


Раздел

[Сетевые запросы](#)

Комментарии

Поделиться

[Редактировать на GitHub](#) → [Сетевые запросы](#) 8-го сентября 2019

Fetch: прерывание запроса

Как мы знаем, метод `fetch` возвращает промис. А в JavaScript в целом нет понятия «отмены» промиса. Как же прервать запрос `fetch`?

Для таких целей существует специальный встроенный объект: `AbortController`, который можно использовать для отмены не только `fetch`, но и других асинхронных задач.

Использовать его достаточно просто:

- Шаг 1: создаём контроллер:

```
1 let controller = new AbortController();
```

Контроллер `controller` – чрезвычайно простой объект.

- Он имеет единственный метод `abort()` и единственное свойство `signal`.
- При вызове `abort()`:
 - генерируется событие с именем `abort` на объекте `controller.signal`
 - свойство `controller.signal.aborted` становится равным `true`.

Все, кто хочет узнать о вызове `abort()`, ставят обработчики на `controller.signal`, чтобы отслеживать его.

Вот так (пока без `fetch`):

```
1 let controller = new AbortController();
2 let signal = controller.signal;
3
4 // срабатывает при вызове controller.abort()
5 signal.addEventListener('abort', () => alert("отмена!"));
6
7 controller.abort(); // отмена!
8
9 alert(signal.aborted); // true
```

- Шаг 2: передайте свойство `signal` опцией в метод `fetch`:

```
1 let controller = new AbortController();
2 fetch(url, {
3   signal: controller.signal
4 });
```

Метод `fetch` умеет работать с `AbortController`, он слушает событие `abort` на `signal`.

- Шаг 3: чтобы прервать выполнение `fetch`, вызовите `controller.abort()`:

```
1 controller.abort();
```

Вот и всё: `fetch` получает событие из `signal` и прерывает запрос.

Когда `fetch` отменяется, его промис завершается с ошибкой `AbortError`, поэтому мы должны обработать её, например, в `try...catch`:

Раздел

Сетевые запросы

Комментарии

Поделиться



Редактировать на GitHub



```
1 // прервать через 1 секунду
2 let controller = new AbortController();
3 setTimeout(() => controller.abort(), 1000);
4
5 try {
6   let response = await fetch('/article/fetch-abort/demo
7     signal: controller.signal
8   });
9 } catch(err) {
10   if (err.name == 'AbortError') { // обработать ошибку
11     alert("Прервано!");
12   } else {
13     throw err;
14   }
15 }
```

AbortController – масштабируемый, он позволяет отменить несколько вызовов fetch одновременно.

Например, здесь мы запрашиваем много URL параллельно, и контроллер прерывает их все:

```
1 let urls = [...]; // список URL для параллельных fetch
2
3 let controller = new AbortController();
4
5 let fetchJobs = urls.map(url => fetch(url, {
6   signal: controller.signal
7 }));
8
9 let results = await Promise.all(fetchJobs);
10
11 // если откуда-то вызвать controller.abort(),
12 // то это прервёт все вызовы fetch
```

Если у нас есть собственные асинхронные задачи, отличные от fetch, мы можем использовать один AbortController для их остановки вместе с fetch.

Нужно лишь слушать его событие abort:

```
1 let urls = [...];
2 let controller = new AbortController();
3
4 let ourJob = new Promise((resolve, reject) => { // наша
5   ...
6   controller.signal.addEventListener('abort', reject);
7 });
8
9 let fetchJobs = urls.map(url => fetch(url, { // запросы
10   signal: controller.signal
11 }));
12
13 // ожидать выполнения нашей задачи и всех запросов
14 let results = await Promise.all([...fetchJobs, ourJob])
15
16 // вызов откуда-нибудь ещё:
17 // controller.abort() прервёт все вызовы fetch и наши з.
```

Так что AbortController существует не только для fetch, это универсальный объект для отмены асинхронных задач, в fetch встроена интеграция с ним.

Раздел

[Сетевые запросы](#)

Комментарии

Поделиться



[Редактировать на GitHub](#)



© 2007–2020 Илья Кантор | [о проекте](#) | [связаться с нами](#) | [пользовательское соглашение](#) | [политика конфи](#)

