

## **ПМЗ Разработка модулей ПО.**

**РО 3.2 Разрабатывать модули с применением DOM API,  
Regexp, HTTP**

**Тема 1. Протокол HTTP.**

**Лекция 2. Основы HTTP: HTTP/1.0 и  
HTTP/1.1.**

# Цель занятия:

Сформировать представление о принципах работы протокола HTTP, научиться различать версии HTTP/1.0 и HTTP/1.1, понимать структуру HTTP-запросов и ответов, применять знания на практике при работе с простыми запросами.

# **Учебные вопросы:**

- 1. Структура HTTP-сообщений.**
- 2. Отличия HTTP/1.0 от HTTP/1.1**
- 3. Практическое применение.**

# 1. Структура HTTP-сообщений.

HTTP-сообщения бывают двух типов:

- Запрос (Request) → от клиента к серверу.
- Ответ (Response) → от сервера клиенту.

# **Формат HTTP-запроса**

## **1. Стартовая строка**

**<Метод> <Путь к ресурсу> <Версия протокола>**

**Пример:**

**GET /index.html HTTP/1.1**

## 2. Заголовки (Headers)

Передают служебную информацию.

Примеры:

- Host: www.example.com — адрес сервера.
- User-Agent: Chrome/117 — информация о клиенте.
- Accept: text/html — какие форматы данных клиент готов принять.

3. Пустая строка (разделитель).

#### **4. Тело запроса (Body) (необязательно)**

- Обычно присутствует при методах POST, PUT.
- Содержит данные формы, JSON, XML и т. д.

# Формат HTTP-ответа

## 1. Стартовая строка

<Версия протокола> <Статус-код> <Пояснение>

Пример:

HTTP/1.1 200 OK

## 2. Заголовки (Headers)

Примеры:

Content-Type: text/html — тип возвращаемых данных.

Content-Length: 512 — длина тела ответа в байтах.

Set-Cookie: session\_id=12345 — установка cookies.

**3. Пустая строка (разделитель).**

**4. Тело ответа (Body)**

Содержит запрошенный ресурс (HTML-страница, JSON, изображение и т. д.).

## Методы HTTP

HTTP определяет множество методов, каждый со своей задачей:

- GET. Запрос ресурса (страницы, данных). Тело запроса обычно отсутствует.
- POST. Отправка данных на сервер (формы, JSON, файлы). Использует тело запроса.
- HEAD. Аналог GET, но без тела ответа (полезно для проверки заголовков/метаданных).
- PUT. Полная замена ресурса на сервере.
- DELETE. Удаление ресурса на сервере.
- OPTIONS. Запрос поддерживаемых сервером методов.
- PATCH. Частичное обновление ресурса.

# **Основные методы GET и POST.**

## **GET**

- Параметры передаются в URL (`?key=value`).
- Ограничение длины запроса.
- Используется для получения данных.
- Запросы кэшируются браузером.

## **POST**

- Данные передаются в теле запроса.
- Нет строгого ограничения на объём (кроме серверных настроек).
- Используется для отправки форм, загрузки файлов, API-запросов.

## Примеры:

```
C:\Users\user>curl -v http://microoft.com
*   Trying 20.70.246.20:80...
* Connected to microoft.com (20.70.246.20) port 80 (#0)
> GET / HTTP/1.1
> Host: microoft.com
> User-Agent: curl/7.83.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 301 Moved Permanently
< Content-Length: 0
< Date: Sun, 07 Sep 2025 12:32:01 GMT
< Server: Kestrel
< Location: http://www.microsoft.com/
< Strict-Transport-Security: max-age=31536000
<
* Connection #0 to host microoft.com left intact

C:\Users\user>
```

## Примеры: Cookie

```
C:\Users\user>curl -v http://good73.net
*   Trying 109.197.199.91:80...
*   Connected to good73.net (109.197.199.91) port 80 (#0)
> GET / HTTP/1.1
> Host: good73.net
> User-Agent: curl/7.83.1
> Accept: */*
>
*   Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Server: nginx/1.5.2
< Date: Sun, 07 Sep 2025 10:28:12 GMT
< Content-Type: text/html; charset=windows-1251
< Transfer-Encoding: chunked
< Connection: keep-alive
< Keep-Alive: timeout=30
< X-Powered-By: PHP/5.2.17
< Set-Cookie: PHPSESSID=m32p2kj9v9s42t5lk9tb7k39r1; path=/
< Expires: Thu, 10 Nov 1981 08:52:00 GMT
< Cache-Control: no-cache
< Pragma: no-cache
<
```

