

Раздел

[Сетевые запросы](#)

Навигация по уроку

Получение сообщений

Переподключение

Идентификатор сообщения

Статус подключения:  
readyState

Типы событий

Полный пример


Итого

Комментарии

Поделиться



Редактировать на GitHub

 → [Сетевые запросы](#) 11-го мая 2020

## Server Sent Events

Спецификация [Server-Sent Events](#) описывает встроенный класс `EventSource`, который позволяет поддерживать соединение с сервером и получать от него события.

Как и в случае с `WebSocket`, соединение постоянно.

Но есть несколько важных различий:

WebSocket	EventSource
Двунаправленность: и сервер, и клиент могут обмениваться сообщениями	Однонаправленность: данные посылает только сервер
Бинарные и текстовые данные	Только текст
Протокол WebSocket	Обычный HTTP

`EventSource` не настолько мощный способ коммуникации с сервером, как `WebSocket`.

Зачем нам его использовать?

Основная причина: он проще. Многим приложениям не требуется вся мощь `WebSocket`.

Если нам нужно получать поток данных с сервера: неважно, сообщения в чате или же цены для магазина – с этим легко справится `EventSource`. К тому же, он поддерживает автоматическое переподключение при потере соединения, которое, используя `WebSocket`, нам бы пришлось реализовывать самим. Кроме того, используется старый добрый HTTP, а не новый протокол.

### Получение сообщений

Чтобы начать получать данные, нам нужно просто создать `new EventSource(url)`.

Браузер установит соединение с `url` и будет поддерживать его открытым, ожидая события.

Сервер должен ответить со статусом 200 и заголовком `Content-Type: text/event-stream`, затем он должен поддерживать соединение открытым и отправлять сообщения в особом формате:

```
1 data: Сообщение 1
2
3 data: Сообщение 2
4
5 data: Сообщение 3
6 data: в две строки
```

- Текст сообщения указывается после `data:`, пробел после двоеточия необязателен.
- Сообщения разделяются двойным переносом строки `\n\n`.
- Чтобы разделить сообщение на несколько строк, мы можем отправить несколько `data:` подряд (третье сообщение).

На практике сложные сообщения обычно отправляются в формате JSON, в котором перевод строки кодируется как `\n`, так что в разделении сообщения на несколько строк обычно нет нужды.

Например:

```
1 data: {"user":"Джон","message":"Первая строка\nВторая"}
```

Раздел

Сетевые запросы

Навигация по уроку

Получение сообщений

Переподключение

Идентификатор сообщения

Статус подключения:  
readyState

Типы событий

Полный пример

Итого

Комментарии

Поделиться



Редактировать на GitHub



...Так что можно считать, что в каждом `data:` содержится ровно одно сообщение.

Для каждого сообщения генерируется событие `message`:

```
1 let eventSource = new EventSource("/events/subscribe");
2
3 eventSource.onmessage = function(event) {
4   console.log("Новое сообщение", event.data);
5   // этот код выведет в консоль 3 сообщения, из потока ,
6 };
7
8 // или eventSource.addEventListener('message', ...)
```

## Кросс-доменные запросы

`EventSource`, как и `fetch`, поддерживает кросс-доменные запросы. Мы можем использовать любой URL:

```
1 let source = new EventSource("https://another-site.com/");
```

Сервер получит заголовок `Origin` и должен будет ответить с заголовком `Access-Control-Allow-Origin`.

Чтобы послать авторизационные данные, следует установить дополнительную опцию `withCredentials`:

```
1 let source = new EventSource("https://another-site.com/");
2   withCredentials: true
3 });
```

Более подробное описание кросс-доменных заголовков вы можете прочитать в главе [Fetch: запросы на другие сайты](#).

## Переподключение

После создания `new EventSource` подключается к серверу и, если соединение обрывается, — переподключается.

Это очень удобно, так как нам не приходится беспокоиться об этом.

По умолчанию между попытками возобновить соединение будет небольшая пауза в несколько секунд.

Сервер может выставить рекомендуемую задержку, указав в ответе `retry`: (в миллисекундах):

```
1 retry: 15000
2 data: Привет, я выставил задержку переподключения в 15
```

Поле `retry`: может посылаться как вместе с данными, так и отдельным сообщением.

Браузеру следует ждать именно столько миллисекунд перед новой попыткой подключения. Или дольше, например, если браузер знает (от операционной системы) что соединения с сетью нет, то он может осуществить переподключение только когда оно появится.

- Если сервер хочет остановить попытки переподключения, он должен ответить со статусом 204.
- Если браузер хочет прекратить соединение, он может вызвать `eventSource.close()`:

```
1 let eventSource = new EventSource(...);
2
```

```
3 eventSource.close();
```

Также переподключение не произойдёт, если в ответе указан неверный Content-Type или его статус отличается от 301, 307, 200 и 204. Браузер создаст событие "error" и не будет восстанавливать соединение.

#### На заметку:

После того как соединение окончательно закрыто, «переоткрыть» его уже нельзя. Если необходимо снова подключиться, просто создайте новый EventSource.

## Идентификатор сообщения

Когда соединение прерывается из-за проблем с сетью, ни сервер, ни клиент не могут быть уверены в том, какие сообщения были доставлены, а какие – нет.

Чтобы правильно возобновить подключение, каждое сообщение должно иметь поле id:

```
1 data: Сообщение 1
2 id: 1
3
4 data: Сообщение 2
5 id: 2
6
7 data: Сообщение 3
8 data: в две строки
9 id: 3
```

Получая сообщение с указанным id, браузер:

- Установит его значение свойству eventSource.lastEventId.
- При переподключении отправит заголовок Last-Event-ID с этим id, чтобы сервер мог переслать последующие сообщения.

