

HTTP и современный Web



Григорий
Вахмистров



Григорий Вахмистров

Backend Developer в Tennisi.bet

План занятия

1. [Введение](#)
2. [Web & HTTP](#)
3. [Веб-технологии](#)
4. [Гиперссылки и формы](#)
5. [Frontend](#)
6. [Итоги](#)
7. [Домашнее задание](#)



Введение

Введение

Мы приступаем к изучению самого популярного фреймворка для Java - **Spring Framework** (Spring).

Наша задача - научиться им пользоваться и понять, как он устроен внутри. Именно это будет определять то, насколько профессионально вы сможете использовать фреймворк.



Введение

Начиная с [первого релиза \(24 марта 2004\)](#), Spring неразрывно связан с созданием веб-приложений.

Мы поговорим и в целом о web, и о frontend в частности, чтобы понимать, как работает клиентская часть. От этого будет зависеть как и какой backend мы сможем написать.

Замечание о стиле кодирования

Классы и интерфейсы из Spring и многие другие, которые мы будем использовать, имеют зачастую очень длинные имена, например, `AnnotationConfigApplicationContext` (и это ещё "средних" размеров). Поэтому мы активно в лекциях будем использовать `var` из Java 11, чтобы код помещался на слайдах.

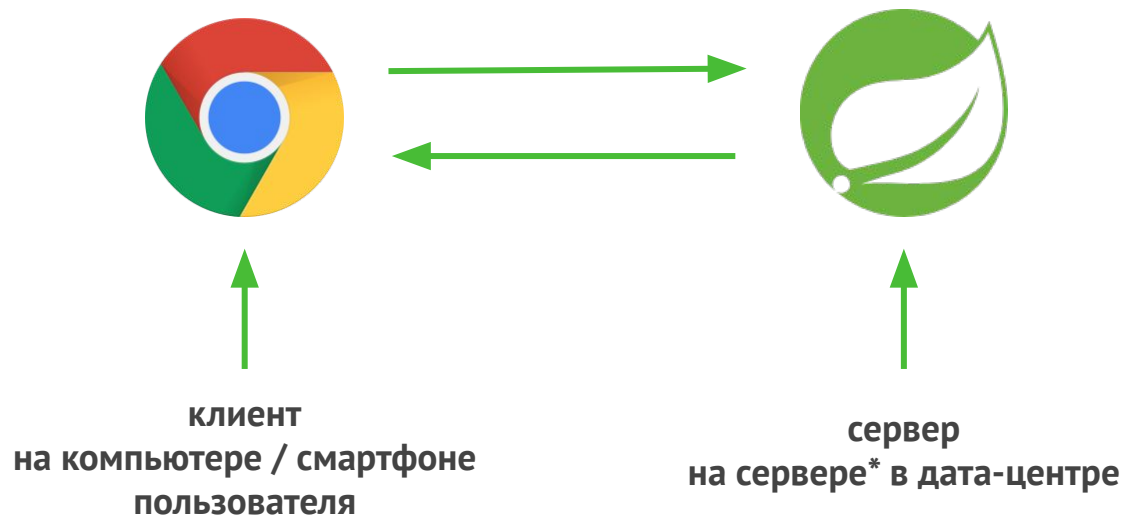
В реальной жизни вам нужно исходить из Code Conventions, принятых на проекте.



Web & HTTP

Клиент-серверная модель

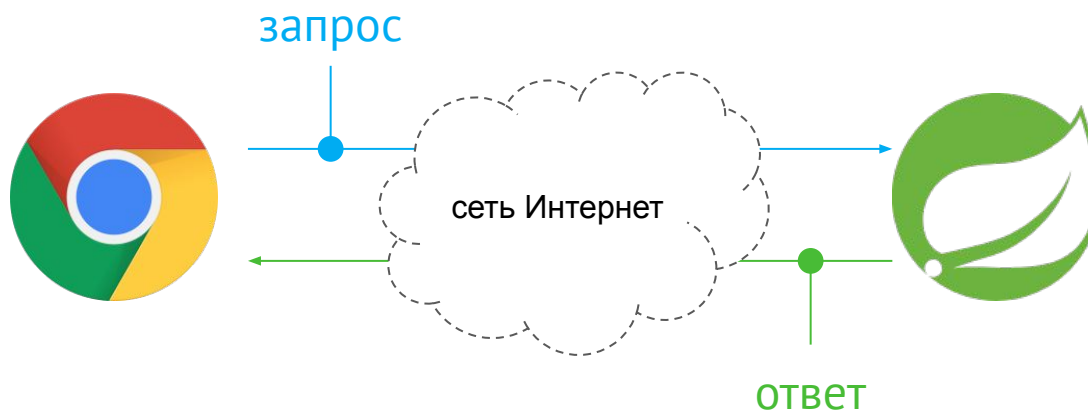
Общая схема работы выглядит следующим образом:



*Под сервером понимается как приложение, так и виртуальная / физическая машина

HTTP

Мы будем рассматривать взаимодействие по протоколу HTTP.
В качестве формата обмена данными могут служить HTML, CSS, изображения, аудио, видео и другие (JSON/XML).



Web

Web (World Wide Web) - система связанных веб-страниц (посредством гиперссылок), доступных через сеть Интернет.

Ключевые вещи:

- HTTP используется для передачи данных между сервером и клиентом;
- Клиент указывает URL* для доступа к конкретному компоненту системы (например, странице);
- HTML - один из форматов представления документов.

