Stream API - Request Object

The request object is used to fetch resources from the server. A request object is created by using Resquest() constructor provided by the Resquest interface. So when the new Request object is created we are allowed to pass a ReadableStream to the body of the Request object such types of requests are known as streaming requests. This request object is then passed to the fetch() function to fetch the resources.

Syntax

```
const myobject = new Request(URL, {
   method: 'POST',
   body: Stream,
   headers: {'Content-Type'},
   duplex: 'half',
});
```

Here the Request() constructor contains the following parameters –

- **URL** – Address of the resource.
- **method** – It represents the HTTP request method like GET, POST, etc.
- **body** – Contains the ReadableStream object.
- **headers** – Contains the suitable headers for the body.
- **duplex** – Set to half to make duplex stream.

Example

In the following program, we create a streaming request. So for that first we create a readable stream with the help of the ReadableStream() constructor along with the start() function which implements the ReadableStream logic and other operations. Then we create a request object using the Request() constructor along with the options: the method option contains the POST request to send the request, the body option contains the stream, the headers option contains the appropriate header, and the duplex option is set to half to make it a duplex stream. After creating a request object now we pass the object in the fetch() function to make a request and this function handles the response using then() and an error(if occurs) using the catch() function. Here in place of https://exampleApi.com/, you can use a valid API/URL which sends/receive data in the form of chunks.

```
<script>
   // Create a readable stream using the ReadableStream constructor()
   const readStream = new ReadableStream({
      start(controller) {
         // Here implement your ReadableStream
         // Also with the help of controller, you can enqueue data and
         // signal the end of the stream
      },
   });

   // Create a request objecct using Request() constructor
   const request = new Request('https://exampleApi.com/', {
      // Set the method
      method: 'POST',

      // Passing the stream to the body of the request
      body: stream,

      // Setting suitable header
      headers: {'Content-Type': 'application/octet-stream'},
      duplex: 'half'
   });

   // After creating a request object pass the object
   // to fetch() function make a request or perform operations
   fetch(request)
   .then(response => {
      // Handling the response
   })
   .catch(error => {
      // Handling any errors if occur
   });
</script>
```


Restrictions

Streaming requests is a new feature so it has some restrictions and they are –

Half duplex − To execute a streaming request we have to set the duplex option to half. If you do not set this option in your streaming request, then you will get an error. This option tells that the request body is a duplex stream, where the duplex stream is a stream which receives data(writeable) and sends data(readable) simultaneously.

Required CORS and trigger a preflight − As we know that a streaming request contains a stream in the request body but does not have a "Content-Length" header. So for such type of request, CORS is required and they always trigger a preflight. Also, no-cors streaming requests are not allowed.

Does not work on HTTP/1.x − If the connection is HTTP/1.x, then based on the HTTP/1.x rules it will reject fetch. According to the HTTP/1.x rules, the request and response bodies should need to send a Content-Length headers. So that the other party can keep a record of how much data is received or can change the format to use chunked encoding. Chunked encoding is common but for the requests, it is very rare.

Server-side incompatibility − Some servers do not support streaming requests. So always use only those servers that support streaming requests like NodeJS, etc.

Conclusion

So this is how we can create a Request object for streams or we can say that this is how we can create a streaming request using the fetch() function. Streaming requests are useful for sending large files, real-time data processing, media streaming, continuous data feeds, etc. Now in the next article, we will learn about the response body in Stream API.