Лекция 4

Углубленный Web

Повторение - мать учения

Подробнее тут https://vc.ru/selectel/76371-chto-proishodit-kogda-polzovatel-nabiraet-v-brauzere-adres-sayta

Базовый сценарий работы web-приложения

- Пользователь вводит URL
- Браузер загружает страницу - HTML документ
- Браузер анализирует (парсит) HTML и загружает доп. ресурсы
- Браузер отображает (рендерит) HTML-страницу

URL - unified resource locator

`http://mi-ami.ru:8080/profile/account.html?gender=male&age=13#comments`

- **http** - протокол
- **mi-ami.ru** - доменное имя (DNS имя сервера)
- **8080** - TCP порт
- **/profile/account.html** - путь до документа
- **?gender=male&age=13** - query-параметры (параметры запроса)
- **#comments** - якорь

Документы

Документ - это тело ответа HTTP-запроса. Он может иметь несколько типов (MIME-типы):

- text/html
- text/css
- text/javascript
- image/png
- video/mp4
- и так далее…

Документы

По смыслу документы можно разделить на статические и динамические.

Статические:

- Файлы на дисках сервера, зачастую с постоянным адресом

Динамические:

- Создаются на каждый запрос
- Содержимое зависит от внешних факторов (пользователя, времени и тд)
- Адрес может меняться (может быть и постоянным)

Ресурсы

```
1 <link rel="stylesheet" href="/css/index.css">
2 <script src="http://code.jquery.com/jquery-2.1.4.js">
3 </script>
4 <img src="pictures/network.png" width="200" >
```

Ресурсы

```
1 .slide {
2  background-image: url(../pictures/network.png)
3 }
4
5 @font-face {
6  font-family: Terminus;
7  src: url(fonts/terminus.ttf);
8 }
```

Протоколы

Существует большое множество различных сетевых протоколов связи.
Самые распространенные:

- TCP
- UPD
- **HTTP (работает поверх TCP)**
- FTP
- SSH

HTTP - HyperText Transfer Protocol

Основой HTTP является технология «клиент-сервер»: всегда есть клиент, который посылает запрос, и сервер который получает запрос и отдает нужный ответ.

Изначально использовался для передачи исключительно HTML, но вскоре был расширен при помощи MIME-типов.

Отсюда делаем вывод, что каждый запрос браузера за ресурсами - это HTTP-запрос.

Структура HTTP-запроса

Каждое HTTP-сообщение состоит:

- метод (GET, POST, PUT, DELETE и тд);
- URL запроса (адрес ресурса);
- заголовки - характеризуют тело сообщения, параметры передачи и прочие сведения;
- тело - может отсутствовать.

Структура HTTP-ответа

Ответ как правило состоит также из тела и заголовков, а также из статуса ответа. Различают 5 видов статусов:

- **1xx** - информативный статус
- **2xx** - успешный статус
- **3xx** - перенаправление
- **4xx** - клиентская ошибка
- **5xx** - ошибка сервера

Пример HTTP-запроса

```
1 GET http://www.ru/robots.txt HTTP/1.0
2 Accept: text/html, text/plain
3 User-Agent: curl/7.64.1
4 If-Modified-Since: Fri, 24 Jul 2015 22:53:05
  GMT
```

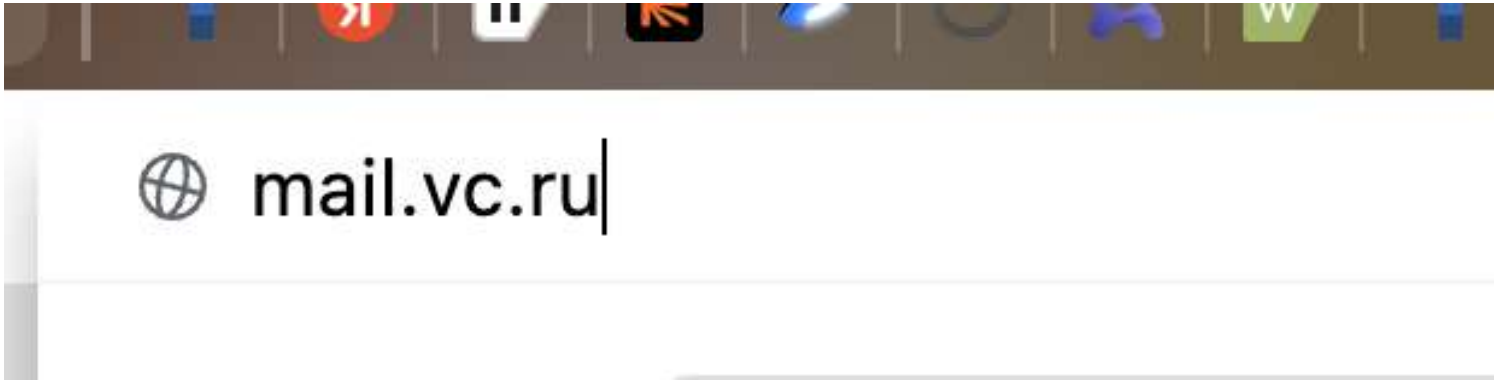
Пример HTTP-ответа

```
1 HTTP/1.1 404 Not Found
2 Server: nginx/1.5.7
3 Date: Sat, 25 Jul 2015 09:58:17 GMT
4 Content-Type: text/html; charset=iso-8859-1
5 Connection: close
6
7 <!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
8 <HTML><HEAD>...
```

“Пользователь вводит ссылку в браузер…”

Подробнее тут https://vc.ru/selectel/76371-chto-proishodit-kogda-polzovatel-nabiraet-v-brauzere-adres-sayta

Ну, вводи уже



Поиск сервера

Работу сайта обеспечивает сервер. У сервера есть адрес - **IP-адрес.**

Откуда браузеру взять IP-адрес сервера?

Такая информация хранится в распределенной системе серверов — **DNS** (**Domain Name System**). Система работает как общая «контактная книга», хранящаяся на распределенных серверах и устройствах в интернете.

Оптимизация поиска

На самом деле браузер не сразу обращается к DNS-серверу. Сначала браузер пытается найти запись об IP-адресе сайта в ближайших местах, чтобы сэкономить время:

1. История подключений (если пользователь уже посещал сайт);
2. В операционной системе (какие-то приложения обращались к сайту);
3. Кэш роутера;
4. Кэш провайдера;
5. Бесконечность не предел, вплоть до столкновения с DNS-сервером.

DNS-сервер


.
.org
.ru
.com
.io
apple.com
vc.ru
vscale.io
ya.ru
mail.vc.ru
store.apple.com
taxi.ya.ru

Устанавливается соединение

Как только браузер узнал IP-адрес нужного сервера, он пытается установить с ним соединение. В большинстве случаев для этого используется специальный протокол — TCP.

Для установления TCP соединения используется система трех рукопожатий.

Тройное рукопожатие

- Устройство пользователя отправляет специальный запрос на установку соединения с сервером — называется SYN-пакет.
- Сервер в ответ отправляет запрос с подтверждением получения SYN-пакета — называется SYN/ACK-пакет.
- В конце устройство пользователя при получении SYN/ACK-пакета отправляет пакет с подтверждением — ACK-пакет. В этот момент соединение считается установленным.

Остальные шаги

1. Браузер отправляет HTTP-запрос по установленному соединению;
2. Сервер получает запрос и обрабатывает его (бизнес-логика, рендер, проксирование и тд);
3. Сервер отправляет ответ браузера. Это может быть любой (!) ресурс;
4. Браузер обрабатывает полученный ответ и «рисует» веб-страницу;

Как происходит рендер (кратко)

Сначала браузер загружает **только основную структуру HTML-страницы**. Затем последовательно проверяет все теги и отправляет дополнительные GET-запросы для получения с сервера различных элементов — картинки, файлы, скрипты, таблицы стилей и так далее. Поэтому по мере загрузки страницы браузер и сервер продолжают обмениваться между собой информацией.

