

Раздел

Сетевые запросы

Навигация по уроку

referrer, referrerPolicy

mode

credentials

cache

redirect

integrity



keepalive

Комментарии

Поделиться



Редактировать на GitHub

 → Сетевые запросы 8-го сентября 2019

Fetch API

На данный момент мы уже многое знаем про `fetch`.

Давайте рассмотрим оставшуюся часть API, чтобы охватить все возможности.

На заметку:

Заметим: большинство этих возможностей используются редко. Вы можете пропустить эту главу и, несмотря на это, нормально использовать `fetch`.

Тем не менее, полезно знать, что вообще может `fetch`, чтобы, когда появится необходимость, вернуться и прочитать конкретные детали.

Нижеследующий список – это все возможные опции для `fetch` с соответствующими значениями по умолчанию (в комментариях указаны альтернативные значения):

```
1 let promise = fetch(url, {
2   method: "GET", // POST, PUT, DELETE, etc.
3   headers: {
4     // значение этого заголовка обычно ставится автомат
5     // в зависимости от тела запроса
6     "Content-Type": "text/plain;charset=UTF-8"
7   },
8   body: undefined // string, FormData, Blob, BufferSource
9   referrer: "about:client", // или "" для того, чтобы не
10  // или URL с текущего источника
11  referrerPolicy: "no-referrer-when-downgrade", // no-ref
12  mode: "cors", // same-origin, no-cors
13  credentials: "same-origin", // omit, include
14  cache: "default", // no-store, reload, no-cache, forced-c
15  redirect: "follow", // manual, error
16  integrity: "", // контрольная сумма, например "sha256-
17  keepalive: false, // true
18  signal: undefined, // AbortController, чтобы прервать
19  window: window // null
20 });
```

Довольно-таки внушительный список, не так ли?

В главе [Fetch](#) мы разобрали параметры `method`, `headers` и `body`.

Опция `signal` разъяснена в главе в [Fetch: прерывание запроса](#).

Теперь давайте пройдемся по оставшимся возможностям.

referrer, referrerPolicy

Данные опции определяют, как `fetch` устанавливает HTTP-заголовок `Referer`.

Обычно этот заголовок ставится автоматически и содержит URL-адрес страницы, с которой пришёл запрос. В большинстве случаев он совсем неважен, в некоторых случаях, с целью большей безопасности, имеет смысл убрать или укоротить его.

Опция `referrer` позволяет установить любой `Referer` в пределах текущего источника или же убрать его.

Чтобы не отправлять `Referer`, нужно указать значением пустую строку:

Раздел

Сетевые запросы

Навигация по уроку

referrer, referrerPolicy

mode

credentials

cache

redirect

integrity

keepalive

Комментарии

Поделиться



Редактировать на GitHub



```
1 fetch('/page', {
2   referrer: "" // не ставить заголовок Referer
3 });
```

Для того, чтобы установить другой URL-адрес (должен быть с текущего источника):

```
1 fetch('/page', {
2   // предположим, что мы находимся на странице https://
3   // мы можем установить любое значение Referer при усл
4   referrer: "https://javascript.info/anotherpage"
5 });
```

Опция `referrerPolicy` устанавливает общие правила для `Referer`.

Выделяется 3 типа запросов:

1. Запрос на тот же источник.
2. Запрос на другой источник.
3. Запрос с HTTPS to HTTP (с безопасного протокола на небезопасный).

В отличие от настройки `referrer`, которая позволяет задать точное значение `Referer`, настройка `referrerPolicy` сообщает браузеру общие правила, что делать для каждого типа запроса.

Возможные значения описаны в [спецификации Referrer Policy](#):

- **"no-referrer-when-downgrade"** – это значение по умолчанию: `Referer` отправляется всегда, если только мы не отправим запрос из HTTPS в HTTP (из более безопасного протокола в менее безопасный).
- **"no-referrer"** – никогда не отправлять `Referer`.
- **"origin"** – отправлять в `Referer` только текущий источник, а не полный URL-адрес страницы, например, посылать только `http://site.com` вместо `http://site.com/path`.
- **"origin-when-cross-origin"** – отправлять полный `Referer` для запросов в пределах текущего источника, но для запросов на другой источник отправлять только сам источник (как выше).
- **"same-origin"** – отправлять полный `Referer` для запросов в пределах текущего источника, а для запросов на другой источник не отправлять его вообще.
- **"strict-origin"** – отправлять только значение источника, не отправлять `Referer` для HTTPS → HTTP запросов.
- **"strict-origin-when-cross-origin"** – для запросов в пределах текущего источника отправлять полный `Referer`, для запросов на другой источник отправлять только значение источника, в случае HTTPS → HTTP запросов не отправлять ничего.
- **"unsafe-url"** – всегда отправлять полный URL-адрес в `Referer`, даже при запросах HTTPS → HTTP.

