Создание Web-сервисов на Python

Московский физико-технический институт и Mail.Ru Group

Неделя 1

Содержание

1. Общее представление о WEB

1.1. Общее теоретическое представление о WEB

На раннем этапе развития сетей оборудование одного производителя не могло работать с оборудованием другого производителя:

- оборудование было несоместимо;
- ПО было несовместимо;
- были различными протоколы для передачи информации.

Необходимо было ввести стандарты. Просто создать сеть, которая объединяла два компьютера, но представьте, что вам необходимо построить сеть в которой объединяются все компьютеры в мире. Такая сеть должна быть надежной и работать, даже если некоторые узлы выйдут из строя; сеть должна иметь возможность расти, например, подключать новые страны. Она должна обеспечить качество обслуживания: скачать без проблем большие файлы или передавать данные без задержек. В этой сети необходимо обеспечить безопасность передачи данных.

Для решения такой сложной задачи ее необходимо декомпозировать. Компьютерные сети разделили на несколько уровней взаимодействия. Каждый уровень решает одну или несколько связанных задач; каждый нижележащий уровень предоставляет интерфейс вышележащему уровню.

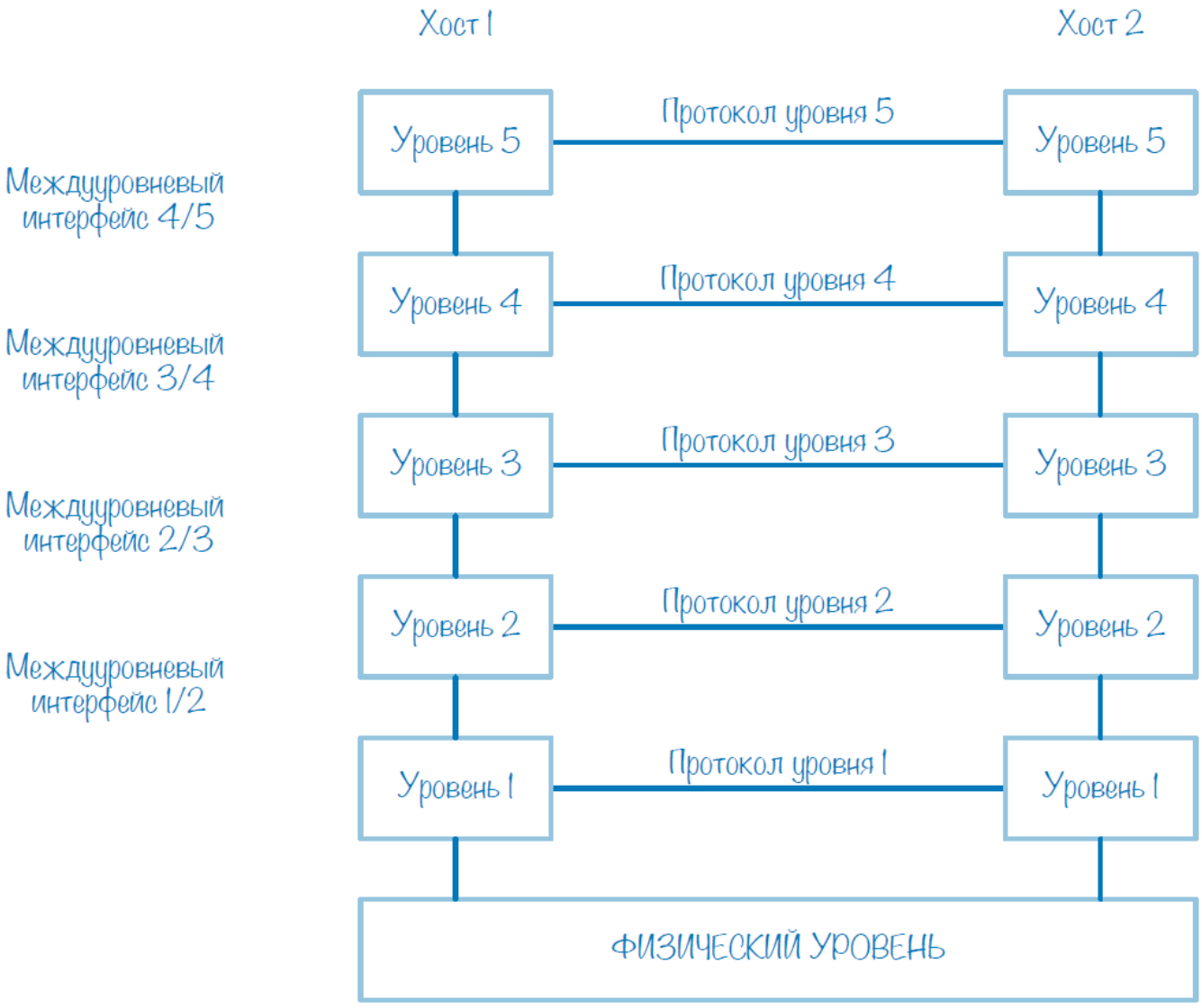

Рис. 1

Каждый уровень решает конкретную задачу, выполняется изоляция уровней друг от друга: если что-то меняется на одном из уровней, то поменяется только этот уровень и на остальных уровнях всё остается без изменений.

1.1.1. Термины многоуровневой модели

Интерфейс – набор примитивных операций, которые нижний уровень предоставляет верхнему. На схеме уровень 3 предоставляет интерфейс уровню 4.
Протокол – это правила и соглашения, используемые для связи уровня N одного компьютера с уровнем N другого компьютера. На схеме уровень 4 хоста 1 общается с 4-ым уровнем хоста 2 по протоколу 4-го уровня.

Пользователи и вышестоящие уровни взаимодействуют с интерфейсом уровня, то есть уровень предоставляет некоторый набор функций. Протокол же является реализацией этого взаимодействия. Это означает, что можно заменить один протокол другим и на вышестоящих уровнях ничего менять не понадобиться.
Также важным понятием является понятие **инкапсуляции** – включение сообщений вышестоящего уровня в сообщение нижестоящего уровня.

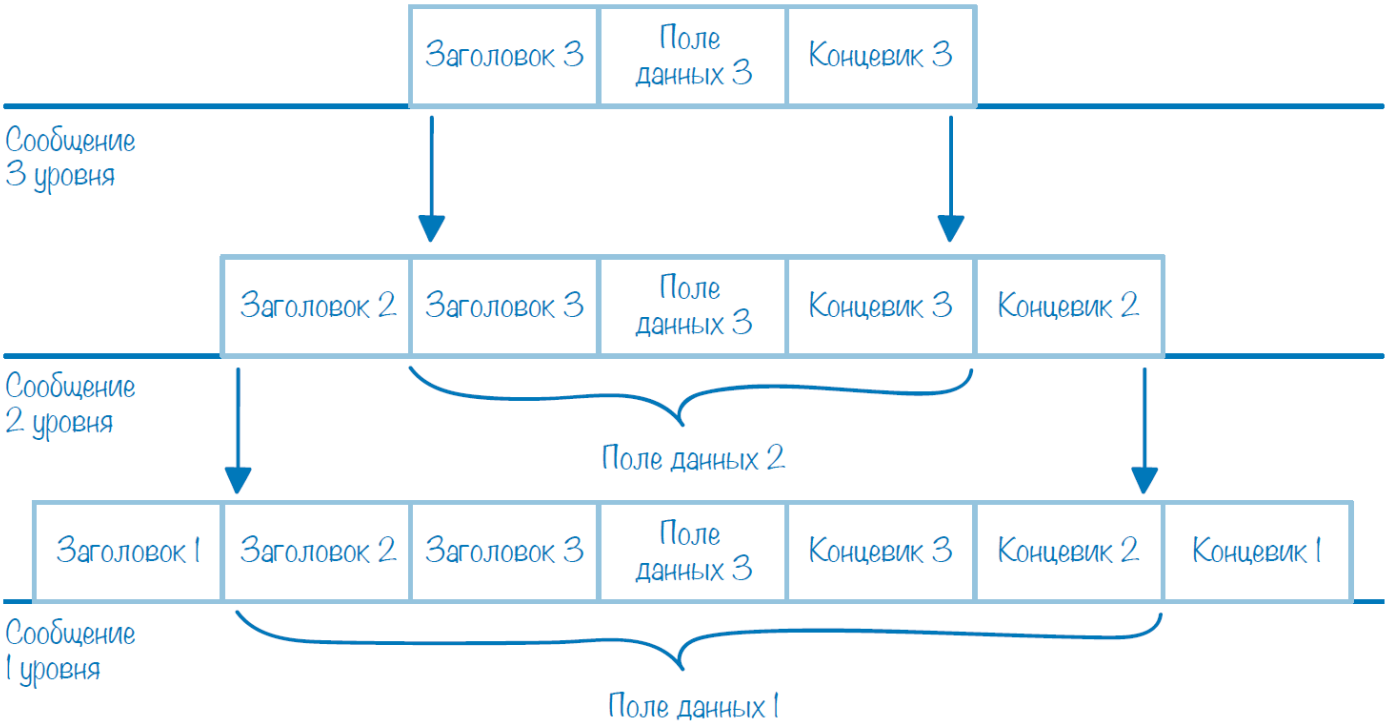

Рис. 2

Сообщение состоит из заголовка данных и концевика.
Допустим, мы хотим отправить сообщение от одного хоста к другому. Формируем сообщение и передаем его третьему уровню. Наше сообщение помещается в поле данных, а также обрамляется заголовком и концевиком третьего уровня; после этого передается на второй уровень. Сообщение 3 уровня помещается в поле данных, а также обрамляется заголовком и концевиком второго уровня, и так далее. К самому первому уровню сообщение обрастает заголовками и концевиками всех уровней; после этого сообщения передаются по каналам связи.
Когда сообщение приходит к адресату, оно передается на первый уровень. Первый уровень обрабатывает заголовок и концевик, извлекает сообщение из поля данных и передает его второму уровню. и т.д., пока мы не получим исходное сообщение.
Для решения задачи построения большой сети используется декомпозиция на уровни.
Остаются вопросы: сколько уровней должно быть в сети, какие это должны быть уровни, какие функции должны выполняться в этой сети и как они должны быть распределены по уровню. Это всё задается архитектурой сети. Можно придумать множество разных архитектур, но существуют общепринятые стандарты организации сетей. Существуют две основные модели:

1. модель взаимодействия открытых систем OSI;
2. модель TCP/IP.

1.2. Модель TCP/IP

Модель TCP включает 4 уровня: прикладной, транспортный, сетевой и уровень сетевых интерфейсов.

Уровни TCP/IP	
Прикладной уровень	HTTP, SMTP, FTP, SSH
Транспортный уровень	TCP, UDP
Сетевой уровень	IP, ICMP, OSPF, ARP
Уровень сетевых интерфейсов	Ethernet, Token ring

Рис. 3

Уровень сетевых интерфейсов (Network access layer) TCP/IP рассматривает любую подсеть, входящую в составную сеть, как средство транспортировки пакетов между двумя соседними узлами. Задачи этого уровня:

- упаковка IP-пакета в единицу передаваемых данных промежуточной сети;
- преобразования IP-адресов в адреса технологии данной промежуточной сети.