Как только рендеринг завершен — пользователю отобразится полностью загруженная страница сайта.

Критические этапы рендера

Подробнее тут

https://developer.mozilla.org/ru/docs/Web/Performance/Critical_rendering_path

Критические этапы рендера

Критические этапы рендеринга (Critical Rendering Path) - это последовательность шагов, которые выполняет браузер, когда преобразуется HTML, CSS и JavaScript в пиксели, которые вы видите на экране.

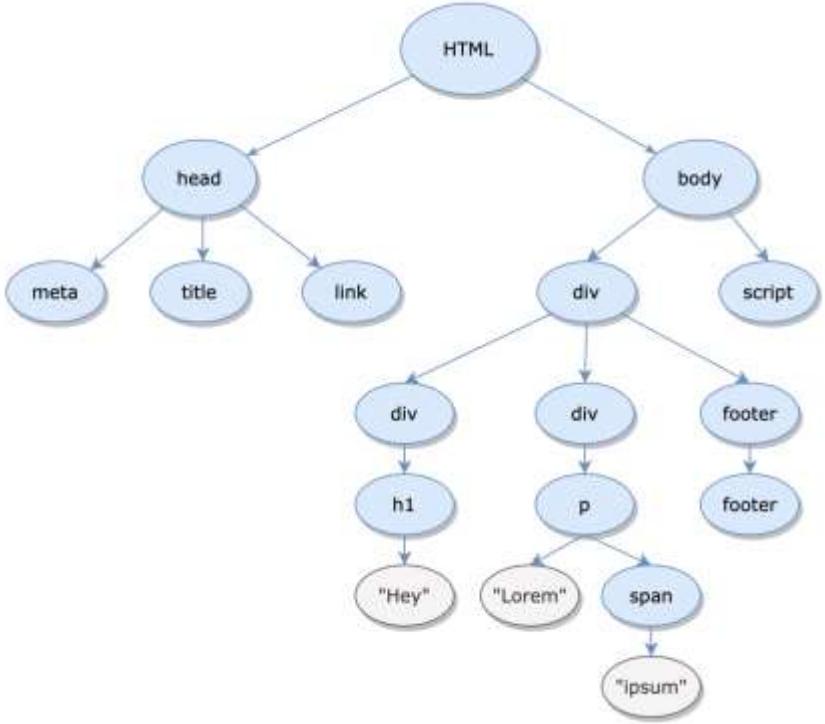
1. Загрузка HTML;
2. DOM;
3. CSSOM;
4. Дерево рендера (render tree);
5. Компоновка (layout);
6. Отрисовка (paint).

Критические этапы рендера (базово)

Браузер парсит загружаемый HTML, преобразуя полученные байты документа в DOM-дерево. Браузер создает новый запрос каждый раз, когда он находит ссылки на внешние ресурсы, будь то файлы стилей, скриптов или ссылки на изображения.
Браузер продолжает парсить HTML и создавать DOM до тех пор, пока запрос на получение HTML не подходит к концу.
После завершения парсинга DOM, браузер конструирует CSS модель.
Как только эти модели сформированы, браузер строит дерево рендера (render tree), в котором вычисляет стили для каждого видимого элемента страницы.
После формирования дерева происходит компоновка (layout), которая определяет положение и размеры элементов этого дерева.
Как только этап завершён - страница рендерится. Или "отрисовывается" (paint) на экране.

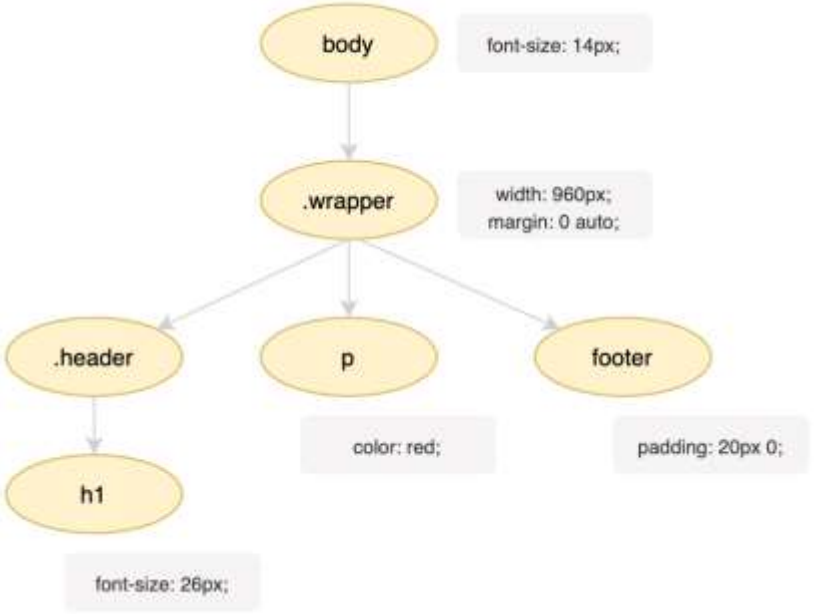
Document Object Model

Построение DOM инкрементально. Ответ в виде HTML превращается в токены, которые превращаются в узлы (nodes), которые формируют DOM дерево. Простейший узел начинается с startTag-токена и заканчивается токеном endTag. Узлы содержат всю необходимую информацию об HTML-элементе, соответствующем этому узлу. Узлы (nodes) связаны с Render Tree с помощью иерархии токенов: если какой-то набор startTag и endTag-токенов появляется между уже существующим набором токенов, мы получаем узел (node) внутри узла (node), то есть получаем иерархию дерева DOM.



CSS Object Model

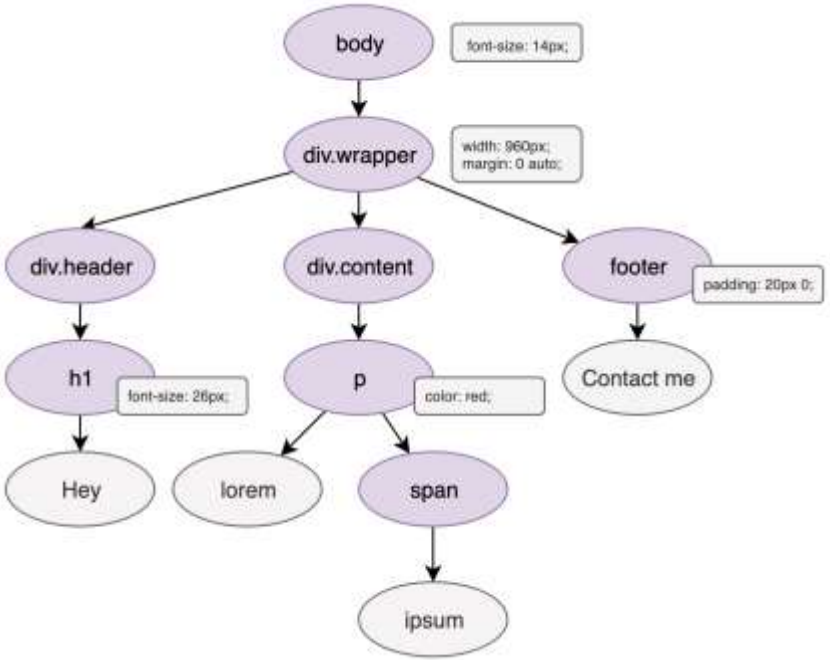
DOM несёт в себе всё содержимое страницы. CSSOM содержит все стили страницы, то есть данные о том, как стилизовать DOM. CSSOM похож на DOM, но всё же отличается. Если формирование DOM инкрементально, CSSOM - нет. CSS блокирует рендер: браузер блокирует рендеринг страницы до тех пор, пока не получит и не обработает все CSS-правила. CSS блокирует рендеринг, потому что правила могут быть перезаписаны, а значит, необходимо дождаться построения CSSOM, чтобы убедиться в отсутствии дополнительных переопределений.



