Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ
И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Предмет: операционные системы и среды

Отчёт по курсовой работе

По теме “Учебные или альтернативные версии серверов сетевых служб”

Выполнил:
студент гр. 853505
Мирончик А.Э.
Проверил:
Ассистент КИ Протько М. И.

Минск 2021

Содержание

1. Постановка задачи

Цель – написать свою простую имплементацию HTTP сервера с поддержкой метода GET. Для этого был выбран язык программирования Java

2. Теория

HTTP (англ. *HyperText Transfer Protocol* — «протокол передачи гипертекста») — протокол прикладного уровня передачи данных, изначально — в виде гипертекстовых документов в формате HTML, в настоящее время используется для передачи произвольных данных.

Основой HTTP является технология «клиент-сервер», то есть предполагается существование:

- Потребителей (клиентов), которые инициируют соединение и посылают запрос;
- Поставщиков (серверов), которые ожидают соединения для получения запроса, производят необходимые действия и возвращают обратно сообщение с результатом.

Программное обеспечение

Всё программное обеспечение для работы с протоколом HTTP разделяется на три большие категории:

- Серверы как основные поставщики услуг хранения и обработки информации (обработка запросов);
- Клиенты — конечные потребители услуг сервера (отправка запроса);
- Прокси (посредники) для выполнения транспортных служб.

Для отличия конечных серверов от прокси в официальной документации используется термин «исходный сервер» (англ. *origin server*). Один и тот же программный продукт может одновременно выполнять функции клиента, сервера или посредника в зависимости от поставленных задач. В спецификациях протокола HTTP подробно описывается поведение для каждой из этих ролей.

Структура HTTP-сообщения

Каждое HTTP-сообщение состоит из трёх частей, которые передаются в указанном порядке:

1. Стартовая строка (англ. *Starting line*) — определяет тип сообщения;

2. Заголовки (англ. *Headers*) — характеризуют тело сообщения, параметры передачи и прочие сведения;
3. Тело сообщения (англ. *Message Body*) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой.

Тело сообщения может отсутствовать, но стартовая строка и заголовок являются обязательными элементами. Исключением является версия 0.9 протокола, у которой сообщение запроса содержит только стартовую строку, а сообщения ответа — только тело сообщения.

Для версии протокола 1.1 сообщение запроса обязательно должно содержать заголовок *Host.*

Стартовая строка

Стартовые строки различаются для запроса и ответа. Строка запроса выглядит так:

`GET URI` — для версии протокола 0.9;

`Метод URI HTTP/Версия` — для остальных версий.

Здесь:

- Метод (англ. *Method*) — тип запроса, одно слово заглавными буквами. В версии HTTP 0.9 использовался только метод `GET`, список методов для версии 1.1 представлен ниже.
- URI определяет путь к запрашиваемому документу.
- Версия (англ. *Version*) — пара разделённых точкой цифр. Например: `1.0`.

Пример запроса:

```
GET /wiki/HTTP HTTP/1.0
Host: ru.wikipedia.org
```

Стартовая строка ответа сервера имеет следующий формат: `HTTP/Версия КодСостояния Пояснение`, где:

- Версия — пара разделённых точкой цифр, как в запросе;
- Код состояния (англ. *Status Code*) — три цифры. По коду состояния определяется дальнейшее содержимое сообщения и поведение клиента;
- Пояснение (англ. *Reason Phrase*) — текстовое короткое пояснение к коду ответа для пользователя. Никак не влияет на сообщение и является необязательным.

Пример ответа сервера:

```
HTTP/1.0 200 OK
```

Методы

Метод HTTP (англ. *HTTP Method*) — последовательность из любых символов, кроме управляющих и разделителей, указывающая на основную операцию над ресурсом. Обычно метод представляет собой короткое английское слово, записанное заглавными буквами. Название метода чувствительно к регистру!

Сервер может использовать любые методы, не существует обязательных методов для сервера или клиента.

Если сервер не распознал указанный клиентом метод, то он должен вернуть статус `501` (Not Implemented). Если серверу метод известен, но он неприменим к конкретному ресурсу, то возвращается сообщение с кодом `405` (Method Not Allowed). В обоих случаях серверу следует включить в сообщение ответа заголовок `Allow` со списком поддерживаемых методов.

Кроме методов `GET` и `HEAD`, часто применяется метод `POST`.

