Spring MVC Form Submission

"Code with Passion!"

Topics

- 2-phase form submission handling
- Command/form objects
- @ModelAttribute
- Validation
- Redirection (in form submission handling)

2-Phase Form Submission Handling

Form Display & Submission Handling

- It is a two-phase process

- Phase 1: Handle initial form display request (GET request)
  - (1.1) Creation and initialization of Command object
  - (1.2) Display the form page

- Phase 2: Handle the form submission (POST request)
  - (2.1) Data binding and type conversion
  - (2.2) Validation
  - (2.3) Business logic handling with input data
  - (2.4) Redirect

Phase 1: Handle initial form display request

```
@Controller
@RequestMapping(value="/account")
public class AccountController {

    // Phase 1: Handle initial form display request
    @GetMapping
    public String getCreateForm(Model model) {

        // (1.1) Create command object “account”
        model.addAttribute(“account”, new Account());

        // (1.2) Return a logical view “account/createForm”, which results in
        // displaying “account/createForm.jsp”
        return "account/createForm";

    }
```

Phase 1: ./account/createForm.jsp Displayed (JSP)

```
<form:form modelAttribute="account" action="account" method="post">
   <fieldset>
      <legend>Account Fields</legend>
      <p>
            <form:label             for="name" path="name"
                              cssErrorClass="error">Name</form:label><br/>
            <form:input path="name" /> <form:errors path="name" />
      </p>
      <p>
            <form:label for="balance" path="balance"
                        cssErrorClass="error">Balance</form:label><br/>
            <form:input path="balance" /> <form:errors path="balance" />
      </p>
      <p>
            <form:label for="equityAllocation" path="equityAllocation"
                           cssErrorClass="error">Equity Allocation</form:label><br/>
            <form:input path="equityAllocation" /> <form:errors path="equityAllocation" />
      </p>
      <p>
            <form:label for="renewalDate" path="renewalDate"
                        cssErrorClass="error">Renewal Date</form:label><br/>
            <form:input path="renewalDate" /> <form:errors path="renewalDate" />
      </p>
      <p>
            <input type="submit" />
      </p>
   </fieldset>
</form:form>
```

Phase 2: Form is submitted

```
<form:form modelAttribute="account" action="account" method="post">
   <fieldset>
      <legend>Account Fields</legend>
      <p>
            <form:label            for="name" path="name"
                              cssErrorClass="error">Name</form:label><br/>
            <form:input path="name" /> <form:errors path="name" />
      </p>
      <p>
            <form:label for="balance" path="balance"
                        cssErrorClass="error">Balance</form:label><br/>
            <form:input path="balance" /> <form:errors path="balance" />
      </p>
      <p>
            <form:label for="equityAllocation" path="equityAllocation"
                           cssErrorClass="error">Equity Allocation</form:label><br/>
            <form:input path="equityAllocation" /> <form:errors path="equityAllocation" />
      </p>
      <p>
            <form:label for="renewalDate" path="renewalDate"
                        cssErrorClass="error">Renewal Date</form:label><br/>
            <form:input path="renewalDate" /> <form:errors path="renewalDate" />
      </p>
      <p>
            <input type="submit" />
      </p>
   </fieldset>
</form:form>
```

sends the post request to /account URL

Phase 2: Handle form submission

```
@Controller
@RequestMapping(value="/account")
public class AccountController {

    ...

    // Phase 2: Handle form  submission
    @PostMapping
    // (2.1) Data Binding and type conversion
    // (2.2) Validation are performed by Spring framework before this method gets called
    public String create(@Valid Account account, BindingResult result) {

        if (result.hasErrors()) {
            return "account/createForm";
        }

        (2.3) Some business logic handling
        this.accounts.put(account.assignId(), account);

        (2.4) Redirect
        return "redirect:/account/" + account.getId();
    }
```

Lab:

Exercise 1: Simple form
4946_spring4_mvc_form.zip

Command/Form Objects

Model Attribute Gets Created In Initial Form Display Request Handling

```
// Handle initial form request
@GetMapping
public String getCreateForm(Model model) {
        // “Account” object will be accessible as form object in the “account/createForm.jsp”
        model.addAttribute("account", new Account());
        return "account/createForm";
}

// Handle form submission
@PostMapping
public String create(Account account, BindingResult result) {
        // If there is an error either in validation or data binding,
        // display the "account/createForm.jsp" page again.
        if (result.hasErrors()) {
                return "account/createForm";
        }

        // If there is no error, add the newly created account into
        // the "accounts" table and then redirect to the /account/{id},
        // which will be handled in the "getView(..) method below.
        this.accounts.put(account.assignId(), account);
        return "redirect:/account/" + account.getId();
}
```