Дерево рендера (Render Tree)

Дерево рендера охватывает сразу и содержимое страницы, и стили: это место, где DOM и CSSOM деревья комбинируются в одно дерево. Для построения дерева рендера браузер проверяет каждый узел (node) DOM, начиная от корневого (root) и определяет, какие CSS-правила нужно присоединить к этому узлу.

Дерево рендера охватывает только видимое содержимое. Например, секция head (в основном) не содержит никакой видимой информации, а потому может не включаться в дерево. Кроме того, если у какого-то узла стоит свойство display: none, оно так же не включается в дерево (как и потомки этого узла).



Компоновка (layout)

В тот момент, когда дерево рендера (render tree) построено, становится возможным этап компоновки (layout).
Layout — это рекурсивный процесс определения положения и размеров элементов из Render Tree. Он начинается от корневого Render Object, которым является, и проходит рекурсивно вниз по части или всей иерархии дерева высчитывая геометрические размеры дочерних render object'ов. Корневой элемент имеет позицию (0,0) и его размеры равны размерам видимой части окна, то есть размеру viewport'а.

К концу процесса layout каждый render object имеет свое положение и размеры.

Подводя промежуточный итог: **браузер знает что, как и где рисовать**.
Следовательно — осталось только нарисовать.

Отрисовка (paint)

Последний этап в нашем списке - **отрисовка (paint)** пикселей на экране. Когда дерево рендера (render tree) создано, компоновка (layout) произошла, пиксели могут быть отрисованы. При первичной загрузке документа (onload) весь экран будет отрисован. После этого будут перерисовываться только необходимые к обновлению части экрана, так как браузер старается оптимизировать процесс отрисовки, избегая ненужной работы. Так, если у вас в документе есть два элемента, перерисовываться будет только тот, который вы изменили.

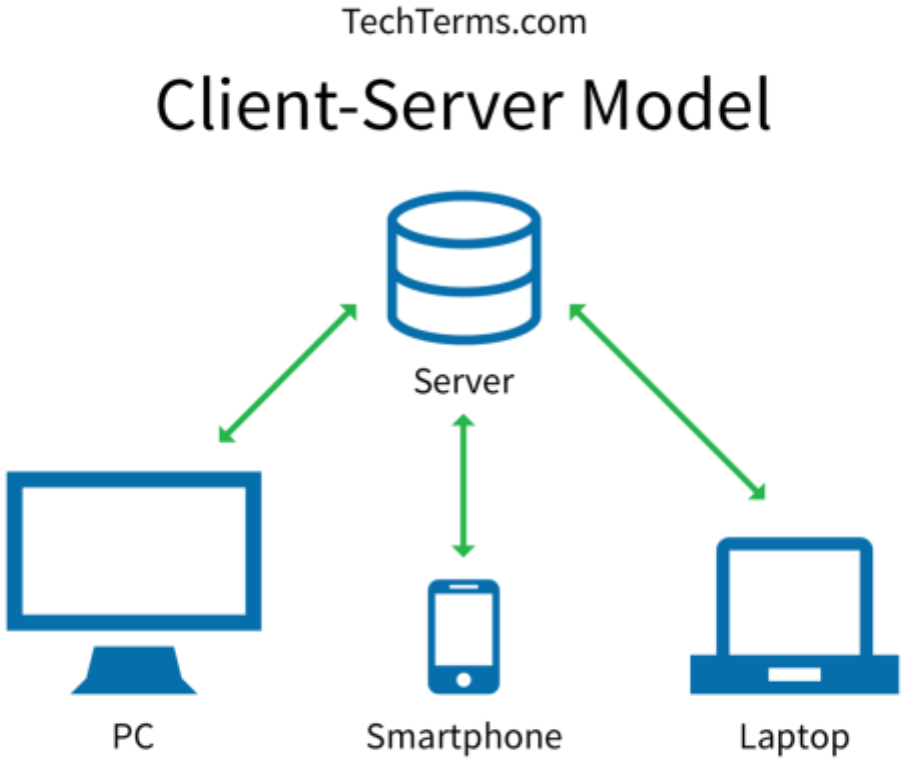
Клиент-серверная модель

Подробнее тут: https://habr.com/ru/post/495698/

Клиент-сервер

Клиент – серверная модель — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами

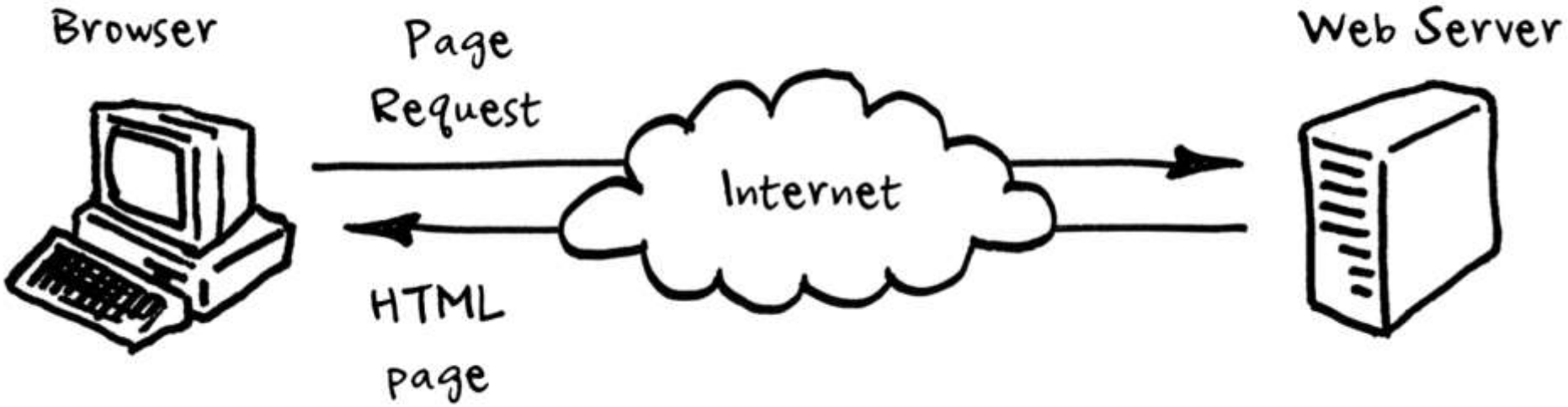
Клиент-сервер



История веб-приложения

Подробнее тут: нигде

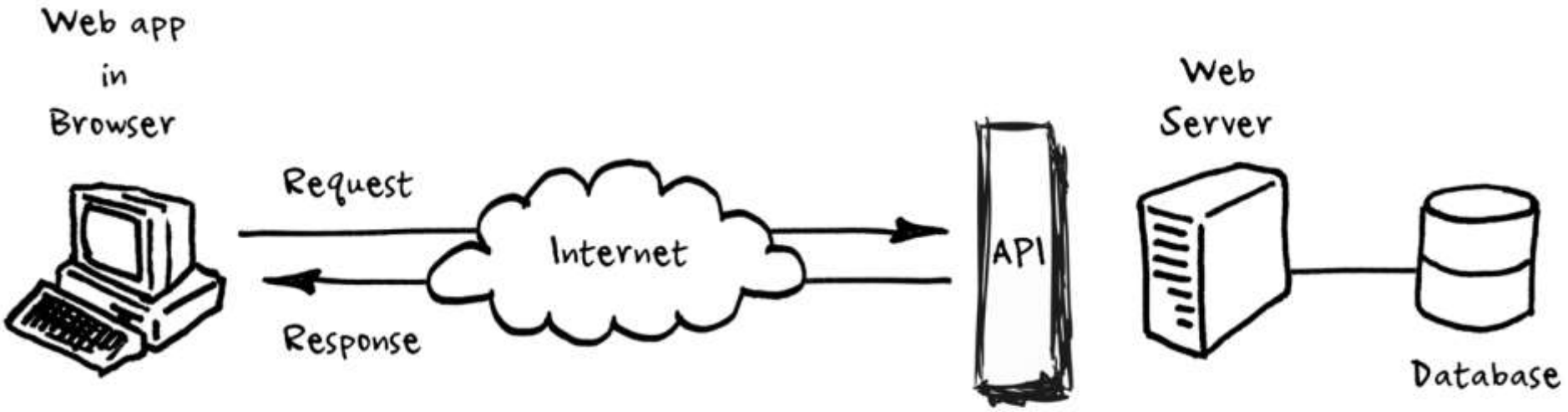
Статические веб-страницы



Статические веб-страницы

- Frontend
  a. Отображение статических HTML-страниц (HTML, CSS);
  b. Контент на страницах, переходы по гиперссылкам
- Backend
  a. Хранение статических документов и отдача по запросу по протоколу HTTP

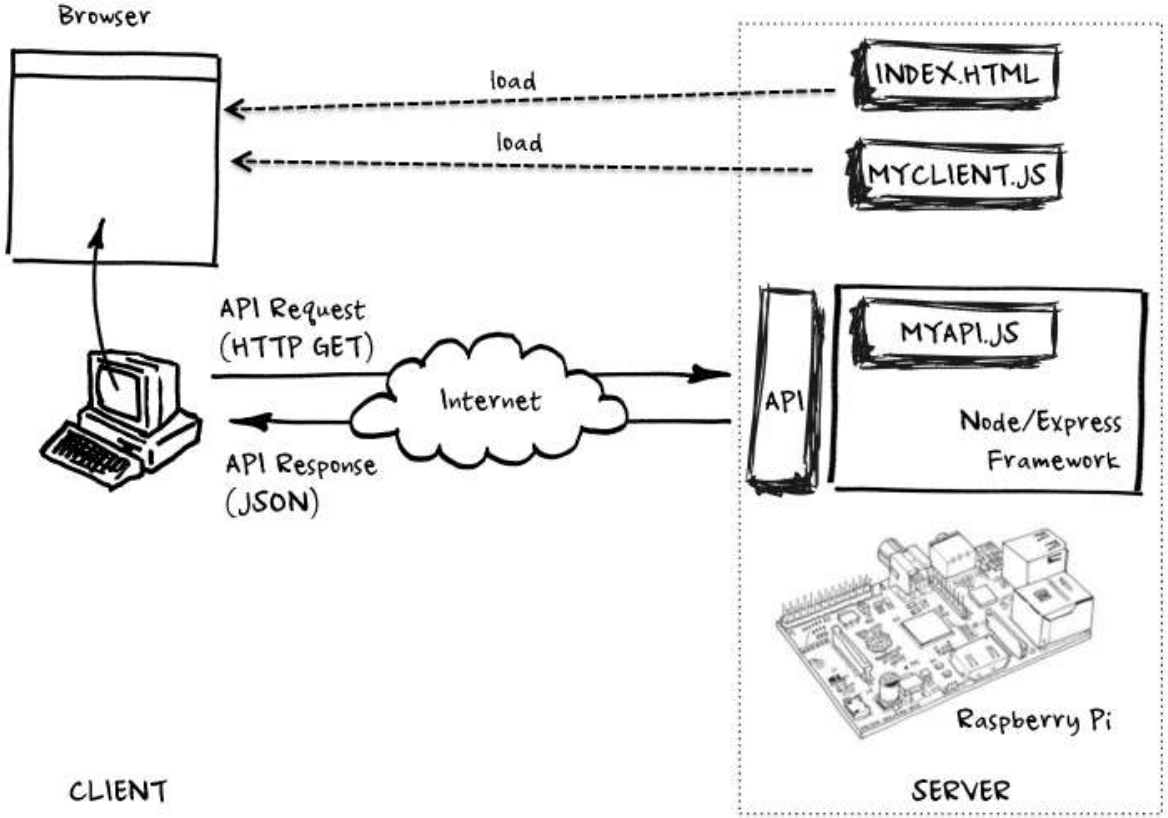
Динамические веб-сервисы



Динамические веб-сервисы

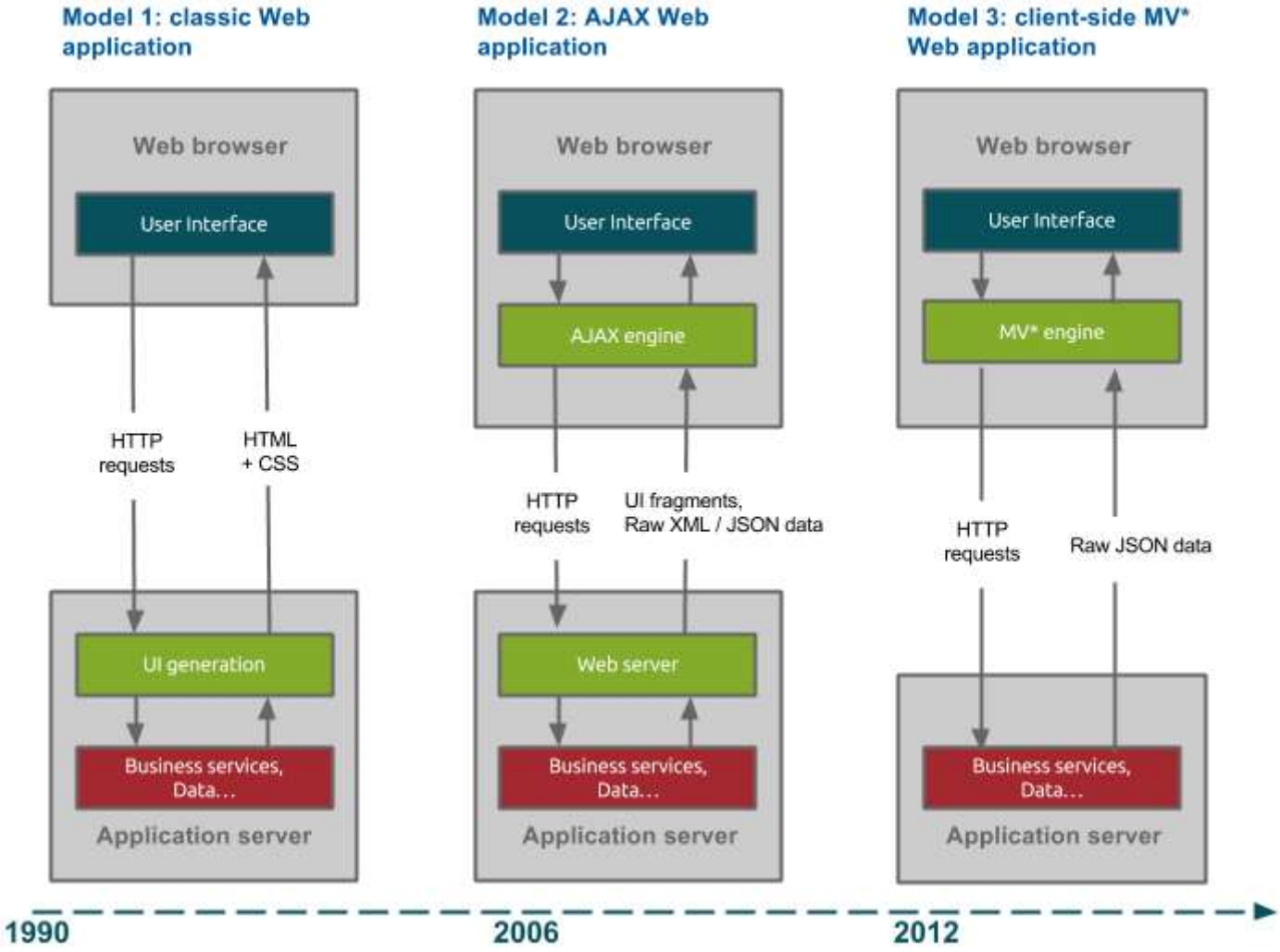
- Frontend
  a. Отображение статических HTML-страниц (HTML, CSS)
  b. Контент на страницах, переходы по гиперссылкам
  c. Взаимодействие с сервисом посредством форм
- Backend
  a. Хранение статических документов и отдача по запросу по протоколу HTTP
  b. Обработка пользовательских запросов и генерация динамических страниц
  c. Хранение данных в базе данных

