**Сервер** – компьютер, который предоставляет информацию другим компьютерам. Такое сообщение называется **ответом** (***response***).

**Клиент** – компьютер, который запрашивает информацию у сервера. Такое сообщение называется **запросом** (***request***). Сервер тоже может быть клиентом.

***Internet Protocol (IP)*** – регламентирует подготовку данных к передаче и их доставку адресату.

***Transmission Control Protocol (TCP)*** – регламентирует доставку и проверяет, что IP передал все данные адресату по определенному порту. Если нет, они запрашиваются и отправляются повторно.

**Сетевой порт (*port*)** – целое неотрицательное число, которое закрепляется за конкретной программой.

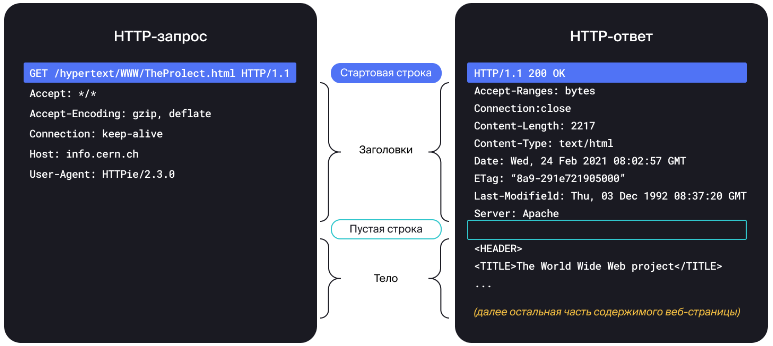
***HyperText Transfer Protocol (HTTP)*** – протокол передачи гипертекста использует протоколы *TCP* и *IP*, вводит понятия клиента и сервера и описывает правила их взаимодействия.

***Uniform Resource Locator (URL)*** – уникальный адрес на ресурс или гиперссылка.



***Domain Name System (DNS)*** – система доменных имен позволяет привязать понятное имя к *IP*-адресу и применяться вместо него.

***HyperText Markup Language (HTML)*** – язык гипертекстовой разметки используется для создания сайтов и позволяет оставлять в тексте метки (**теги**) в угловых скобках *< >*.



Каждое обращение к серверу начинается со стартовой строки. В ней указывается имя протокола и его методы взаимодействия с сервером. Основные методы, применяемые в *HTTP*-запросах:

* *POST* – используется для передачи данных на сервер.
* *GET* – используется для получения ресурса.
* *PUT* – используется для передачи данных серверу аналогично *POST*. Разница в том, что он заменяет существующий ресурс целиком или создает новый ресурс при его отсутствии.
* *PATCH* – используется для частичного обновления данных ресурса.
* *DELETE* – используется для удаления ресурса.

Методы соглашения ***CRUD*** (*Create*, *Read*, *Update*, *Delete*) сопоставляются соответственно с методами *POST*, *GET*, *PUT*, *DELETE*.

В ответе сервера на запрос **в стартовой строке** указывается **код ответа** (***status code***). Он состоит из трехзначного числа и названия. Коды успеха начинаются с 2, коды клиентских ошибок – с 4, коды серверных ошибок – с 5.

Дополнительная информация передается **в заголовке** (***headers***). Каждый заголовок состоит из названия и значения, отделенного двоеточием. Примеры имен заголовков:

* *Accept* – используется только в запросах и указывает серверу список типов данных, которые клиент может от него принять и корректно обработать. Значение *\*/\** означает любые типы данных.
* *Host* – используется только в запросах и указывает адрес сервера, которому нужно отправить запрос.
* *Content-Length* – указывает размер тела сообщения в байтах.
* *Content-Type* – указывает тип передаваемых данных.

Также вместе с запросом или ответом могут передаваться какие-либо данные. Они помещаются в **тело** (***body***) и отделяются пустой строкой.

**Веб-приложения** (***web applications***) – это сайты, состоящие из динамических страниц и представляющие возможность интерактивного взаимодействия с ними. **Фронтенд часть** веб-приложения решает, как отобразить данные в браузере, а также вовремя обновляет их. Серверы, обрабатывающие запросы фронтенда веб-приложения, называются **веб-сервисами**, или **веб-службами** (***web services***).

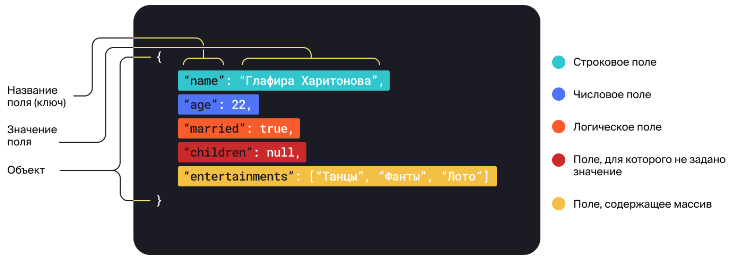
***Web Application Programming Interface*** *(****Web API****)* – это набор *URL* и совокупность соглашений о том, как именно работать с ними другим программам. Например, приложения могут взаимодействовать друг с другом через *API* по принципу черного ящика.

