HIGHER SCHOOL OF ECONOMICS
NATIONAL RESEARCH UNIVERSITY

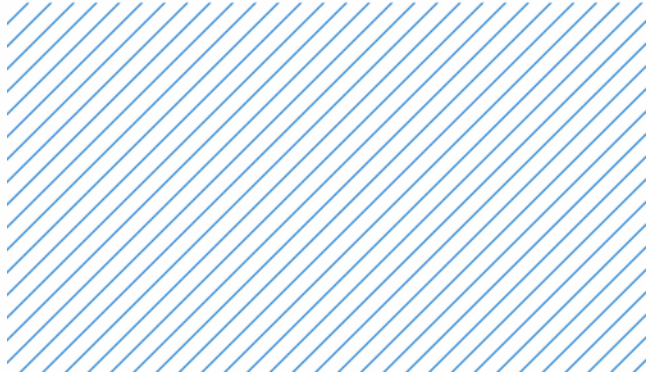
Лекция 18

Основы сетевого программирования

Программирование на языке Java

Роман Гуров

ВШЭ БИ 2021



Сеть

Пусть у нас есть несколько компьютеров (или других подобных устройств)

Как научить эти компьютеры обмениваться друг с другом информацией?

Нужно соединить эти компьютеры между собой
и определить некоторый общий для всех способ общения по этому соединению

Такая группа соединённых компьютеров называется *сетью*, а способ общения – *протоколом*

Протоколы разбиваются на *слои*, начиная от низкоуровневых, работающих непосредственно с сетевым проводом, заканчивая высокоуровневыми, которые может придумать для себя разработчик приложения

Протоколы разных уровней объединяются для произведения сетевого взаимодействия
(подобно тому, как потоки ввода-вывода могли оборачивать друг друга)

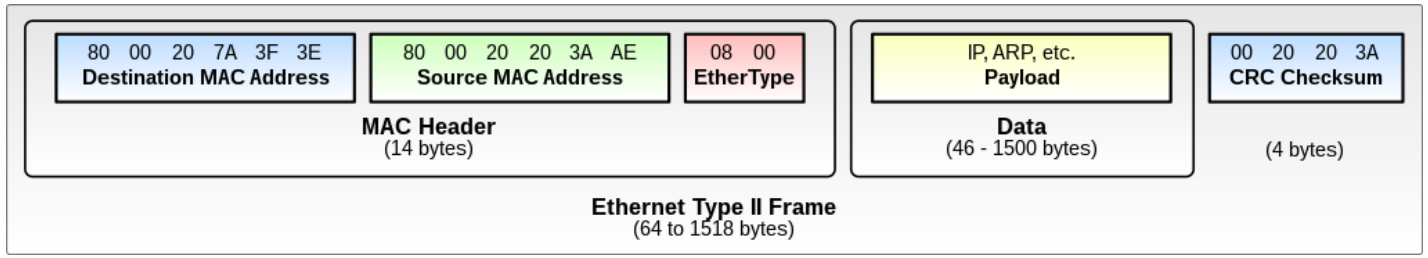
Протокол Ethernet

Ethernet – самый низкоуровневый протокол, который работает на уровне сетевого кабеля

На этот протокол ложится ответственность за корректную передачу и приём сигналов на контакты проводов

Но помимо этого, протокол предоставляет формат, в котором данные передаются

Данные отправляются по сети в виде некоторых наборов – *фреймов*:



До 1500 байтов полезных данных

Для определения пункта назначения пакета используется *MAC-адрес* – уникальный идентификатор сетевого устройства, записанный в него производителем

Адресация через MAC-адрес работает внутри локальных сетей, когда небольшое количество устройств соединены друг с другом в одну полностью общую сеть

Но на этом не сделать интернет, в котором миллионы устройств разбросаны по всему миру

Протокол IP

IP (Internet Protocol) – протокол, работающий над протоколом Ethernet,
и предоставляющий полноценную адресацию сетевых устройств

Именно благодаря нему и работает сеть Интернет, соединяющая устройства со всего мира

IP-адрес – уникальный числовой идентификатор устройства в компьютерной сети, работающей по протоколу IP

Пакет (так на этом уровне называется *фрейм*) протокола IP будет проходить через цепь маршрутизаторов, которые непосредственно соединены между собой, и каждый из них будет определять, на какой следующий маршрутизатор фрейму надо отправиться, до тех пор, пока не будет достигнут адресат

Пример IP-адреса: `74.125.205.94` (4 байта: 0-255 каждый)

Также, в дальнейшем нам пригодится понятие *порта*

Поскольку на одном компьютере может быть сразу несколько сетевых сервисов, должна быть какая-то возможность различать, для чего именно адресован пакет с данными

Порт – число, задающее номер окошка на компьютере, в которое отправляются данные

Некоторое серверное приложение ожидает принятия данных на конкретном порту, а клиенты, зная порт, отправляют данные на него

TCP и UDP

TCP и **UDP** – протоколы, работающие на основе IP и предоставляющие возможность указания *порта*

Собственно, на этих протоколах все приложения и общаются

Лучше, чем на Википедии, не описать:

TCP – ориентированный на соединение протокол, что означает необходимость «рукопожатия» для установки соединения между двумя хостами. Как только соединение установлено, пользователи могут отправлять данные в обоих направлениях.