Пусть пакет пришел на роутер 3 и сеть, к которой подключен хост построена на основе Ethernet.

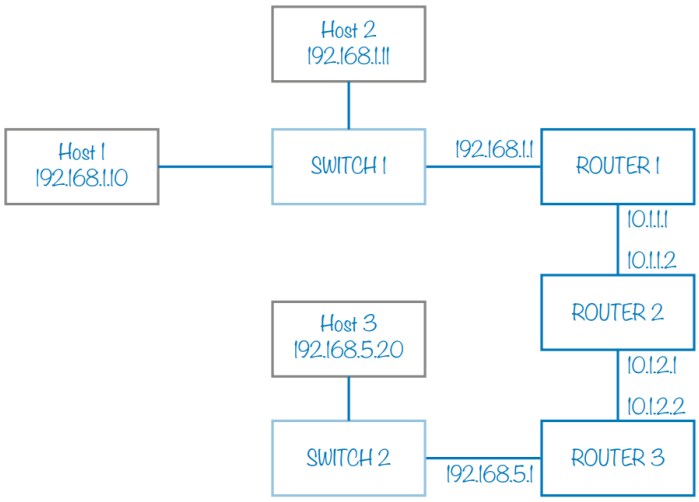

Рис. 4

Чтобы передать пакет от роутера 3 к хосту 3, на этом уровне произойдет преобразование IP-адреса хост 3 в адрес локальной технологии, его mac-адрес. После этого пакет доставится адресату.

Сетевой уровень (network layer) Уровень служит для образования единой транспортной системы, которая объединяет несколько сетей, причём эти сети могут использовать совершенно разные технологии передачи данных. Протоколы сетевого уровня поддерживают интерфейсы с уровнем выше, получая от него запросы на передачу данных по составой сети.

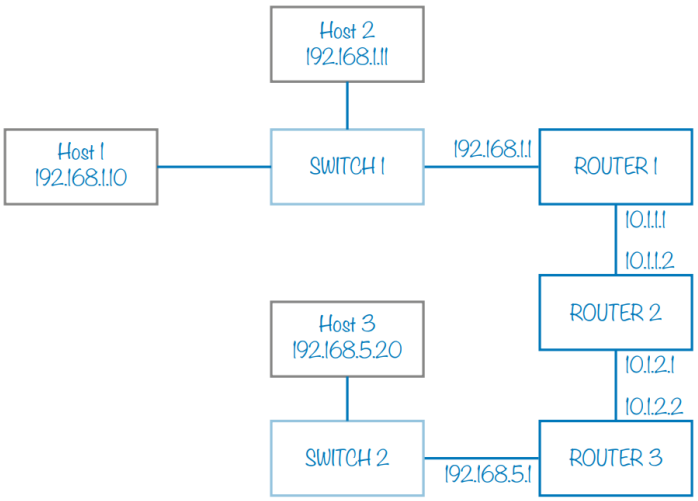

Рис. 5

Необходимо доставить сообщение от хоста 1 к хосту 3. Задача сетевого уровня – найти путь и доставить пакет от роутера 1 к роутеру 3. Доставкой от хоста 1 к роутеру 1 и от роутера 3 к хосту 3 занимается локальная технология, например, Ethernet. Примером протокола сетевого уровня является IP.

Транспортный уровень (Transport layer) Обеспечивает передачу данных между процессами. Особенностью транспортного уровня является управление надежностью. Уровень предоставляет приложениям или верхним уровням стека передачу данных с той степенью надежности, которая им требуется.

Например, нужно передать очень большой файл. Этот файл будет передаваться кусочками и необходимо обеспечить, чтобы эти все кусочки пришли в правильном порядке.
Примером протоколов транспортного уровня являяются TCP и UDP.

Прикладной уровень (Application layer) Набор разнообразных протоколов, с помощью которых пользователи сети получают доступ к разделяемым ресурсам. В модели считается, что если приложению нужны какие-то функции представления информации, например, шифрование, то оно должно само их реализовывать. Пример протоколов прикладного уровня: протокол HTTP или FTP.

Модель OSI

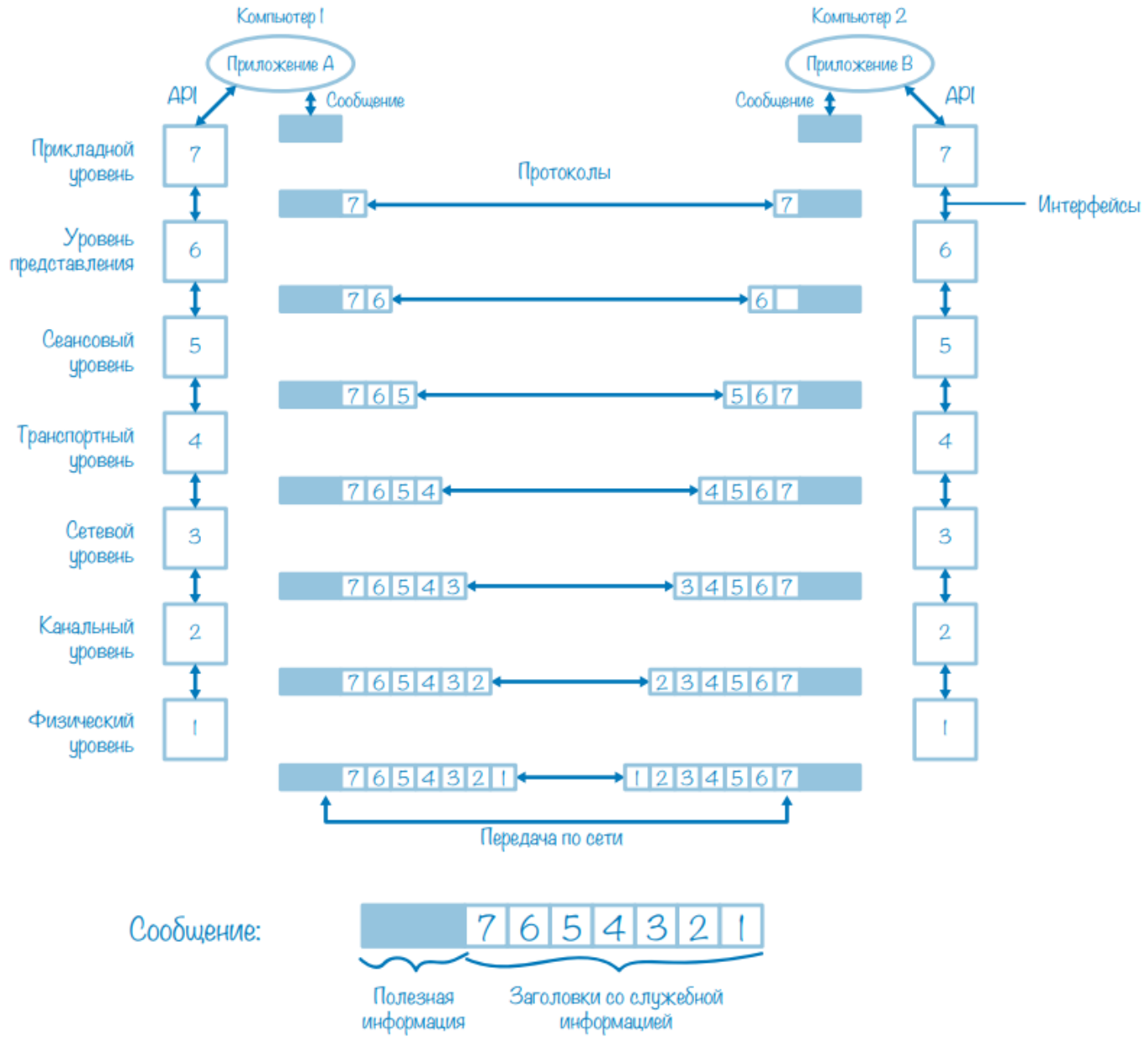

Рис. 6

Это модель взаимодействия открытых систем Open Systems Interconnection, стандарт, принятый организацией по стандартизации ISO.
Модель описывает семь уровней и назначение каждого из них. Обычно нумерация начинается с уровня, который находится ближе всего к среде передачи. Уровни называются так: физический, канальный, сетевой, транспортный, сеансовый, уровень представления и прикладной уровень.
Модели OSI и TCP/IP во многом похожи. Достоинством модели OSI является то, что она теоретически проработана. Она используется для описания различных типов сетей, но на практике широко не применяется. Достоинством модели TCP/IP является стек протоколов, который ши-

роко используется на практике и лежит в основе интернета. Модель теоретически не проработана и непригодна для описания сетей, где не работает стек протоколов TCP/IP.

1.3. Транспортный уровень и его протоколы

Задачи транспортного уровня:

- передача данных между процессами в сети
- предоставление различного уровня надежности передачи данных, независимо от надежности сети

Для адресации на транспортном уроне используются порты. Каждое приложение на хосте имеет свой порт. Номера портов не должны повторяться. Формат записи вместе с IP-адресом: 192.168.0.1:8080

Пример

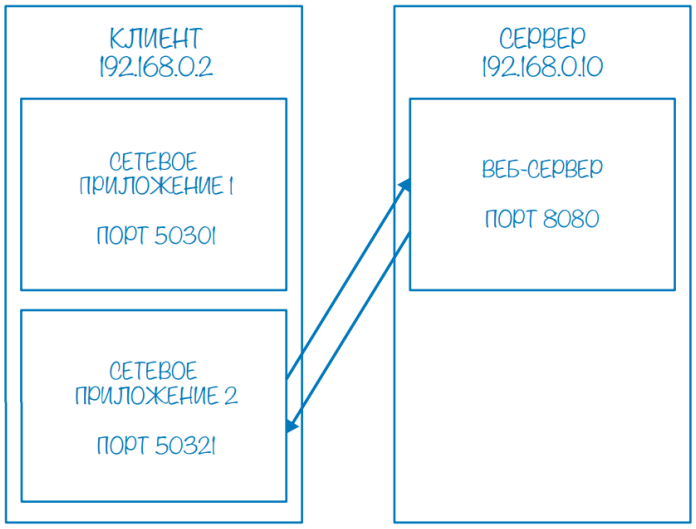

Рис. 7

Есь веб-сервер с IP-адресом 192.168.0.10, который работает на 8080 порту, есть клиент, который хочет подключиться к веб-серверу. Клиент открывает сетевое приложение (браузер), ОС назначает ему порт 50301, приложение открывает соединение с сервером и запрашивает веб-страницу. Если клиент откроет ещё одно сетевое приложение, то ОС назначит ему уже другой порт 50321. Приложение открывает соединение с веб-сервером и запрашивает веб-страницу. Когда от сервера придут данные, на основе порта ОС определит, какому приложению они предназначены.

