

## CS520 Web Programming

Bits and Pieces of Web Programming

Chengyu Sun  
California State University, Los Angeles

## Overview

- ◆ Logging
- ◆ File upload
- ◆ Email
- ◆ Message bundles
- ◆ Deployment
- ◆ Spring
  - Input validation
  - Exception handling

## 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
- ◆ Easy configuration

## Java Logging Libraries

- ◆ Logging implementations
  - Log4j - <http://logging.apache.org/log4j/docs>
  - java.util.logging in JDK
- ◆ Logging API
  - Apache Commons Logging (JCL) - <http://commons.apache.org/logging/>
  - Simply 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. not extra library dependency
- ◆ Commons Logging
  - Widely used
  - Determines logging implementation at runtime
- ◆ SLF4j
  - Static binding
  - Gaining popularity

## Log4j Example

- ◆ Library dependency
- ◆ Logger creation
- ◆ Configuration
  - Message levels
  - Output format

## Log4j Configuration File

- ◆ `log4j.xml` or `log4j.properties`
- ◆ Appender
  - Output type
  - Output format
- ◆ Logger
  - Class
  - Message level

## Log4j PatternLayout

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

## 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.txt.gz"
Content-Type: application/x-gzip

????????????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
  - Treat a uploaded file as a `String` or `byte[]` field
    - Spring Reference Documentation, Chapter 13.8 - <http://static.springsource.org/spring/docs/2.5.x/reference/mvc.html>
  - Use `MultipartHttpServletRequest` and `MultipartFile`
    - <http://static.springsource.org/spring/docs/2.5.x/api/org.springframework.web.multipart.MultipartHttpServletRequest.html>

## File Upload Examples in CSNS

- ◆ `UploadFilesController`
- ◆ `AbstractMessageController`

## Store Uploaded Files

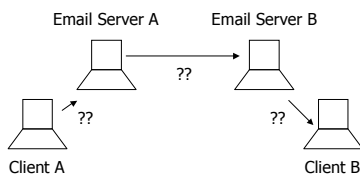
- ◆ In database
  - BLOB, CLOB
  - BINARY VARCAR, VARCHAR
- ◆ On disk
- ◆ *Pros and Cons??*

## 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/2.5.x/api/org/springframework/mail/SimpleMailMessage.html>
    - ◆ No attachment, no special character encoding
  - MimeMailMessage

## Configure Mail Sender

```
<bean id="mailSender"
      class="org.springframework.mail.javamail.JavaMailSenderImpl">
  <property name="host" value="localhost">
</bean>
```

- ◆ Additional properties
  - port
  - username, password

## Email Examples in CSNS

- ◆ ResetPasswordController
- ◆ EmailToListController

## 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

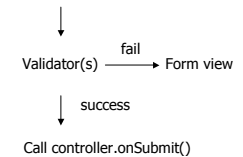
```
<spring:message code="msg1"
  arguments="Chengyu" />

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

## Input Validation in Spring

- ◆ `org.springframework.validation`
  - Validator
  - Errors

Handle input:  
Bind the request parameters to the command object



## 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
  - ...

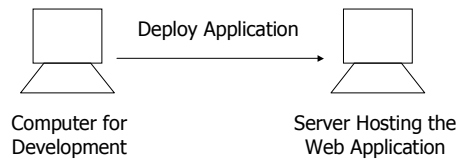
## Controller Exception Handling in Spring

- ◆ An exception resolver catches all exceptions thrown by controllers and chooses the a view to display based on the exception type
  - `SimpleMappingExceptionHandler`
    - ◆ <http://static.springsource.org/spring/docs/2.5.x/api/org/springframework/web/servlet/handler/SimpleMappingExceptionHandler.html>

## ExceptionHandler in CSNS

```
<bean id="exceptionResolver"
      class="csns.spring.handler.ExceptionResolver">
  <property name="exceptionMappings">
    <props>
      <prop key="AccessDeniedException">exception/access</prop>
    </props>
  </property>
  <property name="defaultErrorView" value="exception/default" />
  <property name="exceptionAttribute" value="exception" />
  <property name="defaultStatusCode" value="500" />
</bean>
```

## Application Deployment



## WAR Files

- ◆ Web application ARchive
- ◆ A JAR file for packaging, distributing, and deploying Java web applications
- ◆ Create WAR files
  - The command line tool `jar`
  - Eclipse Export -> Web -> WAR file
  - Ant task `<war>`

## Deploy WAR Files to a Tomcat Server

- ◆ The Manager interface
- ◆ The Tomcat `deploy` Ant task
  - See `build.xml` in the `i18n` example