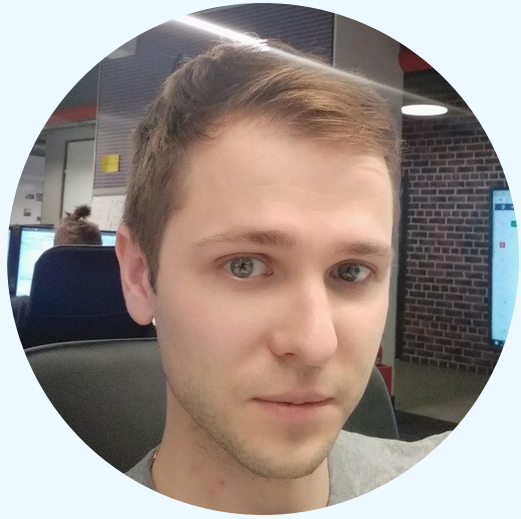
НЕТОЛОГИЯ

ХРАНЕНИЕ СОСТОЯНИЯ НА КЛИЕНТЕ И ОТПРАВКА НА СЕРВЕР

АЛЕКСАНДР РУСАКОВ



АЛЕКСАНДР РУСАКОВ

Веб-разработчик

a_s_rusakov@mail.ru

@a_s_rusakov

ПЛАН ЗАНЯТИЯ

1. `localStorage`
2. `document.cookie`
3. Сравнение `localStorage`, `sessionStorage`, `cookie`
4. Кроссдоменные запросы, CORS, `XHR.withCredentials`

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Вопрос: какая разница между синхронным и асинхронными запросами в JavaScript?

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Ответ: при синхронном JS запросы выполняются последовательно друг за другом, а при асинхронном одновременномогут выполняться разные операции.

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Вопрос: что позволяет делать объект `XMLHttpRequest`?

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Ответ: объект `XMLHttpRequest` (или, сокращенно, XHR) дает возможность браузеру делать HTTP-запросы к серверу без перезагрузки страницы.

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Вопрос: какие вы знаете основные свойства `XMLHttpRequest`?

ВСПОМНИМ ПРОШЛЫЕ ЗАНЯТИЯ

Ответ:

- `status`;
- `statusText`;
- `responseText`;
- `onreadystatechange`;
- `readyState`.

`localStorage`

ЗАЧЕМ НУЖЕН `localStorage`?

`localStorage` — это специальный объект BOM, с помощью которого можно долговременно хранить информацию на клиенте. Для каждого домена свое собственное хранилище.

Для доступа к данным предоставляется удобный интерфейс для чтения и записи пар ключ-значение (строки).

ИСПОЛЬЗОВАНИЕ

Обычно существуютlocalStorage используют для:

- Хранения идентификаторов пользователя/сессии, персональной информации;
- Для удобства пользователя: для сохранения какой-то информации, которую пользователь ввел, но еще не отправил на сервер;
- Для быстродействия: если есть информация в `localStorage`, то не нужно делать лишний запрос на сервер.

Общий размер памяти под `localStorage` обязательно ограничен браузером. Современные НЕ мобильные браузеры предоставляют до 10 Мб под `localStorage` для каждого домена, мобильные обычно меньше.

ЗАПИСЬ И ЧТЕНИЕ

Давайте начнем с записи в `localStorage`. Это можно сделать двумя различными способами — с помощью метода `setItem()` или непосредственно с помощью свойства объекта.

Это пригодится при выполнении домашнего задания.

```
1  localStorage.setItem('lastname', 'Иванов');
2  // или как со свойством объекта
3  localStorage['lastname'] = 'Петров';
4  // можно и без квадратных скобок
5  localStorage.lastname = 'Сидоров'
```

ЗАПИСЬ И ЧТЕНИЕ

Теперь даже после перезагрузки страницы или браузера целиком фамилию можно будет прочитать снова либо через метод `getItem()`, либо через свойство объекта:

Это пригодится при выполнении домашнего задания.

```
let lastname = localStorage.getItem('lastname');
console.log(lastname); // 'Сидоров'

// или через чтения свойства объекта
console.log(localStorage.lastname); // 'Сидоров'
```

У объекта `localStorage` есть еще полезные свойство `length` – это общее количество ключей, которые определены в данный момент в `localStorage`.