1.3.1. Протокол UDP (User Datagram Protocol)

Особенности UDP:

- Работает без установления логического соединения;
- Нет гарантии доставки данных;
- Нет гарантии сохранения исходного порядка дейтаграмм;
- Гарантирует корректность данных внутри одной дейтаграммы.

Формат заголовка UDP дейтаграммы:

Рис. 8

Дейтаграммы могут иметь различную длину, не превышающую длину поля данных протокола IP. Протокол максимально простой и практически не имеет накладных расходов на передачу данных, таких как дополнительной информации для проверки корректности данных. Используется в случае, когда минимальная задержка при передаче данных очень важна, при этом возможна потеря небольшого количества данных.

1.3.2. Протокол TCP (Transmission control protocol)

Transmission Control protocol – протокол управления передачей, является надежным протоколом передачи данных. Особенности TCP:

- Работает с установления логического соединения;
- Гарантирует доставку данных;
- Гарантирует сохранения порядка следования пакетов.

Формат заголовка TCP-пакета:

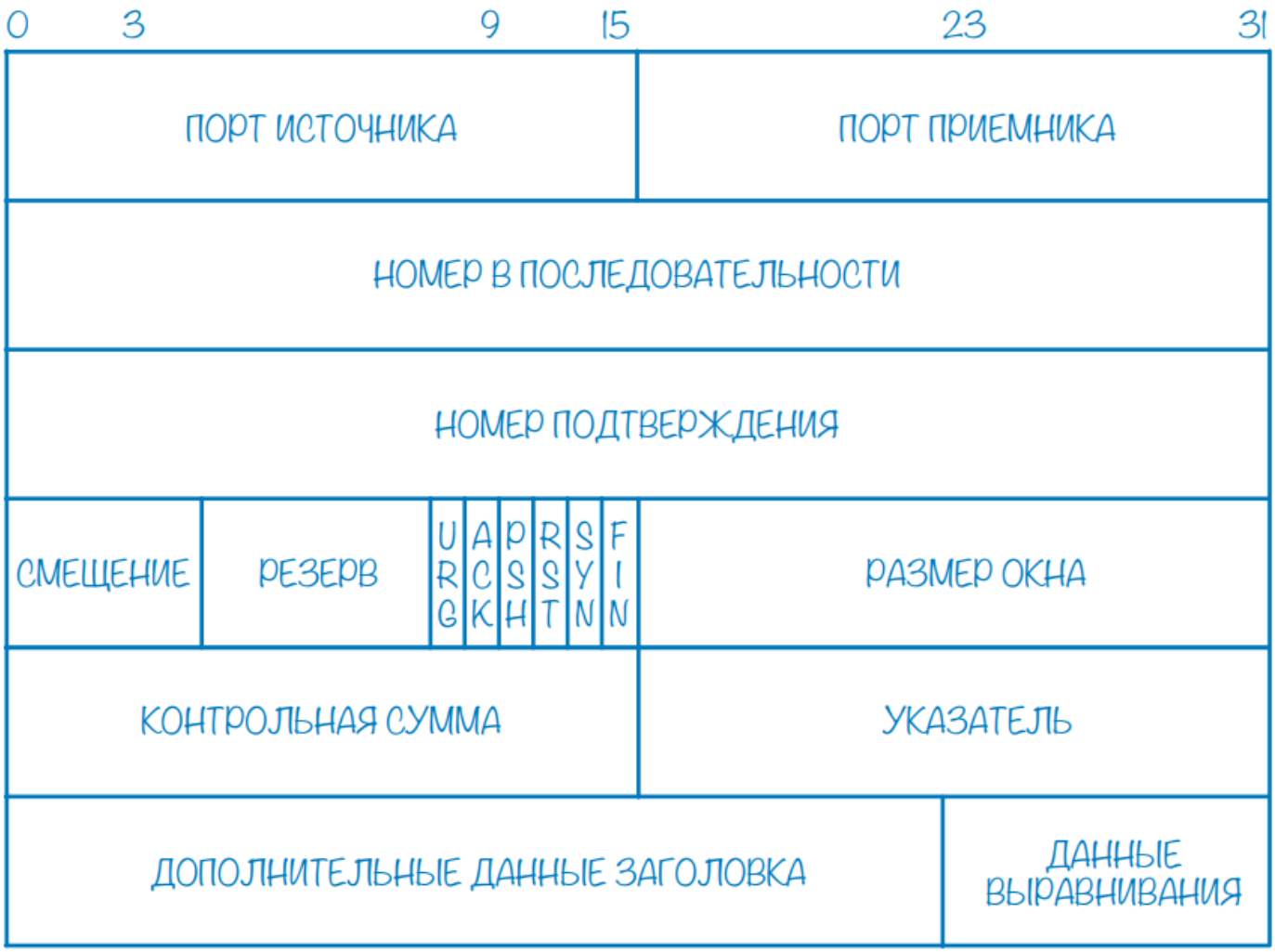

Рис. 9

Рассмотрим поля, которые понадобится для понимания процессов логического соединения.
Поле номер в последовательности или sequence number: 32-битовое поле, содержимое которого определяет положение данных TCP-пакета внутри исходящего потока данных. В момент установления логического соединения каждый из двух партнеров генерирует свой начальный номер последовательности, партнеры обмениваются начальными номерами и подтверждают их получение.
Номер подтверждения (acknowledgment number): 32-битовое поле, которое определяет количество принятых данных исходящего потока.

1.3.3. Логическое соединение

Для надежной передачи данных между двумя процессами необходимо установить логическое соединение. «Соединение» является договоренностью о параметрах между двумя процессами. Взаимодействие партнеров с использованием протокола TCP строится в три этапа:

- установление логического соединения;
- обмен данными;
- закрытие соединения.

Дайте рассмотрим процесс установления логического соединения. Пакеты представлены тремя полями: номер в последовательности, номер подтверждения и флаги, а также числом, характеризующим длину пакета (реально такого поля нет).

Рис. 10

На первом шаге A играет роль клиента, а B – сервера. A посылает B пакет с установленным флагом SYN и начальным значением номера в последовательности 1000. Флаг SYN означает, что пакет является запросом на установление логического соединения. B готов установить соединение и отвечает пакетом, который подтверждает правильный прием запроса: поле номер подтверждения на 1 больше начального номера в последовательности, и с флагом ACK (информирует о готовности установить соединение). Установлен флаг SYN и значение 3000 начального номера в последовательности. Флаг ACK означает, что TCP-пакет содержит в поле номер подтверждения верные данные.
На третьем шаге A подтверждает правильность приема TCP-пакета от B.
Рассмотрим процесс обмена данными.

Рис. 11

TCP-модуль, который принимает данные, всегда подтверждает их прием, вычисляя значение поля номер подтверждения в заголовке ответного пакета как сумму пришедшего номера в последовательности и длины правильно принятых данных. Посылка данных и подтверждение принятых от него данных реализуется в рамках одного TCP-пакета.
Рассмотрим процесс закрытия соединение по инициативе A.

Рис. 12

A посылает партнеру пакет с установленным флагом FIN. FIN означает, что TCP-пакет представляет собой запрос на закрытие логического соединения и является признаком конца потока

данных. Когда B от A принимает запрос на закрытие соединения B отправляет пакет подтверждения, в котором номер подтверждения на 1 больше значения принятого номера последовательности. После этого посылка данных от A невозможна, но B имеет данные для передачи A и получает подтверждение. Затем B формирует пакет с флагом FIN. После подтверждения его приёма соединение считается закрытым.

1.4. tcpdump + nc + telnet

Воспользуемся системной утилитой nc (netcut) и начнем слушать на IP-адресе 127.0.0.1, порт 3000. Подключимся с помощью системной утилиты telnet к IP-адресу 127.0.0.1 к 3000 порту.

Соединение дуплексное: можем отправлять данные в две стороны.

Используем утилиту tcpdump.

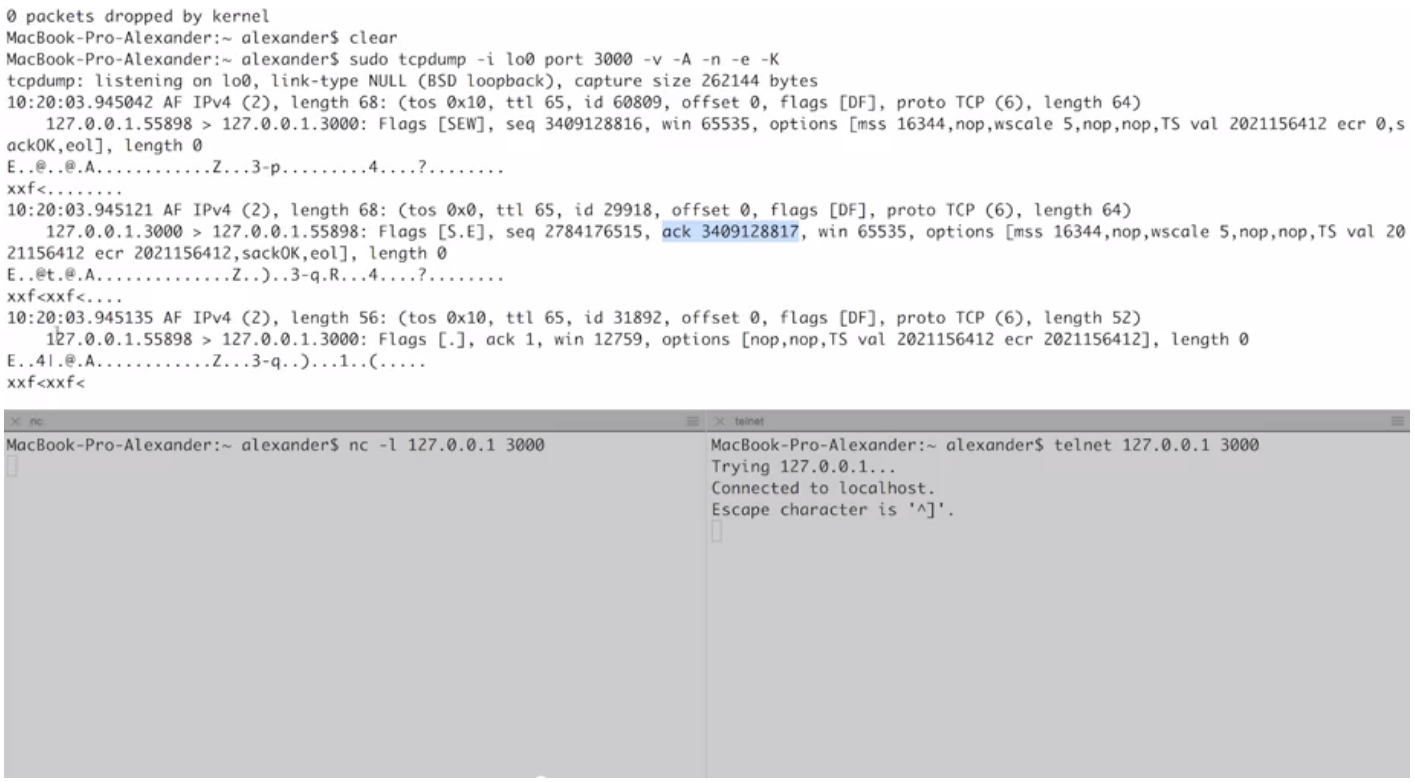

Рис. 13

Первая строка – это IP-пакет, версия протокола v4 и длина 68. Пришли данные от 127.0.0.1 к 127.0.0.1.3000 с флагами S и некоторыми служебными флагами. Установили номер в последовательности 3409128816, win – это размер окна; видим что длина данных (length) равна нулю. То есть мы ничего не передали.

