The BalusC Code

Code depot of a Java EE developer

Thursday, November 8, 2007

MultipartFilter

Fast menu ▼

Upload and store files

Update: if you're using Servlet 3.0 or newer, then there are built-in ways to process file uploads. You may find it more useful: Uploading files in Servlet 3.0.

Downloading files is made relatively easy using a FileServlet, but uploading files is a bit harder. Entering/selecting the raw absolute file path in `input type="text"` and sending it to the server so that it can be used in a `File` object isn't going to work, as the server doesn't have access to the client's file system. That will work only if the server as well as the client runs on the same machine and that wouldn't occur in real life.

To browse and select a file for upload you need a `input type="file"` field in the form. As stated in the HTML specification you have to use the `POST` method and the `enctype` attribute of the form have to be set to `"multipart/form-data"`.

```
<form action="myServlet" method="post" enctype="multipart/form-data">
    <input type="file" name="file" />
    <input type="submit" />
</form>
```

After submitting such a form the binary multipart form data is available in the `HttpServletRequest#getInputStream()`. For testing purposes you can read the stream using the following snippet:

```
BufferedReader reader = new BufferedReader(new InputStreamReader(request.getInputStream()));
String line = null;
while ((line = reader.readLine()) != null) {
    System.out.println(line);
}
```

Parsing such a stream requires precise background knowledge of how multipart form data requests are structured. The standard Servlet API namely doesn't provide builtin facilities to parse them. The form fields aren't available as parameter of the request (e.g. `request.getParameter("name")` will always return null), they are included in the binary stream. The uploaded files are also included in the binary stream. To create a perfect multipart parser you'll have to write a lot of code. But don't feel disappointed, there are lot of 3rd party multipart parsers available. A commonly used one is the Apache Commons FileUpload. It can parse the multipart form data into several `FileItem` objects (misleading class name by the way, I'd rather call it `MultipartItem`). You'll have to filter the parameters and files out yourself.

Back to top

MultipartFilter

To save you the effort, I've played around with it a while and wrote a `MultipartFilter` which automatically detects if the request is a multipart form data request, parses the request and store the parameters back in the parameter map of `HttpServletRequest` and stores the uploaded files as attributes of the `HttpServletRequest`. This way you can just continue writing the Servlet logic as usual.

It makes use of the Apache Commons FileUpload API 1.2. So you need at least the following JAR's (newer versions are allowed) in the classpath, e.g. in /WEB-INF/lib:

- commons-fileupload-1.2.jar
- commons-io-1.3.2.jar

Here is the code. The stuff is tested in a Java EE 5.0 environment with Tomcat 6.0 with Servlet 2.5, JSP 2.1 and JSTL 1.2.

```
/*
 * net/balusc/webapp/MultipartFilter.java
 *
 * Copyright (C) 2007 BalusC
 *
 * This program is free software: you can redistribute it and/or modify it under the terms of the
 * GNU Lesser General Public License as published by the Free Software Foundation, either version 3
 * of the License, or (at your option) any later version.
 *
 * This library is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without
 * even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 * Lesser General Public License for more details.
 *
 * You should have received a copy of the GNU Lesser General Public License along with this library.
 * If not, see <http://www.gnu.org/licenses/>.
 */

package net.balusc.webapp;

import java.io.IOException;
import java.util.Collections;
import java.util.Enumeration;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletRequestWrapper;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.fileupload.disk.DiskFileItemFactory;
import org.apache.commons.fileupload.servlet.ServletFileUpload;

/**
 * Check for multipart HttpServletRequests and parse the multipart form data so that all regular
 * form fields are available in the parameterMap of the HttpServletRequest and that all form file
 * fields are available as attribute of the HttpServletRequest. The attribute value of a form file
 * field can be an instance of FileItem or FileUploadException.
 * <p>
 * This filter requires at least the following JAR's (newer versions are allowed) in the classpath,
 * e.g. in /WEB-INF/lib.
 * <ul>
 * <li>commons-fileupload-1.2.jar</li>
 * <li>commons-io-1.3.2.jar</li>
 * </ul>
 * <p>
 * This filter should be definied as follows in the web.xml:
 * <pre>
 * &lt;filter&gt;
 *     &lt;description&gt;
 *         Check for multipart HttpServletRequests and parse the multipart form data so that all
 *         regular form fields are available in the parameterMap of the HttpServletRequest and that
 *         all form file fields are available as attribute of the HttpServletRequest. The attribute
 *         value of a form file field can be an instance of FileItem or FileUploadException.
 *     &lt;/description&gt;
 *     &lt;filter-name&gt;multipartFilter&lt;/filter-name&gt;
 *     &lt;filter-class&gt;net.balusc.webapp.MultipartFilter&lt;/filter-class&gt;
 *     &lt;init-param&gt;
 *         &lt;description&gt;
 *             Sets the maximum file size of the uploaded file in bytes. Set to 0 to indicate an
 *             unlimited file size. The example value of 1048576 indicates a maximum file size of
 *             1MB. This parameter is not required and can be removed safely.
 *         &lt;/description&gt;
 *         &lt;param-name&gt;maxFileSize&lt;/param-name&gt;
 *         &lt;param-value&gt;1048576&lt;/param-value&gt;
 *     &lt;/init-param&gt;
 * &lt;/filter&gt;
 * &lt;filter-mapping&gt;
 *     &lt;filter-name&gt;multipartFilter&lt;/filter-name&gt;
 *     &lt;url-pattern&gt;/*&lt;/url-pattern&gt;
 * &lt;/filter-mapping&gt;
 * </pre>
 *
 * @author BalusC
 * @link http://balusc.blogspot.com/2007/11/multipartfilter.html
 */
public class MultipartFilter implements Filter {

    // Init ---------------------------------------------------------------------------------------

    private long maxFileSize;

    // Actions ------------------------------------------------------------------------------------

    /**
     * Configure the 'maxFileSize' parameter.
     * @throws ServletException If 'maxFileSize' parameter value is not numeric.
     * @see javax.servlet.Filter#init(javax.servlet.FilterConfig)
     */
    public void init(FilterConfig filterConfig) throws ServletException {
        // Configure maxFileSize.
        String maxFileSize = filterConfig.getInitParameter("maxFileSize");
        if (maxFileSize != null) {
            if (!maxFileSize.matches("^\\d+$")) {
                throw new ServletException("MultipartFilter 'maxFileSize' is not numeric.");
            }
            this.maxFileSize = Long.parseLong(maxFileSize);
        }
    }

    /**
     * Check the type request and if it is a HttpServletRequest, then parse the request.
     * @throws ServletException If parsing of the given HttpServletRequest fails.
     * @see javax.servlet.Filter#doFilter(
     *      javax.servlet.ServletRequest, javax.servlet.ServletResponse, javax.servlet.FilterChain)
     */
    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
        throws ServletException, IOException
    {
        // Check type request.
        if (request instanceof HttpServletRequest) {
            // Cast back to HttpServletRequest.
            HttpServletRequest httpRequest = (HttpServletRequest) request;

            // Parse HttpServletRequest.
            HttpServletRequest parsedRequest = parseRequest(httpRequest);

            // Continue with filter chain.
            chain.doFilter(parsedRequest, response);
        } else {
            // Not a HttpServletRequest.
            chain.doFilter(request, response);
        }
    }

    /**
     * @see javax.servlet.Filter#destroy()
     */
    public void destroy() {
        // I am a boring method.
    }

    // Helpers ------------------------------------------------------------------------------------

    /**
     * Parse the given HttpServletRequest. If the request is a multipart request, then all multipart
     * request items will be processed, else the request will be returned unchanged. During the
     * processing of all multipart request items, the name and value of each regular form field will
     * be added to the parameterMap of the HttpServletRequest. The name and File object of each form
     * file field will be added as attribute of the given HttpServletRequest. If a
     * FileUploadException has occurred when the file size has exceeded the maximum file size, then
     * the FileUploadException will be added as attribute value instead of the FileItem object.
     * @param request The HttpServletRequest to be checked and parsed as multipart request.
     * @return The parsed HttpServletRequest.
     * @throws ServletException If parsing of the given HttpServletRequest fails.
     */
    @SuppressWarnings("unchecked") // ServletFileUpload#parseRequest() does not return generic type.
    private HttpServletRequest parseRequest(HttpServletRequest request) throws ServletException {

        // Check if the request is actually a multipart/form-data request.
        if (!ServletFileUpload.isMultipartContent(request)) {
            // If not, then return the request unchanged.
            return request;
        }

        // Prepare the multipart request items.
        // I'd rather call the "FileItem" class "MultipartItem" instead or so. What a stupid name ;)
        List<FileItem> multipartItems = null;

        try {
            // Parse the multipart request items.
            multipartItems = new ServletFileUpload(new DiskFileItemFactory()).parseRequest(request);
            // Note: we could use ServletFileUpload#setFileSizeMax() here, but that would throw a
            // FileUploadException immediately without processing the other fields. So we're
            // checking the file size only if the items are already parsed. See processFileField().
        } catch (FileUploadException e) {
            throw new ServletException("Cannot parse multipart request: " + e.getMessage());
        }

        // Prepare the request parameter map.
        Map<String, String[]> parameterMap = new HashMap<String, String[]>();

        // Loop through multipart request items.
        for (FileItem multipartItem : multipartItems) {
            if (multipartItem.isFormField()) {
                // Process regular form field (input type="text|radio|checkbox|etc", select, etc).
                processFormField(multipartItem, parameterMap);
            } else {
                // Process form file field (input type="file").
                processFileField(multipartItem, request);
            }
        }

        // Wrap the request with the parameter map which we just created and return it.
        return wrapRequest(request, parameterMap);
    }

    /**
     * Process multipart request item as regular form field. The name and value of each regular
     * form field will be added to the given parameterMap.
     * @param formField The form field to be processed.
     * @param parameterMap The parameterMap to be used for the HttpServletRequest.
     */
    private void processFormField(FileItem formField, Map<String, String[]> parameterMap) {
        String name = formField.getFieldName();
        String value = formField.getString();
        String[] values = parameterMap.get(name);

        if (values == null) {
            // Not in parameter map yet, so add as new value.
            parameterMap.put(name, new String[] { value });
        } else {
            // Multiple field values, so add new value to existing array.
            int length = values.length;
            String[] newValues = new String[length + 1];
            System.arraycopy(values, 0, newValues, 0, length);
            newValues[length] = value;
            parameterMap.put(name, newValues);
        }
    }

    /**
     * Process multipart request item as file field. The name and FileItem object of each file field
     * will be added as attribute of the given HttpServletRequest. If a FileUploadException has
     * occurred when the file size has exceeded the maximum file size, then the FileUploadException
     * will be added as attribute value instead of the FileItem object.
     * @param fileField The file field to be processed.
     * @param request The involved HttpServletRequest.
     */
    private void processFileField(FileItem fileField, HttpServletRequest request) {
        if (fileField.getName().length() <= 0) {
            // No file uploaded.
            request.setAttribute(fileField.getFieldName(), null);
        } else if (maxFileSize > 0 && fileField.getSize() > maxFileSize) {
            // File size exceeds maximum file size.
            request.setAttribute(fileField.getFieldName(), new FileUploadException(
                "File size exceeds maximum file size of " + maxFileSize + " bytes."));
            // Immediately delete temporary file to free up memory and/or disk space.
            fileField.delete();
        } else {
            // File uploaded with good size.
            request.setAttribute(fileField.getFieldName(), fileField);
        }
    }

    // Utility (may be refactored to public utility class) ---------------------------------------

    /**
     * Wrap the given HttpServletRequest with the given parameterMap.
     * @param request The HttpServletRequest of which the given parameterMap have to be wrapped in.
     * @param parameterMap The parameterMap to be wrapped in the given HttpServletRequest.
     * @return The HttpServletRequest with the parameterMap wrapped in.
     */
    private static HttpServletRequest wrapRequest(
        HttpServletRequest request, final Map<String, String[]> parameterMap)
    {
        return new HttpServletRequestWrapper(request) {
            public Map<String, String[]> getParameterMap() {
                return parameterMap;
            }
            public String[] getParameterValues(String name) {
                return parameterMap.get(name);
            }
            public String getParameter(String name) {
                String[] params = getParameterValues(name);
                return params != null && params.length > 0 ? params[0] : null;
            }
            public Enumeration<String> getParameterNames() {
                return Collections.enumeration(parameterMap.keySet());
            }
        };
    }

}
```