*URL - uniform resource location

Общая схема

Браузер и сервер могут быть написаны на разных языках и работать на разных платформах.

Поэтому:

- **нужен транспорт для доставки информации** (набора байт) от одного приложения к другому;
- **нужны правила интерпретации этих байт** — приложения должны понимать друг друга.

Протокол

Протокол – это “правила общения” двух сторон.

Пример: приезжая в аэропорт, вы проходите фиксированную процедуру перед посадкой в самолёт:

- проверка документов;
- контроль безопасности;
- проверка билета перед посадкой.

Если вы вдруг приедете без документов, то вы не выполните условия протокола, и сотрудники аэропорта – сторона, с которой вы взаимодействуете, вас не поймёт и не пропустит.

Протокол

В мире приложений всё так же: договорённости устанавливаются на множестве уровней:

- **от физического** — какие электромагнитные сигналы пересылаются;
- **до уровня приложений** — в каком формате и какие данные передаются.

OSI vs TCP/IP



Про TCP/IP вы уже знаете, нас интересует **HTTP**.

RFC

RFC (Request For Comments) – специальный тип документов, которые описывают стандарты, протоколы, форматы и т.д. Он используется в качестве транспорта для данных уровня приложения.

HTTP 1.1: <https://tools.ietf.org/html/rfc2616>

HTTP 2.0: <https://tools.ietf.org/html/rfc7540>

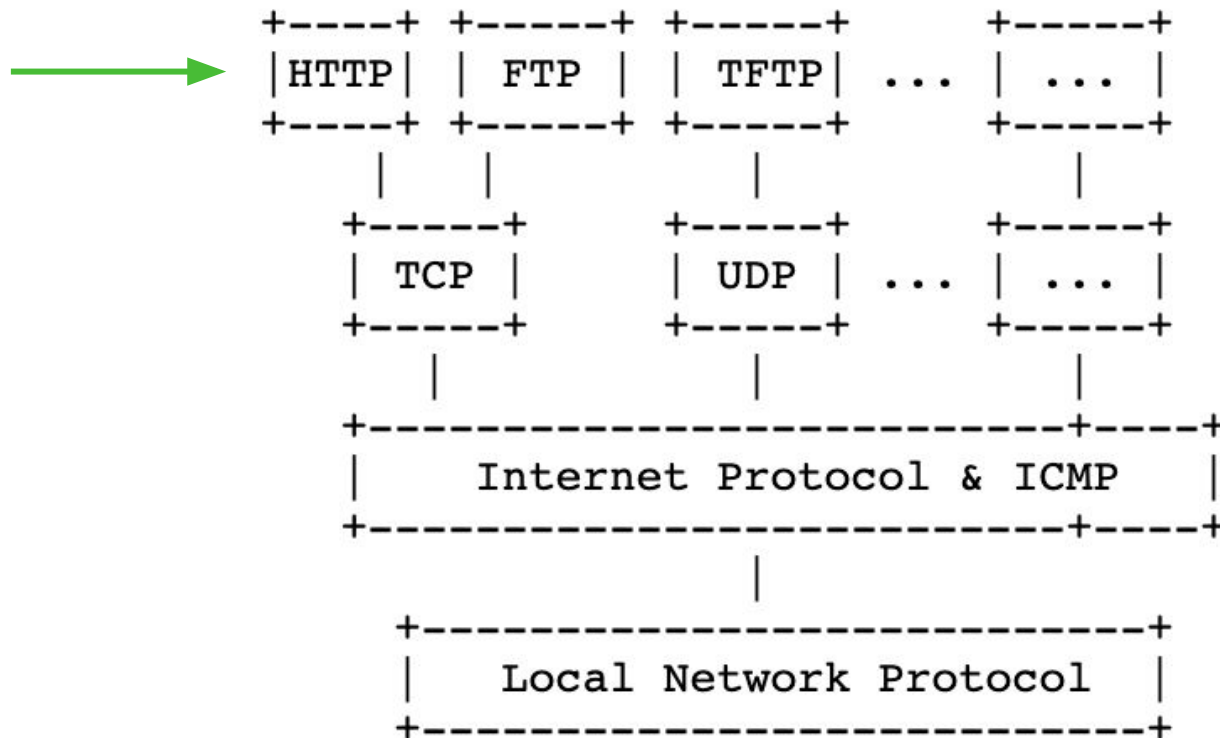
*RFC читать достаточно полезно, поскольку это первоисточник.

RFC

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext.

Вольный перевод: HTTP - протокол прикладного уровня для распределённых гипермедиа информационных систем. Это протокол общего назначения, который может быть использован для многих задач помимо передачи гипертекста.

HTTP - место в стеке протоколов



HTTP

На сегодняшний день есть две версии протокола:

- HTTP 1.1 – текстовая. Значит запросы и ответы можно **интерпретировать как строки** в "человекопонятном" формате
- HTTP 2 – бинарная. Значит запросы и ответы нельзя интерпретировать как строки

Рассмотрим пока версию 1.1.

HTTP

В рамках протокола определяется понятие «сообщения» (message) - единица передачи информации.

Сообщение может быть либо запросом, либо ответом:

4.1 Message Types

HTTP messages consist of requests from client to server and responses from server to client.