Следующий пакет – ответ на соединение с флагом S (SYN) и точкой (это ACK). Нам сообщают, что готовы к установления логического соединения. Номер в последовательности ack на 1 больше предыдущего номера в последовательности seq.

Далее telnet посылает подтверждение об установлении соединения, после этого соединение считается установленным.
Теперь отправим данные.

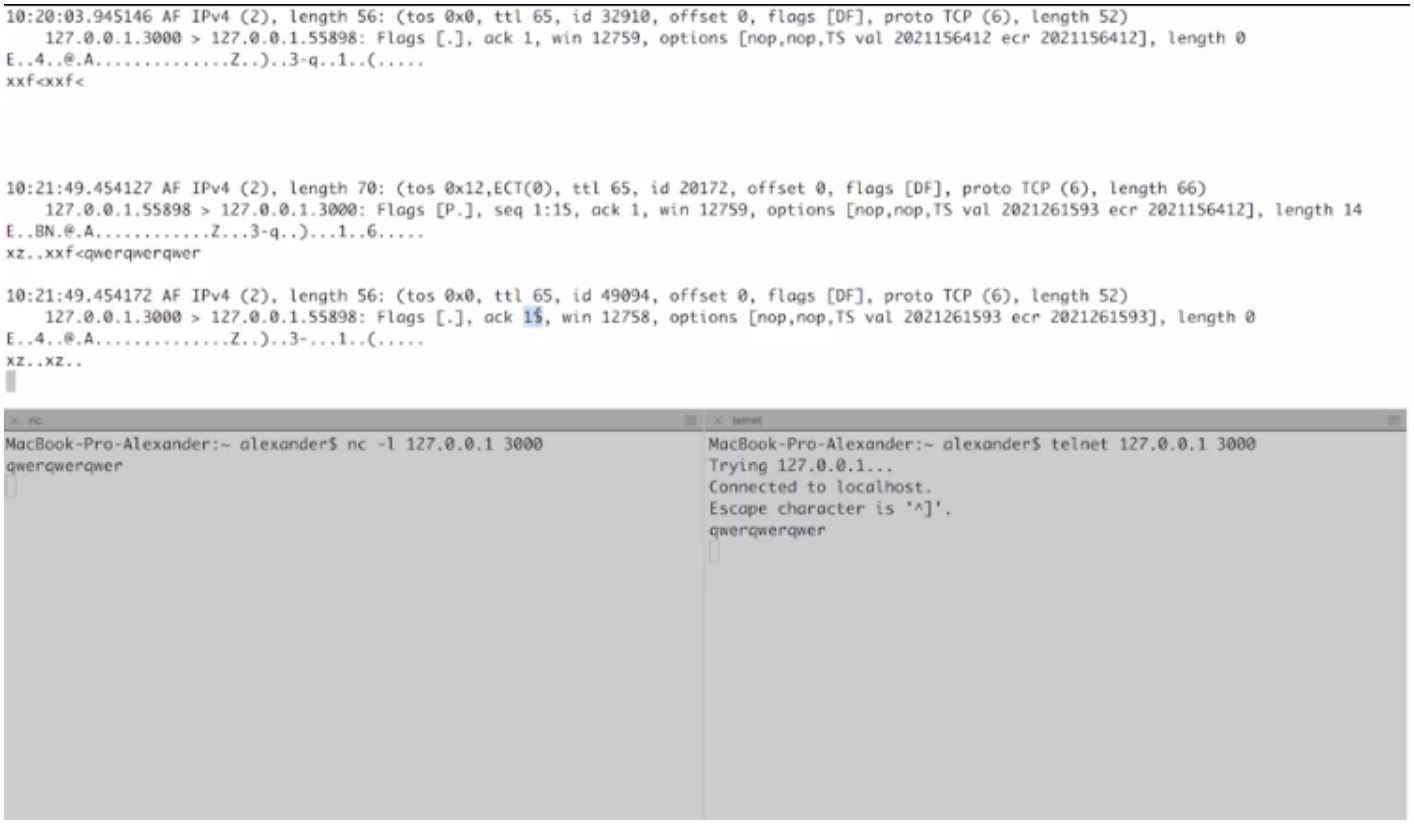

Рис. 14

Приходит IP-пакет размером 70, внутри него пакет TCP. Здесь указаны относительные первого числа цифры. Передаем данные от telnet к netcut, то есть от какого-то порта к 3000 размером 14 байт. Номер в последовательности это 1-15. Далее нам приходит пакет длиной 0, который является подтверждением того, что он принял 15 байт. Так же происходит если сделать передачу в обратном направлении.
Пусть мы хотим разорвать логическое соединение.

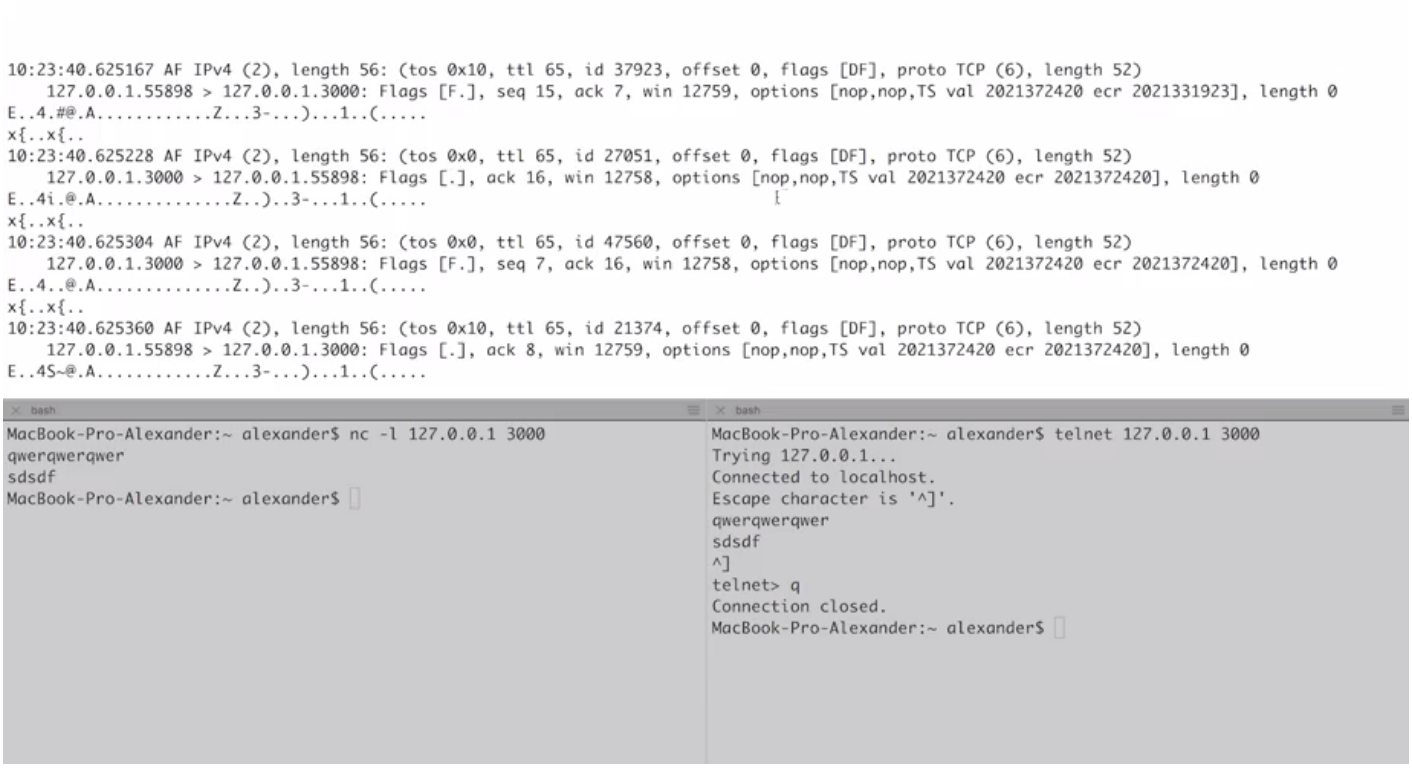

Рис. 15

Обменялись четыремя пакетами. Инициатор разорвать соединение: telnet (55898), он отправляет флаг F длиной 0. Ему отвечают TCP-пакетом ack также длиной 0. В свою очередь другой клиент также отправляет ему пакет FIN, и ему отвечают подтверждением. После этого можно считать что соединение закрыто.

Посмотрим, что происходит в случае UDP.