Add and configure the filter as follows in the web.xml:

```
<filter>
    <description>
        Check for multipart HttpServletRequests and parse the multipart form data so that all
        regular form fields are available in the parameterMap of the HttpServletRequest and that
        all form file fields are available as attribute of the HttpServletRequest. The attribute
        value of a form file field can be an instance of FileItem or FileUploadException.
    </description>
    <filter-name>multipartFilter</filter-name>
    <filter-class>net.balusc.webapp.MultipartFilter</filter-class>
    <init-param>
        <description>
            Sets the maximum file size of the uploaded file in bytes. Set to 0 to indicate an
            unlimited file size. The example value of 1048576 indicates a maximum file size of
            1MB. This parameter is not required and can be removed safely.
        </description>
        <param-name>maxFileSize</param-name>
        <param-value>1048576</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>multipartFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

That's all, folks!

Back to top

Basic use example

Here is a basic use example of a servlet, a form and JSP file which demonstrates the working of the `MultipartFilter`. Thanks to the `MultipartFilter` you can just use `HttpServletRequest#getParameter()` and `#getParameterValues()` for regular form fields. The uploaded file is available by `HttpServletRequest#getAttribute()`. If it is an instance of `FileItem`, then the upload was succesful, else if it is an instance of `FileUploadException`, then the upload was failed. The only cause can be that the file size exceeded the configured maximum file size.