УДАЛЕНИЕ ЗАПИСЕЙ

`localStorage.clear()` — полностью очищает текущее хранилище. Например, его нужно вызывать во время log out с сайта, чтобы удалить все связанные с пользователем данные.

Кроме полной очистки хранилища, можно удалить конкретный ключ и связанные с ним значения с `removeItem()` или с помощью оператора `delete`.

```
localStorage.removeItem('user');
delete localStorage.user;
```

РАБОТА С ОБЪЕКТАМИ

В качестве значения в `localStorage` могут выступать только строки. А что, если необходимо записать объект?
Для этого необходимо использовать сериализация/десериализация с помощью `JSON.stringify()` и `JSON.parse()` соответственно.

```
function saveUser(user) {
  localStorage.user = JSON.stringify(user);
}

function getUser() {
  return JSON.parse(localStorage.user);
}
```

РАБОТА С ОБЪЕКТАМИ

Причем данная реализация `getUser()` не очень безопасная, потому что вызов `JSON.parse()` с любыми некорректными данными приведет к исключению. Поэтому при чтении объектов из `localStorage` лучше всегда использовать `try-catch`:

```
function getUser() {
  try {
    return JSON.parse(localStorage.getItem('user'));
  } catch (e) {
    return null;
  }
}
```

ПЕРЕПОЛНЕНИЕ

Если постоянно записывать новые данные в `localStorage`, то можно достичь того самого ограничения общего размера. Или этого можно достичь разовой записью, например, вот так:

```
localStorage.bigdata = new Array(1e7).join('x')
// Uncaught DOMException:
// Failed to set the 'bigdata' property on 'Storage':
// Setting the value of 'bigdata' exceeded the quota.
```

СОБЫТИЯ НА ИЗМЕНЕНИЯ `localStorage`

Можно подписаться на событие, которое возникает во время измнения содержимого `localStorage`. Это событие с названием storage у объекта `window`.

Это событие не будет возникать на той же вкладке (или странице, если у браузера нет вкладок), где вносятся изменения, т.к. оно является способом для других вкладок на том же домене использовать хранилище для синхронизации любых внесенных изменений.

СОБЫТИЯ НА ИЗМЕНЕНИЯ `localStorage`

Рассмотрим пример, в котором пользователь, у которого сайт открыт сразу в нескольких вкладках, поменял тему со светлой на темную. Сделаем так, чтобы в остальных вкладках тема тоже изменилась:

```
// 1) при изменении пользователем темы на сайте выполнилась строка скрипта:
localStorage.theme = "dark";

// 2) в другой вкладке сработал следующий код:
window.addEventListener('storage', (e) => {
  console.log(e.key, e.oldValue, e.newValue); // будет выведено: "theme", "dark", "light"
  document.body.classList = [e.newValue]; // 3) проставим класс темы для body
});
```

КАК СОХРАНИТЬ ЗНАЧЕНИЕ В `localStorage`?

Вопрос: необходимо запомнить имя пользователя в `localStorage`.
Выберите подходящие варианты:

1. `localStorage.addItem('userName', 'Пупкин');`
2. `localStorage.userName = 'Пупкин';`
3. `localStorage = 'userName=Пупкин';`

КАК СОХРАНИТЬ ЗНАЧЕНИЕ В `localStorage`?

Ответ: 2.

РЕЗЮМИРУЕМ

`localStorage`:

- является специальным объектом BOM, с помощью которого можно долговременно хранить информацию на клиенте;
- доступен в рамках домена, на котором установлен;
- ограничен только общий размер хранимых данных;
- имеет удобный интерфейс для работы;
- не имеет автоматической отправки на сервер.

```
document.cookie
```

ЗАЧЕМ НУЖНЫ COOKIES?

Проблема: сервер хочет принимать данные только от авторизованных пользователей. Но как это проверить?
Для решения этой задачи и были придуманы куки.

Куки — это пары строк ключ-значение, которые могут сохраняться даже после закрытия браузера.

Для индентификации конкретного пользователя сервер записывает в куки специальный идентификатор (строку или число), который определяют сессию и соответственно конкретного пользователя для сервера. А затем, вместе с каждым запросом браузер отправляет на сервер куки автоматически.