Model Attribute is used when Form page gets displayed in Phase 1 (JSP)

```
<form:form modelAttribute="account" action="account" method="post">
 <fieldset>
  <legend>Account Fields</legend>
  <p>
    <form:label    for="name" path="name" cssErrorClass="error">Name</form:label><br/>
    <form:input path="name" /> <form:errors path="name" />
  </p>
  <p>
    <form:label for="balance" path="balance" cssErrorClass="error">Balance</form:label><br/>
    <form:input path="balance" /> <form:errors path="balance" />
  </p>
  <p>
    <form:label for="equityAllocation" path="equityAllocation" cssErrorClass="error">Equity Allocation</form:label><br/>
    <form:input path="equityAllocation" /> <form:errors path="equityAllocation" />
  </p>
  <p>
    <form:label for="renewalDate" path="renewalDate" cssErrorClass="error">Renewal Date</form:label><br/>
    <form:input path="renewalDate" /> <form:errors path="renewalDate" />
  </p>
  <p>
    <input type="submit" />
  </p>
</fieldset>
</form:form>
```

Form Objects

- Form objects bind HTTP request parameters to bean properties
  - With type conversion and validation

Form Object gets used in Form Submission Handling

```
// Handle initial form request
@GetMapping
public String getCreateForm(Model model) {

	model.addAttribute("account", new Account());
	return "account/createForm";
}

// Handle form submission
@PostMapping
// Form values are bound to the properties of “Account” form object.
public String create(@ModelAttribute(“account”) Account account, BindingResult result) {
	// If there is an error either in validation or data binding,
	// display the "account/createForm.jsp" page again.
	if (result.hasErrors()) {
		return "account/createForm";
	}

	// If there is no error, add the newly created account into
	// the "accounts" table and then redirect to the /account/{id},
	// which will be handled in the "getView(..) method below.
	this.accounts.put(account.assignId(), account);
	return "redirect:/account/" + account.getId();
}
```

@ModelAttribute

@ModelAttribute

- *@ModelAttribute* has two usage scenarios in controllers

- Usage #1 - annotating a method for creating model attribute
  - *@ModelAttribute* provides reference data for the model

- Usage #2 - annotating a method parameter for accessing existing model attribute (if it exists already) or create a model attribute (id it does not exist)
  - *@ModelAttribute* maps a model attribute to the specific, annotated method parameter

@ModelAttribute annotating a method (Usage Model #1) for creating model attr

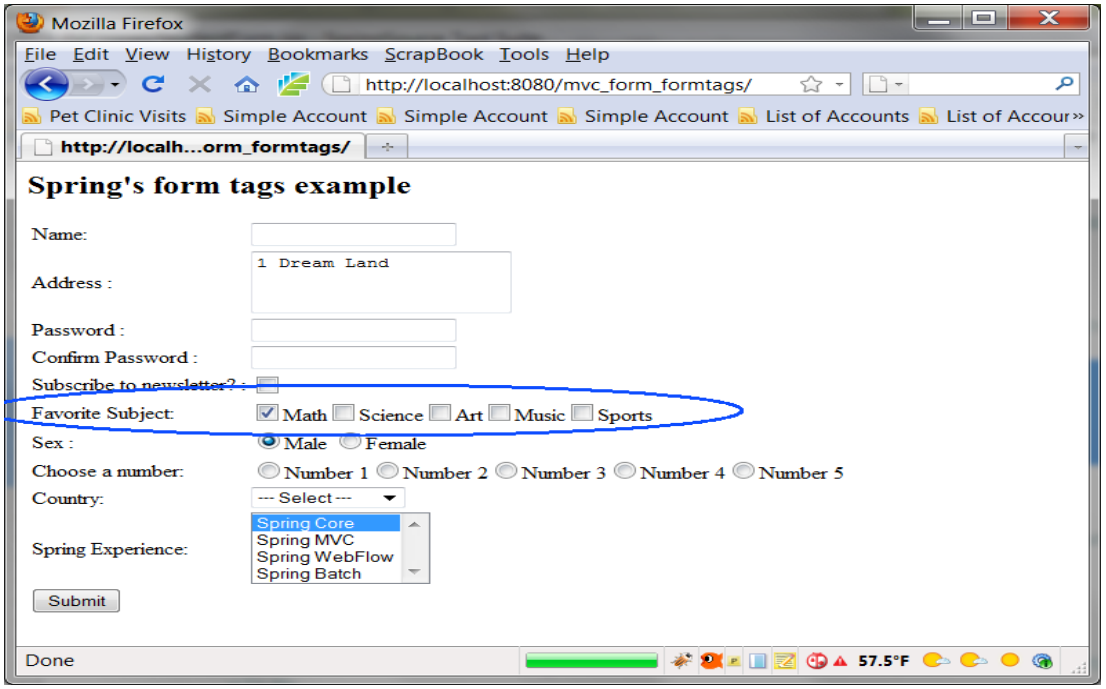
- Used to create a Model attribute through a method
- *@ModelAttribute ("<attribute-name>")* annotated methods are executed before the chosen *@RequestMapping* annotated handler method.
  - > They effectively pre-populate the implicit model with specific attributes, often loaded from a database
  - > It could be then accessible in a view
- (Example in the following slide)

