15th January 2020

Resumable file upload

With `fetch` method it's fairly easy to upload a file.

How to resume the upload after lost connection? There's no built-in option for that, but we have the pieces to implement it.

Resumable uploads should come with upload progress indication, as we expect big files (if we may need to resume). So, as `fetch` doesn't allow to track upload progress, we'll use XMLHttpRequest.

Not-so-useful progress event

To resume upload, we need to know how much was uploaded till the connection was lost.

There's `xhr.upload.onprogress` to track upload progress.

Unfortunately, it won't help us to resume the upload here, as it triggers when the data is *sent*, but was it received by the server? The browser doesn't know.

Maybe it was buffered by a local network proxy, or maybe the remote server process just died and couldn't process them, or it was just lost in the middle and didn't reach the receiver.

That's why this event is only useful to show a nice progress bar.

To resume upload, we need to know *exactly* the number of bytes received by the server. And only the server can tell that, so we'll make an additional request.

Algorithm

1. First, create a file id, to uniquely identify the file we're going to upload:

   ```
   1  let fileId = file.name + '-' + file.size + '-' + +file.lastModifiedDate;
   ```

   That's needed for resume upload, to tell the server what we're resuming.

   If the name or the size or the last modification date changes, then there'll be another `fileId` .

2. Send a request to the server, asking how many bytes it already has, like this:

   ```
   1  let response = await fetch('status', {
   2    headers: {
   3      'X-File-Id': fileId
   4    }
   5  });
   ```

```
6
7  // The server has that many bytes
8  let startByte = +await response.text();
```

This assumes that the server tracks file uploads by `X-File-Id` header. Should be implemented at server-side.

If the file doesn't yet exist at the server, then the server response should be `0`

3. Then, we can use `Blob` method `slice` to send the file from `startByte`:

```
1  xhr.open("POST", "upload", true);
2
3  // File id, so that the server knows which file we upload
4  xhr.setRequestHeader('X-File-Id', fileId);
5
6  // The byte we're resuming from, so the server knows we're resuming
7  xhr.setRequestHeader('X-Start-Byte', startByte);
8
9  xhr.upload.onprogress = (e) => {
10   console.log(`Uploaded ${startByte + e.loaded} of ${startByte + e.total}`)
11 };
12
13 // file can be from input.files[0] or another source
14 xhr.send(file.slice(startByte));
```

Here we send the server both file id as `X-File-Id`, so it knows which file we're uploading, and the starting byte as `X-Start-Byte`, so it knows we're not uploading it initially, but resuming.

The server should check its records, and if there was an upload of that file, and the current uploaded size is exactly `X-Start-Byte`, then append the data to it.

Here's the demo with both client and server code, written on Node.js.

It works only partially on this site, as Node.js is behind another server named Nginx, that buffers uploads, passing them to Node.js when fully complete.

But you can download it and run locally for the full demonstration:

Result server.js uploader.js index.html

Choose file No file chosen Upload (Resumes automatically)
Stop upload
Progress indication

As we can see, modern networking methods are close to file managers in their capabilities – control over headers, progress indicator, sending file parts, etc.

We can implement resumable upload and much more.

Previous lesson Next lesson

Share

Tutorial map

Comments

- If you have suggestions what to improve - please submit a GitHub issue or a pull request instead of commenting.
- If you can't understand something in the article – please elaborate.
- To insert a few words of code, use the `<code>` tag, for several lines – use `<pre>`, for more than 10 lines – use a sandbox (plnkr, JSBin, codepen…)