УСТАНОВКА

Устанавливать куки можно либо с помощью JavaScript кода, либо со стороны сервера с помощью HTTP-заголовка `Set-Cookie`.

```
Set-Cookie: sessionid=XXXyyyZZZ
```

Для работы с куками силами JavaScript в браузере существует специальный BOM-объект `document.cookie`.

Через него можно как устанавливать новые значения, так и читать существующие.

УСТАНОВКА

Запись в куки осуществляется с помощью присваивания в `document.cookie`. Причем такое присваивание не перезаписывает старые cookie, а добавляет новую. Новую куку нужно записывать в виде одной строки, где ключ отделен от значения знаком `=`.

Это пригодится при выполнении домашнего задания.

```
document.cookie = 'firstname=Иван';
document.cookie = 'lastname=Петров';
```

Если значение содержит пробелы, точки с запятой или запятые, то его необходимо предобработать с помощью функции `encodeURIComponent()`:

```
document.cookie = 'user=' + encodeURIComponent('Иван Иваныч Иванов; 1945 г.');
```

ПОЛУЧЕНИЕ

А вот прочитать можно только сразу все куки, которые установлены на сайте. Это одна большая строка, в которой куки разделены между собой знаком `;`.

```
console.log(document.cookie);
// firstname=Иван; lastname=Петров
```

Это достаточно парадоксально. Куки — это пары ключ-значение, но прочитать их можно только все сразу в виде одной строки. Для более удобной работы с куками лучше использовать дополнительные фукнции-помощники (или целую библиотеку) для чтения конкретной куки и создания новой.

ПОЛУЧЕНИЕ

Это пригодится при выполнении домашнего задания.

Рассмотрим функцию, которую можно использовать для получения значения куки по имени:

```
const getCookie = (name) => {
  const value = "; " + document.cookie;
  let parts = value.split("; " + name + "=");
  if (parts.length === 2) {
    return parts
      .pop()
      .split(";")
      .shift();
  }
}
```

ПОЛУЧЕНИЕ

В этой функции в строке 3 строка, которая представляет собой куки и их значения с добавленной ';' в конце, разбивается на 2 части функцией `split`. Разделителем `split` служит `;name=`, где `name` - имя куки.

В строке 4 проверяется, разделилась ли строка на 2, т.е. встретился ли в ней разделитель `;name=`. Если нет, то нет такой куки. Если да, то происходит следующее:

- `parts.pop()` возвращает 2ю часть строки, где как раз содержится значение искомой куки вместе с другими (стр. 6)
- `split(";")` - полученная подстрока разбивается на отдельные куки разделителем ';' (стр. 7)
- `shift()` - берется первый элемент из получившегося массива кук (стр. 8), где содержится значение искомой куки (при этом имя куки отсутствует, т.к. оно служило разделителем).

РАЗЛИЧНЫЕ ОПЦИИ ДЛЯ КУК

На самом деле у куки кроме значения может быть масса дополнительных параметров, которые можно указать при создании куки.

```
document.cookie = 'name=value;
                   Expires=Mon, 01 May 2018 21:41:37 GMT;
                   Path=/api; Domain=.mysite.com';
```

– `Expires` — определяет время жизни куки. Если не указывать, то кука исчезнет после закрытия браузера;

– `Path` — для какого пути (и всех путей, начинающихся на этот путь) кука будет автоматически подставляться в запрос. Если не указано — значит, берется текущий путь;

– `Domain` — домен, на котором доступна кука. Можно указывать текущий домен или поддомены. Если не указано, то будет взят текущий домент. Если как в примере (с точкой в начале), то кука установлена для домена и всех поддоменов.

УДАЛЕНИЕ КУКИ

Удалить куку явно нельзя, но можно установить ей дату окончания существование в далекое прошлое и браузер удалит такую куку автоматически:

```
document.cookie = 'name=; Expires=Thu, 01 Jan 1970 00:00:00 GMT';
```

В этом случае значение можно вообще не указывать.

ОГРАНИЧЕНИЯ КУКИ

Все браузеры накладывают на куки ограничения по размеру, как на размер отдельной куки, так и на общее количество:

- имя и значение не должны превышать 4кб;
- общее количество кук на домен имеет ограничение — не более 50. В некоторых браузерах еще меньше.

КАК УСТАНОВИТЬ КУКИ?