Веб-приложения



Динамические веб-сервисы

- Frontend
  a. Хранение и доступ к статическому контенту (файлы стилей, скрипты)
  b. Генерация и отображение пользовательского интерфейса
  c. Взаимодействие с пользователем и выполнение запросов к API
  d. Обновление пользовательского интерфейса в ответ на действия пользователя
- Backend
  a. Реализация публичного API
  b. Хранение данных в базе данных и работа с ними


Model 1: classic Web application
Web browser
User Interface
HTTP requests
HTML + CSS
UI generation
Business services, Data…
Application server
Model 2: AJAX Web application
Web browser
User Interface
AJAX engine
HTTP requests
UI fragments, Raw XML / JSON data
Web server
Business services, Data…
Application server
Model 3: client-side MV* Web application
Web browser
User Interface
MV* engine
HTTP requests
Raw JSON data
Business services, Data…
Application server
1990
2006
2012

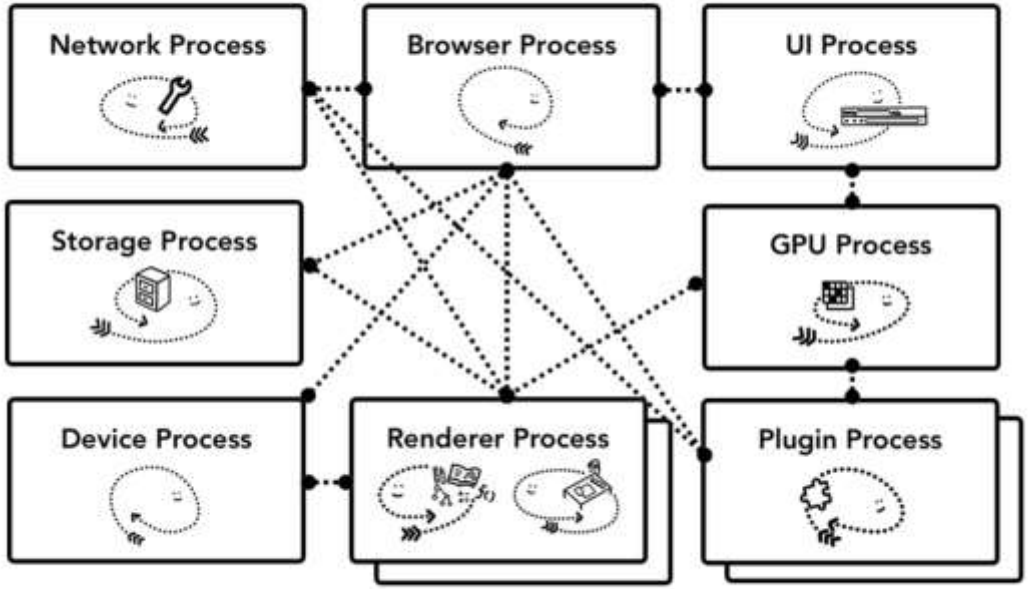
Node && Npm

Подробнее тут https://nodejs.dev/

Что находится внутри браузера?

Что находится внутри браузера?

Что находится внутри браузера?



Ограничения JS в браузере

- Нельзя взаимодействовать с файловой системой
- Нет доступа к сетевым функциям, кроме того, что предоставляет сам браузер
- Нет возможности организовывать многопоточные вычисления. Есть воркеры, но они имеют определенные ограничения
- Нельзя создавать новые процессы / запускать программы (открытие новых вкладок не считается)

Node.js

Node.js - исполнение JavaScript кода НЕ в браузере.

Программная платформа, основанная на движке V8, превращающая JavaScript из узкоспециализированного языка в язык общего назначения. Node.js добавляет возможность JavaScript взаимодействовать с устройствами ввода-вывода через свой API, написанный на C++, подключать другие внешние библиотеки, написанные на разных языках, обеспечивая вызовы к ним из JavaScript-кода.

Npm

Npm - Node Package Manager

Менеджер пакетов, входящий в состав Node.js.

NPM состоит из двух основных частей:

- инструмент CLI (command-line interface – интерфейс командной строки) для публикации и загрузки пакетов
- онлайн-репозиторий, в котором размещаются пакеты JavaScript.

Как подключить библиотеки

- Вариант с CDN

```
<script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.11.6/dist/umd/popper.min.js"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha2/dist/js/bootstrap.min.js"></script>
```

- Вариант с Npm

```
npm install bootstrap@5.3.0-alpha2
```

Npm (как работает)

- Скачиваем нужные нам зависимости

```
1 npm install bootstrap
```

- У нас появляется директория node_modules со всеми зависимостями
- Подключаем нужный нам пакет

```
1 import { ... } from 'bootstrap'
```


СОЛНЦЕ
НЕЙТРОННАЯ ЗВЕЗДА
ЧЕРНАЯ ДЫРА
ТВОЯ ПАПКА NODE_MODULES
САМЫЕ ТЯЖЕЛЫЕ ОБЪЕКТЫ ВО ВСЕЛЕННОЙ

Npm (создаем свой модуль)

`npm init` - инициализируем свое приложение

- `package.json` - файл с описанием приложения
- `package-lock.json` - файл с версиями зависимостей

Npm (создаем свой модуль)