HTTP-message = Request | Response ; HTTP/1.1 messages

Request

A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource, the identifier of the resource, and the protocol version in use.

```
Request      = Request-Line           ; Section 5.1
               *(( general-header      ; Section 4.5
                   | request-header    ; Section 5.3
                   | entity-header ) CRLF) ; Section 7.1
\r\n → CRLF
               [ message-body ]       ; Section 4.3
```

Response

After receiving and interpreting a request message, a server responds with an HTTP response message.

```
Response     = Status-Line           ; Section 6.1
               *(( general-header      ; Section 4.5
                   | response-header   ; Section 6.2
                   | entity-header ) CRLF) ; Section 7.1
\r\n → CRLF
               [ message-body ]       ; Section 7.2
```


\r\n

Напоминаем, что **\r** - это возврат каретки (Carriage Return - CR),
а **\n** - это перенос строки (Line Feed - LF).

Из лекций по работе с файлами вы должны помнить, что для индикации окончания строки (line) используют либо CRLF, либо LF. В HTTP выбран именно CRLF.



HTTP Server (попробуем написать свой сервер)

```
public class Main {  
    public static void main(String[] args) {  
        final var validPaths :List<String> = List.of("/index.html", "/spring.svg", "/spring.png");  
  
        try (final var serverSocket = new ServerSocket( port: 9999)) {  
            while (true) {  
                try (  
                    final var socket :Socket = serverSocket.accept();  
                    final var in = new BufferedReader(new InputStreamReader(socket.getInputStream()));  
                    final var out = new BufferedOutputStream(socket.getOutputStream());  
                    ) {...}  
                }  
            } catch (IOException e) {  
                e.printStackTrace();  
            }  
        }  
    }  
}
```

бесконечный
цикл
обработки
запросов

Мы специально захардкодили некоторые значения, чтобы "уместить" код на слайде.

В реальности, их, конечно же, нужно выносить.

HTTP

В рамках HTTP определяются:

- **Методы** GET, POST, PUT, DELETE (и т.д.) определяют логическое назначение действия + ограничения на запрос. Например, у GET тело запроса должно быть пустое.
- **Статус-коды** ответов:
 - **1xx** — информационные;
 - **2xx** — успешно;
 - **3xx** — перенаправление;
 - **4xx** — ошибки клиента;
 - **5xx** — ошибки сервера.

Код обработки подключения
(внутри while)

```
// read only request line for simplicity
// must be in form GET /path HTTP/1.1
final var requestLine :String = in.readLine();
final var parts :String[] = requestLine.split( regex: " ");

if (parts.length != 3) {
    // just close socket
    continue;
}

final var path :String = parts[1];
if (!validPaths.contains(path)) {
    out.write((
        "HTTP/1.1 404 Not Found\r\n" +
        "Content-Length: 0\r\n" +
        "Connection: close\r\n" +
        "\r\n"
    ).getBytes());
    out.flush();
    continue;
}

final var filePath :Path = Path.of( first: ".", ...more: "public", path);
final var mimeType :String = Files.probeContentType(filePath);
final var length :long = Files.size(filePath);
out.write((
    "HTTP/1.1 200 OK\r\n" +
    "Content-Type: " + mimeType + "\r\n" +
    "Content-Length: " + length + "\r\n" +
    "Connection: close\r\n" +
    "\r\n"
).getBytes());
Files.copy(filePath, out);
out.flush();
```

HTTP

Так устроены почти все сервера - бесконечный цикл, в котором обслуживаются запросы клиентов.

Конечно, в других серверах "навешено" дополнительной функциональности, но в HTTP-серверах суть та же:

- на входе:
 - request line
 - headers
 - body
- на выходе:
 - status line
 - headers
 - body

HTML Document - Chromium

HTML Document x +

localhost:9999/index.html

Incognito

HTML Document

интерпретируется
как HTML
документ

заголовки запроса

заголовки запроса

1 requests | 4

Elements Console Sources Network Performance >> | Settings

Filter Hide data URLs

All XHR JS CSS Img Media Font Doc WS Manifest Other Has blocked cookies

Blocked Requests

Name x Headers Preview Response Initiator Timing

index.html

General

Request URL: http://localhost:9999/index.html

Request Method: GET

Status Code: 200 OK

Remote Address: [::1]:9999

Referrer Policy: no-referrer-when-downgrade

Response Headers view parsed

HTTP/1.1 200 OK

Content-Type: text/html

Content-Length: 344

Connection: close

Request Headers view parsed

GET /index.html HTTP/1.1

Host: localhost:9999

Connection: keep-alive

Pragma: no-cache

Cache-Control: no-cache

именно благодаря этому



Developer Tools

[Developer Tools](#) - это инструмент, который встроен в Chrome, позволяющий в том числе просматривать [отправляемые запросы](#).

Открывается по нажатию клавиши F12.