Вопрос: как установить куки с названием user со значением «Вася Пупкин» с истекающей датой 1 Мая 2019?

1. `document.cookie = 'user=' + encodeURIComponent("Вася Пупкин") + "; Expires=Mon, 01 May 2019 00:00:00 GMT;'`
2. `document.setCookie = 'user=' + encodeURIComponent("Вася Пупкин") + "; Expires=Mon, 01 May 2019 00:00:00 GMT;'`
3. `document.cookie = 'user="Вася, Пупкин"; Expires=Mon, 01 May 2018 00:00:00 GMT;''`

КАК УСТАНОВИТЬ КУКИ?

Ответ: 1 или 3й вариант.

Обратите внимание, в 3м варианте значение будет сохранено вместе с кавычками.

РЕЗЮМИРУЕМ

Cookies:

- представляют собой пары строк ключ-значение, которые сохраняются даже после закрытия браузера (хранятся как файлы в папке с настройками браузера);
- доступны в рамках домена (поддомена), на котором установлены (можно настраивать);
- могут быть установлены с сервера (заголовок `Set-Cookie`);
- имеют настройку времени жизни (опция `Expires`);
- автоматически отправляются на сервер (в зависимости от опции `Path`);
- имеют существенные ограничения на количество/размер одной куки/общий размер.

СРАВНЕНИЕ `localStorage`, `sessionStorage`, `cookie`

`Cookies`

- **+** могут быть установлены с сервера (заголовок `Set-Cookie`);
- **+** могут быть недоступны на клиенте (опция `HttpOnly`);
- **+** имеют настройку времени жизни (опция `Expires`);
- **+/-** автоматически отправляются на сервер (в зависимости от опции `Path`);
- **-** существенные ограничения на количество/размер одной куки/общий размер;
- **-** неудобный интерфейс.

`localStorage`

- **+** ограничен только общий размер хранимых данных;
- **+** удобный интерфейс;
- **+/-** нет автоматической отправки на сервер;
- **+** значения не могут быть установлены с сервера.

`sessionStorage`

— **+** обладает всеми свойствами `localStorage`, кроме времени жизни (до завершения работы вкладки или окна браузера).

КРОССДОМЕННЫЕ ЗАПРОСЫ, CORS, `XHR.withCredentials`

МЕХАНИЗМ CORS ДЛЯ КРОССДОМЕННЫХ ЗАПРОСОВ

Представим, что нам необходимо запросить данные о залогированном пользователе для нашего веб-приложения по учету финансов (пусть он будет расположен по адресу www.financeManager.ru) с другого сайта (www.example.ru). Посмотрим на пример:

```
1  const invocation = new XMLHttpRequest();
2  const url = 'http://www.example.ru/public-data/';
3
4  function getData() {
5    if (invocation) {
6      invocation.open('GET', url, true);
7      invocation.onreadystatechange = handler;
8      invocation.send();
9    }
10 }
```

Но вместо данных в консоли мы видим ошибку:

> Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at www.example.ru

Почему так получилось?

В целях безопасности браузеры ограничивают кроссдоменные запросы, инициируемые скриптами. Например, `XMLHttpRequest` (и `Fetch API`) следуют политике одного источника (same-origin policy). Это значит, что веб-приложения, использующие такие API, могут запрашивать HTTP-ресурсы только с того домена, с которого были загружены, пока не будут использованы CORS-заголовки.

Таким образом, **Cross-Origin Resource Sharing (CORS)** — механизм, использующий дополнительные HTTP-заголовки, чтобы дать возможность веб-приложению, работающему на одном домене, получить доступ к выбранным ресурсам с сервера на другом источнике (домене).

Говорят, что агент пользователя (браузер) делает кроссдоменный запрос, если источник текущего документа отличается от запрашиваемого ресурса **доменом, протоколом или портом**.

С точки зрения безопасности разработчики стандарта `XMLHttpRequest` предусмотрели все возможные варианты, при помощи которых злоумышленники могли бы сломать какой-нибудь сервер, работающий по старому стандарту и не ожидающий новых видов запросов и заголовков (что мы и увидим далее).

«ПРОСТЫЕ» И «ПРЕДВАРИТЕЛЬНО ПРОВЕРЯЕМЫЕ» (`preflighted`) ЗАПРОСЫ

С точки зрения CORS существует два вида запросов: «простые» и «предварительно проверяемые».