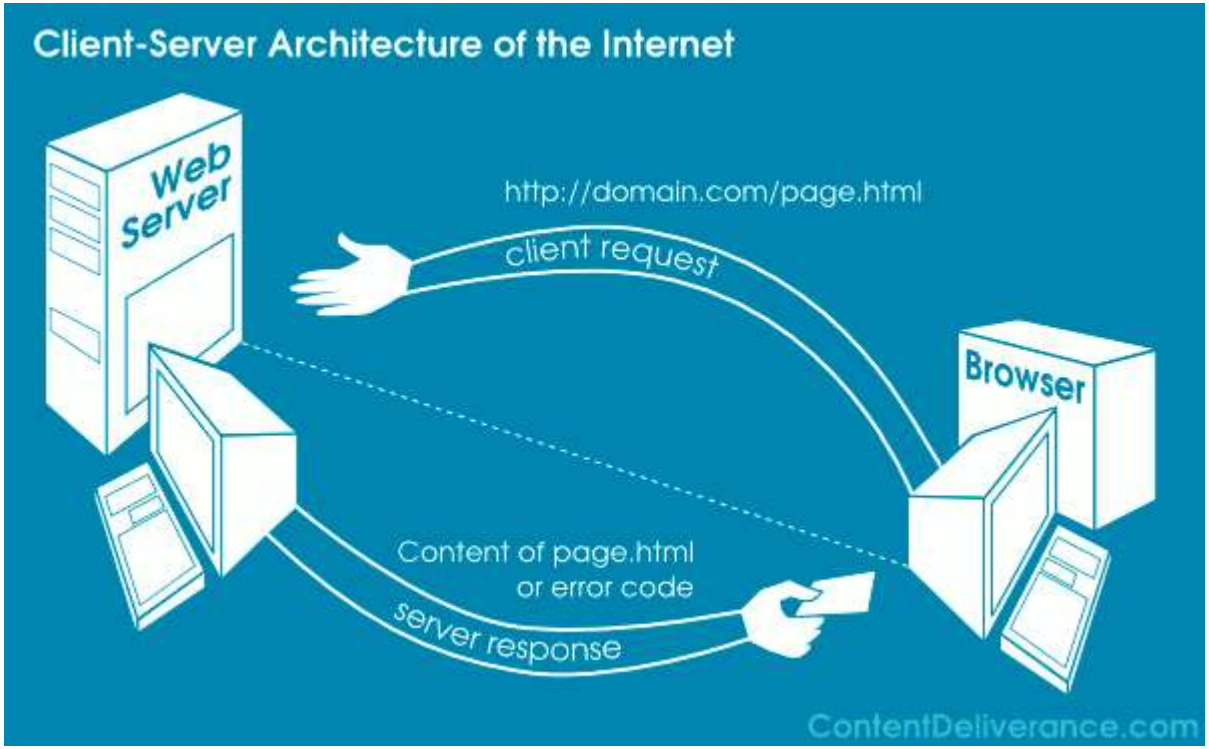
```
{
  "name": "awesome",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bootstrap": "^5.2.3"
  },
  "devDependencies": {
    "typescript": "^4.9.5"
  }
}

```

Работа с сетью

Подробнее тут https://developer.mozilla.org/ru/docs/Web/Guide/AJAX

Протокол HTTP



Ajax

Ajax означает Асинхронный JavaScript и XML.

В основе технологии лежит использование XMLHttpRequest, который позволяет нам отправлять и получать информацию в различных форматах включая XML, HTML и даже текстовые файлы. Самое привлекательное в Ajax — это его асинхронный принцип работы. С помощью этой технологии можно осуществлять взаимодействие с сервером без необходимости перезагрузки страницы. Это позволяет обновлять содержимое страницы частично, в зависимости от действий пользователя.

Что передаем по сети

- Текстовые данные
- JSON
- Бинарные данные
- Файлы

JSON

JSON - это общий формат для представления значений и объектов. Первоначально он был создан для JavaScript, но многие другие языки также имеют библиотеки, которые могут работать с ним.

```json
{
  "name": "John",
  "age": 30,
  "isAdmin": false,
  "courses": ["html", "css", "js"],
  "wife": null
}
```

Blob

Blob - это объект в JavaScript, который позволяет работать с бинарными данными.

```
1 const binary = new Uint8Array(1024 * 1024); // 1 MB данных
2 const blob = new Blob([binary], {type: 'application/octet-stream'});
```

FormData

FormData - объект JavaScript, который позволяет работать с формами и файлами.

```
1 const fileInput = document.querySelector('input[type=file]');
2 const file = fileInput.files[0].file;
3 const formdata = new FormData();
4 formdata.append('file', file);
```

XMLHttpRequest

`XMLHttpRequest (XHR)` – это встроенный в браузер объект, который даёт возможность делать HTTP-запросы к серверу без перезагрузки страницы.

XHR поддерживает 2 режима работы:

- Синхронный
- Асинхронный

XMLHttpRequest

- Создаем XHR
- Инициализируем его
- Посылаем запрос
- Слушаем ответ

1. XMLHttpRequest (создаем)

```
1 let xhr = new XMLHttpRequest();
```

2. XMLHttpRequest (инициализируем)

```
1 xhr.open('GET', '/article/xmlhttprequest/example/load');
```

3. XMLHttpRequest (посылаем запрос)

```
1 xhr.send();
```

4. XMLHttpRequest (слушаем ответ)

```
1 xhr.onload = function() {
2   alert(`Загружено: ${xhr.status} ${xhr.response}`);
3 };
4
5 xhr.onerror = function() {
6   alert(`Ошибка соединения`);
7 };
8
9 xhr.onprogress = function(event) {
10   alert(`Загружено ${event.loaded} из ${event.total}`);
11 };
```

XMLHttpRequest (слушаем ответ)

После получения ответа мы можем достать его из xhr

- `status` - HTTP статус ответа
- `statusText` - текстовых HTTP статус ответа
- `response` - тело ответа

XMLHttpRequest (асинхронный запрос)

```
// 1. Создаём новый XMLHttpRequest-объект
let xhr = new XMLHttpRequest();

// 2. Настраиваем его: GET-запрос по URL /article/.../load
xhr.open('GET', '/article/xmlhttprequest/example/load');

// 3. Отсылаем запрос
xhr.send();

// 4. Этот код сработает после того, как мы получим ответ сервера
xhr.onload = function() {
  if (xhr.status != 200) { // анализируем HTTP-статус ответа, если статус не 200, то произошла ошибка
    alert(`Ошибка ${xhr.status}: ${xhr.statusText}`); // Например, 404: Not Found
  } else { // если всё прошло гладко, выводим результат
    alert(`Готово, получили ${xhr.response.length} байт`); // response -- это ответ сервера
  }
};

xhr.onerror = function() {
  alert("Запрос не удался");
};
```

XMLHttpRequest (синхронный запрос)

```
let xhr = new XMLHttpRequest();

xhr.open('GET', '/article/xmlhttprequest/hello.txt', false);

try {
  xhr.send();
  if (xhr.status != 200) {
    alert(`Ошибка ${xhr.status}: ${xhr.statusText}`);
  } else {
    alert(xhr.response);
  }
} catch(err) { // для отлова ошибок используем конструкцию try...catch вместо onerror
  alert("Запрос не удался");
}
```

XMLHttpRequest (тип ответа)

- text - строка
- arrayBuffer - ArrayBuffer бинарные данные
- blob - Blob бинарные данные
- document - XML-документ
- json - JSON (автоматически парситься)

```
1 xhr.responseType = 'json';
```

XMLHttpRequest (состояние запроса)

- `UNSET = 0` - исходное состояние
- `OPENED = 1` - вызван метод open
- `HEADERS_RECEIVED = 2` - получены заголовки отета
- `LOADING = 3` - ответ в процессе передачи
- `DONE = 4` - запрос завершен

```
xhr.onreadystatechange = function() {
  if (xhr.readyState == 3) {
    // загрузка
  }
  if (xhr.readyState == 4) {
    // запрос завершён
  }
};

```

XMLHttpRequest (отмена запроса)

```
1 xhr.abort();
```

XMLHttpRequest (HTTP-заголовки)

```
1 // Устанавливаем заголовок для передачи в запрос
2 xhr.setRequestHeader('Content-Type',
3 'application/json');
4 // Получаем заголовки из запроса
5 xhr.getResponseHeader('Content-Type')
6
7 // Получаем все заголовки из запроса
8 xhr.getAllResponseHeaders()
```

