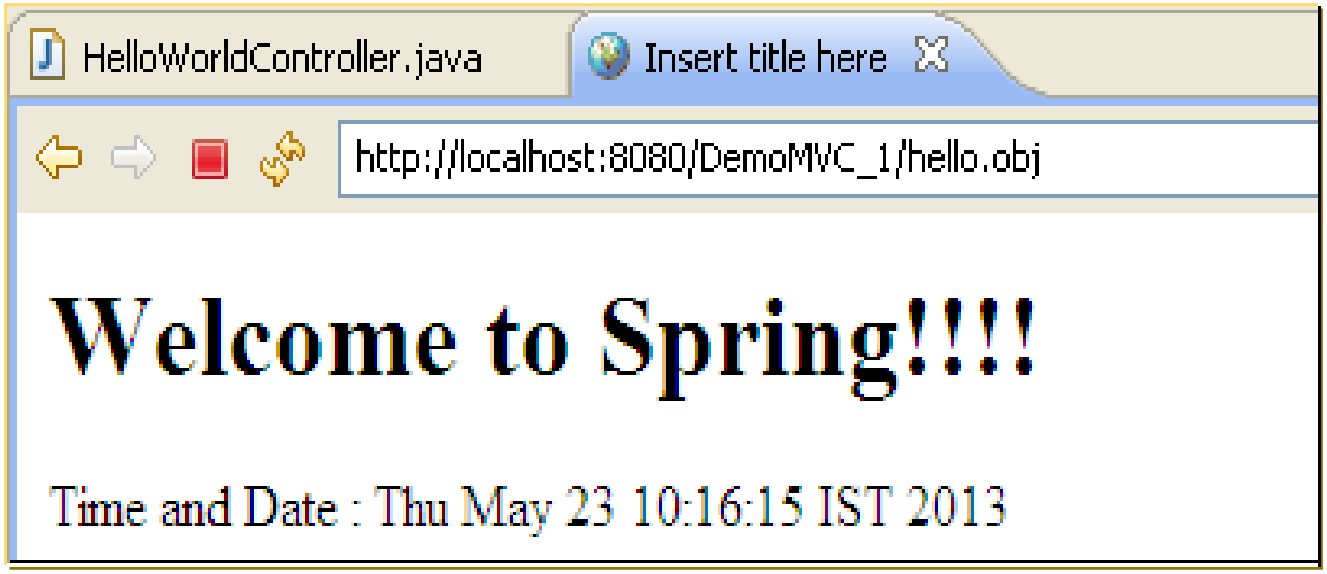
- When DispatcherServlet receives a model and a view name, it will resolve the logical view name into a view object for rendering.
- DispatcherServlet resolves views from one or more view resolvers.
- A view resolver is a bean configured in the web application context that implements the ViewResolver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

Resolving Views: The ViewResolver

View resolver	How it works
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

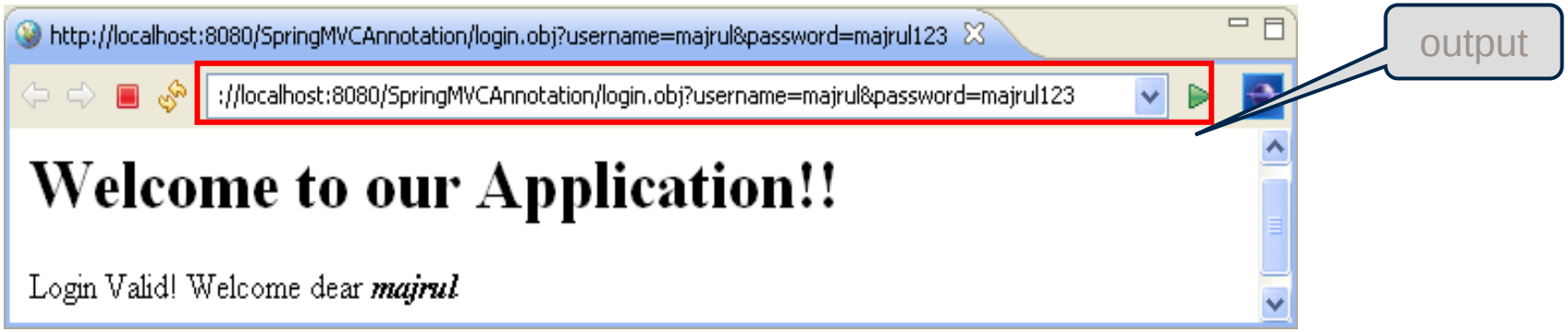
- Refer DemoMVC_1



Handling User Input

```
@Controller
public class LoginFormController {
   @RequestMapping(value = "/login", method = RequestMethod.GET)
   public String onSubmit(@RequestParam("username") String username,
                          @RequestParam("password") String password, Model model) {

      model.addAttribute("username", username);
      if (username.equals("majrul") && password.equals("majrul123"))
            return "success";
      else  return "failure";
}}
```



