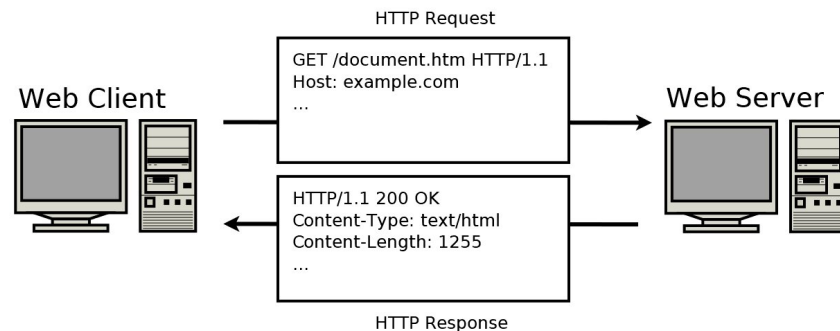


Часть 1. Теоретическая.



HTTP servers

https://developer.mozilla.org/ru/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server

HTTP

<https://developer.mozilla.org/ru/docs/Web/HTTP/Overview>

HTTP 1.1

<http://www.lib.ru/WEBMASTER/rfc2068/>

Заголовки HTTP

<https://habr.com/ru/post/412813/>

Сообщения HTTP

<https://developer.mozilla.org/ru/docs/Web/HTTP/Messages>

HTTP сессия

<https://developer.mozilla.org/ru/docs/Web/HTTP/Session>

XML

https://developer.mozilla.org/ru/docs/Web/XML/XML_introduction

JSON

https://developer.mozilla.org/ru/docs/Web/JavaScript/Reference/Global_Objects/JSON

MIME-типы

https://developer.mozilla.org/ru/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types

Часть 2. Практическая.

Рассмотрим модель простейшего сервера, который может обрабатывать запросы от нескольких клиентов.

```
package iit.mal.wt23.lab02;

import ...

public class LocalHttpServer {

    public static final int RESP_LENGTH = 400;

    private final ExecutorService pool;
    private final int port;
    private boolean stopped;

    public LocalHttpServer(int port, int poolSize) {
        this.port = port;
        this.pool = Executors.newFixedThreadPool(poolSize);
    }

    public void run() {...}

    private void processSocket(Socket socket) {
        try (socket;
            var inputStream = new DataInputStream(socket.getInputStream());
            var outputStream = new DataOutputStream(socket.getOutputStream())) {...} catch (IOException | InterruptedException e) {...}
    }

    public void setStopped(boolean stopped) { this.stopped = stopped; }
}
```

```
public void run() {
    try {
        var server = new ServerSocket(port);
        while (!stopped) {
            var socket :Socket = server.accept();
            System.out.println("Socket accepted");
            pool.submit(() -> processSocket(socket));
        }
    } catch (IOException e) {
        throw new RuntimeException(e);
    }
}
```

```
private void processSocket(Socket socket) {
    try (socket;
        var inputStream = new DataInputStream(socket.getInputStream());
        var outputStream = new DataOutputStream(socket.getOutputStream())) {
        step 1 handle request
        System.out.println("Request: " + new String(inputStream.readNBytes(RESP_LENGTH)));

        Thread.sleep( millis: 10000);
        step 2 handle response
        var body :byte[] = Files.readAllBytes(Path.of( first: "resources", ...more: "example.html"));
        var headers :byte[] = """
            HTTP/1.1 200 OK
            content-type: text/html
            content-length: %s
            """.formatted(body.length).getBytes();
        outputStream.write(headers);
        outputStream.write(System.lineSeparator().getBytes());
        outputStream.write(body);
    } catch (IOException | InterruptedException e) {
        // TODO: log error message
        e.printStackTrace();
    }
}
```

В приведённом ниже примере, один клиент передаёт три запроса асинхронно.

```
package iit.mal.wt23.lab02;

import ...

public class httpClient {

    public static void main(String[] args) throws IOException, InterruptedException, ExecutionException {
        var httpClient : HttpClient = HttpClient.newBuilder()
            .version(HttpClient.Version.HTTP_1_1)
            .build();

        var request : HttpRequest = HttpRequest.newBuilder()
            .uri(URI.create("http://localhost:9000"))
            .header( name: "content-type", value: "application/json")
            .POST(ofFile(Path.of( first: "resources", ...more: "example.json")))
            .build();

        var response : CompletableFuture<HttpResponse<String>> = httpClient.sendAsync(request, HttpResponse.BodyHandlers.ofString());
        var response2 : CompletableFuture<HttpResponse<String>> = httpClient.sendAsync(request, HttpResponse.BodyHandlers.ofString());
        var response3 : CompletableFuture<HttpResponse<String>> = httpClient.sendAsync(request, HttpResponse.BodyHandlers.ofString());

        //...

        System.out.println(response3.get().body());
    }
}
```

