Сетевые протоколы. Основы.

TCP - Transmission Control Protocol для передачи информации где информация должна быть передана полностью, то есть вся информация и **последовательность** важна. Скорость ниже чем на UDP так как при каждом отправлении части информации есть проверка, при отсуствии заново запрашивается.

UDP - User Datagram Protocol для передачи потоковых данных где потеря части информации не важна. Тут скорость выше чем на TCP, так как данные передаются и не проверяются получен ли он.

Скорость **UDP** по сравнению с **TCP** зависит от конкретного случая использования, но в общем:

- **UDP быстрее TCP примерно в 3–10 раз** в большинстве сценариев передачи данных.
- В **локальных сетях (LAN)** разница меньше — **примерно в 2–3 раза**.
- В **нагруженных сетях или Интернете** TCP может быть медленнее **в 5–10 раз** из-за подтверждений и повторных передач.

Сокет - это не протокол. Это интерфейс для сетевых приложений(реализация зависит от ОС) для установки соединения и обмена данными.

Доменное имя - это такая строка, где под капотом есть IP адрес. Это нужно для того чтобы не нужно было запоминать IP адреса.

DNS - система доменных имен. Там хранится "словарь". Это такой посредник который определяет относится ли какое-то доменное имя к IP адресу.

Порт - от 0 до 65536. Это такое окошко, где приложения ждут или передают данные.

📌 В TCP и UDP всего **65 536 (от 0 до 65 535)**:

- **0–1023** — **зарезервированные (well-known)** (например, 80 — HTTP, 443 — HTTPS).
- **1024–49 151** — **зарегистрированные (registered)** (например, 3306 — MySQL).
- **49 152–65 535** — **динамические (ephemeral)**, назначаются временно.

*Если не закрывать Scanner или типа такое, то поток не будет закрыт и невозможно будет использовать этот поток.

OSI vs TCP/IP

// получается **TCP/IP** это такая же концепция как и **OSI**(The **Open Systems Interconnection** model) которая описывает принципы работы сетей

Критерий	OSI (7 уровней)	TCP/IP (4 уровня)
Происхождение	Теоретическая модель (ISO, 1984 г.)	Практическая модель (разработана для ARPANET/Интернета)
Уровни	7 уровней:	4 уровня:
	1. Физический	1. Сетевой интерфейс (объединяет физический и канальный)
	2. Канальный	2. Интернет (IP)
	3. Сетевой	3. Транспортный (TCP/UDP)
	4. Транспортный	4. Прикладной (HTTP, FTP, DNS и т.д.)
	5. Сеансовый	
	6. Представления	
	7. Прикладной	
Цель	Универсальное описание сетей	Реализация работающего стека протоколов (Интернет)
Использование	Редко на практике, но полезна для обучения	Основа современного интернета

Пишем первый мини-эхо сервер

Есть понятия:

Сервер - слушает входящие подключения, обрабатывает запросы клиентов и отправляет ответы.

Клиент - инициирует соединение с сервером для получения или отправки данных.

Сервер "ждет" входящих запросов, а клиент "звонит" серверу.

Сокеты в Java

ServerSocket

Это класс для создания серверного сокета который слушает определенный порт на машине. Это "точка входа" для клиентов, желающих установить соединение с сервером

После создания объекта ServerSocket сервер переходит в состояние ожидания входящих соединений. Метод accept() блокирует выполнение до тех пор, пока не поступит запрос на подключение. Когда клиент пытается подключиться, accept() возвращает новый объект типа Socket

Socket

Это класс представляет активное соединение между двумя сторонами. Он содержит информацию о подключении (ip-адресс, порт) и представляет методы для передачи данных.

На стороне клиента создается объект Socket, который "соединяется" с сервером. На серверной стороне после вызова accept(), также получается объект Сокет Socket, через который происходит обмен данными.

ServerSocket - для прослушивания и приема новых подключений

Socket - для непосредственного обмена данными между уже установленными точками соединения

Потоки ввода-вывода (Input/Output Streams)

После установления соединения через сокет необходимо обмениваться данными. Для этого используются потоки:

1. InputStream и OutputStream

- **InputStream** – базовый класс для чтения байтов из источника (например, из сокета).
- **OutputStream** – базовый класс для записи байтов в поток (например, в сокет).

2. Обёртки для работы с текстом

Чтобы упростить работу с текстовыми данными, часто используют следующие классы:

- **InputStreamReader:**
  Позволяет преобразовать поток байтов (InputStream) в поток символов (Reader) с учётом выбранной кодировки.
- **BufferedReader:**
  Оборачивает `Reader` и предоставляет возможность чтения данных построчно с буферизацией, что увеличивает производительность.
- **PrintWriter:**
  Удобный класс для записи текстовых данных. Он предоставляет методы для вывода

строк и других типов данных, а также может автоматически сбрасывать (flush) буфер после каждой записи (если включен параметр `autoFlush`).

Пример минимального эхо-сервера

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

public class EchoServer {
    public static void main(String[] args) {
        int port = 12345; // Порт, на котором сервер будет ожидать подключения

        // Создаём ServerSocket для прослушивания указанного порта
        try (ServerSocket serverSocket = new ServerSocket(port)) {
            System.out.println("Сервер запущен и ожидает подключения на порту
" + port);

            // Сервер работает в бесконечном цикле, принимая подключения от
клиентов
            while (true) {
                // Метод accept() блокируется до появления нового клиента
                try (Socket clientSocket = serverSocket.accept();
                     // Создаём BufferedReader для чтения строк из входящего
потока клиента
                     BufferedReader in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
                     // Создаём PrintWriter для отправки строк клиенту,
autoFlush включён
                     PrintWriter out = new
PrintWriter(clientSocket.getOutputStream(), true)) {

                    System.out.println("Клиент подключился: " +
clientSocket.getInetAddress());

                    String inputLine;
                    // Читаем данные от клиента построчно
                    while ((inputLine = in.readLine()) != null) {
                        System.out.println("Получено от клиента: " +
inputLine);
                        // Отправляем обратно полученную строку с префиксом
"Эхо: "
                        out.println("Эхо: " + inputLine);
                    }

                    System.out.println("Клиент отключился");
```

```
                } catch (IOException e) {
                    System.out.println("Ошибка при обработке клиента: " +
e.getMessage());
                }
            }
        } catch (IOException e) {
            System.out.println("Ошибка запуска сервера: " + e.getMessage());
        }
    }
}
```

Пример минимального клиента

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

public class EchoClient {
    public static void main(String[] args) {
        String host = "localhost"; // Адрес сервера (может быть IP-адрес или
доменное имя)
        int port = 12345;          // Порт сервера

        try (Socket socket = new Socket(host, port);
             // Поток для чтения данных, полученных от сервера
             BufferedReader in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
             // Поток для отправки данных на сервер
             PrintWriter out = new PrintWriter(socket.getOutputStream(),
true);
             // Для чтения пользовательского ввода из консоли
             BufferedReader stdIn = new BufferedReader(new
InputStreamReader(System.in))) {

            System.out.println("Подключено к серверу " + host + " на порту " +
port);
            String userInput;
            System.out.println("Введите сообщение (для завершения введите
'exit'):");

            while ((userInput = stdIn.readLine()) != null) {
                if ("exit".equalsIgnoreCase(userInput)) {
                    break;
                }
                // Отправляем введённое сообщение на сервер
```

```
                out.println(userInput);
                // Читаем ответ от сервера и выводим его
                System.out.println("Ответ сервера: " + in.readLine());
            }
        } catch (IOException e) {
            System.out.println("Ошибка клиента: " + e.getMessage());
        }
    }
}
```