@ModelAttribute annotating a method (Usage Model #1)

```
// Create “subjectList” model attribute from the return value
// of the “populateSubjectList()” method. The “subjectList” model
// attribute is then used in the view as reference data.
@ModelAttribute("subjectList")
public List<String> populateSubjectList() {

        //Data referencing for web framework checkboxes
        List<String> subjectList = new ArrayList<String>();
        subjectList.add("Math");
        subjectList.add("Science");
        subjectList.add("Art");
        subjectList.add("Music");

        return subjectList;
}

@ModelAttribute("insuranceCommand")
public InsuranceCommand getInsuranceCommand(){
        InsuranceCommand insuranceCommand = new InsuranceCommand();
        return insuranceCommand;
}
```

@ModelAttribute annotating a method (Usage Model #1)

```
<tr>
	<td>Favorite Subject:</td>
	<td><form:checkboxes items="${subjectList}"
		path="subject" /></td>
	<td><form:errors path="subject" cssClass="error" /></td>
</tr>
```



@ModelAttribute annotating a method parameter (Usage Model #2)

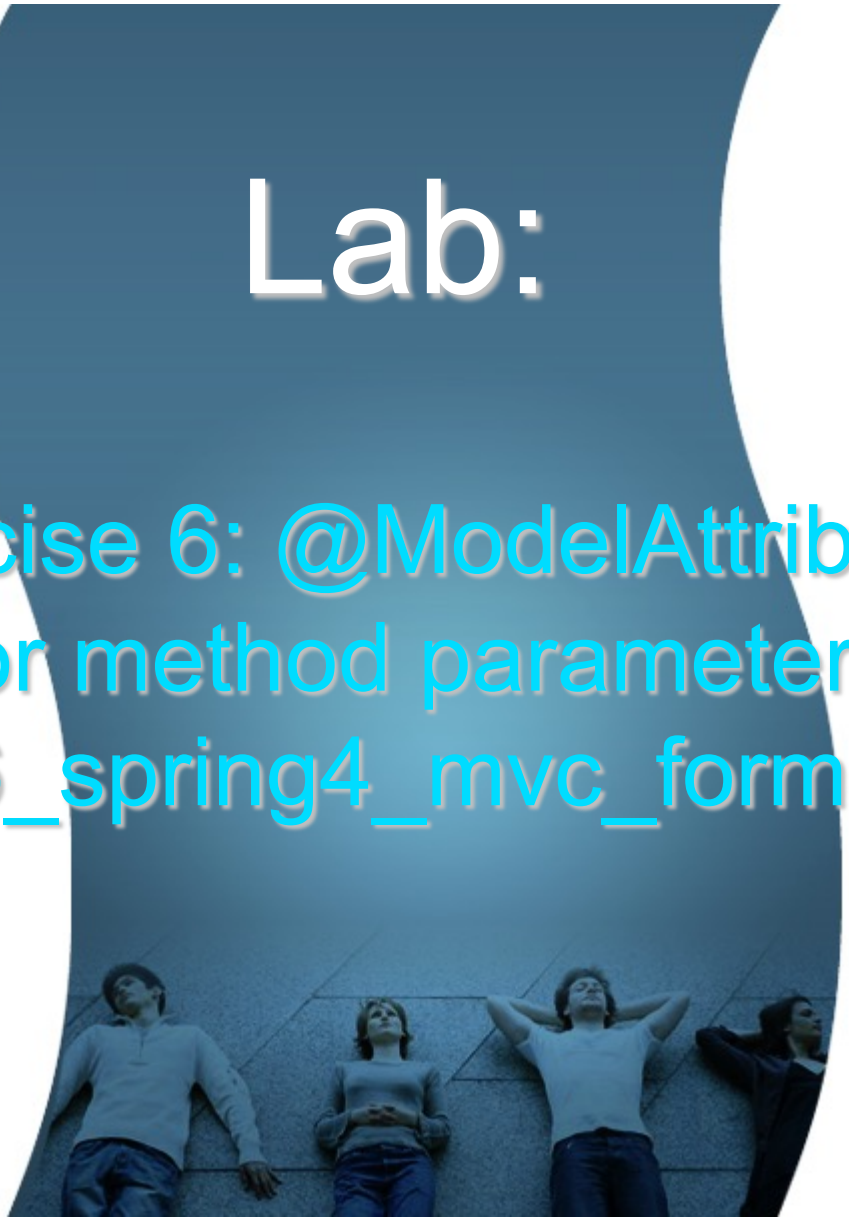
- If model attribute already exists, you can access it
  - Example: once a model attribute is created via Usage model #1, such an attribute can then be accessed through *@ModelAttribute* annotated handler method parameters in a handler method, potentially with binding and validation applied to it
- Otherwise (if the model attribute does not exist), it gets created and you can access it

