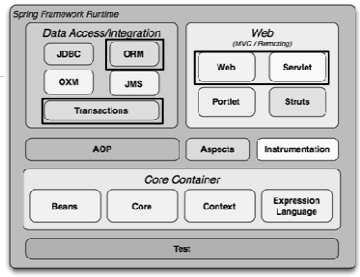
CS520 Web Programming
Spring – Web MVC

Chengyu Sun
California State University, Los Angeles

Spring Framework



Roadmap

- Introduction to Spring MVC
- Database access
- Controllers
- Message bundle and input validation

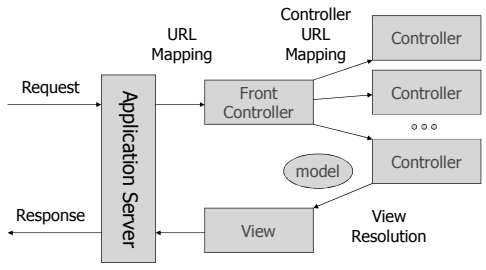
The SpringMVC Example

- Spring and Hibernate from Scratch
  - http://csns.calstatela.edu/wiki/content/cysun/course_materials/cs520/sham/

Add Spring MVC to A Maven Webapp

- Spring dependencies
  - `spring-webmvc`
- Front controller
  - `DispatcherServlet` in web.xml
- Bean configuration file
  - /WEB-INF/<servlet-name>-servlet.xml

Request Processing in Spring Web MVC



Spring Controllers

- Spring controllers are *beans* annotated with `@Controller`
- In <servlet-name>-servlet.xml
  - `<mvc:annotation-driven>`
  - `<context:component-scan>`

Controller URL Mapping

- Requests are mapped to controller methods using `@RequestMapping`
  - `value`: URL pattern(s)
  - Mapping can be further refined by using `method`, `params`, and `headers`
  - http://static.springsource.org/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-ann-requestmapping

View

- A controller method returns a view name, which will be resolved to a view implementation by a view resolver

Resolve JSP Views

```
<bean id="viewResolver"
   class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/jsp/" />
      <property name="suffix" value=".jsp" />
</bean>
```

Prefix + ViewName + Suffix = JSP File

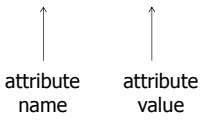
Why not just return the path to the JSP file??

View Technologies and Resolvers

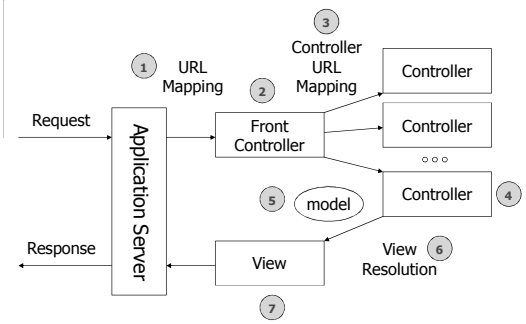
- http://static.springsource.org/spring/docs/current/javadoc-api/org/springframework/web/servlet/ViewResolver.html

Model

- Model objects are passed to view using a `ModelMap`
  - `put( String, Object )`



Request Processing Recap



Database Access Dependencies

- Hibernate
- Spring ORM
- Database connection pooling
  - DBCP vs. Tomcat JDBC
- Database driver

Configuration

- Hibernate JPA
  - persistence.xml
- Spring
  - web.xml
  - applicationContext.xml

Spring Transaction Management

- Transaction management code is added through AOP
- <context:annotation-config> enables annotations like `@Autowired` and `@PersistenceContext`
- <tx:annotation-driven> enables annotations like `@Transactional`

Model Classes and Database

- Model classes, e.g. `User`
  - JPA annotations
- Database
  - SQL scripts
  - `hibernate_sequence`

Database Access

- DAO interfaces, e.g. `UserDao`
- DAO implementations, e.g. `UserDaoImpl`

Spring DAO Annotations

- @Repository for DAO implementation classes
- @PersistenceContext injects an entity manager to the class
- @Transactional for methods that write to the database

Controller Examples

- List all users
- Display a selected user
- Add a new user
- Edit a user

Example: List All Users

- Controller basics
  - `@Controller`
  - `@RequestMapping`
  - `@ModelMap`
  - Using DAO