Demo

- Refer DemoMVC_2 application

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

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
}
```

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>
```

Displaying validation errors

jsp

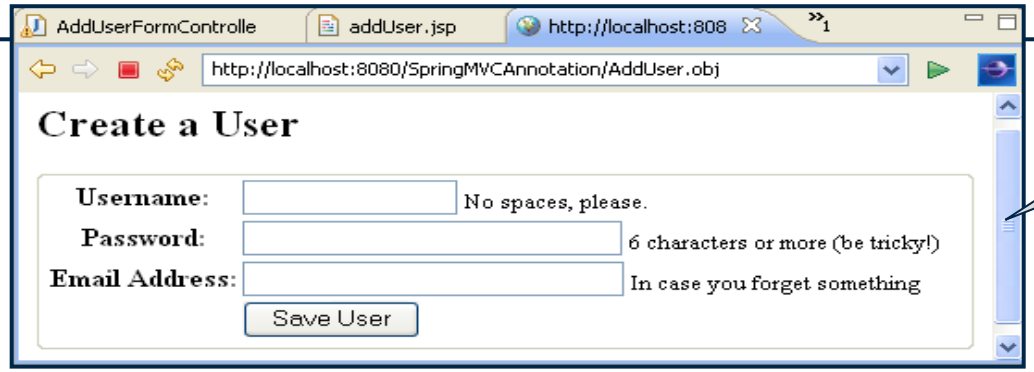
```
<td>
    <sf:password path="password" size="30" showPassword="true"/>
        <small>6 characters or more (be tricky!)</small><br/>
        <sf:errors path="password" />
</td>
```

