

# СОВМЕСТНОЕ ИСПОЛЬЗОВАНИЕ РЕСУРСОВ МЕЖДУ РАЗНЫМИ ИСТОЧНИКАМИ



**ЕВГЕНИЙ КОРЫТОВ** / РУКОВОДИТЕЛЬ ВЕБ РАЗРАБОТКИ, KEENETIC



# ЕВГЕНИЙ КОРЫТОВ

Руководитель веб разработки, Keenetic



[korytov.pro](http://korytov.pro)



[korytoff@gmail.com](mailto:korytoff@gmail.com)



**CORS**

---

# SAME ORIGIN POLICY



*Same Origin Policy – «Принцип одинакового источника» – концепция безопасности разрешает скриптам, находящимся на страницах одного сайта, доступ к методам и свойствам друг друга без ограничений, но предотвращает доступ к большинству методов и свойств для страниц на разных сайтах. Одинаковые источники – это источники, у которых совпадают три признака: домен, порт, протокол.*

---

# CROSS-ORIGIN RESOURCE SHARING



*Cross-origin resource sharing (CORS) – «совместное использование ресурсов между разными источниками» – технология современных браузеров, которая позволяет предоставить веб-странице доступ к ресурсам другого домена.*

# НА ЧТО ОН РАСПРОСТАНЯЕТСЯ

Стандарт затрагивает межсайтовые запросы при:

- Отправке запросов через `XMLHttpRequest` или `fetch`.
- Манипуляции документами во фреймах и окнах.
- И некоторые другие манипуляции.

# РАЗРЕШЕНИЕ ДОЛЖЕН ДАТЬ СЕРВЕР

1. Открываем страницу по адресу `http://site.com/index.html`.
2. Наш скрипт запрашивает данные с сайта `service.com` через `XMLHttpRequest`.
3. Браузер фиксирует различие источников `site.com` и `service.com`.
4. Если браузер поддерживает Same Origin Policy, то не выполняет запрос сразу.
5. Выполняет предварительный запрос `OPTIONS` на этот адрес.
6. Если в заголовках ответа сервера `service.com` нет разрешения для сайта `site.com`, то запрос сваливается с ошибкой (событие `error`).
7. Если разрешение есть, то все выполняется как обычно.

# РАЗРЕШЕНИЕ ЭТО ЗАГОЛОВОК

В протоколе HTTP есть всего две сущности: заголовок и данные. И логично, что разрешение CORS — это просто заголовок.

Если сервер `service.com` готов предоставить доступ сайту `site.com`, он должен послать заголовок:

```
Access-Control-Allow-Origin: http://site.com
```



# РАЗРЕШЕНИЕ ДЛЯ НЕСКОЛЬКИХ САЙТОВ

Если сервер `service.com` готов предоставить доступ нескольким сайтам то можно послать несколько заголовков:

```
Access-Control-Allow-Origin: http://site1.com  
Access-Control-Allow-Origin: http://site2.com  
Access-Control-Allow-Origin: http://site3.com
```

Или один с перечислением через пробел:

```
Access-Control-Allow-Origin: http://site1.com http://site2.com http://site3.com
```

Чтобы предоставить доступ вообще всем источникам, можно использовать wildcard:

```
Access-Control-Allow-Origin: *
```

## ПРИМЕР ОТПРАВКИ ЗАГОЛОВКА НА PHP

Для того чтобы сервис реализованный на PHP смог предоставить CORS доступ:

```
<?php  
    header('Access-Control-Allow-Origin: http://site.com');
```

# РЕЖИМЫ CORS

Существуют два режима проверки:

- С предварительным запросом `OPTIONS` на сервер.
- Упрощенный, когда предварительный запрос `OPTIONS` не делается.

Под упрощенный режим попадают запросы:

- `GET` или `POST`
- Тип данных в случае `GET` : `application/x-www-form-urlencoded`, `multipart/form-data`, или `text/plain`
- В запросе не задано никаких кастомных заголовков `( X-* )`.

