Descarga de archivos

Raúl Estrada

Octubre 2020

1. Overview

A common feature of web applications is the ability to download files.

In this tutorial, **we'll cover a simple example of creating a downloadable file and serving it from a Java Servlet application**.

The file we are using will be from the webapp resources.

2. Maven Dependencies

If using Jakarta EE, then we wouldn't need to add any dependencies. However, if we're using Java SE, we'll need the javax.servlet-api dependency:

```
<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>javax.servlet-api</artifactId>
    <version>4.0.1</version>
    <scope>provided</scope>
</dependency>
```

The latest version of the dependency can be found here.

3. Servlet

Let's have a look at the code first and then find out what's going on:

```
@WebServlet("/download")
public class DownloadServlet extends HttpServlet {
    private final int ARBITARY_SIZE = 1048;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
      throws ServletException, IOException {

        resp.setContentType("text/plain");
        resp.setHeader("Content-disposition", "attachment; filename=sample.txt");

        try(InputStream in = req.getServletContext().getResourceAsStream("/WEB-INF/sample.txt");
          OutputStream out = resp.getOutputStream()) {

            byte[] buffer = new byte[ARBITARY_SIZE];

            int numBytesRead;
            while ((numBytesRead = in.read(buffer)) > 0) {
                out.write(buffer, 0, numBytesRead);
            }
        }
    }
}
```

3.1. Request End Point

@WebServlet("/download") annotation marks the *DownloadServlet* class to serve requests directed at the *"/download"* end-point.

Alternatively, we can do this by describing the mapping in the web.xml file.

3.2. Response *Content-Type*

The *HttpServletResponse* object has a method called as *setContentType* which we can use to set the *Content-Type* header of the HTTP response.

Content-Type is the historical name of the header property. Another name was the MIME type (Multipurpose Internet Mail Extensions). We now simply refer to the value as the Media Type.

This value could be "application/pdf", "text/plain", "text/html", "image/jpg", etc., the official list is maintained by the Internet Assigned Numbers Authority (IANA) and can be found here.

For our example, we are using a simple text file. **The *Content-Type* for a text file is "text/plain".**

3.3. Response *Content-Disposition*

Setting the *Content-Disposition* header in the response object tells the browser how to handle the file it is accessing.

Browsers understand the use of *Content-Disposition* as a convention but it's not actually a part of the HTTP standard. W3 has a memo on the use of *Content-Disposition* available to read here.

The *Content-Disposition* values for the main body of a response will be either "inline" (for webpage content to be rendered) or "attachment" (for a downloadable file).

If not specified, the default *Content-Disposition* is "inline".

Using an optional header parameter, we can specify the filename "sample.txt".

Some browsers will immediately download the file using the given filename and others will show a download dialog containing our predefined value.

The exact action taken will depend on the browser.

3.4. Reading From File and Writing to Output Stream

In the remaining lines of code, we take the *ServletContext* from the request, and use it to obtain the file at "/WEB-INF/sample.txt".

Using *HttpServletResponse#getOutputStream()*, we then read from the input stream of the resource and write to the response's *OutputStream*.

The size of the byte array we use is arbitrary. We can decide the size based on the amount of memory is reasonable to allocate for passing the data from the *InputStream* to the *OutputStream*; the smaller the nuber, the more loops; the bigger the number, the higher memory usage.

This cycle continues until *numByteRead* is 0 as that indicates the end of the file.

3.5. Close and Flush

Stream instances must be closed after use to release any resources it is currently holding. *Writer* instances must also be flushed to write any remaining buffered bytes to it's destination.

Using a *try-with-resources* statement, the application will automatically *close* any *AutoCloseable* instance defined as part of the *try* statement. Read more about try-with-resources here.

We use these two methods to release memory, ensuring that the data we have prepared is sent out from our application.

3.6. Downloading the File

With everything in place, we are now ready to run our Servlet.

Now when we visit the relative end-point *"/download"*, our browser will attempt to download the file as "simple.txt".

4. Conclusion

Downloading a file from a Servlet becomes a simple process. Using streams allow us to pass out the data as bytes and the Media Types inform the client browser what type of data to expect.

It is down to the browser to determine how to handle the response, however, we can give some guidelines with the *Content-Disposition* header.