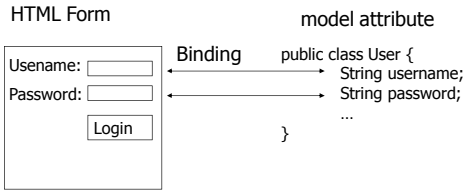
Example: Display A User

- Using `@RequestParam`
  - `required`
- Using `@PathVariable`

Example: Add A User

- Model attribute (a.k.a. command object, form backing object)
  - Concept
  - Binding
  - `@ModelAttribute`
- `@RequestMapping`
  - `method`
- The "redirect" view

Command Object



- Bind form fields to properties of the object

Handling Forms

- One controller method for
  - Creating a model attribute
  - Returning the form view
- Form view
  - Use Spring `<form>` tags to bind the model attribute to the form
- One controller method for
  - Processing the command object (annotated with `@ModelAttribute`)
  - Returning the success view

Spring's `form` Tag Library

- Documentation - http://static.springsource.org/spring/docs/current/spring-framework-reference/html/view.html#view-jsp-formtaglib
- Tag reference – http://static.springsource.org/spring/docs/current/spring-framework-reference/html/spring-form.tld.html

Example: Edit A User

- Store model attribute in session
  - `@SessionAttributes`
  - `SessionStatus`
    - Use `setComplete()` to remove the model attribute in session

Non-Session Model Attributes

Form request	Form Submission
Create a model attribute in controller	Create a model attribute
↓	↓
	Populate the model attribute with submitted data
↓	↓
Use the model attribute to populate the form fields in view	Pass the model attribute to controller

Session Model Attributes

Form request	Form Submission
Create a model attribute in controller	Retrieve the model attribute from session
↓	↓
Save the model attribute in session	Populate the model attribute with submitted data
↓	↓
Use the model attribute to populate the form fields in view	Pass the model attribute to controller

Message Bundles

- Separate text messages from application code and put them into their own files
  - E.g. `messages.properties`

```
error.username.required=A username is required.
error.password.short=The password is too short.
error.username.taken=The username {0} is already taken.
```

Advantages of Using Message Bundles

- Change text messages without modifying the source code
- Internationalization (I18N)
  - `messages.properties`
  - `messages_en_US.properties`
  - `messages_zh_CN.properties`
  - …

Using Message Bundles with JSTL

```
<fmt:setBundle basename="messages" />

<fmt:message key="msg1">
    <fmt:param value="Chengyu" />
</fmt:message>

<fmt:message key="msg2" />
```

Using Message Bundles with Spring

- Declare a `messageSource` bean
- `<spring:message>` tag

```
<spring:message code="msg1"
    arguments="Chengyu" />

<spring:message code="msg2" />
```

Input Validation in Spring

- Add error messages to the message bundle
- Implement a `Validator` and wire it to the controller
- Add a `BindingResult` parameter to the controller method
- Use the validator to validate the model attribute
  - Return the form view if validation fails
- Display errors using `<form:errors>`

Example: Validate Add/Edit User

- Both username and password fields cannot be empty
- *Exercise:*
  - *Cannot use an existing username*

Other Validation Options

- Server-side vs. Client-side Validation
- JavaScript validation
  - jQuery Validation plugin - http://bassistance.de/jquery-plugins/jquery-plugin-validation/
- Commons-validator
  - http://commons.apache.org/validator/
  - Provide both *declarative* and *programmatic* validation

Commons-Validator Declarative Validation Example

```
<form name="fooForm">

    <field property="name" depends="required">
        <arg0 key="fooForm.definition"/>
    </field>

    <field property="difficultyLevel"
           depends="required, integer">
        <arg0 key="fooForm.difficultyLevel"/>
    </field>

</form>
```

Commons-Validator Routines

- http://commons.apache.org/proper/commons-validator/javadocs/api-1.4.0/org/apache/commons/validator/routines/package-summary.html
- Independent of the declarative validation framework
- A set of methods to validate
  - Date and time
  - Numeric values, currency
  - ...

Access HTTP Request, Response, and Session

- Simply add an argument of that type to the controller method
- http://static.springsource.org/spring/docs/current/spring-framework-reference/html/mvc.html#mvc-ann-arguments

Readings

- Spring Framework Reference Document, Chapter 16 - http://static.springsource.org/spring/docs/current/spring-framework-reference/html/mvc.html