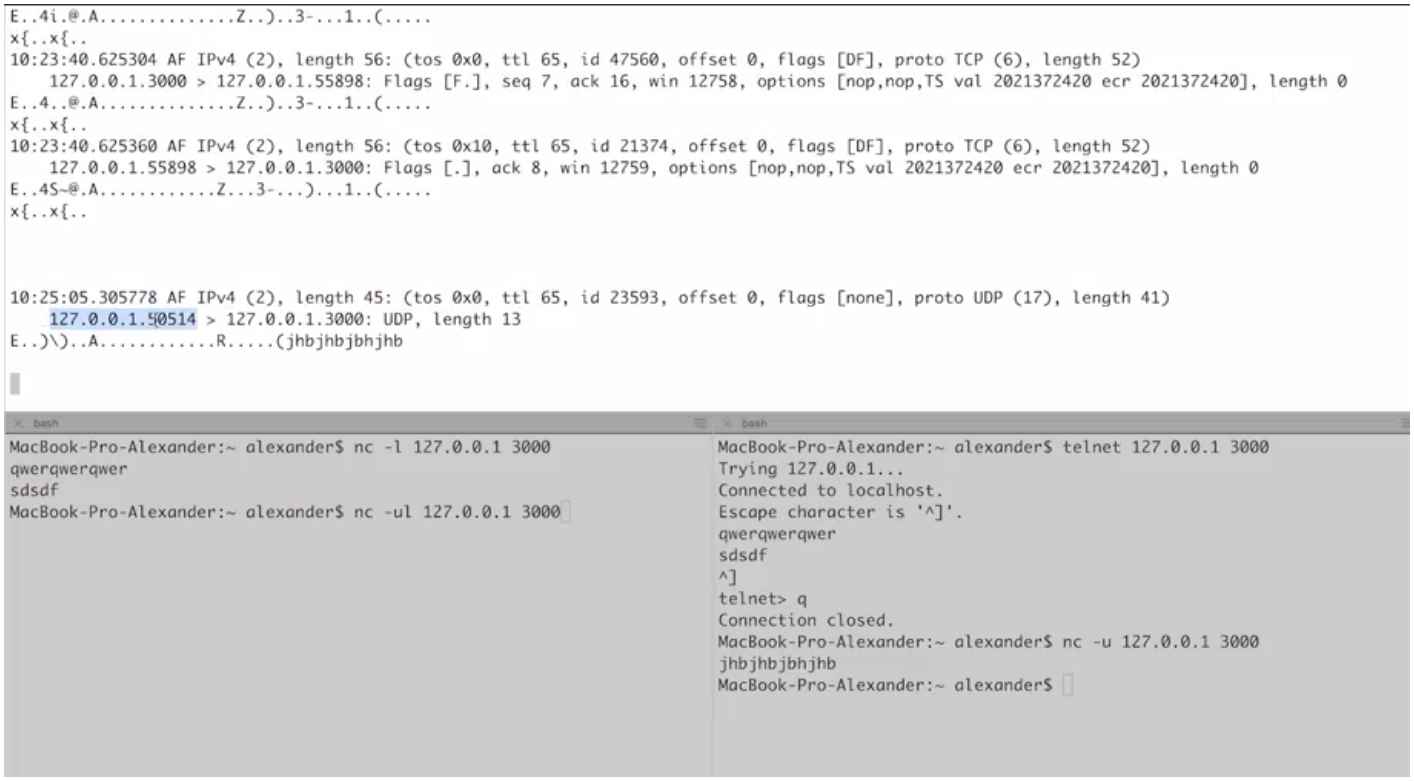

Рис. 16

Также начнём слушать на 3000м порту, но с флагом -u. Видим, что ничего не происходит: соединения не происходило, мы просто сделали вид, что подключились. Отправим какой-то кусок данных. Сам пришел всего один пакет, также в IP, внутри него завернут UDP-пакет. Он отправлен с порта 50514: просто порт, которая заняла системная утилита. Он пришёл в порт 3000 по протоколу UDP длиной 13 байт. Внутри видим переданные данные.

1.5. DNS-протокол

DNS – это распределенная система для получения информации о доменах; чаще всего используется для получения IP-адреса по имени хоста.
Характеристики DNS:

- Распределённость. Ответственность за разные части иерархической структуры несут разные люди или организации; каждый узел сети в обязательном порядке должен хранить только те данные, которые входят в его зону ответственности.
- Кеширование. Узел может хранить некоторое количество данных не из своей зоны ответственности для уменьшения нагрузки на сеть.
- Резервирование. За хранение и обслуживание своих узлов (зон) отвечают (обычно) несколько серверов.

1.5.1. Структура DNS

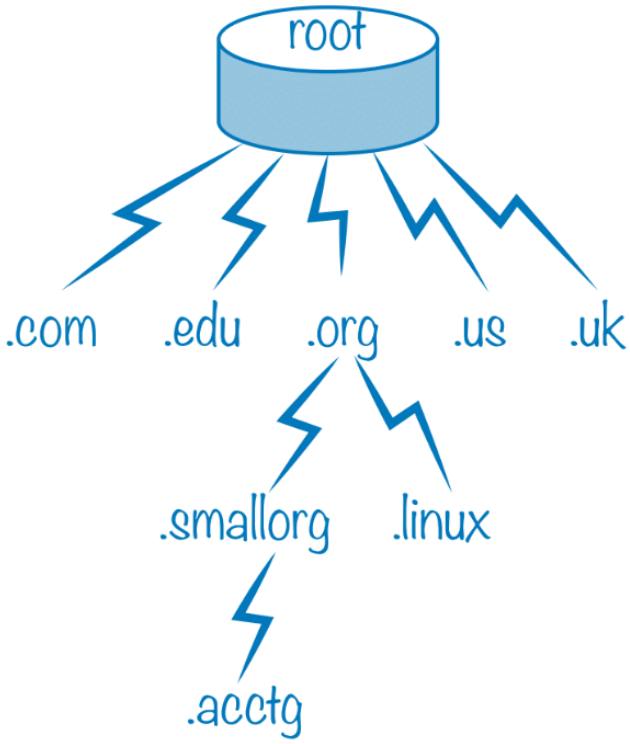

Рис. 17

Есть доменное имя linux.org. Узел верхнего уровня называется корнем (корневой узел не указывается напрямую в адресах). В корневом уровне сформировано несколько категорий, которые делят общую базу данных на части, называемые доменами. По мере дальнейшего роста сети все домены верхнего уровня были поделены на поддомены или зоны.

1.5.2. Схемы разрешения DNS-имен

Существует две схемы разрешения DNS имен: как DNS находит нужный DNS-сервер с записями.
Нерекурсивная. Клиент обращается к корневому DNS-серверу с указанием полного доменного имени, он отвечает клиенту, указывая адрес следующего DNS-сервера. Клиент делает запрос следующего DNS-сервера, который отсылает его к DNS-серверу нужного поддомена и т.д., пока не будет найден DNS-сервер, в котором хранится соответствие запрошенного имени IP-адресу.
Рекурсивная. Клиент запрашивает локальный DNS-сервер. Далее возможны два варианта действий: если локальный DNS-сервер знает ответ, то он сразу же возвращает его клиенту; если локальный сервер не знает ответ, то он выполняет итеративные запросы к корневому серверу и т.д. точно так же, как это делал клиент в предыдущем варианте, а получив ответ, передает его клиенту, который все это время просто ждет его от своего локального DNS-сервера.

1.5.3. Записи в базе данных DNS

DNS-сервер, отвечающий за имена хостов в своей зоне, должен хранить информацию о хостах в базе данных и выдавать ее по запросу с удаленных компьютеров. Обычно, база данных DNS представляет собой текстовый файл, состоящий из исходных записей RR (resource records). Виды записей:

- A-запись. Одна из самых важных записей. Указывает соответствие между IP-адресом и доменом. Формат: myhost.mycompany.com IN A 192.168.0.1
- AAAA-запись. Отображает хост-имя на адрес IPv6. Формат: myhost.mycompany.com IN AAAA 1234:1:2:3:4:567:89cd
- CNAME-запись. Каноническая запись, обычно используемая для алиасов (псевдонимов). Она позволяет отображать несколько хост-имен на заданный IP-адрес. Например, у нас есть несколько имен, которые указывают на один и тот же IP-адрес: необходимо добавить одну A-запись и для всех остальных псевдонимов добавить CNAME-записи.
- NS-запись. В каждой зоне должно быть по крайней мере два сервера DNS. Записи NS служат для их идентификации другими DNS-серверами, которые пытаются преобразовать имена хостов, относящихся к данной зоне.
- SOA-запись. Запись, которая определяет авторитетную информацию о DNS-зоне. Например, указываются параметры:
  - Primary Name Server – имя первичного DNS-сервера зоны;
  - Hostmaster – контактный адрес лица,пответственного за администрирование файла зоны;
  - Serial number – серийный номер файла зоны. 32-разрядное целое число, меняющееся при каждом обновлении зоны;
- PTR-запись. Запись позволяет быстро выполнять обратный поиск с помощью зоны обратного поиска (inaddr.arpa). Она считается обратной A-записью, но не похожа на нее ввиду использования inaddr.arpa в реальной записи. Такая запись для хоста syscrusher.skillet.com с IP-адресом 100.200.252.1 имеет формат: 1.252.200.100.in-addr.arpa IN PTR syscrusher.skillet.com
- MX-запись. С помощью этой записи удаленные сервера электронной почты узнают, куда отправить почту для вашей зоны.
- TXT-запись. Содержит тектовые данные любого вида.

Посмотрим как работает DNS с помощью утилиты dig. Можно посмотреть какие записи у домена mail.ru.

```
MacBook-Pro-Alexander:~ alexander$
MacBook-Pro-Alexander:~ alexander$ dig mail.ru

; <<>> DiG 9.8.3-P1 <<>> mail.ru
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 56141
;; flags: qr rd ra; QUERY: 1, ANSWER: 4, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;mail.ru.                       IN      A

;; ANSWER SECTION:
mail.ru.                35      IN      A       94.100.180.200
mail.ru.                35      IN      A       94.100.180.201
mail.ru.                35      IN      A       217.69.139.200
mail.ru.                35      IN      A       217.69.139.201

;; Query time: 8 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Nov 22 23:21:50 2017
;; MSG SIZE  rcvd: 89

MacBook-Pro-Alexander:~ alexander$
```

Рис. 18

Смотрим вывод команды. По дефолту команда dig делает запрос A-записей. DNS-ответ возвратил 4 A-записи. A-записей может быть больше, чем одна. Если мы обратимся к этим IP-адресам, мы получим доступ к ресурсам mail.ru.
Mail.ru – почтовый сервис, значит у него должна быть MX-запись. Делаем запрос.

```
mail.ru.                35      IN      A       94.100.180.200
mail.ru.                35      IN      A       94.100.180.201
mail.ru.                35      IN      A       217.69.139.200
mail.ru.                35      IN      A       217.69.139.201

;; Query time: 8 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Nov 22 23:21:50 2017
;; MSG SIZE  rcvd: 89

MacBook-Pro-Alexander:~ alexander$ dig mx mail.ru

; <<>> DiG 9.8.3-P1 <<>> mx mail.ru
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 28611
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;mail.ru.                       IN      MX

;; ANSWER SECTION:
mail.ru.                200     IN      MX      10 mxs.mail.ru.

;; Query time: 9 msec
;; SERVER: 8.8.8.8#53(8.8.8.8)
;; WHEN: Wed Nov 22 23:22:24 2017
;; MSG SIZE  rcvd: 45
```

Рис. 19

В ответе получаем MX-запись, которая равна mxs.mail.ru.
Теперь давайте посмотрим что же происходит под капотом, когда мы делаем dig mail.ru. Для этого добавил trace.

```
MacBook-Pro-Alexander:~ alexander$ dig +trace mail.ru

; <<>> DiG 9.8.3-P1 <<>> +trace mail.ru
;; global options: +cmd
.                       159875  IN      NS      a.root-servers.net.
.                       159875  IN      NS      b.root-servers.net.
.                       159875  IN      NS      c.root-servers.net.
.                       159875  IN      NS      d.root-servers.net.
.                       159875  IN      NS      e.root-servers.net.
.                       159875  IN      NS      f.root-servers.net.
.                       159875  IN      NS      g.root-servers.net.
.                       159875  IN      NS      h.root-servers.net.
.                       159875  IN      NS      i.root-servers.net.
.                       159875  IN      NS      j.root-servers.net.
.                       159875  IN      NS      k.root-servers.net.
.                       159875  IN      NS      l.root-servers.net.
.                       159875  IN      NS      m.root-servers.net.
;; Received 228 bytes from 8.8.8.8#53(8.8.8.8) in 14 ms

ru.                     172800  IN      NS      f.dns.ripn.net.
ru.                     172800  IN      NS      e.dns.ripn.net.
ru.                     172800  IN      NS      a.dns.ripn.net.
ru.                     172800  IN      NS      b.dns.ripn.net.
ru.                     172800  IN      NS      d.dns.ripn.net.
;; Received 337 bytes from 202.12.27.33#53(202.12.27.33) in 230 ms

MAIL.RU.                345600  IN      NS      ns1.mail.RU.
MAIL.RU.                345600  IN      NS      ns2.mail.RU.
MAIL.RU.                345600  IN      NS      ns3.mail.RU.
;; Received 151 bytes from 193.232.128.6#53(193.232.128.6) in 71 ms
```

Рис. 20

В самом начале мы обратились к 8888, и он нам отдал список DNS-серверов. Мы прошлись по ним и обратились к IP-адресу 202.12.27.33 на порт 53. Нам ответил m.root-servers.net, и так далее по иерархии до ns-серверов mail.ru, которые отвечают нам A-записями.