controller

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

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";
}}
```



addUser.jsp

Demo

- Refer the following Demos:
  - DemoMVC_3
  - DemoMVC_4
  - DemoMVC_5
  - DemoMVC_6
  - DemoMVC_7
  - DemoMVC_Complete

Introduction to REST web Services

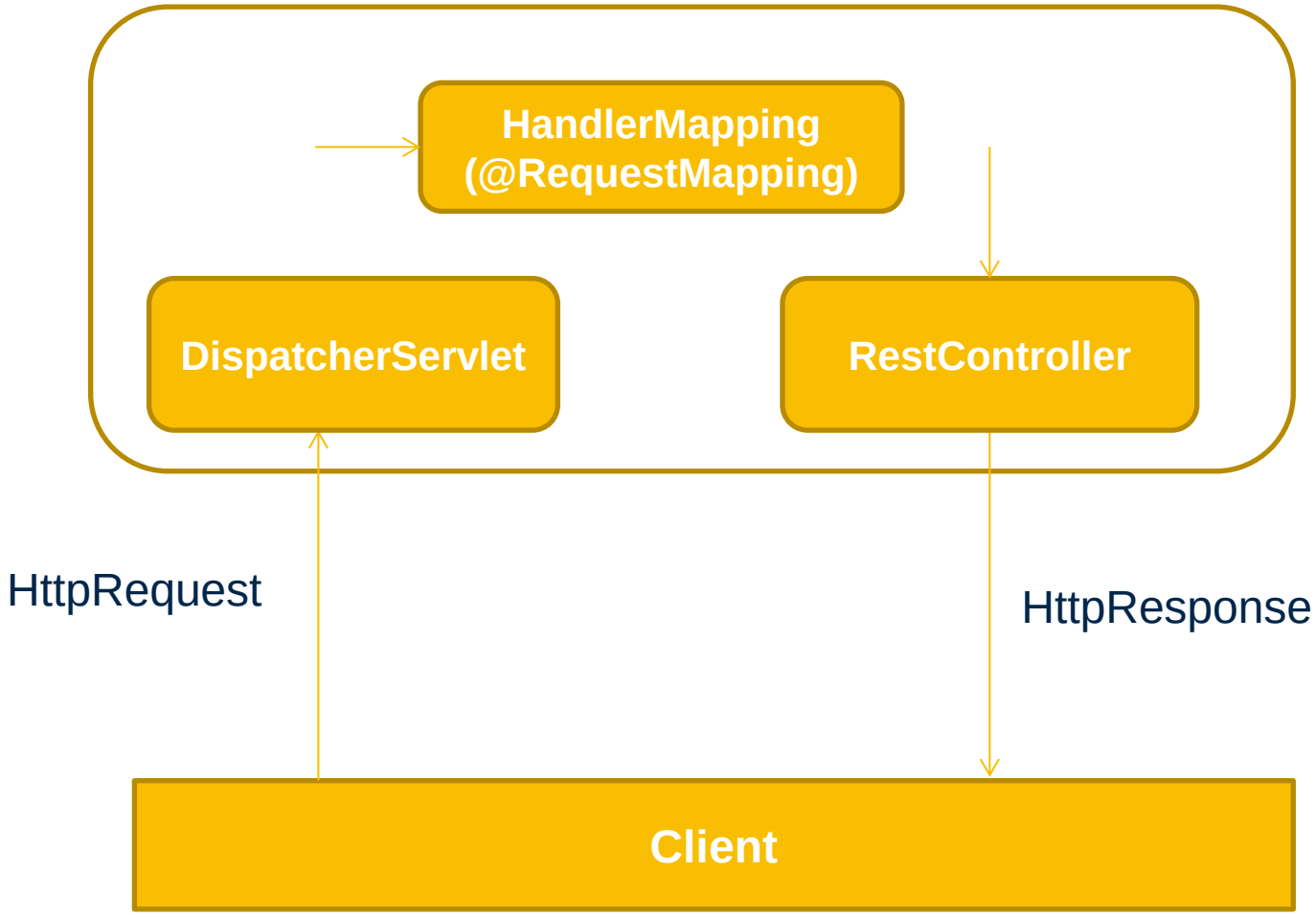
- ReST : Representational state transfer
  - Is an architectural style of designing loosely coupled Web applications that rely on named resources rather than messages.
  - Is a lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP)
  - In a good REST design operations are self-contained, and each request carries with it (transfers) all the information (state) that the server needs in order to complete it.
  - REST leverages aspects of the HTTP protocol to standard business-application needs. So:

Application task	HTTP command
Create	POST
Read	GET
Update	PUT
delete	DELETE

Introduction to REST web Services

- Principles of REST web Services
  - Use HTTP methods explicitly
  - Be stateless
  - Expose directory structure-like URIs
  - Transfer XML, JavaScript Object Notation (JSON), or both

Life cycle of a Request in Spring MVC Restful



Why REST Controller ?

- Traditional Spring MVC controller and the RESTful web service controller differs in the way the HTTP response body is created
- Traditional MVC controller relies on the View technology
- RESTful controller simply returns the object and the object data is written directly to the HTTP response as JSON/XML

Spring 4 support for RESTful web services

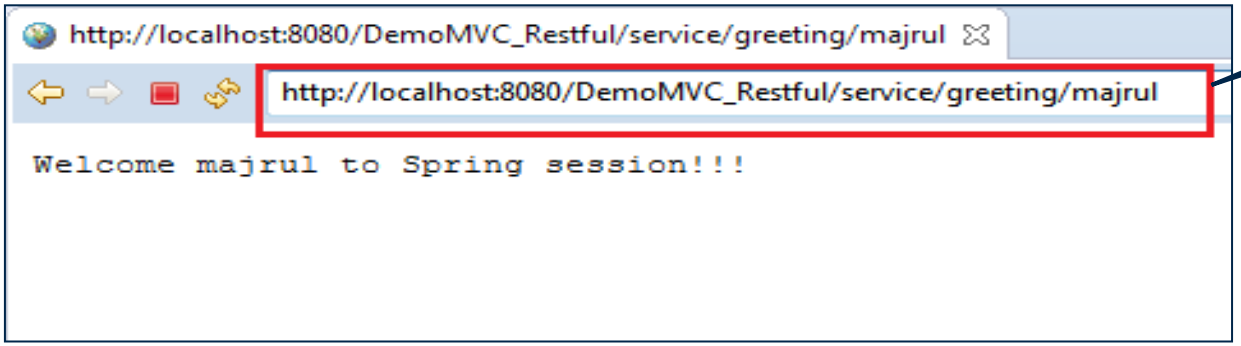
- In Spring 4 REST is built on the top of MVC
  - REST methods: GET, PUT, DELETE, and POST, can be handled by Controllers
  - Using @PathVariable annotation controllers can handle requested parameterized URLs

MVC (RESTful) controller

```
@RestController
@RequestMapping("/service/greeting")
public class SpringRestController {
        @RequestMapping(value = "/{name}", method = RequestMethod.GET)
        public String sayHello(@PathVariable Optional<String> name) {
                String result = “Welcome " + name.get() + " to Spring session!!!";
                return result;
        }
}
```

No “name” required. “Do not repeat yourself” with Java 8 version

output



RESTful URLs – HTTP methods

Demo: DemoMVC_Restful

- DemoMVC_Restful
- SpringRESTWebServices

Summary

- We have so far seen:
  - 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

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