

Формы и форматы передачи данных



Григорий
Вахмистров



Григорий Вахмистров

Backend Developer в Tennisi.bet



План занятия

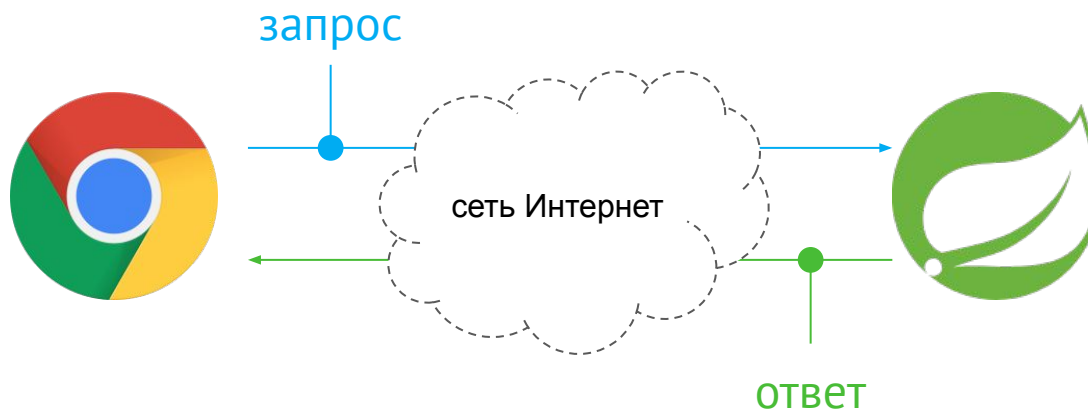
1. [Предисловие](#)
2. [Тестовый сервер](#)
3. [Формы](#)
4. [Итоги](#)
5. [Домашнее задание](#)



Предисловие

Предисловие

На прошлой лекции мы разобрали, что базовая схема взаимодействия выглядит вот так:



Сегодня наша задача рассмотреть, каким образом (в каком формате) передаются данные и как их можно обрабатывать на сервере.



Тестовый сервер



Тестовый сервер

Чтобы получать данные, которые нам присылает клиент в необработанном виде, мы "допишем" сервер.

Этот демо-сервер предназначен только для демонстрации приходящих запросов, поэтому в нём должным образом не обрабатываются ошибки, не структурирована функциональность и т.д.

Тестовый сервер

```
public static final String GET = "GET";
public static final String POST = "POST";

public static void main(String[] args) {
    final var allowedMethods :List<String> = List.of(GET, POST);

    try (final var serverSocket = new ServerSocket( port: 9999)) {
        while (true) {
            try (
                final var socket :Socket = serverSocket.accept();
                final var in = new BufferedInputStream(socket.getInputStream());
                final var out = new BufferedOutputStream(socket.getOutputStream());
            ) {...}
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```


Структура запроса

5 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
               CRLF
               [ message-body ]       ; Section 4.3
```

Request Line

```
// лимит на request line + заголовки
```

```
final var limit = 4096;
```

```
in.mark(limit);
```

```
final var buffer = new byte[limit];
```

```
final var read :int = in.read(buffer);
```

скоро увидим, что большинство production server'ов выставляет подобный лимит

```
// ищем request line
```

```
final var requestLineDelimiter = new byte[]{'\r', '\n'};
```

```
final var requestLineEnd :int = indexOf(buffer, requestLineDelimiter, start: 0, read);
```

```
if (requestLineEnd == -1) {
```

```
    badRequest(out);
```

```
    continue;
```

```
}
```

```
// читаем request line
```

```
final var requestLine :String[] = new String(Arrays.copyOf(buffer, requestLineEnd)).split( regex: " ");
```

```
if (requestLine.length != 3) {
```

```
    badRequest(out);
```

```
    continue;
```

```
}
```

