Fetch

Fetch API предоставляет интерфейс JavaScript для доступа и обработки частей протокола HTTP, таких как запросы и ответы. Оно также предоставляет глобальный метод `fetch()`, который даёт лёгкий, логический способ для извлечения ресурсов асинхронно по сети.

Такая функциональность ранее достигалась с помощью XMLHttpRequest. Fetch представляет собой лучшую альтернативу, которая может быть легко использована другими технологиями .

Оформление запросов

Базовый запрос на получение очень прост в настройке. Рассмотрите следующий код :

```
let myImage = document.querySelector('img');

fetch('/flowers.jpg').then(function(response) {
  return response.blob();
}).then(function(myBlob) {
  let objectURL = URL.createObjectURL(myBlob);
  myImage.src = objectURL;
});
```

Здесь мы забираем изображение по сети и вставляем его в `<img>` элемент. Самое простое использование `fetch()` принимает один аргумент — путь к ресурсу, который вы хотите получить — и возвращает promise, содержащее ответ (объект `Response`).

Это просто HTTP-ответ, а не фактическое изображение. Чтобы извлечь содержимое тела изображения из ответа, мы используем метод `blob()` (см. ниже, как и json()).

Затем, из полученного `Blob` создаётся `objectURL`, который вставляется в `img`.

Снабжение параметрами запроса

Метод `fetch()` может принимать второй параметр - обьект `init`, который позволяет вам контролировать различные настройки:

```
let myHeaders = new Headers();

let myInit = { method: 'GET',
               headers: myHeaders,
               mode: 'cors',
               cache: 'default' };
```

```
7
8  fetch('flowers.jpg', myInit).then(function(response) {
9    return response.blob();
10 }).then(function(myBlob) {
11   let objectURL = URL.createObjectURL(myBlob);
12   myImage.src = objectURL;
13 });
```

Проверка успешности извлечения данных

В методе `fetch()` promise будет отклонён (reject) с `TypeError`, когда случится ошибка сети или не будет сконфигурирован CORS на стороне запрашиваемого сервера, хотя обычно это означает проблемы доступа или аналогичные — для примера, 404 не является сетевой ошибкой. Для достоверной проверки успешности `fetch()` будет включать проверку того, что promise успешен (resolved), затем проверку того, что значение свойства `Response.ok` является true. Код будет выглядеть примерно так:

```
1  fetch('flowers.jpg').then(function(response) {
2    if(response.ok) {
3      return response.blob();
4    }
5    throw new Error('Ошибка сети!');
6  }).then(function(myBlob) {
7    let objectURL = URL.createObjectURL(myBlob);
8    myImage.src = objectURL;
9  }).catch(function(error) {
10   console.log('Какая-то проблема с fetch!' + error.message);
11 });
```

Заголовки

Интерфейс `Headers` позволяет вам создать ваш собственный объект заголовков через конструктор `Headers()`

Это может быть достигнуто путём передачи массива массивов или литерального объекта конструктору:

```
1  myHeaders = new Headers({
2    'Content-Type': 'text/plain',
3    'Content-Length': content.length.toString(),
4    'X-Custom-Header': 'Своё значение заголовка',
5  });
```

Хорошим вариантом использования заголовков является проверка корректности типа контента перед его обработкой. Например:

```
1  fetch(myRequest).then(function(response) {
2      let contentType = response.headers.get('Content-Type');
3      if(contentType && contentType.includes('application/json')) {
4        return response.json();
5      }
6      throw new TypeError('Нет валидного JSON!');
```

```
    throw new TypeError('Нет валидного JSON!');
  })
  .then(function(json) { /* process your JSON further */ })
  .catch(function(error) { console.log(error); });
```

Тело

Тело может содержаться как в запросе, так и в ответе

Следующие методы, которые возвращают промисы, используются для извлечения тела ответа в нужном формате:

- `arrayBuffer()`
- `blob()`
- `json()`
- `text()`
- `formData()`

Спецификации

Спецификации	Статус	Комментарий
Fetch	LS Живой стандарт	Initial definition