Carga de archivos

Raúl Estrada

Octubre 2020

1. Introduction

In this quick tutorial, we'll see how to upload a file from a servlet.

To achieve this, we'll first see the vanilla Jakarta EE solution with file upload capabilities provided by native *@MultipartConfig* annotation.

Then, we'll go over the Apache Commons *FileUpload* library, for earlier versions of the Servlet API.

2. Using Jakarta EE *@MultipartConfig*

Jakarta EE has the ability to support multi-part uploads out of the box.

As such, it's probably a default go-to when enriching a Jakarta EE app with file upload support.

First, let's add a form to our HTML file:

```
<form method="post" action="multiPartServlet" enctype="multipart/form-data">
    Choose a file: <input type="file" name="multiPartServlet" />
    <input type="submit" value="Upload" />
</form>
```

The form should be defined using the *enctype="multipart/form-data"* attribute to signal a multipart upload.

Next, **we'll want to annotate our *HttpServlet* with the correct information using the *@MultipartConfig* annotation**:

```
@MultipartConfig(fileSizeThreshold = 1024 * 1024,
  maxFileSize = 1024 * 1024 * 5,
  maxRequestSize = 1024 * 1024 * 5 * 5)
public class MultipartServlet extends HttpServlet {
    //...
}
```

Then, let's make sure that our default server upload folder is set:

```
String uploadPath = getServletContext().getRealPath("") + File.separator + UPLOAD_DIRECTORY;
File uploadDir = new File(uploadPath);
if (!uploadDir.exists()) uploadDir.mkdir();
```

Finally, **we can easily retrieve our inbound *File* from the *request* using the *getParts()* method**, and save it to the disk:

```
for (Part part : request.getParts()) {
    fileName = getFileName(part);
    part.write(uploadPath + File.separator + fileName);
}
```

Note that, in this example, we're using a helper method getFileName():

```
private String getFileName(Part part) {
    for (String content : part.getHeader("content-disposition").split(";")) {
        if (content.trim().startsWith("filename"))
            return content.substring(content.indexOf("=") + 2, content.length() - 1);
        }
    return Constants.DEFAULT_FILENAME;
}
```

For Servlet 3.1. projects, we could alternatively use the *Part.getSubmittedFileName()* method:

```
fileName = part.getSubmittedFileName();
```

3. Using Apache Commons FileUpload

If we're not on a Servlet 3.0 project, we can use the Apache Commons FileUpload library directly.

3.1. Setup

We'll want to use the following *pom.xml* dependencies to get our example running:

```
<dependency>
    <groupId>commons-fileupload</groupId>
    <artifactId>commons-fileupload</artifactId>
    <version>1.3.3</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.6</version>
</dependency>
```

The most recent versions can be found with a quick search on Maven's Central Repository: commons-fileupload and commons-io.

3.2. Upload Servlet

The three main parts to incorporating Apache's *FileUpload* library go as follows:

- An upload form in a *.jsp* page.
- Configuring your *DiskFileItemFactory* and *ServletFileUpload* object.
- Processing the actual contents of a multipart file upload.

The upload form is the same as the one in the previous section.

Let's move on to creating our Jakarta EE servlet.

In our request processing method, we can wrap the incoming *HttpRequest* with a check to see if it's a multi-part upload.

We'll also specify what resources to allocate to the file upload temporarily (while being processed) on our *DiskFileItemFactory.*

Lastly, **we'll create a *ServletFileUpload* object which will represent the actual file itself**. It will expose the contents of the multi-part upload for final persistence server side:

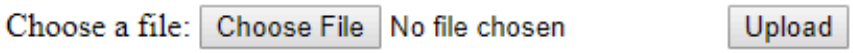
```
if (ServletFileUpload.isMultipartContent(request)) {

    DiskFileItemFactory factory = new DiskFileItemFactory();
    factory.setSizeThreshold(MEMORY_THRESHOLD);
    factory.setRepository(new File(System.getProperty("java.io.tmpdir")));

    ServletFileUpload upload = new ServletFileUpload(factory);
    upload.setFileSizeMax(MAX_FILE_SIZE);
    upload.setSizeMax(MAX_REQUEST_SIZE);
    String uploadPath = getServletContext().getRealPath("")
      + File.separator + UPLOAD_DIRECTORY;
    File uploadDir = new File(uploadPath);
    if (!uploadDir.exists()) {
        uploadDir.mkdir();
    }
    //...
}
```

And, then we can extract those contents and write them to disk:

```
if (ServletFileUpload.isMultipartContent(request)) {
    //...
    List<FileItem> formItems = upload.parseRequest(request);
    if (formItems != null && formItems.size() > 0) {
        for (FileItem item : formItems) {
            if (!item.isFormField()) {
                String fileName = new File(item.getName()).getName();
                String filePath = uploadPath + File.separator + fileName;
                File storeFile = new File(filePath);
                item.write(storeFile);
                request.setAttribute("message", "File "
                  + fileName + " has uploaded successfully!");
            }
        }
    }
}
```

4. Running the Example

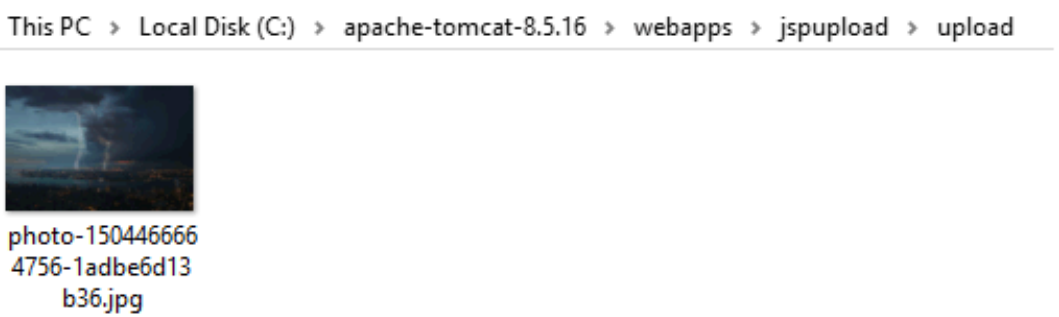
After we've compiled our project into a *.war*, we can drop it into our local Tomcat instance and start it up. From there, we can bring up the main upload view where we're presented with a form:



After successfully uploading our file, we should see the message:

File photo-1504466664756-1adbe6d13b36.jpg has uploaded successfully!

Lastly, we can check the location specified in our servlet:



5. Conclusion

That's it! We've learned how to provide multi-part file uploads using Jakarta EE, as well as Apache's Common *FileUpload* library!