Method & Path

```
final var method :String = requestLine[0];  
if (!allowedMethods.contains(method)) {  
    badRequest(out);  
    continue;  
}  
System.out.println(method);
```

```
final var path :String = requestLine[1];  
if (!path.startsWith("/")) {  
    badRequest(out);  
    continue;  
}  
System.out.println(path);
```

Заголовки

```
// отматываем на начало буфера
in.reset();
// пропускаем requestLine
in.skip(headersStart);

final var headersBytes : byte[] = in.readNBytes( len: headersEnd - headersStart);
final var headers : List<String> = Arrays.asList(new String(headersBytes).split( regex: "\r\n"));
System.out.println(headers);
```



Тело запроса

С телом запроса всё немного интереснее. Согласно спецификации есть специальный заголовок [Content-Length](#), в котором указывается размер тела*.

*В реальности всё немного сложнее: в некоторых случаях этот заголовок может не указываться, в некоторых - **не должен** указываться.

Тело запроса

```
// для GET тела нет
if (!method.equals(GET)) {
    in.skip(headersDelimiter.length);
    // вычитываем Content-Length, чтобы прочитать body
    final var contentLength : Optional<String> = extractHeader(headers, header: "Content-Length");
    if (contentLength.isPresent()) {
        final var length : int = Integer.parseInt(contentLength.get());
        final var bodyBytes : byte[] = in.readNBytes(length);

        final var body = new String(bodyBytes);
        System.out.println(body);
    }
}

out.write((
    "HTTP/1.1 200 OK\r\n" +
    "Content-Length: 0\r\n" +
    "Connection: close\r\n" +
    "\r\n"
).getBytes());
out.flush();
```



Формы

Формы



Мы начнем с обычных веб-формы без JS.

Q: почему именно с них? Ведь современные приложения используют JS.

A: потому что те механизмы, которые используются в формах, будут использоваться в том числе в JS.

IDEA

IDEA позволяет запускать встроенный веб-сервер для веб-страниц.

Для этого нужно навести в правый верхний угол страницы и выбрать один из браузеров:



Обратите внимание:

IDEA запустит веб-сервер на случайном порту, общий адрес будет:

http://localhost:XXXX/web-server/static/default-get.html?_ijt=234qq4ggjjo0

Формы

Форма определяется атрибутом `form`. Ключевые атрибуты формы [определены в спецификации](#):

Content attributes:

[Global attributes](#)

`action` — [URL](#) to use for [form submission](#)

`enctype` — Entry list encoding type to use for [form submission](#)

`method` — Variant to use for [form submission](#)

- `action` - URL, на который будет отправлена форма (по умолчанию тот, с которого загружена страница)
- `enctype` - кодировка данных формы
- `method` - метод передачи (GET - по умолчанию, POST)



Default Values

В HTML, если вы указываете недопустимое значение, то используется значение по умолчанию.

Например, если указать метод не **GET** или **POST**, то будет использован **GET**.

Enctype

Доступные значения:

- `application/x-www-form-urlencoded` (по умолчанию)
- `multipart/form-data`
- `text/plain`

Action

Начнём с `action` - текущее значение нас не устраивает, мы должны указать адрес нашего сервера: `action="http://localhost:9999"`.

```
<form action="http://localhost:9999">  
  <input>  
  <input>  
  <button>Send</button>  
</form>
```

▼ General

Request URL: [http://localhost:9999/?](http://localhost:9999/)

Request Method: GET

Status Code: 🟢 200 OK

Remote Address: [::1]:9999

Referrer Policy: no-referrer-when-downgrade

В консоли нашего сервера:

- `method` - GET
- `path` - `/?`

Name

Согласно спецификации, только значения полей ввода, имеющие атрибут `name`, отправляются на сервер:

```
<form action="http://localhost:9999">
  <input> <!-- without name - никуда не отправится -->
  <input name="title">
  <input name="value">
  <input name="value"> <!-- специально два поля с одним name -->
  <input type="file" name="image"> <!-- для отправки файлов -->
  <button>Send</button>
</form>
```

