Fetch API - Basics

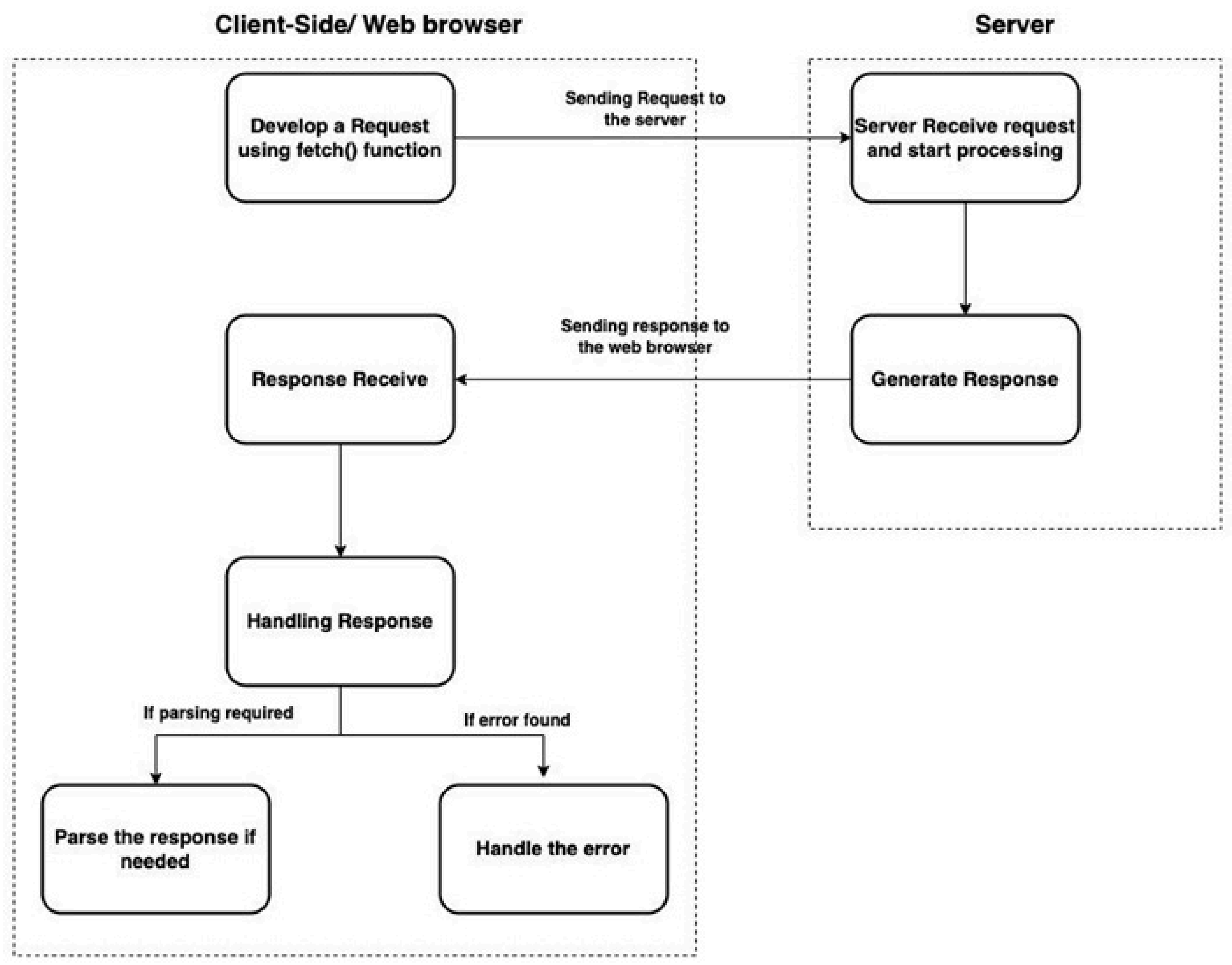
Fetch is a promise-based API which provides a JavaScript interface to access or manipulate requests and responses asynchronously. It is more powerful as compared to XMLHttpRequest using Fetch API we can send data to the server or can request data from the server asynchronously. It also uses Request and Response objects along with CORS and HTTP origin header concepts.

The following are the key components of Fetch API −

- **fetch() function** − To fetch the resources or to create a request Fetch API uses a global method named fetch(). It returns a promise which further resolves to a Response object.
- **Request and Response Object** − The Request object is used to represent a request being sent with all the pieces of information like URL, header, etc. Whereas the Response object is used to represent the response returned by the server including the status code, body and response header.
- **Promises** − Fetch API is based on promises because they handle operations and manage response flow asynchronously. Using promises we can create a chain of operations and can handle successes and errors using .then() and.catch() functions.
- **Customization** − Using Fetch API we can customize the request by specifying the methods, adding a body to the request, setting a header, handling different formats of data, etc.
- **CROS** − Fetch API provide good support to CROS(Cross-Origin Resource Sharing) which allows users to make a request to different domains.