Вот таблица со всеми комбинациями:

Значение	На тот же источник	На другой источник	HTTPS → HTTP
"no-referrer"	-	-	-
"no-referrer-when-downgrade" или "" (по умолчанию)	full	full	-
"origin"	origin	origin	origin
"origin-when-cross-origin"	full	origin	origin
"same-origin"	full	-	-
"strict-origin"	origin	origin	-
"strict-origin-when-cross-origin"	full	origin	-
"unsafe-url"	full	full	full

Раздел

[Сетевые запросы](#)

Навигация по уроку

referrer, referrerPolicy

mode

credentials

cache

redirect

integrity

keepalive

Комментарии

Поделиться



[Редактировать на GitHub](#)



Допустим, у нас есть админка со структурой URL, которая не должна стать известной снаружи сайта.

Если мы отправляем запрос `fetch`, то по умолчанию он всегда отправляет заголовок `Referer` с полным URL-адресом нашей админки (исключение – это когда мы делаем запрос от HTTPS в HTTP, в таком случае `Referer` не будет отправляться).

Например, `Referer: https://javascript.info/admin/secret/paths`.

Если мы хотим, чтобы другие сайты получали только источник, но не URL-путь, это сделает такая настройка:

```
1 fetch('https://another.com/page', {
2   // ...
3   referrerPolicy: "origin-when-cross-origin" // Referer
4 });
```

Мы можем поставить её во все вызовы `fetch`, возможно, интегрировать в JavaScript-библиотеку нашего проекта, которая делает все запросы и внутри использует `fetch`.

Единственным отличием в поведении будет то, что для всех запросов на другой источник `fetch` будет посылать только источник в заголовке `Referer` (например, `https://javascript.info`, без пути). А для запросов на наш источник мы продолжим получать полный `Referer` (это может быть полезно для отладки).

Политика установки Referer (Referer Policy) – не только для fetch

Политика установки Referer, описанная в [спецификации Referrer Policy](#), существует не только для `fetch`, она более глобальная.

В частности, можно поставить политику по умолчанию для всей страницы, используя HTTP-заголовок `Referrer-Policy`, или на уровне ссылки ``.

mode

Опция `mode` – это защита от нечаянной отправки запроса на другой источник:

- `"cors"` – стоит по умолчанию, позволяет делать такие запросы так, как описано в [Fetch: запросы на другие сайты](#),
- `"same-origin"` – запросы на другой источник запрещены,
- `"no-cors"` – разрешены только простые запросы на другой источник.

Эта опция может пригодиться, если URL-адрес для `fetch` приходит от третьей стороны, и нам нужен своего рода «глобальный выключатель» для запросов на другие источники.

credentials

Опция `credentials` указывает, должен ли `fetch` отправлять куки и авторизационные заголовки HTTP вместе с запросом.

- `"same-origin"` – стоит по умолчанию, не отправлять для запросов на другой источник,
- `"include"` – отправлять всегда, но при этом необходим заголовок `Access-Control-Allow-Credentials` в ответе от сервера, чтобы JavaScript получил доступ к ответу сервера, об этом говорилось в главе [Fetch: запросы на другие сайты](#),
- `"omit"` – не отправлять ни при каких обстоятельствах, даже для запросов, сделанных в пределах текущего источника.

cache

Раздел

Сетевые запросы

Навигация по уроку

referrer, referrerPolicy

mode

credentials

cache

redirect

integrity

keepalive

Комментарии

Поделиться



Редактировать на GitHub



По умолчанию `fetch` делает запросы, используя стандартное HTTP-кеширование. То есть, учитываются заголовки `Expires`, `Cache-Control`, отправляется `If-Modified-Since` и так далее. Так же, как и обычные HTTP-запросы.