Query

Важно: всё, что мы вводили, попало в URL (браузер описывает это как Query String Parameters):

<http://localhost:9999/?title=%D0%B2%...&value=%D1%82...&value=%D1%87%...&image=photo.jpg>

Query

<http://localhost:9999/?title=%D0%B2%...&value=%D1%82...&value=%D1%87%...&image=photo.jpg>

Ключевые моменты:

1. параметры идут после **?** склеены через **&** и передаются в формате **key=value**, где **key** - это **name** у поля ввода, а **value** - введённое значение;
2. часть символов закодирована;
3. вместо изображения передалось имя файла.

URL

Общая структура:

foo://example.com:8042/over/there?name=ferret#nose

//_/_/_/

| | | | |

scheme authority path query fragment

параметры передаются в URL'e
при передаче методом **GET**

Percent Encoding

Кодировка, позволяющая закодировать символы, у которых есть специальное назначение:

▼ General	▼ Query String Parameters	view source	view URL encoded
Request URL: http://localhost:9999/?title=%D0%B2%D1%82%D0%BE%D1%80%D0%BE%D0%BF%D0%BE%D0%BB%D0%B5&value=%D1%82%D1%80%D0%B5%D1%82%D1%8C%D0%B5+%D0%BF%D0%BE%D0%BB%D0%B5&value=%D1%87%D0%B5%D1%82%D0%B2%D1%91%D1%80%D1%82%D0%BE%D0%BF%D0%BE%D0%BB%D0%B5&image=photo.jpg	title: второе поле value: третье поле value: четвертое поле image: photo.jpg		браузер показывает декодированные значения
Request Method: GET			
Status Code: 🟢 200 OK			
Remote Address: [::1]:9999			
Referrer Policy: no-referrer-when-downgrade			

Query

В Query обычно передаются поисковые параметры, чтобы пользователь мог скопировать URL и передать другому пользователю, чтобы тот увидел тот же результат*.

Давайте посмотрим, как передаются параметры в Яндекс.Поиск, Avito, Aviasales или других сервисах.

*Или очень похожий, поскольку результаты поисковой выдачи зависят от времени, через которое второй пользователь откроет данную ссылку.

Query

Принимающая сторона (сервер), должна уметь декодировать данные из Percent Encoding в обычное представление.

В Java для этого есть специальный класс [URLDecoder](#) (и парный к нему - [URLEncoder](#)).




Enctype

Попытки **поставить другой enctype** ни к чему не приведут - данные по-прежнему будут отправляться в Percent Encoding в URL'е.

POST

Поставим метод POST и отправим форму (по умолчанию `application/x-www-form-urlencoded`):

▼ General	▼ Form Data	view source	view URL encoded
Request URL: http://localhost:9999/ Request Method: POST Status Code:  200 OK Remote Address: [::1]:9999 Referrer Policy: no-referrer-when-downgrade	title: второе поле value: третье поле value: четвёртое поле image: photo.jpg		браузер показывает декодированные значения

И самое главное:

1. В URL'е данные не передаются, а передаются в теле запроса, но тоже в Percent Encoding
2. Выставляется заголовок `Content-Type: application/x-www-form-urlencoded`
3. Файлы всё равно не отправляются

Ключевое

Благодаря заголовку **Content-Type** сервер сможет понять, в каком виде клиент прислал данные.

На прошлой лекции мы видели влияние этого заголовка на формат отображения данных клиентом. Сейчас (в запросе) он будет влиять на то, как сервер будет пытаться обрабатывать данные.