GET

Используется для запроса содержимого указанного ресурса. С помощью метода `GET` можно также начать какой-либо процесс. В этом случае в тело ответного сообщения следует включить информацию о ходе выполнения процесса.

Клиент может передавать параметры выполнения запроса в URI целевого ресурса после символа «`?`»:
`GET /path/resource?param1=value1&param2=value2 HTTP/1.1`

Согласно стандарту HTTP, запросы типа `GET` считаются идемпотентными, то есть многократное повторение одних и тех же запросов `GET` даст один и тот же результат.

Кроме обычного метода `GET`, различают ещё

- Условный `GET` — содержит заголовки `If-Modified-Since`, `If-Match`, `If-Range` и подобные;
- Частичный `GET` — содержит в запросе `Range`.

Порядок выполнения подобных запросов определён стандартами отдельно.

HEAD

Аналогичен методу `GET`, за исключением того, что в ответе сервера отсутствует тело. Запрос `HEAD` обычно применяется для

извлечения метаданных, проверки наличия ресурса (валидация URL) и чтобы узнать, не изменился ли он с момента последнего обращения.

Заголовки ответа могут кэшироваться. При несовпадении метаданных ресурса с соответствующей информацией в кэше — копия ресурса помечается как устаревшая.

POST

Применяется для передачи пользовательских данных заданному ресурсу.

Например, в блогах посетители обычно могут вводить свои комментарии к записям в HTML-форму, после чего они передаются серверу методом POST и он помещает их на страницу. При этом передаваемые данные (в примере с блогами — текст комментария) включаются в тело запроса. Аналогично с помощью метода `POST` обычно загружаются файлы на сервер.

В отличие от метода `GET`, метод `POST` не считается идемпотентным, то есть многократное повторение одних и тех же запросов `POST` может возвращать разные результаты (например, после каждой отправки комментария будет появляться очередная копия этого комментария).

При результате выполнения `200` (Ok) в тело ответа следует включить сообщение об итоге выполнения запроса. Если был создан ресурс, то серверу следует вернуть ответ `201` (Created) с указанием URI нового ресурса в заголовке `Location`.

Сообщение ответа сервера на выполнение метода `POST` не кэшируется.

Коды состояния

Код состояния является частью первой строки ответа сервера. Он представляет собой целое число из трёх цифр. Первая цифра указывает на класс состояния. За кодом ответа обычно следует отделённая пробелом поясняющая фраза на английском языке, которая разъясняет человеку причину именно такого ответа. Примеры:

```
201 Webpage Created
404 Page Not Found
507 Insufficient Storage
```

Клиент узнаёт по коду ответа о результатах его запроса и определяет, какие действия ему предпринимать дальше. Набор кодов состояния является стандартом, и они описаны в соответствующих документах.

Заголовки

Заголовки HTTP (англ. *HTTP Headers*) — это строки в HTTP-сообщении, содержащие разделённую двоеточием пару параметр-значение. Формат заголовков соответствует общему формату заголовков текстовых сетевых

сообщений ARPA. Заголовки должны отделяться от тела сообщения хотя бы одной пустой строкой.

Все заголовки разделяются на четыре основных группы:

1. General Headers («Основные заголовки») — могут включаться в любое сообщение клиента и сервера;
2. Request Headers («Заголовки запроса») — используются только в запросах клиента;
3. Response Headers («Заголовки ответа») — только для ответов от сервера;
4. Entity Headers («Заголовки сущности») — сопровождают каждую сущность сообщения.

Именно в таком порядке рекомендуется посылать заголовки получателю.

Тело сообщения

Тело HTTP-сообщения (`message-body`), если оно присутствует, используется для передачи тела объекта, связанного с запросом или ответом. Тело сообщения отличается от тела объекта (`entity-body`) только в том случае, когда применяется кодирование передачи, что указывается полем заголовка `Transfer-Encoding`.

```
message-body = entity-body
| <entity-body закодировано согласно
Transfer-Encoding>
```

Поле `Transfer-Encoding` должно использоваться для указания любого кодирования передачи, применённого приложением в целях гарантирования безопасной и правильной передачи сообщения. Поле `Transfer-Encoding` — это свойство сообщения, а не объекта, и, таким образом, может быть добавлено или удалено любым приложением в цепочке запросов/ответов.

Правила, устанавливающие допустимость тела сообщения в сообщении, отличны для запросов и ответов.