Working of Fetch API

Fetch API is used to create HTTP requests from Javascript code in web browsers. So using the following steps we will learn how Fetch API works from sending requests to accepting responses −


Client-Side/ Web browser
Server
Develop a Request using fetch() function
Sending Request to the server
Server Receive request and start processing
Sending response to the web browser
Response Receive
Generate Response
Handling Response
If parsing required
If error found
Parse the response if needed
Handle the error

Following is the step by step explanation of the above flow diagram −

Step 1 − Request initialization: On the client side, a JavaScript program uses the fetch() function to create a request object. In this fetch() function, we pass the resource URL from where we fetch and other optional controls like header, method, body, etc.

Step 2 − Sending request: After initializing the request Fetch API sends the request to the server using the given URL. If the request is a GET request, then the browser sends the request directly to the server. If the request is other than a GET request, then the browser sends a preflight OPTIONS request to check if the server allows the request.

Step 3 − Server Processing: After receiving the request the server processes the request. It can perform various operations on the request like handling requests, retrieving data, etc.

Step 4 − Generating response: Now the server generates the response to the given request. The server response generally contains a status code(e.g 200 for success, 404 for request not found, etc.), response header, and optional body.

Step 5 − Receive Response: The web browser receives the response from the server. Now the Fetch API uses promises to resolve the response object send by the server.

Step 6 – Handling response: Fetch API uses promise-based syntax to handle the responses returned by the server. Using this we can access the response status, body, and header, and can perform an action on the received data.

Step 7 – Parse the Response: If the server response contains textual data, then the JavaScript program uses inbuilt methods like .json(), .text(), .blob(), etc to parse and extract data from the response.

Step 8 – Error Handling: If the server returns an error, then the error is handled by the catch() function.

These are the basic steps to understand the workflow of the fetch API. These steps can vary according to the complexity of the real-time usage. Also as we know that Fetch API is asynchronous so it does not block the execution of other Javascript code while waiting for the response from the server.

Advantages

The following are the advantages of Fetch API –

- **Easy to use** – Fetch API provides simple and straightforward syntax to create asynchronous requests.
- **Promise** – Fetch API uses promises due to which it can easily handle asynchronous operations. Promises provide a precise method to easily handle responses and errors.
- **Modern and browser support** – Fetch API is the modern web standard and it is in-built in web browsers due to which it is supported by almost all modern web browsers and platforms. This makes Fetch API more consistent and predictable as compared to XMLHttpRequest.
- **Streaming and progressive loading** – Fetch API supports streaming responses means we can start processing the response before it is fully loaded. It is generally useful for large files.
- **In-built JSON support** – Fetch API supports JSON data very efficiently. It can parse JSON responses and convert them into the JavaScript object automatically.
- **Integrate with other APIs** – Due to the behaviour of the Fetch API, it can easily integrate with other APIs like Service Worker API, Cache API, etc.
- **More Controls** – Using Fetch API we can easily customize the request with the help of additional parameters like header, method, body, etc.

Disadvantages

The following are the disadvantages of Fetch API −

- **Limited web browser support** − Fetch API is supported by almost all the modern web browsers but it is not supported by the older web browsers. If you are working with an older web browser, then you have to use older methods like XMLHttpRequest, etc.
- **Request Cancellation** − Fetch API does not provide any in-built method to cancel the initiated request.
- **Timeouts** − Fetch API does not provide any specified or in-built method to timeout a request. If you want to enforce a timeout for a request then you have to do it manually.
- **Error Handling** − Fetch API provides limited error-handling methods. It treats any HTTP status code other than 2xx as an error. This behaviour generally works for some specified cases but not for all cases.
- **Progress event for file load** − Fetch API does not provide any in-built event for the file upload. If you want to monitor the progress of file upload then you required extra libraries.
- **Cross-origin Limitation** − As we know that Fetch API follows the browser's same-origin policy so due to this cross-origin request required extra CORS headers on the server side or is subject to CORS preflight checks, which add-on extra complexity to the development.

Conclusion

Hence Fetch API is more powerful and flexible as compared to traditional approaches like XMLHttpRequest. It can easily integrate with other APIs and platforms. It is the commonly used method while working with HTTP requests in web applications. Now in the next article, we will learn about the differences between fetch API and XMLHttpRequest.