- *Надёжность* – TCP управляет подтверждением, повторной передачей и тайм-аутом сообщений. Производятся многочисленные попытки доставить сообщение. Если оно потеряется на пути, сервер вновь запросит потерянную часть. В TCP нет ни пропавших данных, ни (в случае многочисленных тайм-аутов) разорванных соединений.
- *Упорядоченность* – если два сообщения последовательно отправлены, первое сообщение достигнет приложения-получателя первым. Если участки данных прибывают в неверном порядке, TCP отправляет неупорядоченные данные в буфер до тех пор, пока все данные не могут быть упорядочены и переданы приложению.
- *Тяжеловесность* – TCP необходимо три пакета для установки сокет-соединения перед тем, как отправить данные. TCP следит за надёжностью и перегрузками.
- *Потоковость* – данные читаются как поток байтов, не передается никаких особых обозначений для границ сообщения или сегментов.

UDP — более простой, основанный на сообщениях протокол без установления соединения. Протоколы такого типа не устанавливают выделенного соединения между двумя хостами. Связь достигается путём передачи информации в одном направлении от источника к получателю без проверки готовности или состояния получателя.

- *Ненадёжный* — когда сообщение посылается, неизвестно, достигнет ли оно своего назначения — оно может потеряться по пути. Нет таких понятий, как подтверждение, повторная передача, тайм-аут.
- *Неупорядоченность* — если два сообщения отправлены одному получателю, то порядок их достижения цели не может быть предугадан.
- *Легковесность* — никакого упорядочивания сообщений, никакого отслеживания соединений и т. д. Это небольшой транспортный уровень, разработанный на IP.
- *Датаграммы* — пакеты посылаются по отдельности и проверяются на целостность только если они прибыли. Пакеты имеют определенные границы, которые соблюдаются после получения, то есть операция чтения на сокете-получателе выдаст сообщение целиком, каким оно было изначально послано.
- *Нет контроля перегрузок* — UDP сам по себе не избегает перегрузок. Для приложений с большой пропускной способностью возможно вызвать коллапс перегрузок, если только они не реализуют меры контроля на прикладном уровне.

Протокол HTTP

Пока что, рассмотренные протоколы TCP и UDP давали нам лишь возможность отправлять поток байтов (для UDP – датаграммы)

Поверх потока байтов можно построить и какой-то совсем высокоуровневый протокол общения

Hyper**T**ext **T**ransfer **P**rotocol (**HTTP**) – текстовый протокол, работающий над TCP, использующийся веб-браузерами и веб-серверами

Пример общения по протоколу HTTP (получение страницы на википедии):

Запрос клиента:

```
GET /wiki/страница HTTP/1.1
Host: ru.wikipedia.org
User-Agent: Mozilla/5.0 (X11; U; Linux i686; ru; rv:1.9b5) Gecko/2008050509 Firefox/3.0b5
Accept: text/html
Connection: close
(пустая строка)
```

Ответ сервера:

```
HTTP/1.1 200 OK
Date: Wed, 11 Feb 2009 11:20:59 GMT
Server: Apache
X-Powered-By: PHP/5.2.4-2ubuntu5wm1
Last-Modified: Wed, 11 Feb 2009 11:20:59 GMT
Content-Language: ru
Content-Type: text/html; charset=utf-8
Content-Length: 1234
Connection: close
(пустая строка)
(запрошенная страница в HTML)
```

Общепринятый порт для HTTP – 80

Протокол HTTP

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

1. *Стартовая строка* (Starting line) — определяет тип сообщения
2. *Заголовки* (Headers) — характеризуют тело сообщения, параметры передачи и прочие сведения
3. *Тело сообщения* (Message Body) — непосредственно данные сообщения. Обязательно должно отделяться от заголовков пустой строкой

Тело сообщения может отсутствовать, но стартовая строка и заголовок являются обязательными элементами

Стартовая строка запроса выглядит так:
`Метод URI HTTP/Версия`

- *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

Чтобы запросить страницу с википедии, достаточно такого запроса:

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

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

- *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

Например, стартовая строка ответа сервера на запрос слева может выглядеть так:

```
HTTP/1.0 200 OK
```

Статус коды

Существуют множество кодов ответов на HTTP-запрос, которые делятся на пять групп по первой цифре:

Код	Класс	Назначение
1xx	Информационный(англ. **informational**)	Информирование о процессе передачи.
2xx	Успех(англ. **Success**)	Информирование о случаях успешного принятия и обработки запроса клиента. В зависимости от статуса, сервер может ещё передать заголовки и тело сообщения.
3xx	Перенаправление(англ. **Redirection**)	Сообщает клиенту, что для успешного выполнения операции необходимо сделать другой запрос (как правило по другому URI). Адрес, по которому клиенту следует произвести запрос, сервер указывает в заголовке Location.
4xx	Ошибка клиента(англ. **Client Error**)	Указание ошибок со стороны клиента.
5xx	Ошибка сервера(англ. **Server Error**)	Информирование о случаях неудачного выполнения операции по вине сервера.