@ModelAttribute annotating a method parameter (Usage Model #2)

```
@RequestMapping(value="{id}", method=RequestMethod.GET)
public String getView(@PathVariable Long id,
        // Access “subjectList” pre-existing model attribute as “subjectList2”
        @ModelAttribute("subjectList") List<String> subjectList2,
        Model model) {

    Account account = this.accounts.get(id);
    if (account == null) {
        throw new ResourceNotFoundException(id);
    }
    model.addAttribute("account", account);
    model.addAttribute("subjectList2", subjectList2);

    //return "account/view";              // Display account detail in form format
    return "account/view2";             // Display account detail in table format
}
```

Lab:

Exercise 6: @ModelAttribute for method parameter 4946_spring4_mvc_form.zip



Validation

Validation

- To trigger validation of a *@Controller* input, simply annotate the input argument with *@Valid* (from JSR 303)

```
@Controller
public class MyController {

    @RequestMapping(method=RequestMethod.POST)
    public void processFoo(@Valid Account account,
                                        BindingResult result) {

        if (bindingResult.hasErrors()) {
            return ”account/new_account";
        }

}
```

Domain Class with Validation Annotation

```
public class Account {

    private Long id;

    @NotNull
    @Size(min=1, max=25)
    private String name;

    @NotNull
    @NumberFormat(style=Style.CURRENCY)
    private BigDecimal balance = new BigDecimal("1000");

    @NotNull
    @NumberFormat(style=Style.PERCENT)
    private BigDecimal equityAllocation = new BigDecimal(".60");

    @DateTimeFormat(style="S-")
    @Future
    private Date renewalDate = new Date(new Date().getTime() + 31536000000L);

    public Long getId() {
            return id;
    }
```

Controller Class

```
@Controller
@RequestMapping(value="/account")
public class AccountController {

    private Map<Long, Account> accounts =
            new ConcurrentHashMap<Long, Account>();

    // Handle initial form request
    @RequestMapping(method=RequestMethod.GET)
    public String getCreateForm(Model model) {
        model.addAttribute(new Account());
        return "account/createForm";
    }

    // Handle form request
    @RequestMapping(method=RequestMethod.POST)
    public String create(@Valid Account account, BindingResult result) {
        if (result.hasErrors()) {
            return "account/createForm";
        }
        this.accounts.put(account.assignId(), account);
        return "redirect:/account/" + account.getId();
    }
```

Lab:

Exercise 2: Validator 1
Exercise 3: Validator 2
4946_spring4_mvc_form.zip

Redirect (in Form Submission Handling)

Post Redirect Get (PRG)

- Issue an HTTP redirect back to the client, before the view is rendered.
- Use cases
  - case #1: To eliminate the possibility of the user submitting the form data multiple times through refreshing
  - case #2: When one controller has been called with POST'ed data, and the response is actually a delegation to another controller (for example on a successful form submission)
- Return “redirect:<destination-URL>”
  - The UrlBasedViewResolver will recognize this as a special indication that a redirect is needed. The rest of the view name will be treated as the redirect URL.

Example Redirect

```
// Inside of a controller

// Handle initial form request
@RequestMapping(method=RequestMethod.GET)
public String getCreateForm(Model model) {
   model.addAttribute(new Account());
   return "account/createForm";
}

// Handle form request
@RequestMapping(method=RequestMethod.POST)
public String create(@Valid Account account, BindingResult result) {
   if (result.hasErrors())
        return "account/createForm";
   }
   this.accounts.put(account.assignId(), account);
   return "redirect:/account/" + account.getId();
}
```

Lab:

Exercise 7: Redirection and Refreshing a page

4946_spring4_mvc_form.zip