1.6. HTTP-протокол

HTTP – это протокол прикладного уровня для передачи гипертекста.

Все взаимодействие HTTP происходит между клиентом и сервером. Клиент – это конечный потребитель услуг сервера, а сервер – это поставщик услуг хранения обработки информации. Клиенту необходимо получить какие-то данные, он делает запрос к серверу, сервер подготавливает эти данные и отдает клиенту.

1.6.1. Схема HTTP-сеанса

HTTP работает поверх протокола TCP. Для начала необходимо установить соединение с сервером, после этого клиент формирует запрос и передает его на сервер, сервер как-то обрабатывает этот запрос и подготавливает данные, формирует ответ, отправляет этот ответ клиенту, клиент его получает и закрывает TCP-соединение. Протокол не хранит информацию о предыдущих запросах клиентов и ответах сервера,

1.6.2. URI, URL

Единообразный идентификатор ресурса URI (Uniform Resource Identifer) представляет собой короткую последовательность символов, идентифицирующую абстрактный или физический ресурс. URI не указывает на то, как получить ресурс, а только идентифицирует его. Один из самых известных примеров URI — это URL.
Для однозначной идентификации ресурсов в сети Веб используются уникальные идентификаторы URL (Uniform Resource Locator). Имеет следующую структуру:<схема>://<логин>:<пароль>@<хост>: <порт>/<URL-путь>

1.6.3. Структура HTTP-запроса

Теперь рассмотрим структуру HTTP-запроса.

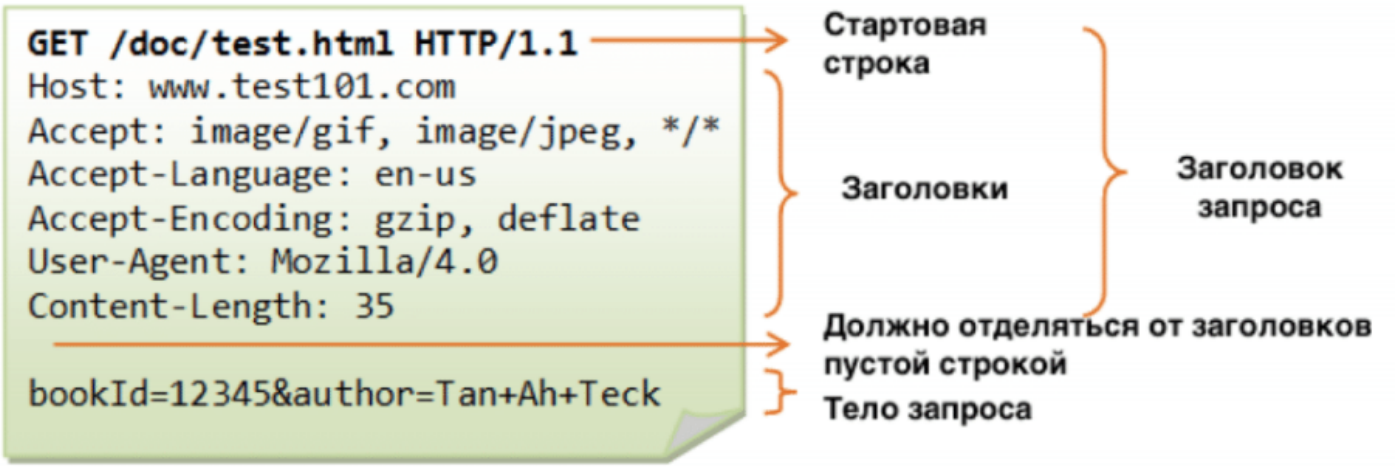

Рис. 21

Совокупность стартовой строки и заголовков называют заголовком запроса. После заголовка запроса идет пустая строка, затем идёт тело запроса.
Стартовая строка является обязательным элементом, так как указывает на тип запроса/ответа, заголовки и тело сообщения могут отсутствовать. Стартовые строки различаются для запроса и ответа. Строка запроса выглядит так: Метод Путь HTTP/Версия протокола.
Методы протокола:

- GET: используется для получения какой-то информации;
- HEAD: аналогично GET-запросу, но без тела запроса;
- POST: отсылает от клиента данные на сервер;
- PATCH: изменять отправленные данные;

- DELETE: удалить данные.
- ...

Поля заголовка, следующие за строкой состояния, позволяют уточнять запрос, т.е. передавать серверу дополнительную информацию. Поле заголовка имеет следующий формат: Имя_поля: Значение. Поля заголовка:

- Host — доменное имя или IP-адрес узла, к которому обращается клиент;
- Referer — URL, откуда перешел клиент;
- Accept — MIME-типы данных, обрабатываемых клиентом;
- Accept-Charset — перечень поддерживаемых кодировок;
- Content-Type — MIME-тип данных, содержащихся в теле запроса;
- Content-Length — число символов, содержащихся в теле запроса;
- Connection — используется для управления TCP-соединением;
- User-Agent — информация о клиенте.

1.6.4. MIME

Спецификация MIME (Multipurpose Internet Mail Extension — многоцелевое почтовоерасширение Internet) первоначально была разработана для того, чтобы обеспечить передачу различных форматов данных в составе электронных писем. До появления MIME устройства, взаимодействующие по протоколу HTTP, обменивались исключительно текстовой информацией. Для описания формата данных используются тип и подтип. Тип определяет, к какому классу относится формат содержимого HTTP-запроса или HTTP-ответа. Подтип уточняет формат (text/html, image/png). Данный спецификатор позволяет передавать от клиента к серверу различные типы данных.

1.6.5. Структура HTTP-ответа

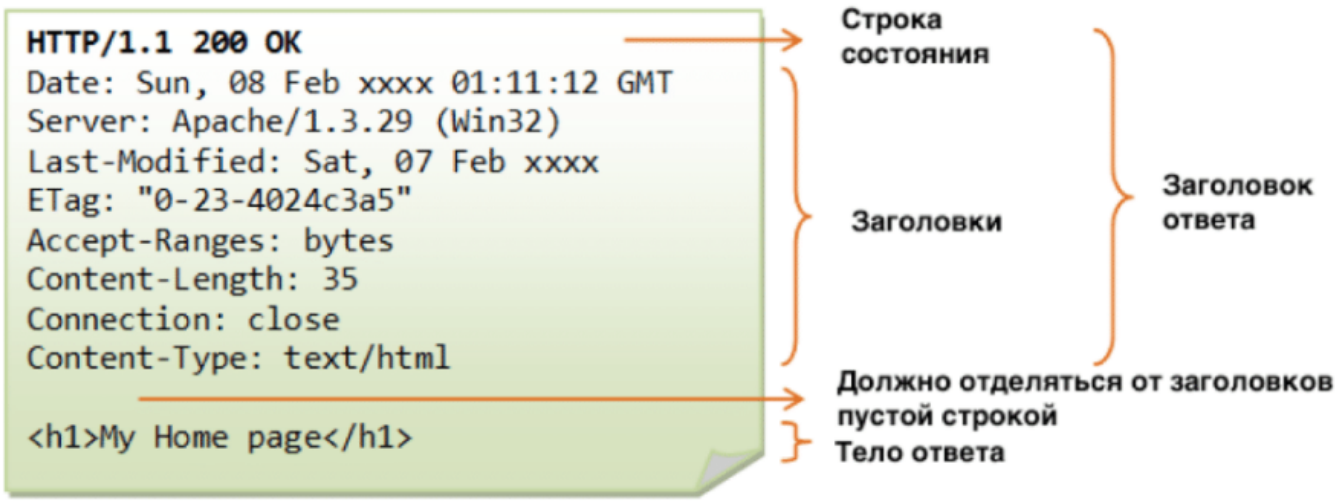

Рис. 22

В совокупности строка состояния и заголовки называются заголовком ответа. После этого идет пустая строка, после пустой строки идет тело ответа. Строка состояния состоит из версии протокола и статуса HTTP-ответа и расшифровки этого ответа. Существует 5 классов ответов:

- 1xx — специальный класс сообщений, называемых информационными. Означает, что сервер продолжает обработку запроса. (101 Switching Protocols);
- 2xx — успешная обработка запроса клиента. (200 Ok, 201 Created);
- 3xx — перенаправление запроса. (301 Moved Permanently, 302 Found);
- 4xx — ошибка клиента. (400 Bad Request, 403 Forbidden, 404 Not Found);
- 5xx — ошибка сервера. (500 Internal Server Error, 502 Bad Gateway).

Рассмотрим поля заголовка ответа:

- Server — имя и номер версии сервера;
- Allow — список методов, допустимых для данного ресурса;
- Content-Type — MIME-тип данных, содержащихся в теле ответа сервера;
- Content-Length — число символов, содержащихся в теле ответа сервера;
- Last-Modifed — дата и время последнего изменения ресурса;

- Expires — дата и время, когда информация станет устаревшей;
- Location — расположение ресурса;
- Cache-Control — директива управления кэшированием.

Давайте попробуем получить страницу example.com, для этого сделаем get-запрос на этот ресурс.

```
MacBook-Pro-Alexander:~ alexander$ telnet example.com 80
Trying 93.184.216.34...
Connected to example.com.
Escape character is '^]'.
GET / HTTP/1.1

HTTP/1.1 400 Bad Request
Content-Type: text/html
Content-Length: 349
Connection: close
Date: Wed, 22 Nov 2017 20:25:59 GMT
Server: ECSF (lga/13AF)

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
        <head>
                <title>400 - Bad Request</title>
        </head>
        <body>
                <h1>400 - Bad Request</h1>
        </body>
</html>
Connection closed by foreign host.
MacBook-Pro-Alexander:~ alexander$
```

Рис. 23

Передаем метод, путь и версию протокола HTTP. Нам пришел ответ 400 Bad request, то есть сервер посчитал, что нас запрос некорректный: у нас не хватает данных.
Теперь отправим корректный запрос.

```
Content-Length: 349
Connection: close
Date: Wed, 22 Nov 2017 20:25:59 GMT
Server: ECSF (lga/13AF)

<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
         "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
        <head>
                <title>400 - Bad Request</title>
        </head>
        <body>
                <h1>400 - Bad Request</h1>
        </body>
</html>
Connection closed by foreign host.
MacBook-Pro-Alexander:~ alexander$ cat ~/Desktop/request.txt
GET / HTTP/1.1
Host: example.com
Connection: keep-alive
Cache-Control: max-age=0
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrom
e/62.0.3202.94 Safari/537.36
Upgrade-Insecure-Requests: 1
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8
MacBook-Pro-Alexander:~ alexander$
MacBook-Pro-Alexander:~ alexander$
MacBook-Pro-Alexander:~ alexander$
MacBook-Pro-Alexander:~ alexander$
```

Рис. 24

Здесь мы указали стартовую строку, host, connection, etc. Отправим эти данные. Пришел успешный HTTP-ответ со статусом 200 с данными длиной 1270 символов и content-type HTML.