При разработке приложений могут использоваться подходы ***API-first*** или ***Code-first***.



***JavaScript Object Notation (JSON)*** – специальный формат для передачи данных. Во фронтенде он используется в *POST*, *PUT* и *PATCH* запросах. В бэкенде он используется в *GET* и *PUT* запросах.

Формат *JSON* поддерживает шесть типов данных. Примитивы – строки, числа, логические значения, *null*. Сложные типы – массивы *[1,2,3,4]* и объекты *{ “value”: 42 }*. Примитивы не могут использоваться сами по себе, они должны входить в состав объекта или массива. Есть поддержка вложенных типов.



**Рассмотрим создание *HTTP*-сервера.**

**Эндпоинт (*end-point*)**, или конечная точка – это комбинация *URL*-адреса и *HTTP*-метода. Каждый эндпоинт отвечает за выполнение конкретной задачи.

В стандартной библиотеке *Java* есть свой *HTTP*-сервер. Он находится в пакете *com.sun.net.httpserver* и реализован главным классом ***HttpServer***.

При передаче по сети информация разбивается на блоки – **сетевые пакеты** (***network packet***). Внутри пакетов есть данные, а также *IP*-адрес и **номер порта**, используемого для передачи данных. Номер порта привязан к определенной программе и принимает значения от 1 до 65535 (216-1), а также уникален в данной операционной системе.

**Сокет (*socket*)** – это средство операционной системы, через которое программа может получить доступ к сети. В качестве параметров для открытия сокета используются *IP*-адрес и номер порта. Одной программе могут соответствовать несколько сокетов.

Метод ***HttpServer.create()*** позволяет создать собственный веб-сервер.

Метод ***HttpServer.bind(InetSocketAddress addr, int backlog)*** привязывает сервер к порту.

Числовой параметр ***backlog*** – это число соединений, которые могут находиться в ожидании, пока сервер обрабатывает текущий запрос. При указании нуля будет использоваться стандартное значение, определенное в операционной системе. Если не указывать конкретный *IP*-адрес, то веб-сервер будет работать на всех доступных *IP*-адресах.

Процесс трансформации Java-объекта в какой-то другой формат называется **сериализацией** (***serialization***). Обратный процесс – **десериализация** (***deserialization***). Популярные форматы для сериализации данных – *JSON*, *XML*, *Protobuf*, *YAML*.

Рассмотрим пример такой трансформации в *JSON* с помощью библиотеки ***GSON***. Метод ***toJson(Object src)*** переводит объект в строковое представление *JSON*. Метод ***fromJson(String json, Class<T> classOfT)*** переводит строковое представление *JSON* в указанный объект класса.

*import com.google.gson.Gson;*

*class Practicum {*

*public static void main(String[] args) {*

*Gson gson = new Gson();*

*String jsonString = gson.toJson(dog); // сериализация*

*Dog dog = gson.fromJson(jsonString, Dog.class); } } // десериализация*

Для создания экземпляра *Gson* с дополнительными настройками используется класс ***GsonBuilder***.

Метод ***create()*** возвращает объект класса *Gson*.

Метод ***setPrettyPrinting()*** включает удобное для чтения отображение данных в формате *json*. По умолчанию эта настройка выключена и *json* выводится в одну строку.

Метод ***serializeNulls()*** включает добавление в json строк со значением null. По умолчанию эта настройка выключена и *json* не добавляет поля вида *"название\_поля": null*.

Метод ***registerTypeAdapter(Type type, Object typeAdapter)*** позволяет реализовать свой сериализатор или десериализатор с заданными параметрами. Для этого нужно написать свой конвертер, унаследованный от абстрактного класса ***TypeAdapter***, и переопределить методы ***write(JsonWriter jsonWriter, Object obj)*** и ***read(JsonReader jsonReader, Object obj)***.

