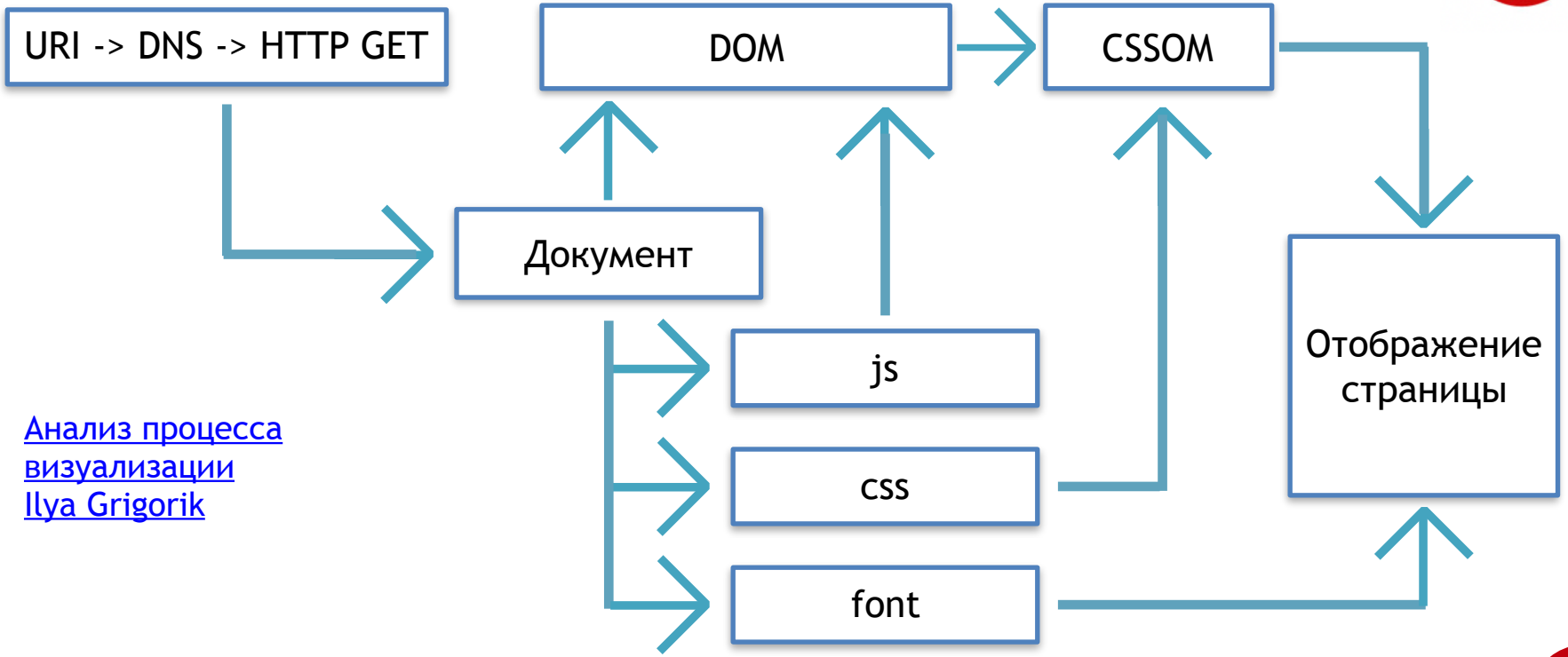
Взаимодействие с сервером

Борис Ребров

Минутка бюрократии

- Внимание
- Отметки о посещении занятий
- Обратная связь о лекциях

Порядок загрузки страницы



[Анализ процесса визуализации Ilya Grigorik]

Включение ресурсов

- JS

```
1. <script async />   не блокирует DOM, выполнится ASAP
2. <script defer />   выполнится после построения DOM
```

```
1. var script = document.createElement('script');
2. script.addEventListener('load', (event) => { });
3. script.src = '/dist/main.js';
4. document.head.appendChild(script);
```

- CSS

```
1. <link href="/dist/touch.css" rel="stylesheet" media="(max-width: 320px)">
2. <link href="landscape.css" as="style" rel="preload" media="(orientation: landscape)">
```

```
1. @import "common.css" screen;
2. @import url('landscape.css') screen and (orientation:landscape);
```

Включение ресурсов: приоритет и предзагрузка

- Низкий приоритет, для следующих экранов

```
<link rel="prefetch">
```

- Высокий приоритет, для текущего экрана (CR)

```
<link rel="preload" as="script">
```