Простыми считаются запросы, удовлетворяющие следующим условиям:

1. Методы: `GET`, `POST` или `HEAD`;
2. Заголовки из списка:
   - `Accept`;
   - `Accept-Language`;
   - `Content-Language`;
   - `Content-Type` со значением `application/x-www-form-urlencoded`, `multipart/form-data` или `text/plain`.

Предварительно проверяемыми считаются все остальные запросы.

Разница между ними заключается в том, что простой запрос можно сформировать и отправить на сервер и без `XMLHttpRequest`, например, при помощи HTML-формы. Поэтому сервера на старых технологиях (без поддержки `XMLHttpRequest`) уже должны были предусмотреть эту потенциальную угрозу.

Запросы с нестандартными заголовками или с методом `DELETE` при помощи отправки формы не создать, поэтому старый сервер может быть к ним не готов и может повести себя небезопасным способом. Для этого и появились предварительно проверяемые запросы, где до основного запроса идет запрос на проверку готовности сервера.

CORS ДЛЯ ПРОСТЫХ ЗАПРОСОВ

Вернемся к примеру выше. Предположим, с нашего сайта (www.financeManager.ru) мы хотим получить данные о пользователе с сайта www.example.ru. Запрос удовлетворяет условиям с прошлого слайда и поэтому является простым запросом.

```
1  const invocation = new XMLHttpRequest();
2  const url = 'http://www.example.ru/public-data/';
3
4  function getData() {
5    if (invocation) {
6      invocation.open('GET', url, true);
7      invocation.onreadystatechange = handler;
8      invocation.send();
9    }
10 }
```

Посмотрим на заголовки запроса браузера:

```
GET /resources/public-data/ HTTP/1.1
Host: www.example.ru
Referer: www.financeManager.ru
Origin: http://www.financeManager.ru
...
```

Самый важный заголовок, на который стоит обратить внимание, это `Origin`, показывающий домен, с которого осуществлён запрос.

Посмотрим теперь на заголовки ответа с сервера http://www.example.ru:

```
HTTP/1.1 200 OK
Date: Fri, 01 Feb 2019 00:23:53 GMT
Server: Apache/2.0.61
Access-Control-Allow-Origin: *
Content-Type: application/xml
...
```

Как мы видим, в ответ сервер отправляет заголовок `Access-Control-Allow-Origin,` в строке 4. В нашем случае сервер отвечает `Access-Control-Allow-Origin: *`, что означает, что к ресурсу может обращаться любой домен в межсайтовом режиме. Если владельцы ресурса http://www.example.ru хотят ограничить доступ к ресурсу только запросами с http://www.financeManager.ru, они отправят обратно:

```
Access-Control-Allow-Origin: http://www.financeManager.ru
```

Обратите внимание, что теперь ни один домен, кроме http://www.financeManager.ru (идентифицируемый заголовком `ORIGIN` в запросе, как в строке 4 выше), не может получить доступ к ресурсу с помощью кроссдоменного запроса. Заголовок `Access-Control-Allow-Origin` должен содержать значение, которое было отправлено в заголовке `Origin` запроса.

Если `Access-Control-Allow-Origin` нет, то браузер считает, что разрешение не получено, и завершает запрос с ошибкой.

ПРЕДВАРИТЕЛЬНО ПРОВЕРЯЕМЫЕ ЗАПРОСЫ

В кросс-доменном `XMLHttpRequest` можно указать не только `GET/POST`, но и любой другой метод, например `PUT`, `DELETE`.

Любое из условий ведёт к тому, что браузер сделает два HTTP-запроса:

- Если метод – не `GET` / `POST` / `HEAD`.
- Если заголовок `Content-Type` имеет значение отличное от `application/x-www-form-urlencoded`, `multipart/form-data` или `text/plain`, например `application/xml`;
- Если устанавливаются другие HTTP-заголовки, кроме `Accept`, `Accept-Language`, `Content-Language`.

Предварительный запрос использует метод `OPTIONS`. Он не содержит тела и содержит название желаемого метода (и при необходимости особые заголовки) в заголовке `Access-Control-Request-Method`.

На этот запрос сервер должен ответить статусом `200`, без тела ответа, указав заголовки:

- `Access-Control-Allow-Method`: метод;
- `Access-Control-Allow-Headers`: разрешённые заголовки (при необходимости).