После запуска сервера получаем сообщение:

```
Server started
```

После запуска клиента в окне сервера получаем следующие сообщения:

```
Server started
Socket accepted
Socket accepted
Socket accepted
Request: POST / HTTP/1.1
Content-Length: 326
Host: localhost:9000
User-Agent: Java-http-client/18
content-type: application/json

{
  "id": 25,
  "startRow": {
    "method": "GET",
    "url": "www.google.com",
    "version": 1.1
  },
  "headers": [
    {
```

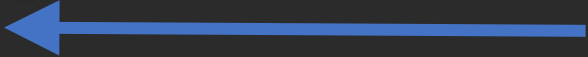
В это же время на клиенте отображаем наш ответ от сервера:

```
<!DOCTYPE html>

<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Title</title>
  <script src="path/to/script.js"></script>
  <script src="path/to/script2.js"></script>
  <link rel="stylesheet" href="path/to/simple.css"/>
</head>
<body>
<span>Hello World!</span>
<div>
  <form action="/registration" method="post">
    <label>
      Username:
      <input type="text" name="username">
    </label>
    <br/>
```

В ответе сервера мы получили цифру **326** о чём нам говорит это значение?

```
Request: POST / HTTP/1.1
Content-Length: 326
Host: localhost:9000
User-Agent: Java-http-client/18
content-type: application/json
```



* значения, полученные вами, могут отличаться от примера.

Задание:

Необходимо разработать простейший веб сервер, который будет:

- < иметь возможность обработки HTTP запросов с версиями 1.0 и 1.1;
- < в соответствии с версией, указанной в заголовке запроса, должна производиться его обработка и возвращаться ответ;
- < работать с несколькими заголовками;
- < работать с форматами данных: XML, JSON, HTML;
- < иметь некоторое ограниченное количество статусов, которые он должен возвращать. (200 ok, 404 not found и т.д.);
- < при невозможности обработать запрос от клиента, сервер должен выдать соответствующее сообщение с кодом ошибки и поместить запись об

ошибке в текстовый файл с логом состояний и запросов, при этом работа сервера прерываться не должна;

- ⟨ работать набором клиентов (2 и более), которые отправляют на сервер запросы в разных форматах на получение информации;
- ⟨ иметь возможность подключения по протоколу telnet с указанием параметров запроса в окне терминала.

Часть 3. Контрольные вопросы.

По результатам выполнения лабораторной работы преподаватель надеется, что студент получил достаточные теоретические и практические знания и в состоянии ответить на следующие вопросы:

1. Что такое HTTP? Какие версии и отличия вы знаете?
2. Что такое сообщения HTTP-запроса?
3. Какая структура у HTTP заголовка?
4. Чем отличаются методы GET, POST и HEAD?
5. Что такое состояние сеанса в HTTP?
6. Что вы понимаете под HTTP-запросом и HTTP-ответом? (приведите примеры)
7. Что такое Web Server? Какие веб серверы Вы знаете и чем они отличаются?
8. Что такое Apache Tomcat?
9. Что такое MIME-type? Какие типы вы знаете?
10. Что такое JSON его отличие от XML?
11. Что такое JSON SCHEMA?
12. Какие есть уровни модели OSI?

Для защиты лабораторной работы необходимо:

- < Прочитать и понять теоретическую часть.
- < **Индивидуально** выполнить все задания (теоретические и практические) согласно варианта - если такой присутствует. Принимается первый уникальный код, все остальные признаются плагиатом или репликами первого и не учитываются.
- < Понимать, что делает каждая строка написанного кода.
- < Оформить отчёт, который должен быть распечатан или написан вручную на листах формата А4 (цвет листов не важен) и содержать титульный лист и ход выполнения работы. Выводы о результатах проделанной работы приветствуются.
 - * использование репозитория (git или аналоги) с открытым доступом приветствуется.