Настройка `cache` позволяет игнорировать HTTP-кеш или же настроить его использование:

- **"default"** – `fetch` будет использовать стандартные правила и заголовки HTTP кеширования,
- **"no-store"** – полностью игнорировать HTTP-кеш, этот режим становится режимом по умолчанию, если присутствуют такие заголовки как `If-Modified-Since`, `If-None-Match`, `If-Unmodified-Since`, `If-Match`, или `If-Range`,
- **"reload"** – не брать результат из HTTP-кеша (даже при его присутствии), но сохранить ответ в кеше (если это дозволено заголовками ответа);
- **"no-cache"** – в случае, если существует кешированный ответ – создать условный запрос, в противном же случае – обычный запрос. Сохранить ответ в HTTP-кеше,
- **"force-cache"** – использовать ответ из HTTP-кеша, даже если он устаревший. Если же ответ в HTTP-кеше отсутствует, сделать обычный HTTP-запрос, действовать как обычно,
- **"only-if-cached"** – использовать ответ из HTTP-кеша, даже если он устаревший. Если же ответ в HTTP-кеше отсутствует, то выдаётся ошибка. Это работает, только когда `mode` установлен в `"same-origin"`.

redirect

Обычно `fetch` прозрачно следует HTTP-редиректам, таким как 301, 302 и так далее.

Это можно поменять при помощи опции `redirect`:

- **"follow"** – стоит по умолчанию, следовать HTTP-редиректам,
- **"error"** – ошибка в случае HTTP-редиректа,
- **"manual"** – не следовать HTTP-редиректу, но установить адрес редиректа в `response.url`, а `response.redirected` будет иметь значение `true`, чтобы мы могли сделать перенаправление на новый адрес вручную.

integrity

Опция `integrity` позволяет проверить, соответствует ли ответ известной заранее контрольной сумме.

Как описано в [спецификации](#), поддерживаемыми хеш-функциями являются SHA-256, SHA-384 и SHA-512. В зависимости от браузера, могут быть и другие.

Например, мы скачиваем файл, и мы точно знаем, что его контрольная сумма по алгоритму SHA-256 равна «abcdef» (разумеется, настоящая контрольная сумма будет длиннее).

Мы можем добавить это в настройку `integrity` вот так:

```
1 fetch('http://site.com/file', {
2   integrity: 'sha256-abcdef'
3 });
```

Затем `fetch` самостоятельно вычислит SHA-256 и сравнит его с нашей строкой. В случае несоответствия будет ошибка.

keepalive

Опция `keepalive` указывает на то, что запрос может «пережить» страницу, которая его отправила.

Например, мы собираем статистические данные о том, как посетитель ведёт себя на нашей странице (на что он кликает, части страницы, которые он просматривает), для анализа и улучшения интерфейса.

Когда посетитель покидает нашу страницу – мы хотим сохранить собранные данные на нашем сервере.

Для этого мы можем использовать событие `window.onunload`:

```
1 window.onunload = function() {  
2   fetch('/analytics', {  
3     method: 'POST',  
4     body: "statistics",  
5     keepalive: true  
6   });  
7 };
```

Обычно, когда документ выгружается, все связанные с ним сетевые запросы прерываются. Но настройка `keepalive` указывает браузеру выполнять запрос в фоновом режиме даже после того, как пользователь покидает страницу. Поэтому эта опция обязательна, чтобы такой запрос удался.

У неё есть ряд ограничений:

- Мы не можем посылать мегабайты: лимит тела для запроса с `keepalive` – 64кб.
- Если мы собираем больше данных, можем отправлять их регулярно, «пакетами», тогда на момент последнего запроса в `onunload` их останется немного.
- Этот лимит распространяется на все запросы с `keepalive`. То есть, мы не можем его обойти, послав 100 запросов одновременно – каждый по 64Кбайт.
- Мы не сможем обработать ответ от сервера, если запрос сделан при `onunload`: в тот момент документ уже выгружен, его функции не работают.
- Обычно сервер посылает пустой ответ на такие запросы, так что это не является проблемой.

Проводим [курсы по JavaScript и фреймворкам](#). ✕

💬 Комментарии

перед тем как писать...