

## CS520 Web Programming

Bits and Pieces of Web Programming

Chengyu Sun  
California State University, Los Angeles

## Overview

- ◆ Logging
- ◆ Testing
- ◆ File upload
- ◆ Email
- ◆ Message bundles
- ◆ Input validation

## Logging

- ◆ Use print statements to assist debugging
  - Why do we want to do that when we have GUI debugger??

```
public void foo()
{
    System.out.println( "loop started" );
    // some code that might get into infinite loop
    ...
    System.out.println( "loop finished" );
}
```

## Requirements of Good Logging Tools

- ◆ Minimize performance penalty
- ◆ Support different log output
  - Console, file, database, ...
- ◆ Support different message levels
  - Fatal, error, warn, info, debug, trace
  - Example: logging
- ◆ Easy configuration

## Java Logging Libraries

- ◆ Logging implementations
  - Log4j - <http://logging.apache.org/log4j/>
  - java.util.logging in JDK
- ◆ Logging API
  - Apache Commons Logging (JCL) - <http://commons.apache.org/logging/>
  - Simple Logging Façade for Java (SLF4J) - <http://www.slf4j.org/>

## Choose Your Logging Libraries

- ◆ Log4j
  - Widely used
  - Good performance
  - Easy configuration
- ◆ java.util.logging
  - Part of JDK, i.e. no extra library dependency
- ◆ Commons Logging
  - Determines logging implementation at runtime
- ◆ SLF4j
  - Statically linked to a logging implementation
    - Cleaner design
    - Better performance
    - Less problem

## Using Log4j and SLF4j

- ◆ Library dependencies
- ◆ Coding
  - Creating a `Logger`
  - Logging statements
- ◆ Configuration
- ◆ Output format

## Log4j Configuration File

- ◆ `log4j.xml` or `log4j.properties`
- ◆ Appender
  - Output type
  - Output format
- ◆ Logger
  - Package or class selection
  - Message level

## Log4j PatternLayout

- ◆ <http://logging.apache.org/log4j/1.2/api/docs/org/apache/log4j/PatternLayout.html>

## Testing Basics

- ◆ Unit Testing
- ◆ System Testing
- ◆ Integration Testing
- ◆ User Acceptance Testing (Beta Testing)

## Java Testing Frameworks

- ◆ JUnit
  - <http://www.junit.org/>
  - Widely used and supported
- ◆ TestNG
  - <http://testng.org/>
  - Technical superior to JUnit but not as widely used or supported
  - Example: testing BubbleSort

## Maven Support for JUnit/TestNG

- ◆ Library dependency
- ◆ Directory structure
  - `src/test/java`
  - `src/test/resources`
- ◆ The *surefire* plugin

## Basic TestNG Annotations

- ◆ @Test
  - Method
  - Class
  - Group
- ◆ Annotations for various before/after methods

## Ordering Tests

- ◆ @Test
  - dependsOnMethods
  - dependsOnGroups

## Test Suite

```
testng.xml  
  
<suite name="cs520">  
  <test name="all">  
    <packages>  
      <package name="cs520.testing" />  
    </packages>  
  </test>  
</suite>
```

## TestNG and Spring

- ◆ Test classes inherit from Spring TestNG support classes
- ◆ Specify Spring configuration file using @ContextConfiguration
- ◆ Examples: CSNS2

## More About TestNG

- ◆ TestNG Documentation – <http://testng.org/doc/documentation-main.html>
- ◆ *Next Generation Java Testing* by Cédric Beust and Hani Suleiman

## File Upload – The Form

```
<form action="FileUploadHandler"  
  method="post"  
  enctype="multipart/form-data">  
  
  First file: <input type="file" name="file1" /> <br />  
  Second file: <input type="file" name="file2" /> <br />  
  
  <input type="submit" name="upload" value="Upload" />  
</form>
```

## File Upload – The Request

```
POST / HTTP/1.1
Host: cs.calstatela.edu:4040
[...]
Cookie: SITESERVER=ID=289f7e73912343a2d7d1e6e44f931195
Content-Type: multipart/form-data; boundary=-----146043902153
Content-Length: 509

-----146043902153
Content-Disposition: form-data; name="file1"; filename="test.txt"
Content-Type: text/plain

this is a test file.

-----146043902153
Content-Disposition: form-data; name="file2"; filename="test2.bt.gz"
Content-Type: application/x-gzip

???[unprintable]UC
```

## Apache commons-fileupload

◆ <http://jakarta.apache.org/commons/fileupload/using.html>

```
FileItemFactory fileItemFactory = DiskFileItemFactory();
ServletFileUpload fileUpload = new ServletFileUpload( fileItemFactory );
```

```
List items = fileUpload.parseRequest( request );
for( Object o : items )
{
    FileItem item = (FileItem) items;
    if( ! item.isFormFiled() ) {...}
}
```

## Spring File Upload Support

- ◆ `multipartResolver` bean
  - Support multiple request parser libraries
- ◆ Handle uploaded files
  - Add an `MultipartFile` argument to the controller method
  - CSNS2 Example: `upload()` in `SubmissionController`

## Store Uploaded Files

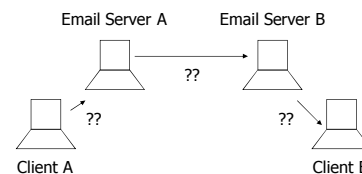
- ◆ In database
  - BLOB, CLOB
  - BINARY VARCHAR, VARCHAR
- ◆ On disk
- ◆ *Pros and Cons??*

## Store Uploaded Files

- ◆ In database
  - ACID
  - BLOB/CLOB types are not very portable
  - Bad performance
- ◆ On disk
  - Not ACID
  - Do not need BLOB/CLOB types
  - Good performance

## How Email Works

◆ SMTP, IMAP, POP



## JavaMail

◆ <http://java.sun.com/products/javamail/>

```
Properties props = System.getProperties();
props.put("mail.smtp.host", mailhost);
Session session = Session.getInstance( props );

Message msg = new MimeMessage(session);
...
Transport.send( msg );
```

## Spring Email Support

- ◆ Declare a `mailSender` bean
- ◆ Mail message classes
  - `SimpleMailMessage`
    - <http://static.springsource.org/spring/docs/current/spring-framework-reference/html/mail.html#mail-usage>
    - No attachment, no special character encoding
  - `MimeMailMessage`
- ◆ CSNS2 Example: `resetPassword()` in `UserController`

## 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

- ◆ Implement a `Validator`
  - CSNS2 Example: `DepartmentValidator`
- ◆ Add a `BindingResult` parameter to the controller method
- ◆ Return the form view if validation fails
- ◆ Display errors using `<form:errors>`

## Example: Validate Add/Edit User

- ◆ Add error messages to the message bundle
- ◆ Implement a validator and wire it to the controller
- ◆ Validate
- ◆ Display error messages

## Other Validation Options

- ◆ JavaScript 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/validator/api-1.3.1/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
  - ...