XMLHttpRequest (POST-запрос)

```
<form name="person">
  <input name="name" value="Петя">
  <input name="surname" value="Васечкин">
</form>

<script>
  // заполним FormData данными из формы
  let formData = new FormData(document.forms.person);

  // добавим ещё одно поле
  formData.append("middle", "Иванович");

  // отправим данные
  let xhr = new XMLHttpRequest();
  xhr.open("POST", "/article/xmlhttprequest/post/user");
  xhr.send(formData);

  xhr.onload = () => alert(xhr.response);
</script>
```

XMLHttpRequest (POST-запрос)

```
let xhr = new XMLHttpRequest();

let json = JSON.stringify({
  name: "Вася",
  surname: "Петров"
});

xhr.open("POST", '/submit')
xhr.setRequestHeader('Content-type', 'application/json; charset=utf-8');

xhr.send(json);
```

XMLHttpRequest (прогресс отправки)

```
1 xhr.upload.onprogress = function(event) {
2   alert(`Отправлено ${event.loaded} из ${event.total} байт`);
3 };
4
5 xhr.upload.onload = function() {
6   alert(`Данные успешно отправлены.`);
7 };
8
9 xhr.upload.onerror = function() {
10   alert(`Произошла ошибка во время отправки: ${xhr.status}`);
11 };
```

Cookie

Подробнее тут https://learn.javascript.ru/cookie


WARNING
THIS WEBSITE USES
cookies

HTTP is a stateless protocol

Cookie

`Cookie` - небольшой фрагмент данных, отправленный веб-сервером и хранимый на компьютере пользователя. Веб-клиент всякий раз при попытке открыть страницу соответствующего сайта пересылает этот фрагмент данных веб-серверу в виде HTTP - запросы

Cookie (спецификация)

- Браузер должен хранить как минимум 4096 байт кук
- Минимум 20 шт. на домен
- Минимум 300 шт. всего
- Имена не чувствительны к регистру

1. Cookie (клиент -> сервер)

```
1 GET / HTTP/1.1
2 Host: example.com
```

2. Cookie (клиент <- сервер)

```
1 HTTP/1.1 200 OK
2 Set-Cookie: name=value
3 Content-Type: text/html
```

3. Cookie (клиент -> сервер)

```
1 GET / HTTP/1.1
2 Host: example.com
3 Cookie: name=value
```

4. Cookie (клиент -> сервер)

```
1 HTTP/1.1 200 OK
2 Set-Cookie: name=value2
3 Content-Type: text/html
```

5. Cookie (клиент -> сервер)

```
1 GET / HTTP/1.1
2 Host: example.com
3 Cookie: name=value2
```

Cookie (использование)

- Аутентификация пользователя
- Хранение настроек пользователя
- Отслеживание сеанса (сессия)
- Сбор статистики
- Проведение экспериментов

Cookie (параметры)

- Expires - дата истечения срока действия куки
- Max-Age - срок действия куки в секундах
- Domain - домен определяет, где доступен файл куки
- Path - урл префикс пути, по которому куки будут доступны
- Secure - куки следует передавать только по HTTPS
- HttpOnly - запрет на получение куки из JavaScript

Cookie (JavaScript)

```
1 alert( document.cookie ); // cookie1=value1; cookie2=value2;...
2
3 document.cookie = "user=John"; // обновляем только куки с именем 'user'
4 alert(document.cookie); // user=John, cookie1=value1; cookie2=value2;...
```

Cookie (XHR)

```
1 let xhr = new XMLHttpRequest();
2 xhr.withCredentials = true;
3
4 xhr.open('POST', 'http://anywhere.com/request');
```

Same Origin Policy

Подробнее тут https://learn.javascript.ru/fetch-crossorigin

Same Origin Policy

Same Origin Policy (правило ограничения домена) - это важное концепция безопасности работы web-приложения. Она призвана ограничивать возможности пользовательских сценариев из определенного источника по доступу к ресурсам и информации из других источников.

Same Origin Policy (источник)

Два URL считаются имеющими один источник (“same origin”), если у них одинаковый протокол, домен и порт

Same Origin Policy (одинаковый источник)

- http://site.com
- http://site.com/
- http://site.com/my/page.html

Same Origin Policy (разные источник)

- http://site.com
- http://**www**.site.com (другой домен)
- http://site.**org** (другой домен)
- **https**://site.com (другой протокол)
- http://site.com:**8080** (другой порт)

Same Origin Policy (типы взаимодействия с ресурсами)

- Запись в ресурсы - переходы по ссылкам, редиректы, сабмит форм
- Встраивание ресурсов в другие ресурсы - добавление JavaScript, CSS, Images и тд с помощью HTML тегов
- Чтение из других ресурсов - чтение ответов на запросы, получение доступа к содержимому встроенного ресурса

Same Origin Policy (типы взаимодействия с ресурсами)

- Cross Origin “Запись” - обычно разрешена
- Cross Origin “Встраивание” - обычно разрешено
- Cross Origin “Чтение” - по-умолчанию запрещено

Cookie (samesite)

Samesite — сравнительно новый параметр кук, предоставляющий дополнительный контроль над их передачей согласно Origin policy. Важно заметить, что данная настройка работает только с **secure** cookies.

Возможные значения: Strict, Lax, None.

Cookie (strict)

Cookies с `samesite=strict` **никогда** не отправятся, если пользователь пришел не с этого же сайта.

Важно помнить, что cookies не будут пересылаться также при навигации высокого уровня (т.е. даже при переходе по ссылке)

Пример: vk.com -> site.com -> cookies не передаются


SERVER
API
CLIENT
CORS ERROR
imgflip.com

CORS

Cross-Origin Resource Sharing (CORS) standard — спецификация, позволяющая обойти ограничения, которые Same Origin Policy накладывает на кросс-доменные запросы

CORS (XHR)

```
1 // Находимся на https://evil.com/
2 const xhr = new XMLHttpRequest();
3 xhr.open('GET', 'https://e.mail.ru/messages/inbox/', false);
4 xhr.send();
5
6 console.log(xhr.responseText)
```

CORS (браузер)

Браузер играет роль доверенного посредника:

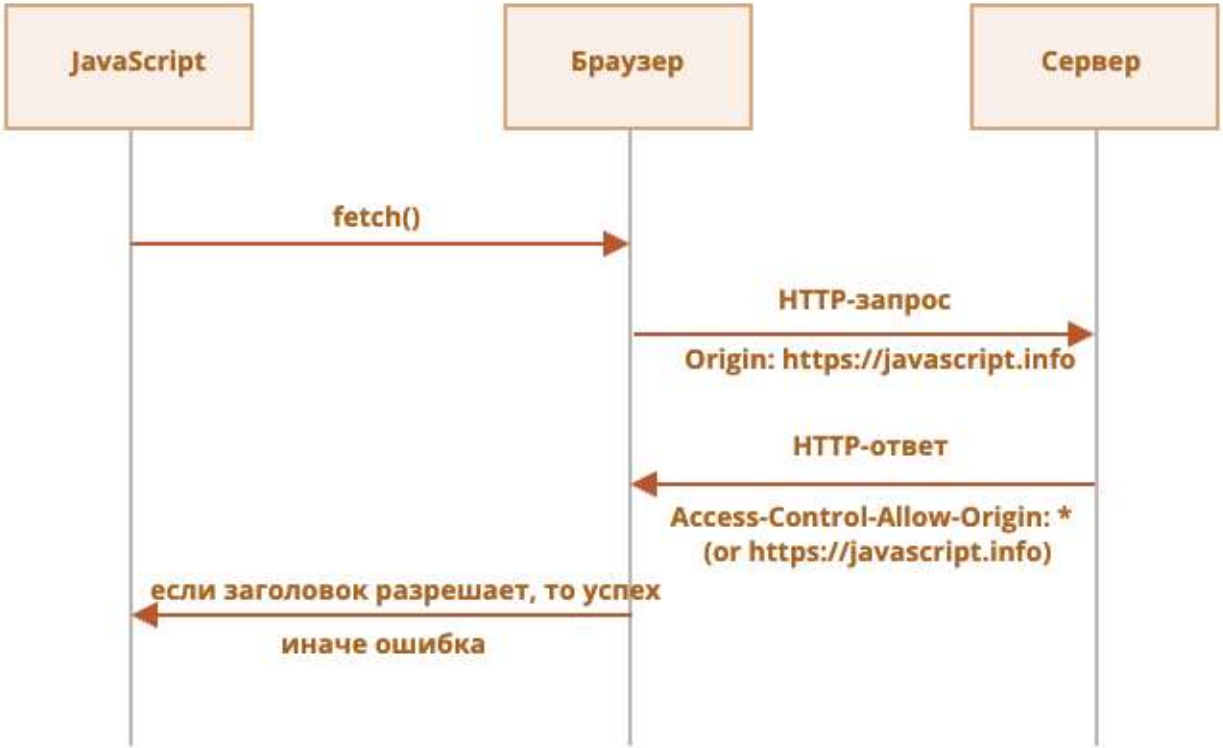
- Он гарантирует, что к запросу на другой источник добавляется правильный заголовок Origin.
- Он проверяет наличие разрешающего заголовка **Access-Control-Allow-Origin** в ответе и, если всё хорошо, то JavaScript получает доступ к ответу сервера, в противном случае – доступ запрещается с ошибкой.

CORS (простые запросы)

Запросы считаются простыми, если они удовлетворяют двум условиям:

- Простой метод: GET, POST или HEAD
- Простые заголовки
  - Acceps
  - Accept-Language
  - Content-Language
  - Content-Type
    - application/x-www-form-urlencoded
    - multipart/form-data
    - text/plain

CORS (простые запросы)



1. CORS (запрос сервиса)

Например, мы запрашиваем https://anywhere.com/request со страницы https://javascript.info/page

```
1 GET /request
2 Host: anywhere.com
3 Origin: https://javascript.info
```

2. CORS (ответ сервиса)

```
1 200 OK
2 Content-Type:text/html; charset=UTF-8
3 Access-Control-Allow-Origin: https://javascript.info
```

CORS (непростые запросы)

Остальные запросы считаются непростыми. При их отправке нужно понять согласен ли сервер на обработку таких запросов. Эти запросы всегда отсылаются со специальным заголовком Origin.

CORS (непростые запросы)

При отправке непростого запроса, браузер сделает на самом деле два HTTP-запроса.

- «Предзапрос» (английский термин «preflight») OPTIONS. Содержит название желаемого метода в заголовке **Access-Control-Request-Method**, а если добавлены особые заголовки, то и их тоже — в **Access-Control-Request-Headers**.
- Основной HTTP-запрос с заголовком Origin

CORS (OPTIONS запрос)

Предварительный запрос использует метод OPTIONS, который содержит 3 заголовка

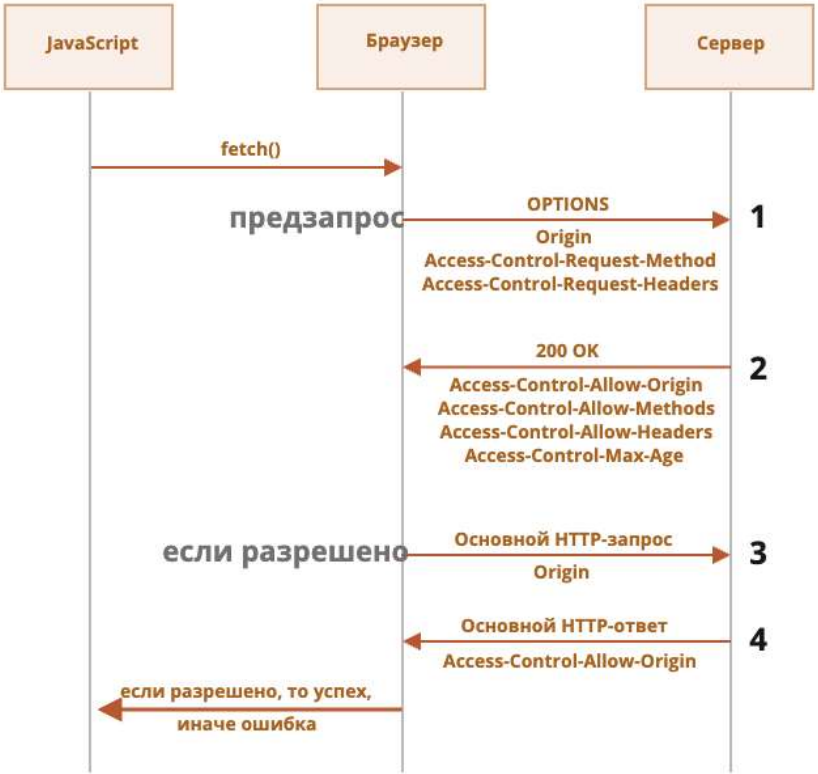
- Origin - содержит домен источника
- Access-Control-Request-Method - содержит HTTP метод запроса
- Access-Control-Request-Headers - содержит список HTTP заголовков запроса

CORS (OPTIONS запрос)

Если сервер согласен принять запрос, то он должен ответить 3 заголовками

- Access-Control-Allow-Origin - список разрешенных источников
- Access-Control-Allow-Method - содержит список разрешенных методов
- Access-Control-Allow-Headers - содержит список разрешенных заголовков
- Access-Control-Max-Age - указывает количество секунд, на которое можно кешировать решение

CORS (непростые запросы)



1. CORS (предзапрос сервиса)

Например, мы запрашиваем https://site.com/service.json со страницы https://javascript.info/page с методом PATCH и заголовками Content-Type и API-Key

```
1 OPTIONS /service.json
2 Host: site.com
3 Origin: https://javascript.info
4 Access-Control-Request-Method: PATCH
5 Access-Control-Request-Headers: Content-Type,API-Key
```

2. CORS (ответ сервиса на предзапрос)

```
200 OK
Access-Control-Allow-Origin: https://javascript.info
Access-Control-Allow-Methods: PUT,PATCH,DELETE
Access-Control-Allow-Headers: API-Key,Content-Type,If-Modified-Since,Cache-Control
Access-Control-Max-Age: 86400
```

3. CORS (запрос сервиса)

```
1 PATCH /service.json
2 Host: site.com
3 Content-Type: application/json
4 API-Key: secret
5 Origin: https://javascript.info
```

4. CORS (ответ сервиса)

```
1 200 OK
2 Content-Type:text/html; charset=UTF-8
3 Access-Control-Allow-Origin: https://javascript.info
```

CORS (дополнительные заголовки)

```
1 200 OK
2 Content-Type:text/html; charset=UTF-8
3 Access-Control-Allow-Origin: https://javascript.info
4 Access-Control-Expose-Headers: X-Uid, X-Secret
```

CORS (данные авторизации)

```
1 200 OK
2 Content-Type:text/html; charset=UTF-8
3 Access-Control-Allow-Origin: https://javascript.info
4 Access-Control-Allow-Credentials: true
```

Как еще можно решить проблему корсов?