```
package mypackage;

import java.io.File;
import java.io.IOException;
import java.util.Arrays;
import java.util.Collection;
import java.util.Map;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.commons.fileupload.FileItem;
import org.apache.commons.fileupload.FileUploadException;
import org.apache.commons.io.FilenameUtils;

public class MyServlet extends HttpServlet {

    // Init ---------------------------------------------------------------------------------------

    private File uploadFilePath;

    // Actions ------------------------------------------------------------------------------------

    public void init() throws ServletException {
        // Configure uploadFilePath.
        String uploadFilePathParam = getServletConfig().getInitParameter("uploadFilePath");
        if (uploadFilePathParam == null) {
            throw new ServletException("MyServlet 'uploadFilePath' is not configured.");
        }
        uploadFilePath = new File(uploadFilePathParam);
        if (!uploadFilePath.exists()) {
            throw new ServletException("MyServlet 'uploadFilePath' does not exist.");
        }
        if (!uploadFilePath.isDirectory()) {
            throw new ServletException("MyServlet 'uploadFilePath' is not a directory.");
        }
        if (!uploadFilePath.canWrite()) {
            throw new ServletException("MyServlet 'uploadFilePath' is not writeable.");
        }
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        // Do nothing, just show the form.
        forward(request, response);
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        // Prepare bean.
        MyForm myForm = new MyForm();
        // Process request.
        process(request, myForm);

        // Store bean in request.
        request.setAttribute("myForm", myForm);