Например, переопределим формат конвертирования для экземпляра даты *LocalDate*:

*class LocalDateAdapter extends TypeAdapter<LocalDate> {*

*private static final DateTimeFormatter formatterWriter = DateTimeFormatter.ofPattern("dd--MM--yyyy");*

*private static final DateTimeFormatter formatterReader = DateTimeFormatter.ofPattern("dd.MM.yyyy");*

*@Override*

*public void write(final JsonWriter jsonWriter, final LocalDate localDate) throws IOException {*

*jsonWriter.value(localDate.format(formatterWriter)); }*

*@Override*

*public LocalDate read(final JsonReader jsonReader) throws IOException {*

*return LocalDate.parse(jsonReader.nextString(), formatterReader); } }*

*Gson gson = new GsonBuilder()*

*.registerTypeAdapter(LocalDate.class, new LocalDateAdapter())*

*.create();*

Для создания эндпоинта нужно создать класс, реализующий интерфейс ***HttpHandler*** и переопределить метод ***handle(HttpExchange httpExchange)***.

Метод ***httpExchange.createContext(String path, HttpHandler handler)*** используется, чтобы связать *URL* путь и обработчик запроса.

Метод ***httpExchange.start()*** используется для запуска сервера.

Метод ***httpExchange.sendResponseHeaders (int rCode, long responseLength)*** отправляет ответ, который можно сформировать заранее.

Если в ответ нужно включить тело, то требуется создать экземпляр класса ***OutputStream*** с помощью метода ***httpExchange.getResponseBody()***. Он возвращает объект ***OutputStream***, в который нужно записать поток байтов с помощью метода ***getBytes()***.

В итоге базовый сервер будет выглядеть так:

*public class Practicum {*

*private static final int PORT = 8080;*

*public static void main(String[] args) throws IOException {*

*HttpServer httpServer = HttpServer.create(); // создаем сервер*

*httpServer.bind(new InetSocketAddress(PORT), 0); // назначаем сокет*

*httpServer.createContext("/hello", new HelloHandler()); // связываем путь и обработчик*

*httpServer.start(); // запускаем сервер*

*System.out.println("HTTP-сервер запущен на " + PORT + " порту!"); }*

*static class HelloHandler implements HttpHandler {*

*@Override*

*public void handle(HttpExchange httpExchange) {*

*try {*

*String response = "Hey! Glad to see you on our server.";*

*httpExchange.sendResponseHeaders(200, 0);*

*try (OutputStream os = httpExchange.getResponseBody()) {*

*os.write(response.getBytes()); } } // преобразование тела в байты*

*catch(Exception e) {*

*e.printStackTrace(); }*

*finally {*

*httpExchange.close(); } // закрытие потока*

Метод ***httpExchange.getRequestBody()*** возвращает тело запроса в формате ***InputStream***, которые клиент отправил на сервер.

*InputStream inputStream = httpExchange.getRequestBody();*

*String body = new String(inputStream.readAllBytes(), StandardCharsets.UTF\_8);*

Метод ***httpExchange.getRequestMethod()*** возвращает *HTTP*-метод, который клиент использовал при отправке запроса. С помощью него можно сделать из одного эндпоинта несколько.

*String method = httpExchange.getRequestMethod();*

*String response;*

*switch(method) {*

*case "POST":*

*response = "Вы использовали метод POST!";*

*break;*

*case "GET":*

*response = "Вы использовали метод GET!";*

*break;*

*default:*

*response = "Вы использовали какой-то другой метод!"; }*

Метод ***httpExchange.getRequestHeaders()*** возвращает заголовки, которые передал клиент, в виде *Map<String, List<String>>*.

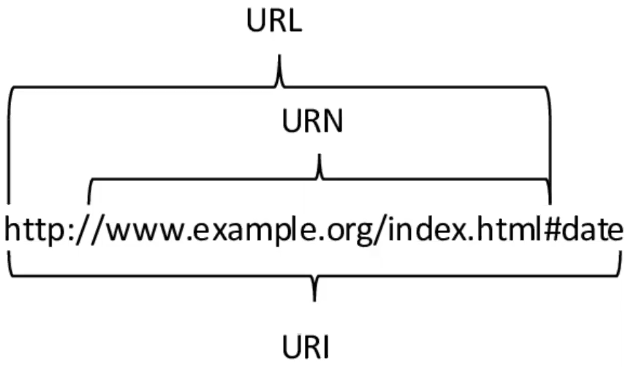
Метод ***httpExchange.getRequestURI()*** возвращает *URI*, использованный клиентом при вызове сервера. Из него можно получить данные пути с помощью метода ***getPath()*** и использовать в ответе.

Метод ***httpExchange.getResponseHeaders()*** возвращает объект класса ***Headers***, в который можно добавить заголовки с помощью метода ***set(String)***.

**Рассмотрим создание *HTTP*-клиента.**

Для этого нам понадобятся:

* Класс ***java.net.URI*** – описывает адрес интересующего ресурса. Его можно создать с помощью статического метода ***create(String)***.