## Примеры: GET-запрос с параметрами.

```
C:\Users\user>curl -vk "https://httpbin.org/get?user=neo&city=zion"
*   Trying 54.166.11.78:443...
* Connected to httpbin.org (54.166.11.78) port 443 (#0)
* schannel: disabled automatic use of client certificate
* ALPN: offers http/1.1
* ALPN: server accepted http/1.1
> GET /get?user=neo&city=zion HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.83.1
> Accept: */*
```

```
< HTTP/1.1 200 OK
< Date: Sun, 07 Sep 2025 12:36:57 GMT
< Content-Type: application/json
< Content-Length: 315
< Connection: keep-alive
< Server: gunicorn/19.9.0
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
<
{
  "args": {
    "city": "zion",
    "user": "neo"
  },
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.83.1",
    "X-Amzn-Trace-Id": "Root=1-68bd7c66-14462e630a30d804684028ea"
  },
  "origin": "37.99.47.107",
  "url": "https://httpbin.org/get?user=neo&city=zion"
}
* Connection #0 to host httpbin.org left intact
```

# Примеры: POST-запрос

```
C:\Users\user>curl -vk -X POST -d "username=neo&password=trinity" https://httpbin.org/post
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 34.238.12.187:443...
* Connected to httpbin.org (34.238.12.187) port 443 (#0)
* schannel: disabled automatic use of client certificate
* ALPN: offers http/1.1
* ALPN: server accepted http/1.1
> POST /post HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.83.1
> Accept: */*
> Content-Length: 29
> Content-Type: application/x-www-form-urlencoded
<
```

```
< HTTP/1.1 200 OK
< Date: Sun, 07 Sep 2025 12:39:57 GMT
< Content-Type: application/json
< Content-Length: 458
< Connection: keep-alive
< Server: gunicorn/19.9.0
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
<
{
    "args": {},
    "data": "",
    "files": {},
    "form": {
        "password": "trinity",
        "username": "neo"
    },
    "headers": {
        "Accept": "*/*",
        "Content-Type": "application/x-www-form-urlencoded"
    }
}
```

 Краткий вывод:

- HTTP-сообщения имеют чёткую структуру: стартовая строка, заголовки, тело.
- Методы запроса определяют, что именно клиент хочет сделать с ресурсом.
- На практике чаще всего применяются GET (получение данных) и POST (отправка данных).

## **2. Отличия HTTP/1.0 от HTTP/1.1.**

### **1. Поддержка постоянного соединения**

#### **HTTP/1.0:**

- По умолчанию после каждого запроса соединение закрывается.
- Для удержания соединения нужно явно указывать заголовок Connection: keep-alive.

#### **HTTP/1.1:**

- Постоянные соединения стали стандартом (keep-alive по умолчанию).
- Это уменьшило накладные расходы на установку TCP-сессий.

## 2. Обработка нескольких запросов

**HTTP/1.0:** только один запрос за соединение.

**HTTP/1.1:** появилась поддержка pipeline — клиент может отправлять несколько запросов подряд без ожидания ответа на каждый (хотя реально в браузерах почти не применяется из-за проблем с блокировкой ответов).

### **3. Заголовки и новые возможности**

**HTTP/1.1** ввёл множество новых заголовков:

- Host — обязателен (позволяет обслуживать несколько сайтов на одном IP-адресе → виртуальный хостинг).
- Cache-Control, ETag, If-Modified-Since, Transfer-Encoding, Range и др.
- В HTTP/1.0 многие из этих возможностей отсутствовали.

## 4. Передача данных

HTTP/1.0: размер тела ответа определялся только по заголовку Content-Length.

HTTP/1.1: появилась поддержка chunked transfer encoding (данные передаются частями, удобно для динамически генерируемого контента).

## **Код состояния**

**HTTP/1.0:** ограниченный набор статус-кодов.

**HTTP/1.1:** расширен список (например, 100 Continue, 206 Partial Content, 409 Conflict, 410 Gone и др.).

## 6. Оптимизация работы с ресурсами

HTTP/1.1: возможность запрашивать часть ресурса (Range requests) — удобно для докачки файлов, потокового видео.