Мы будем активно им пользоваться.

spring.png (401×136) - Chromium

spring.png (401×136) x +

localhost:9999/spring.png

Incognito

Elements Console Sources Network Performance >> ⚙️ ⋮ ×

⛔ 🔍 ☒ Preserve log ☒ Disable cache Online ⬆️ ⬇️ ⚙️

Filter ☐ Hide data URLs

All XHR JS CSS Img Media Font Doc WS Manifest Other ☐ Has blocked cookies

☐ Blocked Requests

Name	×	Headers	Preview	Response	Initiator	Timing
spring.png		General				
Request URL: http://localhost:9999/spring.png						
Request Method: GET						
Status Code: 200 OK						
Remote Address: [::1]:9999						
Referrer Policy: no-referrer-when-downgrade						
Response Headers view source						
Connection: close						
Content-Length: 9440						
Content-Type: image/png ← именно благодаря этому						
Request Headers view source						
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9						
Accept-Encoding: gzip, deflate, br						
Accept-Language: en-US,en;q=0.9						
Cache-Control: no-cache						
Connection: keep-alive						

1 requests | 9

интерпретируется как изображение

HTTP

Меняем Content-Type:

```
final var filePath :Path = Path.of( first: ".", ...more: "public", path);  
final var mimeType = "text/plain"; // Files.probeContentType(filePath);  
final var length :long = Files.size(filePath);
```