# ЗАГОЛОВКИ CORS

Кроме основного заголовка `Access-Control-Allow-Origin` сервер может передать еще следующие заголовки:

- `Access-Control-Allow-Methods` — указав какие типы запросов доступны.
- `Access-Control-Allow-Headers` — указав какие кастомные заголовки можно использовать.
- `Access-Control-Allow-Credentials` — указывает допустимость передачи кук и данных HTTP аутентификации.
- и некоторые другие.

Так как реализовать поддержку CORS нужно на back-end стороне, то мы не будем глубже изучать тему заголовков и особенностей.



**JSONP**

# CORS ПОЯВИЛСЯ ГОРАЗДО ПОЗЖЕ SOP

Как обходили ограничения Same Origin Policy до появления CORS?

Использовали тег `<script>`.

# SOP НЕ РАСПРОСТРАНЯЕТСЯ НА ТЕГ <script>

Мы можем подключать к своему HTML-документу любой скрипт с любого источника.

Но как правило асинхронные HTTP запросы мы используем для получения данных. Например в формате JSON.

Допустим мы хотим получить данные книги и отобразить на странице.

Они доступны в формате JSON по адресу:

<http://netology-fbb-store-api.herokuapp.com/book/fbb-0001>

## КАК ВЫГЛЯДЯТ ДАННЫЕ

```
1 {  
2   "author": "Филип Фрай",  
3   "price": 946,  
4   "title": "История земли XXX вв.",  
5   "id": "fbb-0001"  
6 }
```



# ПОДКЛЮЧИМ ИХ В НАШ ДОКУМЕНТ

Укажем путь к данным в теге `<script>`:

```
<script src="http://netology-fbb-store-api.herokuapp.com/book/fbb-0001"></script>  
<div class="book"></div>
```

## БРАУЗЕР НЕ ПОНЯЛ НАШУ ЗАДУМКУ

Данные были получены. Ограничение SOP мы обошли. Но к задаче пока не приблизились:

**Uncaught SyntaxError: Unexpected token:**

Браузер просто посчитал наш объект блоком кода. Но даже если бы он интерпретировал эти данные как объект, как бы мы к нему потом обратились?

Тут нужен немного иной подход. И он есть: JSONP.

# JSON С ДОБИВКОЙ



*«JSONP или «JSON with padding» (JSON с добивкой) — это дополнение к базовому формату JSON. Он предоставляет способ запросить данные с сервера, находящегося в другом домене — операцию, запрещённую в типичных веб-браузерах из-за политики ограничения домена.*

## ДОБЬЕМ НАШИ ДАННЫЕ

Может быть стоит немного подождать и продолжить, когда теги появятся?

Попробуем:

```
1  parseBook( {  
2    "author": "Филип Фрай",  
3    "price": 946,  
4    "title": "История земли XXX вв.",  
5    "id": "fbb-0001"  
6  } );
```

## БЛИЗКО, НО ПОКА НЕ ГОТОВО>

Опять ошибка, но уже другая:

```
Uncaught ReferenceError: parseBook is not defined
```

И я думаю что вы уже знаете с чем она связана и как её решить.

# СОЗДАДИМ ФУНКЦИЮ

В файле `app.js` реализуем функцию `parseBook`:

```
1 function parseBook(book) {  
2   const target = document.querySelector('.book');  
3   target.innerHTML = `Книга ${book.title}, автор ${book.author.name}`;  
4 }
```

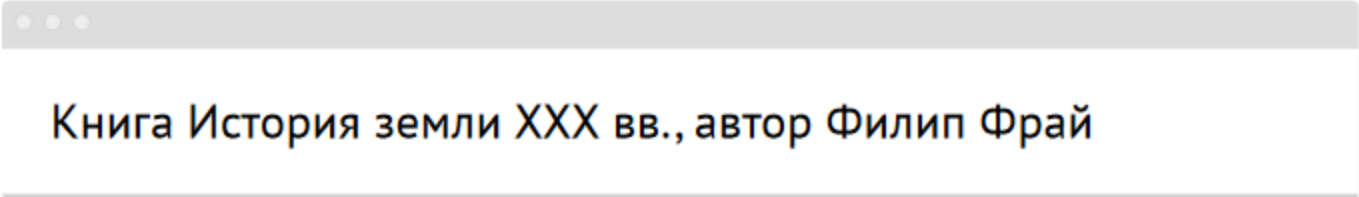
# ВСЕ В СБОРЕ

```
<script src="./app.js"></script>
```

```
<script src="http://netology-fbb-store-api.herokuapp.com/book/fbb-0001"></script>
```

```
<div class="book"></div>
```

# МЫ ОБОШЛИ CORS



Книга История земли XXX вв., автор Филип Фрай

Но это пока только промежуточный итог. Потому что пока все выглядит весьма статично. И до гибкости `XMLHttpRequest` а уж тем более легкости `fetch` очень далеко.