- Resource Hints (WD)

```
<link rel="dns-prefetch">
<link rel="preconnect">
<link rel="prerender">
```

События загрузки

- DOMContentLoaded
  - Блокирующие ресурсы загружены
  - Построение DOM завершено

```
1. document.addEventListener("DOMContentLoaded", (event) => { });
```

- load
  - Все внешние ресурсы загружены
  - Построение CSSOM завершено
- unload/beforeunload

```
1. window.addEventListener("load", (event) => { });
2. window.addEventListener("beforeunload", (event) => {
3.  event.preventDefault();
4.  event.returnValue = '';
5. });
```

URI

https://example.com/path/page.ext?query=1#second

схема хост путь запрос фрагмент

- window.location
- window.URL
  - protocol, hostname, pathname, search, hash
  - port, host, searchParams, password, username
  - href

```
1. const baseUrl = new URL('https://example.com');
2. const myUrl = new URL('path/page.ext?query=1#second');
3. window.location.assign(myUrl);
```

Same-origin policy

- Origin – хост, протокол и порт
- Можно понижать уровень домена

```
1. document.domain = 'example.com';
```

- Если источники различаются
  - Можно отправлять запросы и встраивать контент
  - Нельзя программно читать ответы
  - Нет доступа к DOM и свойству location
- Для iframe есть X-Frame-Options

```
1. X-Frame-Options: dany
2. X-Frame-Options: allow-from https://example.com/
```

CORS

- Техника, управления доступом к ресурсам из внешних источников
- Набор HTTP-заголовков
  - Access-Control-Allow-*
  - Access-Control-Request-*

```
1. Access-Control-Allow-Origin: https://example.com
```

- Реквизиты доступа (credentials)
  - Не будет отправлены по-умолчанию
  - Будут отправлены только с разрешения источника

CORS: Простые запросы

- Методы
  - GET, POST, HEAD
- Заголовки
  - Accept, Accept-Language, Content-Language
  - Content-Type
    - application/x-www-form-urlencoded
    - multipart/form-data
    - text/plain

```
1. GET /public HTTP/1.1
2. Host: my-example.com
3. Origin: https://example.com
```

```
1. HTTP/1.1 200 OK
2. Access-Control-Allow-Origin: *
```

CORS: Preflight

- Непростой запрос
  - Отличается метод
  - Присутствуют другие заголовки
  - Другой Content-Type
- Используется дополнительный запрос OPTION

```
OPTIONS /public HTTP/1.1
Host: my-example.com
Access-Control-Request-Method: POST
Access-Control-Request-Headers:
X-CSRF-Toket, Content-Type
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Methods: POST, GET, OPTIONS
Access-Control-Allow-Headers:
X-CSRF-Toket, Content-Type
Access-Control-Max-Age: 600
```

```
POST /public HTTP/1.1
Host: my-example.com
X-CSRF-Toket: some-value
Origin: https://example.com
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://example.com
```

CORS: Credentials

- Реквизиты доступа
  - Cookie
  - Authorization
  - TLS-сертификат
- Для простых и «preflight» запросов
  - Ответ на простой нельзя прочесть
  - Целевой запрос не будет отправлен, если в preflight нет разрешения

```
POST /public HTTP/1.1
Host: my-example.com
Cookie: key=some-value
Origin: https://example.com
```

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: https://example.com
Access-Control-Allow-Credentials: true
```

Выполнение запросов

	Встроенное поведение	Обработка результата	Отмена запроса
Переход между страницами	Да	Нет *	Нет
Отправка формы	Да	Нет *	Нет
Добавление ресурса	Да *	Да *	Нет
XHR/Fetch	Нет	Да	Да *

Выполнение запросов: XHR

- XMLHttpRequest и ajax – «исторические» названия
- К счастью, доступен не только XML

```
1.  const myRequest = new XMLHttpRequest;
2.  myRequest.addEventListener('readystatechange', (event) => {
3.      if (myRequest.readyState != XMLHttpRequest.DONE) return;
4.      if (myRequest.status == 200) {
5.          console.log(JSON.parse(myRequest.responseText));
6.      } else {
7.          console.log(myRequest.responseText);
8.      }
9.  });
10. myRequest.open('GET', '/test.json', true);
11. myRequest.send();
```

Выполнение запросов: XHR

```
1.  const myRequest = new XMLHttpRequest;
2.  XMLHttpRequest.prototype.readyState;
3.  // 0  UNSENT
4.  // 1  OPENED
5.  // 2  HEADERS_RECEIVED
6.  // 3  LOADING
7.  // 4  DONE
8.  XMLHttpRequest.prototype.onreadystatechange = () => {};