2. Работа с HTTP из python

2.1. Библиотека requests

Импортируем библиотеку и попробуем выполнить простой get-запрос.

```python
import requests

# Make a Request

r = requests.get('http://httpbin.org/get')
print(r.text)
```

В итоге мы получили HTTP-ответ со следующей структурой:


```
{
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "close",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.18.1"
  },
  "origin": "185.136.79.217",
  "url": "http://httpbin.org/get"
}
```


Допустим, мы хотим выполнить post-запрос.

```
r = requests.post('http://httpbin.org/post')
print(r.text)
```


```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {},

  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "close",
    "Content-Length": "0",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.18.1"
  },
  "json": null,
  "origin": "185.136.79.217",
  "url": "http://httpbin.org/post"

}
```


Чтобы передать Get-параметры, нам необходимо указать params.

```python
# Passing Parameters
payload = {'key1': 'value1', 'key2': 'value2'}
r = requests.get('http://httpbin.org/get', params=payload)
print(r.text)
```


```
{
  "args": {
    "key1": "value1",
    "key2": "value2"
  },

  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "close",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.18.1"
  },

  "origin": "185.136.79.217",
  "url": "http://httpbin.org/get?key1=value1&key2=value2"
}
```


Чтобы передать данные в Post Put Patch запросах, нам нужно передать в соответствующие методы поля data.

```
r = requests.put('http://httpbin.org/put', data = {'key':'value'})
print(r.text)
```


```
{
  "args": {},
  "data": "",
  "files": {},
  "form": {
    "key": "value"
  },

  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "close",
    "Content-Length": "9",
    "Content-Type": "application/x-www-form-urlencoded",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.18.1"
  },

  "json": null,
  "origin": "185.136.79.217",
  "url": "http://httpbin.org/put"
}
```


Если мы хотим передать json, это можно сделать двумя путями: руками реализовать json и передать в поле data, либо использовть встроенный в библиотеку механизм и передать поле json, которое сделает все само.

```
import json
url = 'http://httpbin.org/post'
r = requests.post(url, data=json.dumps({'key':'value'}))
r = requests.post(url, json={'key':'value'})
print(r.text)
```


```
{
  "args": {},
  "data": "{\"key\": \"value\"}",
  "files": {},
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "close",
    "Content-Length": "16",
    "Content-Type": "application/json",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.18.1"

  },

  "json": {
    "key": "value"
  },

  "origin": "185.136.79.217",
  "url": "http://httpbin.org/post"
}
```


В поле data пришел реализованный json, а в поле json отобразились ключ и значение.

Рассмотрим, как передать файл на сервер. Для этого необходимо объявить переменную files, которая будет являться словарём, где ключ – это ключ, по которому сможем получить файл на сервере. Дальше идет имя файла и его дескриптор. Потом эту переменную необходимо передать в поле files.

```
# POST a Multipart-Encoded File
url = 'http://httpbin.org/post'
files = {'file':
         ('test.txt',
          open('/Users/alexander/Desktop/test.txt',
               'rb'))}

r = requests.post(url, files=files)
print(r.text)
```


```
{
  "args": {},
  "data": "",
  "files": {
    "file": "test content\n"
  },
  "form": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "close",
    "Content-Length": "157",
    "Content-Type": "multipart/form-data; boundary=a6d397e696144b588e9a4aa1cff723fb",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.18.1"
  },
  "json": null,
  "origin": "185.136.79.217",
  "url": "http://httpbin.org/post"
}
```


Рассмотрим, как передать заголовки.

```python
# Headers
url = 'http://httpbin.org/get'
headers = {'user-agent': 'my-app/0.0.1'}

r = requests.get(url, headers=headers)
print(r.text)
```


```
{
  "args": {},

  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "close",
    "Host": "httpbin.org",
    "User-Agent": "my-app/0.0.1"

  },
  "origin": "185.136.79.217",
  "url": "http://httpbin.org/get"

}
```


Мы посмотрели, как же нам передать различные параметры в запрос, теперь давайте рассмотрим ответы. Существует несколько способов получения HTTP-ответа: текст, контент и json. Текст вернет данные с типом string, контент – массив байт, json – словарь.

```
# Response Content
r = requests.get('http://httpbin.org/get')
print(type(r.text), r.text)
print(type(r.content), r.content)
print(type(r.json()), r.json())
```

```
{

<class 'str'> {
  "args": {},
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "close",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.18.1"
  },
  "origin": "185.136.79.217",
  "url": "http://httpbin.org/get"
}

<class 'bytes'> b'{\n  "args": {}, \n  "headers": {\n    "Accept": "*/*", \n
"Accept-Encoding":
"gzip, deflate", \n    "Connection": "close", \n    "Host": "httpbin.org", \n
"User-Agent":
"python-requests/2.18.1"\n  }, \n  "origin": "185.136.79.217", \n  "url":
"http://httpbin.org/get"\n}\n'
<class 'dict'> {'args': {}, 'headers': {'Accept': '*/*', 'Accept-Encoding': 'gzip,
deflate',
'Connection': 'close', 'Host': 'httpbin.org', 'User-Agent': 'python-requests/2.18.1'},
'origin':
'185.136.79.217', 'url': 'http://httpbin.org/get'}
}
```

Также можно посмотреть статус ответа и можно сравнить этот ответ с набором ответов из библиотеки requests.

```
# # Response Status Codes
print(r.status_code)
print(r.status_code == requests.codes.ok)
```

```
200

True
```

Иногда очень удобно при некорректном HTTP-ответе получать исключение.

```
bad_r = requests.get('http://httpbin.org/status/404')
print(bad_r.status_code)
bad_r.raise_for_status()
```

```
404

---------------------------------------------------------------------------

HTTPError                                 Traceback (most recent call last)
<ipython-input-14-9b812f4c5860> in <module>()
      1 bad\_r = requests.get('http://httpbin.org/status/404')
      2 print(bad\_r.status\_code)
----> 3 bad\_r.raise\_for\_status()

/Users/alexander/Development/deep-completion/env/lib/python3.6/site-packages/
requests/models.py in raise\_for\_status(self)

    935
    936         if http\_error\_msg:
--> 937             raise HTTPError(http\_error\_msg, response=self)
    938
    939     def close(self):
httpError: 404 Client Error: NOT FOUND for url: http://httpbin.org/status/404
```

Можем получить заголовки HTTP-ответа.

```
# Response Headers
print(r.headers)
```

```
{'Connection': 'keep-alive', 'Server': 'meinheld/0.6.1', 'Date': 'Sun, 03 Dec 2017
08:46:02 GMT',
'Content-Type': 'application/json', 'Access-Control-Allow-Origin': '*',
'Access-Control-Allow-Credentials': 'true', 'X-Powered-By': 'Flask',
'X-Processed-Time':
'0.000767946243286', 'Content-Length': '267', 'Via': '1.1 vegur'}
```

Допустим мы хотим обратиться к github.com. Мы обращались к HTTP, но попали на HTTPS. Рассмотрим, почему так происходит: в history появилась история, что был ответ 301, то есть мы были перенаправлены.

```
# Redirection and History
r = requests.get('http://github.com')
print(r.url)
print(r.status_code)
print(r.history)
```

```
https://github.com/

200

[<Response [301]>]
```

Если нам не нужно такое поведение, нужно запретить перенаправление:

```
r = requests.get('http://github.com', allow_redirects=False)
print(r.status_code)
print(r.history)
```

```
301

[]
```

Теперь рассмотрим cookie. Это некоторые данные, которые будут передаваться в каждом запросе. Это может быть нужно, например, для авторизации. Для того, чтобы передать cookie, мы объявляем словарь, ключ которого – имя куки, а значение – значение куки.

```python
# Cookies

url = 'http://httpbin.org/cookies'
cookies = dict(cookies_are='working')
r = requests.get(url, cookies=cookies)
print(r.text)
```

```
{

  "cookies": {

    "cookies_are": "working"

  }

}
```

Когда мы объявляем сессию, все действия в ней будут выполняться с теми же параметрами, что и предыдущие. Например, мы объявили сессию и сделали в ней get-запрос. Сервер установил нам cookie. Все следующие запросы будут выполняться в этой сессии с этой cookie.

```python
# Session Objects
s = requests.Session()
s.get('http://httpbin.org/cookies/set/sessioncookie/123456789')
r = s.get('http://httpbin.org/cookies')
print(s.cookies)
print(r.text)
```

```
<RequestsCookieJar[<Cookie sessioncookie=123456789 for httpbin.org/>]>

{

  "cookies": {

    "sessioncookie": "123456789"

  }

}
```

То же самое происходит со всеми параметрами, например, заголовками.

```
s = requests.Session()
s.headers.update({'x-test': 'true'})
r = s.get('http://httpbin.org/headers', headers={'x-test2': 'true'})
print(r.text)
```


```
{
  "headers": {
    "Accept": "*/*",
    "Accept-Encoding": "gzip, deflate",
    "Connection": "close",
    "Host": "httpbin.org",
    "User-Agent": "python-requests/2.18.1",
    "X-Test": "true",
    "X-Test2": "true"
    }
}
```


3. Сбор данных со сторонних сайтов. Регулярные выражения

3.1. Введение в обработку данных

Зачем могут быть нужны данные со сторонних сайтов?

- Авторизация OAuth: VK, Facebook, и др: сначала регистрируем свою приложение на сайте ВКонтакте, получаем некоторые идентификационные данные. Затем, когда мы хотим авторизовать пользователя, мы передаем эти данные ВКонтакте, пользователь отмечает разрешение для нашего приложения, а мы обратно получаем данные пользователя. Мы обмениваемся данными с сайтом ВК;
- Автоматизация: курс, поиск, reCAPTCHA: мы хотим разместить капчу на сайте, чтобы отделить пользователей от ботов и запретить ботам публиковать комменты, показываем картинку recaptcha, пользователь вводит, что на ней изображено, а мы передаем эту строку на сайт recaptcha, обмениваясь с ним данными;
- Мониторинг сайта;
- Микросервисная архитектура: приложение состоит из небольших кусочков, которые обмениваются друг с другом данными по протоколу.