localhost:9999/index.html - Chromium

localhost:9999/index.html x +

localhost:9999/index.html

Incognito

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport"
    content="width=device-
width, user-scalable=no,
initial-scale=1.0, maximum-
scale=1.0, minimum-scale=1.0">
  <meta http-equiv="X-UA-
Compatible" content="ie=edge">
  <title>HTML Document</title>
</head>
<body>
  <h1>HTML Document</h1>
</body>
</html>
```

↑

не интерпретируется
как HTML документ

Elements Console Sources Network Performance >> ⚙️ ⋮ ✕

⛔ ⛔ 🔍 ☒ Preserve log ☒ Disable cache Online ⌵ ⬆️ ⬇️ ⚙️

Filter ☐ Hide data URLs

All XHR JS CSS Img Media Font Doc WS Manifest Other ☐ Has blocked cookies

☐ Blocked Requests

Name	×	Headers	Preview	Response	Initiator	Timing
index.html		▼ General				
Request URL: http://localhost:9999/index.html						
Request Method: GET						
Status Code: 🟢 200 OK						
Remote Address: [::1]:9999						
Referrer Policy: no-referrer-when-downgrade						
▼ Response Headers view source						
Connection: close						
Content-Length: 344						
Content-Type: text/plain ← именно благодаря этому						
▼ Request Headers view source						
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9						
Accept-Encoding: gzip, deflate, br						
Accept-Language: en-US,en;q=0.9						
Cache-Control: no-cache						
Connection: keep-alive						
Host: localhost:9999						

1 requests | 4

spring.png (401×136) - Chromium

spring.png (401×136) x +

localhost:9999/spring.png

Incognito

Elements Console Sources Network Performance

Filter ☐ Hide data URLs

All XHR JS CSS Img Media Font Doc WS Manifest Other ☐ Has blocked cookies

☐ Blocked Requests

Name x Headers Preview Response Initiator Timing

spring.png

General

Request URL: http://localhost:9999/spring.png

Request Method: GET

Status Code: 200 OK

Remote Address: [::1]:9999

Referrer Policy: no-referrer-when-downgrade

Response Headers view source

Connection: close

Content-Length: 9440

Content-Type: text/plain ← почему?

Request Headers view source

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9

Accept-Encoding: gzip, deflate, br

Accept-Language: en-US,en;q=0.9

Cache-Control: no-cache

Connection: keep-alive

1 requests | 9

интерпретируется как изображение

Content-Type

Заголовок Content-Type определяет тип содержимого в соответствии с [реестром IANA](#). Исходя из значения этого заголовка, браузер решает, как отображать содержимое.

Q: но с изображением не сработало. Почему?

A: браузеры сейчас достаточно умные и включают процесс sniffing: браузер по первым байтам файла видит, что это PNG (Magic Numbers) и игнорирует значение Content-Type.



Веб-технологии

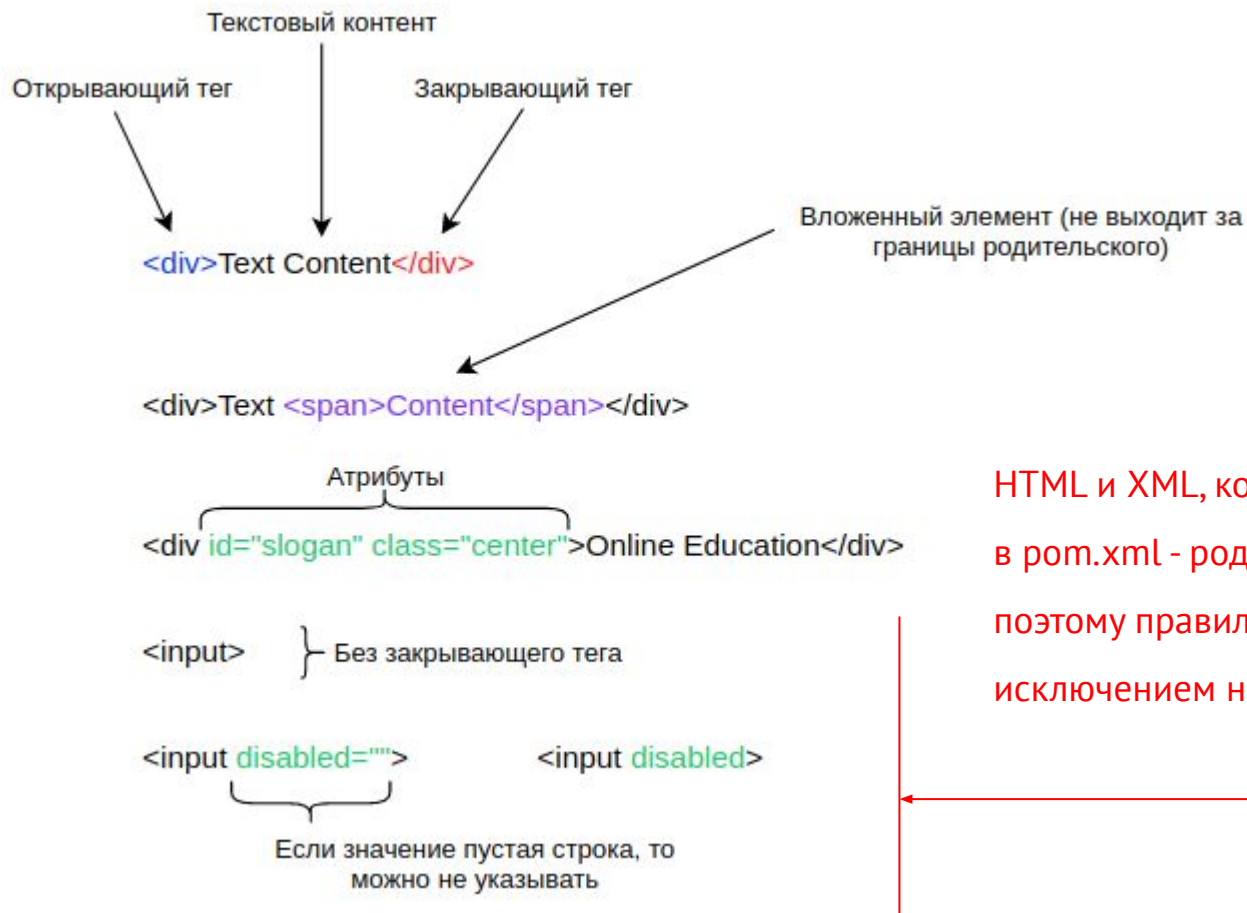
Веб-технологии

В рамках курса нас будут интересовать три ключевых веб-технологии:

- **HTML** — язык, определяющий правила по формированию веб-документов;
- **CSS** — язык, определяющий правила по стилевому оформлению веб-документов;
- **JS** — язык, использующий API браузера для добавления поведения к элементам или переопределения их поведения по умолчанию.

HTML