#### Указывайте id: после data:

Обратите внимание: id указывается сервером после данных data сообщения, чтобы обновление lastEventId произошло после того, как сообщение будет получено.

## Статус подключения: readyState

У объекта EventSource есть свойство readyState, имеющее одно из трёх значений:

```
1 EventSource.CONNECTING = 0; // подключение или переподк.
2 EventSource.OPEN = 1;      // подключено
3 EventSource.CLOSED = 2;     // подключение закрыто
```

При создании объекта и разрыве соединения оно автоматически устанавливается в значение EventSource.CONNECTING (равно 0).

Мы можем обратиться к этому свойству, чтобы узнать текущее состояние EventSource.

## Типы событий

По умолчанию объект EventSource генерирует 3 события:

- message – получено сообщение, доступно как event.data.
- open – соединение открыто.

Раздел

[Сетевые запросы](#)

Навигация по уроку

Получение сообщений

Переподключение

Идентификатор сообщения

Статус подключения:  
readyState

Типы событий

Полный пример

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)

Раздел

[Сетевые запросы](#)

Навигация по уроку

Получение сообщений

Перепоключение

Идентификатор сообщения

Статус подключения:  
readyState

Типы событий

Полный пример

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)

- `error` – не удалось установить соединение, например, сервер вернул статус 500.

Сервер может указать другой тип события с помощью `event`: ... в начале сообщения.

Например:

```
1 event: join
2 data: Боб
3
4 data: Привет
5
6 event: leave
7 data: Боб
```

Чтобы начать слушать пользовательские события, нужно использовать `addEventListener`, а не `onmessage`:

```
1 eventSource.addEventListener('join', event => {
2   alert(`${event.data} зашёл`);
3 });
4
5 eventSource.addEventListener('message', event => {
6   alert(`Сказал: ${event.data}`);
7 });
8
9 eventSource.addEventListener('leave', event => {
10  alert(`${event.data} вышел`);
11 });
```

## Полный пример

В этом примере сервер посылает сообщения 1, 2, 3, затем пока-пока и разрывает соединение.

После этого браузер автоматически перепоключается.

Результат    server.js    index.html



Нажмите кнопку "Старт" для начала  
 Чтобы закончить, нажмите "Стоп".

## Итого

Объект `EventSource` автоматически устанавливает постоянное соединение и позволяет серверу отправлять через него сообщения.

Он предоставляет:

- Автоматическое перепоключение с настраиваемой `retry` задержкой.
- Идентификаторы сообщений для восстановления соединения. Последний полученный идентификатор посылается в заголовке `Last-Event-ID` при пересоединении.
- Текущее состояние, записанное в свойстве `readyState`.

Это делает `EventSource` достойной альтернативой протоколу `WebSocket`, который сравнительно низкоуровневый и не имеет таких встроенных возможностей (хотя их и можно реализовать).

Для многих приложений возможностей `EventSource` вполне достаточно.

Поддерживается во всех современных браузерах (кроме Internet Explorer).

Синтаксис:

```
1 let source = new EventSource(url, [credentials]);
```

Второй аргумент – необязательный объект с одним свойством: `{ withCredentials: true }`. Он позволяет отправлять авторизационные данные на другие домены.

В целом, кросс-доменная безопасность реализована так же как в `fetch` и других методах работы с сетью.

## Свойства объекта `EventSource`

### `readyState`

Текущее состояние подключения: `EventSource.CONNECTING` (=0), `EventSource.OPEN` (=1) или `EventSource.CLOSED` (=2).

### `lastEventId`

`id` последнего полученного сообщения. При переподключении браузер посылает его в заголовке `Last-Event-ID`.

## Методы

### `close()`

Закрывает соединение.

## События

### `message`

Сообщение получено, переданные данные записаны в `event.data`.

### `open`

Соединение установлено.

### `error`

В случае ошибки, включая как потерю соединения, так и другие ошибки в нём. Мы можем обратиться к свойству `readyState`, чтобы проверить, происходит ли переподключение.

Сервер может выставить собственное событие с помощью `event`:. Такие события должны быть обработаны с помощью `addEventListener`, а не `on<event>`.

## Формат ответа сервера

Сервер посылает сообщения, разделённые двойным переносом строки `\n\n`.

Сообщение состоит из следующих полей:

- `data`: – тело сообщения, несколько `data` подряд интерпретируются как одно сообщение, разделённое переносами строк `\n`.
- `id`: – обновляет свойство `lastEventId`, отправляемое в `Last-Event-ID` при переподключении.
- `retry`: – рекомендованная задержка перед переподключением в миллисекундах. Не может быть установлена с помощью JavaScript.
- `event`: – имя пользовательского события, должно быть указано перед `data`:.

Сообщение может включать одно или несколько этих полей в любом порядке, но `id` обычно ставят в конце.

Раздел

Сетевые запросы

Навигация по уроку

Получение сообщений

Переподключение

Идентификатор сообщения

Статус подключения:  
`readyState`

Типы событий

Полный пример

Итого

Комментарии

Поделиться



Редактировать на GitHub

Раздел

[Сетевые запросы](#)

Навигация по уроку

Получение сообщений

Переподключение

Идентификатор сообщения

Статус подключения:  
readyState

Типы событий

Полный пример

Итого

Комментарии

Поделиться



[Редактировать на GitHub](#)



## Комментарии

перед тем как писать...



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