200 – Запрос успешно выполнен

404 – Страница не найдена
403 – Доступ запрещён
400 – Ошибка в текстке запроса

500 – внутренняя ошибка сервиса при обработке запроса

Заголовки

Заголовки описывают некоторые дополнительные параметры запроса/ответа

Примеры заголовков используемых и в запросе, и в ответе:

- `Content-Length`: размер тела запроса или ответа в байтах
- `Content-Type`: MIME-тип данных, которые передаются в теле, например `Content-Type: image/jpeg` (телу не обязательно быть текстовым, оно может состоять и из сырых байтов)

Примеры заголовков используемых только в запросе:

- `Accept-Language`: список языков, на которых допустимо получить ответ, `Accept-Language: ru`
- `User-Agent`: строка с описанием программы, которая отправляет запрос, `User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/101.0.4951.67 Safari/537.36`

Примеры заголовков используемых только в ответе:

- `Server`: строка с информацией о сервере и версиях программ, которые он использует, `Server: Apache/2.2.17 (Win32) PHP/5.3.5`

Типы запросов

Мы уже видели тип запроса `GET`

Такой запрос посылается браузером, когда вы пытаетесь открыть в нём любую страницу и подразумевает то, что вы хотите получить от сервера ресурс по указанному пути

Обычно, `GET`-запрос не содержит тела

Также, есть `POST`-запрос – отправка некоторых данных на сервер по адресу

`POST`-запрос, напротив, чаще содержит тело и используется для отправки данных на сервер
Например – форма регистрации на сайте

Типы запросов этим не исчерпываются, есть ещё: `OPTIONS`, `HEAD`, `PUT`, `PATCH`, `DELETE`, `TRACE`, `CONNECT`

Версии HTTP

В стартовой строке запроса указывается версия протокола

А какие бывают отличия между версиями?

В версии `1.1`, по сравнению с `1.0`, есть такие интересные нововведения:

Обязательно требуется указывать заголовок `Host`, в котором должно быть указано имя сайта (HTTP-запросы присылаются по IP-адресу, на одном адресе может быть несколько сайтов, так указывается конкретный сайт)

Появилась возможность не закрывать TCP-соединение после первого запроса-ответа, а продолжить общение в том же соединении

Для изъявления желания оставлять соединение открытым, передаётся заголовок `Connection: keep-alive`
Используется как в запросах, так и в ответах

Аутентификация

Допустим, нам хочется добавить в сервис специальный запрос, который доступен только администратору

Как в этот запрос принимать и передавать администраторские логин и пароль?

Самый базовый способ – прописать их в адресную строку запроса:

```
admin:password@google.ru
```

В таком случае при отправке запроса учётные данные закодируются в Base64
и будут добавлены к запросу в виде заголовка:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

На самом деле, современные браузеры в целях безопасности игнорируют и стирают учетные данные, прописанные в адресной строке, поэтому заголовок придётся добавлять вручную

Помимо `Basic`, в этом заголовке можно передавать и другие, более продвинутые, способы авторизации

Сети в Java

Как работать с сетью в Java?

Для сетевого взаимодействия используется понятие сокета (socket – розетка, разъём)

Сокет в Java представляется классом [java.net.Socket]:

```
public class Socket implements java.io.Closeable
```

Socket, по сути, представляет собой открытое TCP-соединение, по которому можно принимать и отправлять данные

Для UDP-сокета есть совсем другой класс DatagramSocket

Для открытия клиентского соединения достаточно просто использовать конструктор от хоста и порта:

```
Socket clientSocket = new Socket("google.com", 80);
```

Можно передать как IP-адрес, так и доменное имя сайта

Socket

```
Socket clientSocket = new Socket("google.com", 80);
```

Для ввода-вывода с открытым сокетом, есть методы, возвращающие потоки для этого сокета:

```
public InputStream getInputStream() throws IOException
public OutputStream getOutputStream() throws IOException
```

Теперь, можем попробовать вручную отправить HTTP GET-запрос:

```
Socket clientSocket = new Socket("google.com", 80);
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

out.println("GET / HTTP/1.0");
out.println();

in.lines().forEach(System.out::println);
```

ServerSocket

Роль клиента брать на себя научились, пора побыть сервером

Для создания TCP-сервера есть класс [java.net.ServerSocket]():

```
public class ServerSocket implements java.io.Closeable
```

`ServerSocket` открывает некоторый порт для прослушивания и принимает на нём клиентов:

```
ServerSocket serverSocket = new ServerSocket(7777);
```

Метод `accept()` блокирует исполнение до появления следующего клиента
и возвращает `Socket` соединения с ним:

```
Socket clientSocket = serverSocket.accept();
```

Дальше общение с клиентом идёт по знакомой схеме:

```
PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
```