* Класс ***HttpRequest*** – описывает *HTTP*-запрос к серверу. Статический метод ***newBuilder().build()*** возвращает запрос с настройками по умолчанию.
* Класс ***HttpResponse*** – описывает *HTTP*-ответ к серверу. Методы ***body()*** и ***statusCode()*** позволяют получить соответствующие строковые значения.
* Класс ***HttpClient*** – реализует коммуникацию с сервером по протоколу *HTTP*. Статический метод ***newHttpClient()*** возвращает клиент с настройками по умолчанию. Статический метод ***send(HttpRequest request, HttpResponse.BodyHandler<T> responseBodyHandler)*** отправляет запрос на сервер и возвращает объект типа *HttpResponse*.

Пример простого *HTTP*-клиента:

*public class Practicum {*

*public static void main(String[] args) throws IOException, InterruptedException {*

*// создаём экземпляр URI, содержащий адрес нужного ресурса*

*URI url = URI.create("https://ru.wikipedia.org/wiki/Список\_кодов\_состояния\_HTTP");*

*// создаём объект, описывающий HTTP-запрос*

*HttpRequest request = HttpRequest.newBuilder() // получаем экземпляр билдера*

*.GET() // указываем HTTP-метод запроса*

*.uri(url) // указываем адрес ресурса*

*.version(HttpClient.Version.HTTP\_1\_1) // указываем версию протокола*

*.header("Accept", "text/html") // указываем заголовок Accept*

*.build(); // заканчиваем настройку и создаём http-запрос*

*// HTTP-клиент с настройками по умолчанию*

*HttpClient client = HttpClient.newHttpClient();*

*// получаем стандартный обработчик тела запроса с конвертацией содержимого в строку*

*HttpResponse.BodyHandler<String> handler = HttpResponse.BodyHandlers.ofString();*

*// отправляем запрос и получаем ответ от сервера*

*HttpResponse<String> response = client.send(request, handler);*

*// выводим код состояния и тело ответа*

*System.out.println("Код ответа: " + response.statusCode()); // Можно добавить реализацию в*

*System.out.println("Тело ответа: " + response.body()); } } // зависимости от кода ответа*

Исключения ***ConnectException*** и другие, наследуемые от ***IOException***, возникают из-за проблем с сетевой коммуникацией (перегружен сервер, обрыв соединения, несуществующий адрес, нет связи с сервером).

Исключения ***InterruptedException*** возникает при долгом ожидании ответа от сервера, когда запрос обрабатывается на стороне клиента.

Для передачи данных на сервер используется не только тело запроса, но и **строка запроса** (***query string***). Она находится в конце *URL* и отделяется от пути знаком вопроса **?** , после которого указываются пары в формате ***параметр=значение***, отделенные символом **&** .

Адрес может содержать только латинские буквы, цифры и отдельные знаки пунктуации. Все остальные символы будут преобразованы через **процентное кодирование**.



Также для передачи данных на сервер можно использовать заголовки (***header***). Для этого нужно вызвать метод ***header(String name, String value)*** у билдера в момент формирования запроса. Например, *header("Accept", "application/*json*")* в запросе информирует сервер, что клиент готов получить ответ в формате *json*.

Для работы с форматом GSON существует несколько полезных библиотек:

*import com.google.gson.JsonArray; // описывает JSON-массив*

*import com.google.gson.JsonElement; // описывает любой тип данных JSON*

*import com.google.gson.JsonObject; // описывает JSON-объект*

*import com.google.gson.JsonParser; // разбирает JSON-документ на элементы*

Метод ***JsonParser.parseString(String json)*** разбирает текст на составные элементы и возвращает объект класса ***JsonElement***.

Метод ***JsonObject.get(String name)*** принимает название поля и возвращает его значение в виде ***JsonElement***.

Методы класса ***JsonElement***:

* ***isJsonObject()*** – проверяет, является ли элемент *JSON*-объектом
* ***getAsJsonObject()*** – преобразовывает элемент в *JSON*-объект
* ***getAsJsonArray()*** – преобразовывает элемент в *JSON*-массив
* ***getAsString()*** – преобразовывает *JSON*-элемент в строку
* ***getAsLong()*** – преобразовывает *JSON*-элемент в число
* ***getAsBoolean()*** – преобразовывает *JSON*-элемент в логическое значение

*if (response.statusCode() == 200) {*

*// передаем парсеру тело ответа в виде строки, содержащей данные в формате JSON*

*JsonElement jsonElement = JsonParser.parseString(response.body());*

*if(!jsonElement.isJsonObject()) { // проверяем, точно ли мы получили JSON-объект*

*System.out.println("Ответ от сервера не соответствует ожидаемому.");*

*return; }*

*// преобразуем результат разбора текста в JSON-объект*

*JsonObject jsonObject = jsonElement.getAsJsonObject();*

*// получаем название страны*

*String country = jsonObject.get("country").getAsString();*

*System.out.println("Страна: " + country); }*