HTML (HyperText Markup Language) - предоставляет для создания документов, состоящих из структурированных элементов.



HTML и XML, который мы использовали в rom.xml - родственные языки, поэтому правила похожи (за исключением нижних двух строк)



DOM

HTML-документ - это просто текст. Чтобы он превратился в визуальный интерфейс, необходимо этот документ обработать.

В процесс обработки браузер строит объектную модель документа (DOM - Document Object Model) - т.е. каждый элемент превращается в объект во внутреннем представлении браузера.

Поскольку HTML - иерархическая структура, то и DOM - это дерево элементов, где у каждого элемента (кроме корневого) есть только один родитель и сколько угодно детей (от 0).

Синтаксис HTML-элементов

Перечень элементов определяется спецификацией [HTML](#), 4-ый раздел:

§ 4.1.1. The `html` element

Categories:

None.

Contexts in which this element can be used:

As the document's [document element](#).

Wherever a subdocument fragment is allowed in a compound document.

Content model:

A `<head>` element followed by a `<body>` element.

Tag omission in text/html:

An `<html>` element's [start tag](#) can be omitted if the first thing inside the `<html>` element is not a [comment](#).

An `<html>` element's [end tag](#) can be omitted if the `<html>` element is not immediately followed by a [comment](#).

Content attributes:

[Global attributes](#).

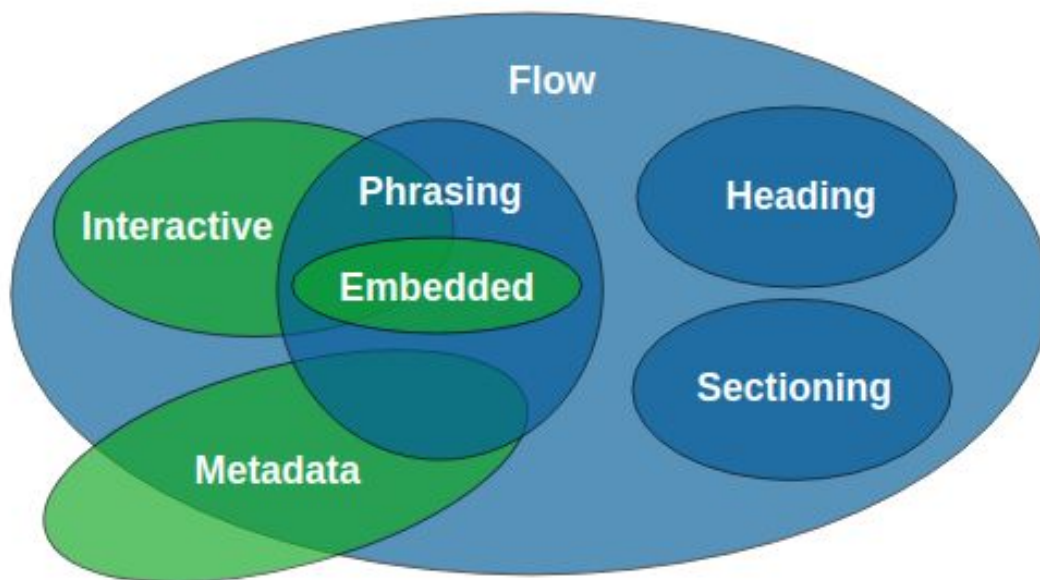
`manifest` — Application cache manifest.

*HTML допускает создание собственных (кастомных) элементов, но это находится за рамками наших интересов.

Категории элементов

Выделяются следующие категории

(нас будут интересовать выделенные зелёным):



Категории элементов

Q: чем они интересны?

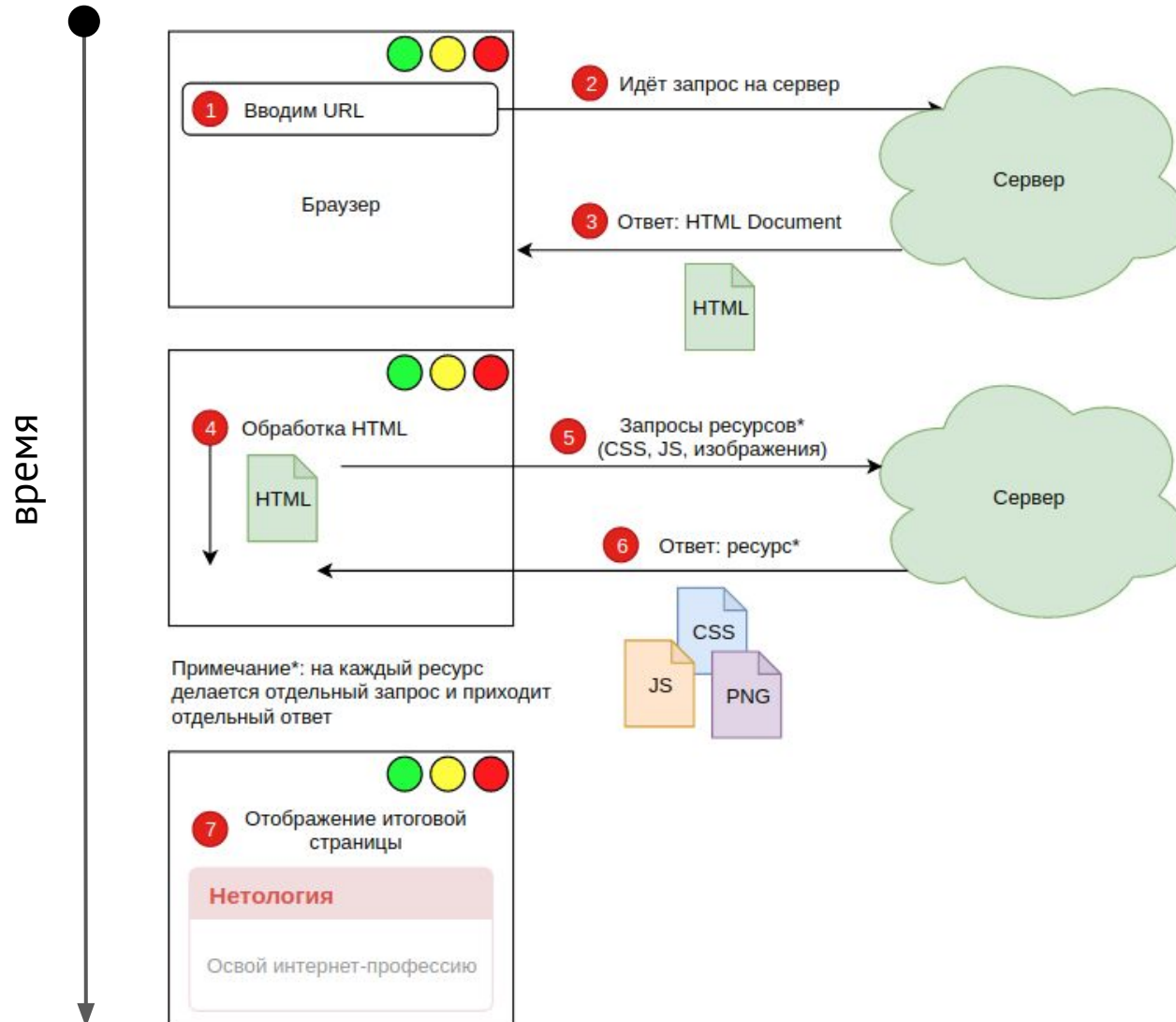
A: они позволяют осуществить взаимодействие с сервером путём отправки HTTP-запросов.

Мы их условно разделим на две категории:

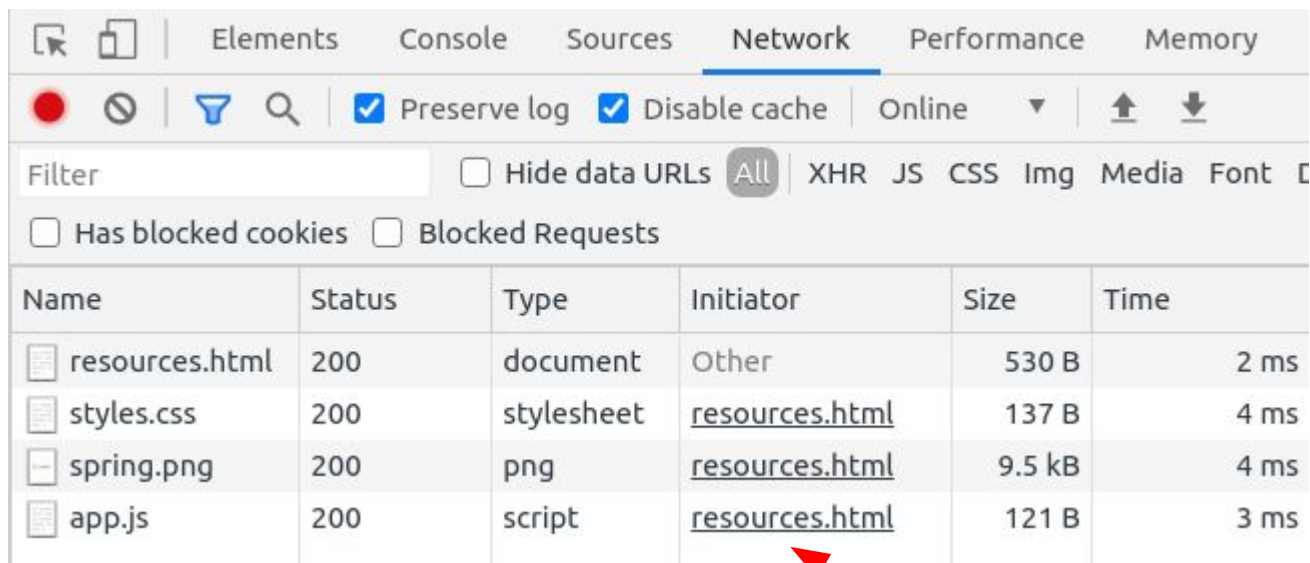
1. Ресурсы (изображения, аудио, видео, CSS, JS)
2. Гиперссылки и формы



Ресурсы (см. /resources.html)



Ресурсы



The screenshot shows the Chrome DevTools Network tab. The top bar includes tabs for Elements, Console, Sources, Network (selected), Performance, and Memory. Below the tabs are icons for a red status indicator, a disabled icon, a filter icon, and a search icon. There are also checkboxes for 'Preserve log' and 'Disable cache', and a dropdown menu set to 'Online'. A 'Filter' input field is present, followed by a 'Hide data URLs' checkbox and a button labeled 'All'. Below these are checkboxes for 'Has blocked cookies' and 'Blocked Requests'. The main table lists network resources with columns for Name, Status, Type, Initiator, Size, and Time. A red arrow points to the 'resources.html' entry in the Initiator column of the 'app.js' row.

Name	Status	Type	Initiator	Size	Time
resources.html	200	document	Other	530 B	2 ms
styles.css	200	stylesheet	resources.html	137 B	4 ms
spring.png	200	png	resources.html	9.5 kB	4 ms
app.js	200	script	resources.html	121 B	3 ms



Гиперссылки и Формы



Гиперссылки

Гиперссылки - это элементы вида:

```
<a href="путь">Текст</a>
```

Переход по гиперссылкам (по обычным, а не по тем, что только выглядят, как гиперссылки) приводит к запросу на сервер с соответствующим URL (см. links.html).

Формы

С формами всё гораздо интереснее:

`<form>`

внутри элементы ввода:

- input (21 тип)
- select
- textarea

`</form>`



Формы

По умолчанию **форма отправляется на тот же адрес, с которого была загружена** с помощью метода GET.

Все элементы ввода, имеющие атрибут name, передаются через параметры запроса (см. forms.html).

Общая схема

Соберём всё в одну картинку:

1. Пользователь вводит URL веб-страницы или переходит по ссылке
2. Сервер отдаёт новый документ, соответствующий этому URL'у
3. Браузер пользователя обрабатывает документ, загружает все ресурсы
4. Пользователь взаимодействует с документом (формами и гиперссылками), что приводит к новым запросам на сервер (по определённому URL'у)
5. Всё повторяется, начиная с шага 2.

Классические веб-приложения

Со статичными файлами (как мы сделали) - всё понятно. Но ведь пользователю на странице может выводиться имя, есть возможность авторизации и т.д. Как это работает?

На самом деле, сервер может просто менять часть контента (тот же `String.replace` прекрасно справится) в зависимости от данных запроса.

Такие приложения раньше называли классическими веб-приложениями, сейчас же их всё чаще неформально называют "олдскульными".



Классические веб-приложения