Query + Form

Попробуем "руками" выставить URL так, чтобы передавался и Query:

```
<form action="http://localhost:9999?value=get-value" method="post">
  <input> <!-- without name - нигуда не отправится -->
  <input name="title">
  <input name="value">
  <input name="value"> <!-- специально два поля с одним name -->
  <input type="file" name="image"> <!-- для отправки файлов -->
  <button>Send</button>
</form>
```


Query + Form

Мы увидим, что **данные придут и в Query, и в теле (POST)**. Таким образом, при отправке методом POST данные одновременно можно передавать и в URL'е, и в теле запроса:

▼ **Query String Parameters** view source view URL encoded

value: get-value

▼ **Form Data** view source view URL encoded

title: второе поле

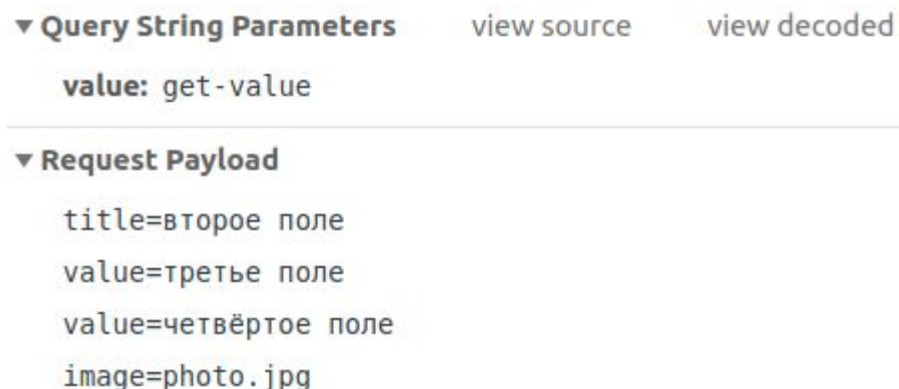
value: третье поле

value: четвертое поле

image: photo.jpg

text/plain

Попробуем отправить с `enctype="text/plain"`:



Данные склеиваются в виде текста через символ переноса строки и отправляются на сервер в формате `key=value`. Данный формат используется достаточно редко, поэтому детально на нём останавливаться не будем.

multipart/form-data

Попробуем отправить с `enctype="multipart/form-data"`:

header

Content-Type: multipart/form-data; boundary=----WebKitFormBoundaryX9hAGfP6eae9AVLx

body

-----WebKitFormBoundaryX9hAGfP6eae9AVLx

Content-Disposition: form-data; name="title"

второе поле

-----WebKitFormBoundaryX9hAGfP6eae9AVLx

Content-Disposition: form-data; name="value"

третье поле

-----WebKitFormBoundaryX9hAGfP6eae9AVLx

Content-Disposition: form-data; name="value"

четвёртое поле

-----WebKitFormBoundaryX9hAGfP6eae9AVLx

Content-Disposition: form-data; name="image"; filename="photo.jpg"

Content-Type: image/jpeg ● \r\n

● \r\n

0000JFIF000`00C0000000

multipart/form-data

Браузер сделал **две вещи**:

1. В заголовке `Content-Type` отправил `boundary` (границу), которым он будет разделять данные каждого поля
2. В теле отправил все поля, разделив их `boundary`. При этом формат передачи:

`(header\r\n)+`

`\r\n\r\n`

`body`

Через Developer Tools этого не видно (самого тела), для этого мы и написали наш сервер.

Ключевое

Благодаря `enctype="multipart/form-data"` мы имеем возможность передавать файлы. Но это работает только при отправке методом `POST`.

Соответственно, на принимающей стороне (сервере) должна быть реализована поддержка декодирования соответствующих полей. Умеет это делать, например, библиотека [FileUpload](#) из состава Apache Commons.



Итоги

Итоги

Сегодня мы посмотрели на то, каким образом отправляются данные из браузера с помощью форм.

Ключевые моменты:

- **GET** передаёт все поля формы в URL'e через Percent Encoding;
- **POST** передаёт все поля формы в Body в кодировке, зависящей от **enctype**, при этом в Query также можно передавать данные;
- файлы можно передавать только через **multipart/form-data** и метод **POST**.



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**.

⌘ нетология

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

Григорий Вахмистров