В HTTP/1.0 этого не было.

## 6. Оптимизация работы с ресурсами

HTTP/1.1: возможность запрашивать часть ресурса (Range requests) — удобно для докачки файлов, потокового видео.

В HTTP/1.0 этого не было.

# 3. Практическое применение

## Анализ HTTP-запросов в браузере

The screenshot shows the Network tab of the Chrome DevTools developer console. The tab bar includes Elements, Console, Sources, Network (which is selected), and a few others. Below the tabs are various controls like Preserve log, Disable cache, and No throttling. A filter bar allows for filtering by type: All, Fetch/XHR, Doc, CSS, JS, Font, Img, Media, Manifest, Socket, and Wasm. The main area displays a timeline with four segments: 50 ms, 100 ms, 150 ms, and 200 ms. A long green horizontal bar spans across these segments, indicating the duration of a single network request. At the bottom, a detailed view of the request to 'example.com' is shown, with tabs for Name, Headers, Preview, and Response. The Headers tab is active, showing the Request URL as 'https://example.co/m/' and the Request Method as 'GET'.

Name	Headers	Preview	Response
example.com	<b>General</b>		
	Request URL	https://example.co/m/	
	Request Method	GET	

# GET-запрос через curl

```
C:\Users\user>curl -v "http://httpbin.org/get?user=neo&city=zion"
*   Trying 13.222.46.84:80...
*   Connected to httpbin.org (13.222.46.84) port 80 (#0)
> GET /get?user=neo&city=zion HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.83.1
> Accept: */*
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
< Date: Sun, 07 Sep 2025 13:51:46 GMT
< Content-Type: application/json
< Content-Length: 314
< Connection: keep-alive
< Server: gunicorn/19.9.0
< Access-Control-Allow-Origin: *
< Access-Control-Allow-Credentials: true
<
{
  "args": {
    "city": "zion",
    "user": "neo"
  },
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Host": "httpbin.org",
    "User-Agent": "curl/7.83.1"
  },
  "method": "GET",
  "url": "http://httpbin.org/get?user=neo&city=zion"
}
```

# POST-запрос через curl

```
C:\Users\user>curl -v -X POST -d "username=neo&password=trinity" http://httpbin.org/post
Note: Unnecessary use of -X or --request, POST is already inferred.
*   Trying 54.166.11.78:80...
* Connected to httpbin.org (54.166.11.78) port 80 (#0)
> POST /post HTTP/1.1
> Host: httpbin.org
> User-Agent: curl/7.83.1
> Accept: */*
> Content-Length: 29
> Content-Type: application/x-www-form-urlencoded
>
* Mark bundle as not supporting multiuse
< HTTP/1.1 200 OK
```

# Работа в hoppscotch.io (веб-аналог Postman)

The screenshot shows the hoppscotch.io web application interface. At the top, the address bar displays 'hoppscotch.io' and the title 'Hoppscotch • Open source API development ecosystem • Hoppscotch'. The main interface is titled 'HOPPSCOTCH' and shows a 'POST Untitled' request. The request URL is set to 'https://httpbin.org/post'. Below the URL, there are tabs for 'Параметры' (Parameters), 'Тело' (Body), 'Заголовки' (Headers), 'Авторизация' (Authorization), 'Пред-скрипт' (Pre-request Script), and 'Post-request Script'. The 'Тело' tab is selected, showing the body type as 'application/json' and a button to 'Переопределить' (Override). The body content is a JSON object:

```
1 {  
2   "name": "Neo",  
3   "role": "The One"  
4 }
```

At the bottom of the request panel, status information is displayed: 'Статус: 200 • OK', 'Время: 949 ms', and 'Размер: 1.27 KB'. The overall layout is clean and follows the design of the Postman application.

<https://hoppscotch.io/>

## **Контрольные вопросы:**

- Из каких частей состоит HTTP-запрос?
- Какие основные методы HTTP вы знаете? В чём разница между GET и POST?
- Из каких частей состоит HTTP-ответ? Что означает статус-код?
- В чём основные отличия HTTP/1.0 от HTTP/1.1?

# **Домашнее задание:**

1. [https://ru.hexlet.io/courses/http\\_protocol](https://ru.hexlet.io/courses/http_protocol)