9.  XMLHttpRequest.prototype.response/responseText

10. XMLHttpRequest.prototype.status/statusText

11. XMLHttpRequest.prototype.upload //EventTarget

12. XMLHttpRequest.prototype.withCredentials
```

Выполнение запросов: XHR

```
1.  XMLHttpRequest.prototype.setRequestHeader(header, value);

2.  XMLHttpRequest.prototype.open(method, url[, async = false[, user[, password]]]);

4.  XMLHttpRequest.prototype.send([body = null]);

6.  XMLHttpRequest.prototype.abort();

8.  XMLHttpRequest.prototype.getResponseHeader(headerName);

10. XMLHttpRequest.prototype.getAllResponseHeaders();
```

```
1.  request.onreadystatechange = () => {
2.      if(this.readyState == this.HEADERS_RECEIVED) {
3.          let headers = request.getAllResponseHeaders().trim().split(/[\r\n]+/);
4.          console.log(headers);
5.      }
6.  }
```

Выполнение запросов: fetch

- Более удобная замена XHR
- Возвращает Promise
- Отдельные объекты запроса, ответа и заголовков
- 404, 500, etc – вернут fullfiled Promise

```
1. const myRequest = fetch('/test.json');
2. myRequest.then(
3.     response => response.ok && response.json()
4. ).then(console.log);
```

Выполнение запросов: fetch

- Принимает строку или объект Request

```
fetch(input[, init]);
```

- Объект инициализации
  - method, body, headers
  - mode, credentials
  - cache, redirect

```
const myInit = {
    method: 'GET',
    mode: 'cors',
    cache: 'default',
    body: JSON.stringify({test: 'value')
};
const myRequest = new Request('/test.json');
fetch(myRequest, myInit).then((response) => {});
```

Выполнение запросов: fetch

- Объект запроса
  - Обычно не создается явно
  - Содержится с свойстве FetchEvent (SW)
  - Сигнатура конструктора совпадает с fetch
  - Наследует интерфейс Body

```
new Request(input[, init]);
```

```
request[
    arrayBuffer ||
    blob        ||
    formData    ||
    json        ||
    text
]().then((result) => {});

```

Выполнение запросов: fetch

- Объект ответа
  - Разрешает Promise возвращаемый fetch
  - Наследует интерфейс Body

```
1. response.prototype.ok // Статус > 200 и < 300

3. response.prototype.status/statusText
```

```
1. fetch('igorek.jpg').then(function(response) {
2.     return response.blob();
3. }).then(function(blob) {
4.     const objectURL = URL.createObjectURL(blob);
5.     const myImage = document.createElement('img');
6.     myImage.src = objectURL;
7. });
```

Выполнение запросов: formData

- Позволяет конструировать объекты с данными формы
- Конструктор принимает ссылку на элемент формы

```
1.  new FormData([form]);

3.  FormData.prototype.append(name, value[, filename]);
4.  FormData.prototype.set(name, value[, filename]);

6.  FormData.prototype.delete(name);

8.  FormData.prototype.get(name);

10. FormData.prototype.has(name);
```

Выполнение запросов: jsonp

- Способ обойти CORS
- Пример запроса:

```
1. function appendScript(src) {
2.   const script = document.createElement('script');
3.   script.src = src;
4.   script.onload = () => document.head.removeChild(script);
5.   document.head.appendChild(script);
6. }

8. appendScript('user?id=123&callback=test');
9. window.test = (data) => console.log(data);
```

- Ответ сервера:

```
1. 'test' in window && window.test({test: 'value'});
```

Обработка данных

- Array.prototype.map/reduce

```
1. [1,2,3].map(num => ++num); // [2,3,4];
2. [1,2,3].reduce((result, num) => result + num}, 0);
```

- Map

```
1. const myMap = new Map();
2. const myObj = {};
3. myMap.set(myObj, true);
4. myMap.get(myObj); // true
5. myMap.get({}); // undefined
```

- Set

```
1. new Set([1,1,2,3,4]); // {1, 2, 3, 4}
```

- Array.from(arrayLike[, mapFn[, thisArg]])

```
1. Array.from(new Set([1,1,2,3,4])); // [1, 2, 3, 4]
```

Домашнее задание № 4

- Отправлять данные сообщений на сервер
- Отображать статус отправки (загружается/загружено) изменяя иконку элемента сообщения
- Настроить публикацию пакета со статикой

Срок сдачи

- ~ 29 октября

Спасибо за внимание!