Присутствие тела сообщения в запросе отмечается добавлением к заголовкам запроса поля заголовка `Content-Length` или `Transfer-Encoding`. Тело сообщения может быть добавлено в запрос, только когда метод запроса допускает тело объекта.

Включается или не включается тело сообщения в сообщение ответа — зависит как от метода запроса, так и от кода состояния ответа. Все ответы на запрос с методом `HEAD` не должны включать тело сообщения, даже если присутствуют поля заголовка объекта (`entity-header`), заставляющие поверить в присутствие объекта. Никакие ответы с кодами

состояния `1xx` (Информационные), `204` (Нет содержимого, No Content), и `304` (Не модифицирован, Not Modified) не должны содержать тела сообщения. Все другие ответы содержат тело сообщения, даже если оно имеет нулевую длину.

3. Реализация

План:

1. Открыть TCP сокет и слушать
2. Принять клиента и прочитать запрос
3. Распарсить запрос
4. Найти запрашиваемый ресурс на диске
5. Отправить ответ

Открыть TCP сокет

Будем использовать класс ServerSocket для обработки TCP соединения.

```
private static final int PORT_NUMBER = 8080;

public static void main( String[] args ) throws Exception {
    ServerSocket serverSocket = null;

    try {
        serverSocket = new ServerSocket(PORT_NUMBER);
    }
    catch (BindException ex)
    {
        System.out.println("Port " + PORT_NUMBER + " is already in use");
        System.out.println(ex.getMessage());
    }
}
```

Принять клиента

```
while (true) {
    try (Socket client = serverSocket.accept()) {
        handleClient(client);
    }
}
```

Чтобы принять соединение от клиента, вызываем метод блокировки accept (). Программа будет ждать клиента на этой линии.

Обработчик клиента и чтение запроса

```
private static void handleClient(Socket client) throws IOException {
    BufferedReader br = new BufferedReader(new
InputStreamReader(client.getInputStream()));
```

```
    StringBuilder requestBuilder = new StringBuilder();
    String line;
    while (!(line = br.readLine()).isBlank()) {
        requestBuilder.append(line + "\r\n");
    }

    String path = parseTheRequest(client, requestBuilder);
}
```

Запрос заканчивается одной пустой строкой (\ r \ n). Клиент отправит пустую строку, но imputStream будет по-прежнему открыт, мы должны читать его, пока не появится одна пустая строка.

Парсинг запроса

```
private static String parseTheRequest(Socket client, StringBuilder requestBuilder){
    String request = requestBuilder.toString();
    String[] requestsLines = request.split("\r\n");
    String[] requestLine = requestsLines[0].split(" ");
    String method = requestLine[0];
    String path = requestLine[1];
    String version = requestLine[2];
    String host = requestsLines[1].split(" ")[1];

    List<String> headers = new ArrayList<>();
    for (int h = 2; h < requestsLines.length; h++) {
        String header = requestsLines[h];
        headers.add(header);
    }

    String accessLog = String.format("Client %s, method %s, path %s, version %s, host
%s, headers %s",
            client.toString(), method, path, version, host, headers.toString());
    System.out.println(accessLog);

    return path;
}
```

Поскольку первая строка (индекс 0) - это GET / HTTP / 1.1, вторая строка - это хост. Заголовки начинаются с третьей строки запроса, поэтому начинаем цикл с 2.

Найти запрашиваемый ресурс на диске

```
Path filePath = getFilePath(path);
if (Files.exists(filePath)) {
    // file exist
    String contentType = guessContentType(filePath);
    sendResponse(client, "200 OK", contentType, Files.readAllBytes(filePath));
} else {
    // 404
    byte[] notFoundContent = "<h1>404 Page Not found :O</h1>".getBytes();
    sendResponse(client, "404 Not Found", "text/html", notFoundContent);
}
```

```
private static Path getFilePath(String path) {
    if (path.equals("/")) {
        path = "/index.html";
    }

    return Paths.get("F:\\University\\sixth-semester\\course2", path);
}

private static String guessContentType(Path filePath) throws IOException {
    return Files.probeContentType(filePath);
}
```

guessContentType - мы должны сообщить браузеру, какой контент мы отправляем. Для этого в Java есть встроенные механизмы. Нам не нужно делать большой switch блок.

getFilePath - прежде чем мы вернем файл, нам нужно знать его местонахождение.

Если пользователю нужен ресурс по умолчанию - возвращаем index.html.