Расмотрим пример предварительно проверяемого запроса, в котором мы в своем веб-приложении по учету финансов добавляем нового пользователя:

```
var invocation = new XMLHttpRequest();
var url = 'http://www.financeManager.ru/post-here/';
var body = '<?xml version="1.0"?><person><name>Василий</name></person>';

function addUser(){
  if (invocation)
    {
      invocation.open('POST', url, true);
      invocation.setRequestHeader('X-PINGOTHER', 'pingpong');
      invocation.setRequestHeader('Content-Type', 'application/xml');
      invocation.onreadystatechange = handler;
      invocation.send(body);
    }
}
```

Строка 3 создает тело с XML для отправки запросом POST в строке 8. В строке 9 устанавливается нестандартный заголовок HTTP-запроса (`X-PINGOTHER: pingpong`). Такие заголовки не являются частью протокола `HTTP / 1.1`, но они могут быть нужны для веб-приложений.

Поскольку в запросе используется тип контента `application / xml`, и поскольку установлен настраиваемый заголовок, этот запрос предварительно проверяется.

РАССМОТРИМ ЗАПРОС `OPTIONS`

```
OPTIONS /post-here/ HTTP/1.1
Access-Control-Request-Method: POST
Access-Control-Request-Headers: X-PINGOTHER, Content-Type
...
```

— Заголовок `Access-Control-Request-Method` как часть предварительного запроса уведомляет сервер о том, что при отправке основного запроса он будет отправлен методом POST;

— Заголовок `Access-Control-Request-Headers` - уведомляет сервер о том, что основной запрос будет отправлен с пользовательскими заголовками `X-PINGOTHER` и `Content-Type`.

Теперь у сервера есть возможность определить, может ли он принять такой запрос или нет.

РАССМОТРИМ ОТВЕТ НА ЗАПРОС OPTIONS

```
HTTP/1.1 200 OK
...
Access-Control-Allow-Origin: http://www.financeManager.ru
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers: X-PINGOTHER, Content-Type
```

- Сервер посылает `Access-Control-Allow-Methods` и говорит, что `POST` и `GET` являются допустимыми методами для запроса соответствующего ресурса.
- Сервер также отправляет `Access-Control-Allow-Headers` со значением `X-PINGOTHER`, `Content-Type`, подтверждая, что это разрешенные заголовки, которые будут использоваться с основным запросом.

После предварительного запроса основной запрос выполняется в обычном режиме.

ЧТЕНИЕ НЕСТАДАРТНЫХ ЗАГОЛОВКОВ ОТВЕТА

По умолчанию скрипт может прочитать из ответа только простые заголовки, такие как:

- `Cache-Control`;
- `Content-Language`;
- `Content-Type`;
- `Expires`;
- `Last-Modified`;
- `Pragma`.

Например, `Content-Type` можно получить всегда, а доступ к нестандартным заголовкам нужно открывать явно.

Чтобы JavaScript мог прочитать HTTP-заголовок ответа, сервер должен указать его имя в `Access-Control-Expose-Headers`.

Например:

```
Access-Control-Expose-Headers: X-My-Custom-Header, X-Another-Custom-Header
```

позволяет заголовкам `X-My-Custom-Header` и `X-Another-Custom-Header` быть прочитанными браузером.

ЗАПРОС С КУКИ И АВТОРИЗУЮЩИМИ ЗАГОЛОВКАМИ

Одна из возможностей `XMLHttpRequest` - это делать «проверенные» запросы, которые осведомлены о файлах `cookie` и информации об HTTP аутентификации.

По умолчанию в кроссдоменных вызовах `XMLHttpRequest` браузеры не отправляют учетные данные. Для этого должен быть установлен специальный флаг для объекта `XMLHttpRequest`.