```
<body>
<h1>Classic Demo</h1>
<p>Current time is: {time}</p>
</body>
```

сами придумали "метку" для замены

```
// special case for classic
if (path.equals("/classic.html")) {
    final var template :String = Files.readString(filePath);
    final var content :byte[] = template.replace(
        target: "{time}",
        LocalDateTime.now().toString()
    ).getBytes();
    out.write((
        "HTTP/1.1 200 OK\r\n" +
        "Content-Type: " + mimeType + "\r\n" +
        "Content-Length: " + content.length + "\r\n" +
        "Connection: close\r\n" +
        "\r\n"
    ).getBytes());
    out.write(content);
    out.flush();
    continue;
}
```

Важно

То, что мы сейчас вам рассказали, используется повсеместно и не зависит от языка.

Равно как отсюда же вытекают все оптимизации и расширенные возможности, которые можно применить:

- заранее читать файлы в память с диска;
- реализовать поддержку "включения" одних файлов в другие (т.е. вынести общие куски HTML в отдельные файлы);
- реализовать поддержку синтаксических конструкций вроде `if`'ов и циклов в HTML



Классические веб-приложения

Ключевой минус классических веб-приложений - backend-разработчик должен заниматься HTML.

Т.е. именно вы, помимо написания кода на Java и работы с БД, должны ещё формировать эти самые HTML-странички.



Frontend



JS

В современном мире всё немного поменялось. JS (JavaScript) стал полноценным языком, который позволяет писать приложения, работающие в браузере*.

*На самом деле, JS можно запускать уже везде.

JS

В чём суть:

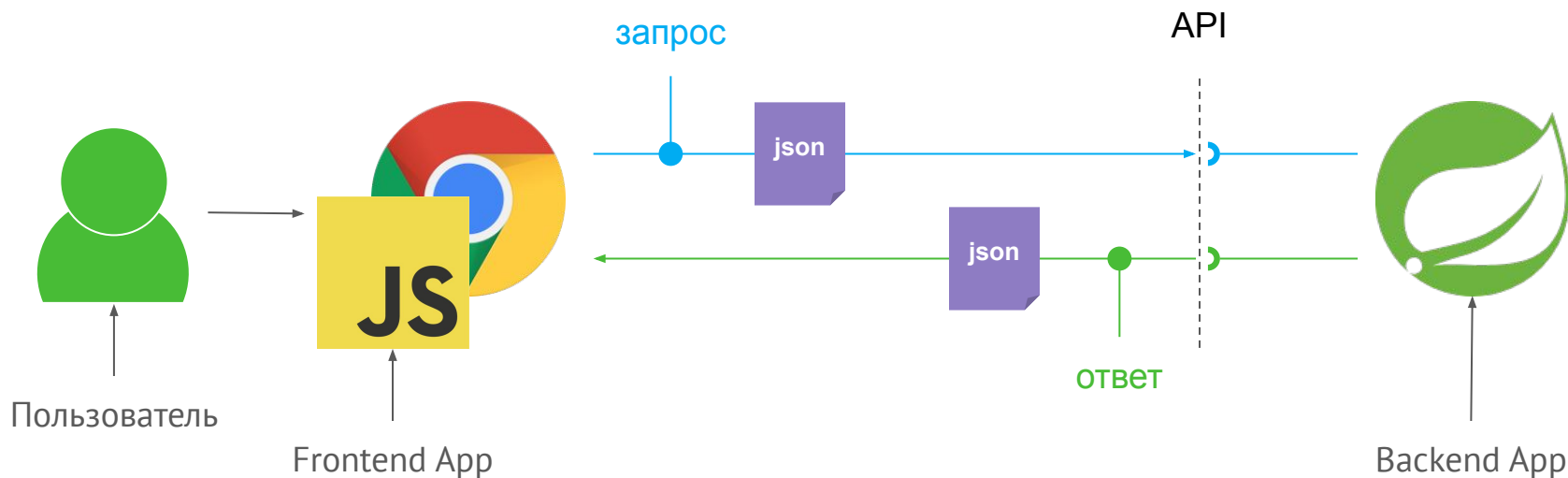
- браузер содержит реализацию движка JS (как JVM для Java);
- браузер предоставляет Web API - набор интерфейсов, позволяющих создавать полноценные приложения

Web API

1. DOM API - взаимодействие с UI (создание/изменение элементов, реагирование на события)
2. XMLHttpRequest/fetch - отправка HTTP-запросов
3. FileReader API - работа с файлами
4. Media Capture and Streams (запись аудио/видео)
5. и др.

JS

Таким образом, становится возможным сделать главное - разделить разработку, определив чёткий интерфейс взаимодействия:





Общая схема

Теперь frontend-программисты пишут свою часть приложения, а backend-программисты - свою.

Поскольку между сторонами (за исключением первой загрузки документа и загрузки файлов) происходит посредством JSON*, то backend-программисту уже не нужно заниматься вопросами отображения информации.

*На самом деле, можно не только JSON, а что угодно.



Sandbox

Поскольку документы (HTML) и ресурсы (JS, CSS) загружаются из сети Интернет, они априори не могут считаться доверенными.

Поэтому браузер ограничивает JS с точки зрения предоставляемых возможностей: например, из JS можно прочитать только те файлы, которые пользователь сам выбрал с помощью элемента выбора файлов или перенёс (Drag & Drop) в окно браузера.



Sandbox

Это ключевая вещь, из-за которой мы рассматриваем эти темы - понимание ограничений позволит вам создавать удобные API.

Ведь какой смысл писать API, которое не смогут использовать frontend-разработчики из-за ограничений браузера?

Разбору возможных форматов передачи данных и ограничений клиента и будет посвящены несколько наших следующих лекций.



Итоги

Итоги

Сегодня мы кратко "прошлись" по основам веб-технологий, для того, чтобы вынести две ключевых мысли:

1. Современная разработка делится на frontend и backend
2. Frontend часто ограничен в доступных инструментах

В связи с этим, мы будем концентрироваться именно на разработке API (которое также учитывает ограничения frontend'а), а не классических приложений.



Домашнее задание

Давайте посмотрим ваше [домашнее задание](#).

- Вопросы по домашней работе задавайте **в чате** мессенджера .
- Задачи можно сдавать **по частям**.
- Зачёт по домашней работе проставляется после того, как **приняты все задачи**.

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

Григорий Вахмистров