Отправить ответ

```
private static void sendResponse(Socket client, String status, String contentType,
byte[] content) throws IOException {
    OutputStream clientOutput = client.getOutputStream();
    clientOutput.write(("HTTP/1.1 \r\n" + status).getBytes());
    clientOutput.write(("ContentType: " + contentType + "\r\n").getBytes());
    clientOutput.write("\r\n".getBytes());
    clientOutput.write(content);
    clientOutput.write("\r\n\r\n".getBytes());
    clientOutput.flush();
    client.close();
}
```

4. Тестирование

Вспомогательные html страницы:

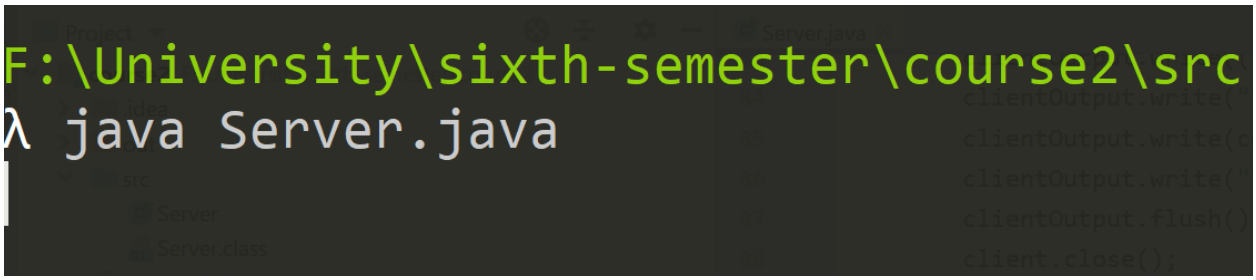
index.html

```
<html>
  <header>
    <title>My homepage!</title>
  </header>
  <body>
    <h1>Welcome!</h1>
    <p><a href="gallery.html">Here</a> you can look at my pictures</p>
  </body>
</html>
```

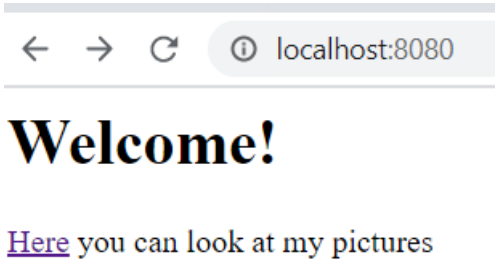
gallery.html

```
<html>
  <head>
    <title>Gallery</title>
  </head>
  <body>
    <h1>My sexi photos<h1>
    <img src="photo.jpg" />
  </body>
</html>
```

Запуск сервера



Браузер



Вывод в консоль:

Client Socket[addr=/0:0:0:0:0:0:0:1,port=50944,localport=8080], method GET, path /, version HTTP/1.1, host localhost:8080, headers [Connection: keep-alive, Cache-Control: max-age=0, sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90", "Google Chrome";v="90", sec-ch-ua-mobile: ?0, Upgrade-Insecure-Requests: 1, User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36, Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9, Sec-Fetch-Site: none, Sec-Fetch-Mode: navigate, Sec-Fetch-User: ?1, Sec-Fetch-Dest: document, Accept-Encoding: gzip, deflate, br, Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7]
Client Socket[addr=/0:0:0:0:0:0:0:1,port=50945,localport=8080], method GET, path /favicon.ico, version HTTP/1.1, host localhost:8080, headers [Connection: keep-alive, Pragma: no-cache, Cache-Control: no-cache, sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90", "Google Chrome";v="90", sec-ch-ua-mobile: ?0, User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36, Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8, Sec-Fetch-Site: same-origin, Sec-Fetch-Mode: no-cors, Sec-Fetch-Dest: image, Referer: http://localhost:8080/, Accept-Encoding: gzip, deflate, br, Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7]

Вывод в консоль:

Client Socket[addr=/0:0:0:0:0:0:0:1,port=50946,localport=8080], method GET, path /gallery.html, version HTTP/1.1, host localhost:8080, headers [Connection: keep-alive, sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90", "Google Chrome";v="90", sec-ch-ua-mobile: ?0, Upgrade-Insecure-Requests: 1, User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36, Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9, Sec-Fetch-Site: same-origin, Sec-Fetch-Mode: navigate, Sec-Fetch-User: ?1, Sec-Fetch-Dest: document, Referer: http://localhost:8080/, Accept-Encoding: gzip, deflate, br, Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7]
Client Socket[addr=/0:0:0:0:0:0:0:1,port=50947,localport=8080], method GET, path /photo.jpg, version HTTP/1.1, host localhost:8080, headers [Connection: keep-alive, sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90", "Google Chrome";v="90", sec-ch-ua-mobile: ?0, User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36, Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8, Sec-Fetch-Site: same-origin, Sec-Fetch-Mode: no-cors, Sec-Fetch-Dest: image, Referer: http://localhost:8080/gallery.html, Accept-Encoding: gzip, deflate, br, Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7]
Client Socket[addr=/0:0:0:0:0:0:0:1,port=50948,localport=8080], method GET, path /favicon.ico, version HTTP/1.1, host localhost:8080, headers [Connection: keep-alive, sec-ch-ua: " Not A;Brand";v="99", "Chromium";v="90", "Google Chrome";v="90", sec-ch-ua-mobile: ?0, User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36, Accept: image/avif,image/webp,image/apng,image/svg+xml,image/*,*/*;q=0.8, Sec-Fetch-Site: same-origin, Sec-Fetch-Mode: no-cors, Sec-Fetch-Dest: image, Referer: http://localhost:8080/gallery.html, Accept-Encoding: gzip, deflate, br, Accept-Language: ru-RU,ru;q=0.9,en-US;q=0.8,en;q=0.7]

5. Исходный код

```
import java.io.*;
import java.net.BindException;
import java.net.ServerSocket;
```

```java
import java.net.Socket;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;


public class Server {

    private static final int PORT_NUMBER = 8080;

    public static void main( String[] args ) throws Exception {
        ServerSocket serverSocket = null;

        try {
            serverSocket = new ServerSocket(PORT_NUMBER);
        }
        catch (BindException ex)
        {
            System.out.println("Port " + PORT_NUMBER + " is already in use");
            System.out.println(ex.getMessage());
        }
        while (true) {
            try (Socket client = serverSocket.accept()) {
                handleClient(client);
            }
        }
    }

    private static void handleClient(Socket client) throws IOException {
        BufferedReader br = new BufferedReader(new
InputStreamReader(client.getInputStream()));

        StringBuilder requestBuilder = new StringBuilder();
        String line;
        while (!(line = br.readLine()).isBlank()) {
            requestBuilder.append(line + "\r\n");
        }

        String path = parseTheRequest(client, requestBuilder);

        Path filePath = getFilePath(path);
        if (Files.exists(filePath)) {
            // file exist
            String contentType = guessContentType(filePath);
            sendResponse(client, "200 OK", contentType,
Files.readAllBytes(filePath));
        } else {
            // 404
            byte[] notFoundContent = "<h1>404 Page Not found :O</h1>".getBytes();
            sendResponse(client, "404 Not Found", "text/html", notFoundContent);
        }

    }

    private static String parseTheRequest(Socket client, StringBuilder
requestBuilder){
        String request = requestBuilder.toString();
        String[] requestsLines = request.split("\r\n");
        String[] requestLine = requestsLines[0].split(" ");
        String method = requestLine[0];
```

```
        String path = requestLine[1];
        String version = requestLine[2];
        String host = requestsLines[1].split(" ")[1];

        List<String> headers = new ArrayList<>();
        for (int h = 2; h < requestsLines.length; h++) {
            String header = requestsLines[h];
            headers.add(header);
        }

        String accessLog = String.format("Client %s, method %s, path %s, version %s,
host %s, headers %s",
                client.toString(), method, path, version, host, headers.toString());
        System.out.println(accessLog);

        return path;
    }

    private static void sendResponse(Socket client, String status, String
contentType, byte[] content) throws IOException {
        OutputStream clientOutput = client.getOutputStream();
        clientOutput.write(("HTTP/1.1 \r\n" + status).getBytes());
        clientOutput.write(("ContentType: " + contentType + "\r\n").getBytes());
        clientOutput.write("\r\n".getBytes());
        clientOutput.write(content);
        clientOutput.write("\r\n\r\n".getBytes());
        clientOutput.flush();
        client.close();
    }

    private static Path getFilePath(String path) {
        if (path.equals("/")) {
            path = "/index.html";
        }

        return Paths.get("F:\\University\\sixth-semester\\course2", path);
    }

    private static String guessContentType(Path filePath) throws IOException {
        return Files.probeContentType(filePath);
    }

}
```

6. Источники

https://ru.wikipedia.org/

https://www.youtube.com/