В примере наше веб-приложение по учету финансов http://www.financeManager.ru выполняет простой GET-запрос к ресурсу , который устанавливает файлы cookie. Наше веб-приложение может содержать следующий JavaScript:

```
const invocation = new XMLHttpRequest();
const url = 'http://www.example.ru/credentialed-content/';

const getDataWithCredentials => () = {
  if (invocation) {
    invocation.open('GET', url, true);
    invocation.withCredentials = true;
    invocation.onreadystatechange = handler;
    invocation.send();
  }
}
```

В строке 7 проставляется флаг `withCredentials` для `XMLHttpRequest`, который нужен для того, чтобы сделать вызов с помощью Cookies. Это «простой» запрос `GET` и он не является предварительно проверяемым, но браузер отклонит любой ответ, который не имеет заголовка `Access-Control-Allow-Credentials: true`, и не сделает ответ доступным для вызывающего сайта.

Таким образом, при запросе с `withCredentials` сервер должен вернуть уже не один (как в случае с простым запросом), а два заголовка:

```
Access-Control-Allow-Origin: домен
Access-Control-Allow-Credentials: true
```

Пример заголовков ответа сервера для нашего примера:

```
HTTP/1.1 200 OK
Content-Type:text/html; charset=UTF-8
Access-Control-Allow-Origin: http://financeManager.ru
Access-Control-Allow-Credentials: true
```

Использование звёздочки `*` в `Access-Control-Allow-Origin` при этом запрещено.

Если этого заголовка не будет, то браузер не даст JavaScript доступ к ответу сервера.

Более подробно о CORS можно прочитать по ссылке в доп. материалах.

ЧТО НЕОБХОДИМО ДОБАВИТЬ?

Вопрос: какой заголовок должен возвращаться в ответе с сервера, чтобы были разрешены кроссдоменные запросы с сайта financeManager.ru?

1. `Access-Control-Allow-Origin: http://financeManager.ru`;
2. `Access-Control-Request-Method: POST`;
3. `Access-Control-Allow-Origin: *`;
4. `Access-Control-Allow-Headers: Content-Type`.

ЧТО НЕОБХОДИМО ДОБАВИТЬ?

Ответ: 1 или 3 (1 - только для указанного домена, 3 для всех).

РЕЗЮМИРУЕМ

— Cross-Origin Resource Sharing (CORS) — механизм, использующий дополнительные HTTP-заголовки, чтобы дать возможность веб-приложению, работающему на одном домене, получить доступ к выбранным ресурсам с сервера на другом домене;

— выделяют простые и предварительно проверяемые запросы (с предварительным запросом `OPTIONS`);

— если сервер отвечает `Access-Control-Allow-Origin: *`, что означает, что к ресурсу может обращаться любой домен в межсайтовом режиме. Также разрешение может быть толко у конкретного домена: `Access-Control-Allow-Origin: домен`;

— флаг `withCredentials` для `XMLHttpRequest` нужен для того, чтобы сделать вызов с помощью Cookies (используется при HTTP аутентификации).

КРАТКИЕ ИТОГИ ЛЕКЦИИ

Итак, сегодня мы узнали, что:

- `localStorage` — это специальный объект BOM, с помощью которого можно долговременно хранить информацию на клиенте. Для доступа к данным предоставляется удобный интерфейс для чтения и записи пар ключ-значение (строки).
- Куки — это пары строк ключ-значение, которые могут сохраняться даже после закрытия браузера, часто используются для авторизации пользователя.
- CORS — это механизм, использующий дополнительные HTTP-заголовки, чтобы дать возможность веб-приложению, работающему на одном домене, получить доступ к выбранным ресурсам с сервера на другом домене.

ДОПОЛНИТЕЛЬНЫЕ МАТЕРИАЛЫ

- https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS

МАТЕРИАЛЫ, ИСПОЛЬЗОВАННЫЕ ПРИ ПОДГОТОВКЕ

- https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
- https://developer.mozilla.org/ru/docs/Web/API/XMLHttpRequest/withCredentials
- https://developer.mozilla.org/en-US/docs/Web/API/Window/localStorage
- https://developer.mozilla.org/ru/docs/Web/API/Window/sessionStorage
- https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies

ДОМАШНЕЕ ЗАДАНИЕ

Давайте посмотрим ваше домашнее задание.

- Вопросы по домашней работе задаем в Slack!
- Работы должны соответствовать принятому стилю оформления кода.
- Зачет по домашней работе проставляется после того, как приняты все **3 задачи**.

НЕТОЛОГИЯ

Задавайте вопросы и напишите отзыв о лекции!

АЛЕКСАНДР РУСАКОВ


a_s_rusakov@mail.ru


@a_s_rusakov