## JSONP - ПРОМЕЖУТОЧНЫЙ ИТОГ

- Вмешательство на сервере нам всё же потребуется.
- Используем только `GET`-запрос.
- Это просто определенный прием, а не интерфейс API и не технология.

## НОВАЯ ЦЕЛЬ

Давайте реализуем работу с JSONP в стиле `fetch`:

1. Создадим функцию `loadData`, которая будет принимать адрес где размещены данные.
2. Функция пусть вернет нам промис, который разрешится данными.

# ЗАГОТОВКА

```
1 function loadData(url) {  
2   return new Promise((done, fail) => {  
3  
4     });  
5 }
```

## ПРИМЕР ИСПОЛЬЗОВАНИЯ

Реализуем функцию показа книги и вызовем её с данными промиса когда они будут получены:

```
1 function showBook(book) {  
2   const target = document.querySelector('.book');  
3   target.innerHTML = `Книга ${book.title}, автор ${book.author.name}`;  
4 }  
5  
6 loadData('http://netology-fbb-store-api.herokuapp.com/book/fbb-0001')  
7 .then(showBook);
```

## ВОПРОСЫ РЕАЛИЗАЦИИ

- Для загрузки данных мы использовали тег `<script>`. Можно ли его создать программно?
- После того как данные были загружены, вызывалась функция в которую они были обернуты. Какая это будет функция?

## СОЗДАНИЕ ТЕГА СКРИПТ

Тэг скрипт такой же тэг как и остальные. Значит его можно создать при помощи `innerHTML` или при помощи `cloneNode`:

```
const script = document.scripts[0].cloneNode();  
script.src = url;  
document.body.appendChild(script);
```

## ФУНКЦИЯ-ДОБИВКА JSONP

```
window.parseBook = done;
```

## КОД В СБОРЕ

```
1 function loadData(url) {  
2   return new Promise((done, fail) => {  
3     window.parseBook = done;  
4  
5     const script = document.scripts[0].cloneNode();  
6     script.src = url;  
7     document.body.appendChild(script);  
8   });  
9 }
```



## ОСТАЛАСЬ ОДНА ПРОБЛЕМА

Если параллельно вызывать `loadData` несколько раз, то разрешится только последний промис.

Решение: создавать каждый раз новую функцию и использовать её.

Для этого её название нужно передавать на сервер через `GET` параметр `callback` или `jsonp`.

## ПРИМЕР АДРЕСА И ОТВЕТА

Пример адреса:

```
http://netology-fbb-store-api.herokuapp.com/book/fbb-0001?jsonp=callback1234
```

Пример ответа:

```
1 callback1234( {  
2   "author": "Филип Фрай",  
3   "price": 946,  
4   "title": "История земли XXX вв.",  
5   "id": "fbb-0001"  
6   });
```

## ОКОНЧАТЕЛЬНАЯ РЕАЛИЗАЦИЯ

```
1 function loadData(url) {  
2   const functionName = randName();  
3   return new Promise((done, fail) => {  
4     window[functionName] = done;  
5  
6     const script = document.scripts[0].cloneNode();  
7     script.src = `${url}?jsonp=${functionName}`;  
8     document.body.appendChild(script);  
9   });  
10 }
```



**НЕТОЛОГИЯ**  
университет интернет-профессий

**Задавайте вопросы и напишите отзыв о лекции!**

**ЕВГЕНИЙ КОРЫТОВ**



[korytov.pro](http://korytov.pro)



[korytoff@gmail.com](mailto:korytoff@gmail.com)