        // Postback.
        forward(request, response);
    }

    // Helpers ------------------------------------------------------------------------------------

    private void process(HttpServletRequest request, MyForm myForm) {
        // Validate text.
        String text = request.getParameter("text");
        if (isEmpty(text)) {
            // No text entered.
            myForm.setError("text", "Please enter some text.");
        }

        // Validate file.
        Object fileObject = request.getAttribute("file");
        if (fileObject == null) {
            // No file uploaded.
            myForm.setError("file", "Please select file to upload.");
        } else if (fileObject instanceof FileUploadException) {
            // File upload is failed.
            FileUploadException fileUploadException = (FileUploadException) fileObject;
            myForm.setError("file", fileUploadException.getMessage());
        }

        // Validate checkboxes.
        String[] check = request.getParameterValues("check");
        if (isEmpty(check)) {
            // No checkboxes checked.
            myForm.setError("check", "Please check one or more checkboxes.");
        }

        // If there are no errors, proceed with writing file.
        if (!myForm.hasErrors()) {
            FileItem fileItem = (FileItem) fileObject;

            // Get file name from uploaded file and trim path from it.
            // Some browsers (e.g. IE, Opera) also sends the path, which is completely irrelevant.
            String fileName = FilenameUtils.getName(fileItem.getName());

            // Prepare filename prefix and suffix for an unique filename in upload folder.
            String prefix = FilenameUtils.getBaseName(fileName) + "_";
            String suffix = "." + FilenameUtils.getExtension(fileName);

            try {
                // Prepare unique local file based on file name of uploaded file.
                File file = File.createTempFile(prefix, suffix, uploadFilePath);

                // Write uploaded file to local file.
                fileItem.write(file);

                // Set the file in form so that it can be provided for download.
                myForm.setFile(file);
            } catch (Exception e) {
                // Can be thrown by uniqueFile() and FileItem#write().
                myForm.setError("file", e.getMessage());
                e.printStackTrace();
            }
        }

        // If there are no errors after writing file, proceed with showing messages.
        if (!myForm.hasErrors()) {
            myForm.setMessage("text", "You have entered: " + text + ".");
            myForm.setMessage("file", "File succesfully uploaded.");
            myForm.setMessage("check", "You have checked: " + Arrays.toString(check) + ".");
        }
    }

    private void forward(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        request.getRequestDispatcher("myForm.jsp").forward(request, response);
    }

    // Utilities (should be refactored to public utility classes) ---------------------------------

    /**
     * Check if the given object is empty. Returns true if the object is null, or if it is an
     * instance of String and its trimmed length is zero, or if it is an instance of an ordinary
     * array and its length is zero, or if it is an instance of Collection and its size is zero,
     * or if it is an instance of Map and its size is zero, or if its String representation is
     * null or the trimmed length of its String representation is zero.
     * @param value The object to be determined on emptiness.
     * @return True if the given object value is empty.
     */
    public static boolean isEmpty(Object value) {
        if (value == null) {
            return true;
        } else if (value instanceof String) {
            return ((String) value).trim().length() == 0;
        } else if (value instanceof Object[]) {
            return ((Object[]) value).length == 0;
        } else if (value instanceof Collection<?>) {
            return ((Collection<?>) value).size() == 0;
        } else if (value instanceof Map<?, ?>) {
            return ((Map<?, ?>) value).size() == 0;
        } else {
            return value.toString() == null || value.toString().trim().length() == 0;
        }
    }

}
```

Add and configure the servlet as follows in the web.xml:

```
<servlet>
    <servlet-name>myServlet</servlet-name>
    <servlet-class>mypackage.MyServlet</servlet-class>
    <init-param>
        <description>
            Set the file path where uploaded files should be stored in. This parameter is
            required.
        </description>
        <param-name>uploadFilePath</param-name>
        <param-value>c:/upload</param-value>
    </init-param>
</servlet>
<servlet-mapping>
    <servlet-name>myServlet</servlet-name>
    <url-pattern>/myServlet</url-pattern>
</servlet-mapping>
```

The form bean:

```
package mypackage;

import java.io.File;
import java.util.HashMap;
import java.util.Map;

public class MyForm {

    // Init ---------------------------------------------------------------------------------

    private String text;
    private File file;
    private String[] check;
    private Map<String, Boolean> checked = new HashMap<String, Boolean>();
    private Map<String, String> errors = new HashMap<String, String>();
    private Map<String, String> messages = new HashMap<String, String>();

    // Getters ------------------------------------------------------------------------------

    public String getText() {
        return text;
    }

    public File getFile() {
        return file;
    }

    public String[] getCheck() {
        return check;
    }

    // Setters ------------------------------------------------------------------------------

    public void setText(String text) {
        this.text = text;
    }

    public void setFile(File file) {
        this.file = file;
    }

    public void setCheck(String[] check) {
        checked = new HashMap<String, Boolean>();
        for (String value : check) {
            checked.put(value, Boolean.TRUE);
        }
        this.check = check;
    }

    // Helpers ------------------------------------------------------------------------------

    public Map<String, Boolean> getChecked() {
        return checked;
    }

    public Map<String, String> getErrors() {
        return errors;
    }

    public Map<String, String> getMessages() {
        return messages;
    }

    public void setError(String fieldName, String message) {
        errors.put(fieldName, message);
    }

    public void setMessage(String fieldName, String message) {
        messages.put(fieldName, message);
    }

    public boolean hasErrors() {
        return errors.size() > 0;
    }

}
```

Finally the JSP file, save it as **myForm.jsp** in the root of the WebContent:

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<!doctype html>

<html lang="en">
    <head>
        <title>Test</title>
    </head>
    <body>
        <jsp:useBean id="myForm" class="mypackage.MyForm" scope="request" />
        <jsp:setProperty name="myForm" property="*" />

        <form action="myServlet" method="post" enctype="multipart/form-data">
```

```
            <label for="text" >Text:</label>
            <input type="text" id="text" name="text" value="${myForm.text}">
            <c:if test="${myForm.errors.text != null}">
                <span style="color: red;">${myForm.errors.text}</span>
            </c:if>
            <c:if test="${myForm.messages.text != null}">
                <span style="color: green;">${myForm.messages.text}</span>
            </c:if>
            <br>

            <label for="file" >File:</label>
            <input type="file" id="file" name="file">
            <c:if test="${myForm.errors.file != null}">
                <span style="color: red;">${myForm.errors.file}</span>
            </c:if>
            <c:if test="${myForm.messages.file != null}">
                <span style="color: green;">${myForm.messages.file}
                    <c:if test="${myForm.file != null}">
                        &nbsp;<a href="file/${myForm.file.name}">Download back</a>.
                    </c:if>
                </span>
            </c:if>
            <br>

            <label for="check1" >Check 1:</label>
            <input type="checkbox" id="check1" name="check" value="check1"
                ${myForm.checked.check1 ? 'checked' : ''}>
            <c:if test="${myForm.errors.check != null}">
                <span style="color: red;">${myForm.errors.check}</span>
            </c:if>
            <c:if test="${myForm.messages.check != null}">
                <span style="color: green;">${myForm.messages.check}</span>
            </c:if>
            <br>

            <label for="check2" >Check 2:</label>
            <input type="checkbox" id="check2" name="check" value="check2"
                ${myForm.checked.check2 ? 'checked' : ''} /></td>
            <br>

            <input type="submit">
        </form>
    </body>
</html>
```

Note: the download link makes use of the FileServlet. Make sure that it points to the same directory as where the file is uploaded. You can configure it as an `init-param`.

Copy'n'paste the stuff, run it on http://localhost:8080/playground/myServlet (assuming that your local development server runs at port 8080 and that the context root of your playground web application project is called 'playground') and see it wonderfully working!

Back to top

Geplaatst door Bauke Scholtz op 8:15 AM

G+1 Recommend this on Google

Labels: Filter, JSP, Servlet, Upload File, UseBean

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)