Давайте рассмотрим автоматизацию. Представим вынужден каждый день заходить на сайт Центробанка, чтобы узнать какой на сегодня курс евро. Давайте автоматизируем этот процесс. Мы импортировали модуль re, который реализует функционал регулярных выражений.

```
import requests
resp = requests.get('http://cbr.ru')
html = resp.text

import re
match = re.search(r'Евро\D+(\d+,\d+)', html)
rate = match.group(1)
rate
```

У этого модуля есть метод search, который позволяет найти шаблон внутри строки. В качестве строки мы будем использовать html-код сайта. Затем мы указываем наш шаблон: мы ищем слово евро, затем мы пропускаем всё, что не является цифрами (+ показывает, что в этом месте не может быть цифры и это может повториться несколько раз). Затем мы ставим круглые скобки, указывая на группу: всё что находится в круглых скобках, модуль запоминает, и потом мы сможем это достать. Здесь мы ищем нечто формата "цифра, запятая, цифра".
Регулярное выражение – это язык поиска и манипуляции с подстроками в тексте. Он состоит из символов и метасимволов. Символы обозначают сами себя, а также есть метасимволы, которые также называется символами-джокерами. Этот символ не обозначает сам себя, но показывает что в этом месте может находиться какой-то другой символ или не находиться другой символ. Например указывает, что после слова "евро" мы пропускаем все, что не является цифрой.
Регулярные выражения используется, когда мы не знаем точно, что ищем, но знаем, на что оно похоже. В нашем примере мы не знали, что курс евро сегодня будет 68 руб, но мы знали, что он находится где-то после слова евро и перед ним не могут быть другие цифры.
Регулярные выражения поддерживаются всеми современными языками программирования и многим текстовыми редакторами и утилитами. Они позволяют искать и заменять строки по каким-либо сложным условиям.

3.2. Поиск с помощью символьных выражений

3.2.1. Синтаксис регулярных выражений

Большинство букв и цифр означают просто сами себя.
Если перед буквой стоит обратный слэш, то как правило, это спецсимвол, который означает какое-то специальное действие. **\d** обозначает цифру от 0 до 9, **\D** означает, что в этом месте регулярного выражения может быть всё что угодно, кроме цифры.
Точка означает, что на ее месте может быть любой сивол, кроме перевода строки.
+ означает, что идущий перед ним символ повторяется один или более раз. * означает, что он повторяется 0 или более раз. ? означает 0 или один раз.
a.*c = ***a***b*c*d*a*b***c**d (от первой a до последней c, как можно больше символов между), a.*?c = ***a***b***c**d***a***b***c**d (от первой a до первой c, все вхождения a...c)
() обозначают группы. Все, что в них находится, регулярные выражения запоминют и потом это можно использовать в дальнейшем. Также группы могут быть вложены.

Примеры

- abc = abc;
- a\db = a0b, a1b, ..., a9b, ba1bc, но не подходят a11b, adb;
- a= azb, aab, a_B, A B, но не подходят a1b, a11b, ab, aaab;
- a.b = a0b, aab, a b, но не подходят ab, a11b, abc;
- a\d?b = ab, a5b, но не подходят a55b, acb, a?b;
- a.*b = ab, a123XYZb, a-b=b;
- a.*?b = ab, a123XYZb, a-b=b.

Возьмем строку из первого примера с евро.

```
import re
html = "Курс евро на сегодня 68,7514, курс евро на завтра 67,8901"
match = re.search(r'Евро\D+(\d+,\d+)', html, re.IGNORECASE)
rate = match.group(1)
rate
```

Без флага IGNORECASE ничего работать не будет, так как в строке ”евро” написано со строчной буквы, а в выражении – с заглавной.
Заменим \D+ на .*, и в нашей строке нашлась последняя часть курса, причём не полностью: от 67 у нас осталась только цифра 7. Мы ищем хотя бы одну цифру, поэтому нам достаточно такого числа.

```
re.search(r'Евро.*(\d+,\d+)', html, re.IGNORECASE).group(1)
```

Если мы добавим знак вопроса, то все снова будет хорошо.

```
re.search(r'Евро.*?(\d+,\d+)', html, re.IGNORECASE).group(1)
```

Давайте попробуем теперь найти использовать вместо re.search findall с теми же самыми регулярными выражениями. Мы найдем оба курса.

```
re.findall(r'Евро\D+(\d+,\d+)', html, re.IGNORECASE)
```

Findall вывел все, что у нас находится в группе. Если же групп в выражении не будет, findall выведет всё, что попало под регулярное выражение.

```
re.findall(r'Евро\D+\d+,\d+', html, re.IGNORECASE)
```

Если же групп в регулярном выражение будет несколько, findall вернет список кортежей. В каждом из кортежей будут все значения групп указанных выражений.

```
re.findall(r'Евро\D+(\d+),(\d+)', html, re.IGNORECASE)
```

Если же мы сделаем вложенные группы, также будут возвращены все группы. Группы нумеруется по открывающимся скобкам: первой группой являются внешние круглые скобки, затем идёт вторая группа и третья группа, в таком порядке они возвращаются.

```
re.findall(r'Евро\D+((\d+),(\d+))', html, re.IGNORECASE)
```

На сайте regex101 можно писать и разбирать регулярные выражения в режиме онлайн.

3.3. Символьные классы и квантификаторы

Иногда нам необходимо, чтобы в каком-то месте регулярного выражения был один символ из имеющегося у нас набора.

- **символьный класс**, один из множества = [abcd1234] = [a-d1-4]
- \d = [0123456789] = [0-9]; \D = обратный \d символьный класс = [^\D] = [0-9]
- {1,2} = **квантификатор**, предыдущий символ должен повторяться от одного до двух раз
- {2, 2} = предыдущий символ должен повторяться два раза
- {, 2} = предыдущий символ должен повторяться от нуля до двух раз
- {2, } = предыдущий символ должен повторяться два и более (до бесконечности) раз

Теперь попробуем найти все автомобильные номера внутри имеющегося у нас текста. Он начинается с буквы, причём не любой, а одной из 12 имеющих одинаковое написание в русском и латинском алфавите, потом ид три цифры, потом две буквы, всё из того же самого множества, затем две или три цифры регионов. Для этого отлично подойдет символьный класс.

```
import re
text = '''
Автомобиль с номером A123BC77 подрезал автомобиль
K654HE197, спровоцировав ДТП с участием еще двух
иномарок с номерами M542OP777 и O007OO77
```

```
    '''
    pattern = r'[АВЕКМНОРСТУХ]\d{3}[АВЕКМНОРСТУХ]{2}\d{2,3}'
    matches = re.findall(pattern, text)
    matches
```

Обсудим плюсы и минусы использование регулярных выражений.

- Во многих случаях компактно и просто, но бывают и плохочитаемыми;
- Штатный модуль Python, всегда доступен;
- Можно не только искать, но и заменять
- Подходят не для всего;
- Не слишком быстрые.

3.4. Сложный поиск и замена

Рассмотрим специальные символьные классы:

- \w = буква, цифра, _;
- \W = [\] = обратный , не "буквоцифра";
- \s = пробел [\f\n\r\t\v];
- \S = не пробел;
- \b = граница между \w и \W (пустая строка);
- \B = позиция внутри слова;
- ^= начало строки, $ = конец строки.

У нас есть список ников, которые мы хотим проверить. Считаем что в них могут быть только цифры, буквы и знак подчеркивания. Давайте попробуем написать регулярное выражение. Мы используем метод re.compile, который позволяет один раз скомпилировать регулярное выражение и затем несколько раз его использовать. Это удобно использовать, если оно встречается несколько раз в программе или используется внутри цикла, чтобы каждый раз оно не компилировалось заново.

```
import re
nicknames = ['sU3r_h4XX0r', 'alёna', 'ivan ivanovich']
reg = re.compile(r'\w+')
for nick in nicknames:
    print('{} nickname: "{}"'.format(
        'valid' if reg.match(nick) else 'invalid',
        nick
    ))
```

Когд мы запустим код, все ники будут валидны, хотя в алене есть ё, а в иване ивановиче – пробел. Дело в том, что мы написали матч который ищет соответствии с начала строки, но необязательно идет до конца строки. Исправить выражение можно так:

```
import re
nicknames = ['sU3r_h4XX0r', 'alёna', 'ivan ivanovich']
reg = re.compile(r'^\w+$', re.ASCII)

for nick in nicknames:
    print('{} nickname: "{}"'.format(
        'valid' if reg.match(nick) else 'invalid',
        nick
    ))
```

Теперь найдем в строке все слова, начинающиеся с большой буквы и заканчивающиеся на ”-на”.

```
import re
text = (
    'Анна и Лена загорали на берегу океана, '
    'когда к ним подошли Яна и Ильнар'
)

re.findall(r'[А-Я]\w*на', text)
```

В данном случае найдутся Анна, Лена, Яна и Ильна, потому что мы же не говорим о нашем регулярные выражения что это должны быть последние буквы слова. У нас просто после каких-то букв идет ”-на”. Тогда мы можем добавить границу слова в начало и в конец регулярного выражения, это будет означает что мы действительно ищем слово, которое начинается с большой буквой и заканчивается на ”-на”. Давайте попробуем.

```
re.findall(r'\b[А-Я]\w*на\b', text)
```

Если у нас будет Полина, написанная большими буквами, она не найдется: мы ищем слова оканчивающиеся на маленькие буквы. Давайте исправим. Попробуем написать группу, в которую входит как ”-на” маленькое, так и ”-на” большое.

```
import re
text = (
    'Анна и Елена загорали на берегу океана, '
    'когда к ним подошли ЯНА, ПОЛИна и Ильнар'
)
re.findall(r'\b[А-Я]\w*(на|НА)\b', text)
```

Нам вернутся только окончания слов, потому что findall, если находит группу внутри строки, возвращает только то, что попало в группу. Это можно исправить, используя специальный синтаксис ?:, это будет значить, что то, что внутри скобок группируется, но не запоминается.

```
re.findall(r'\b[А-Я]\w*(?:на|НА)\b', text)
```

Допустим у нас есть строка, "как защитить металл от процесса коррозии". В этих словах встречаются двойные буквы. Как их найти? Можно использовать обратные ссылки.

```
import re
text="Как защитить металл от процесса коррозии?"

re.findall(r'(\w)\1', text)
```

\1 будет ссылаться на первую группу, которой является любая буква или цифра, то есть в итоге мы получили двойную букву/цифру. После запуска кода мы получаем буквы л, с, р, и.
Теперь попробуем заменить все буквы а на знаки вопроса. Воспользуемся методом re.sub, который принимает регулярные выражения, текст для замены и текст, в котором мы заменяем и возвращает замененный текст.

```
re.sub(r'а', '?', text)
```

Теперь попробуем не просто найти все двойные буквы, а заменить их на заглавные версию. Для этого передаем в re.sub функцию, при помощи которой мы будем обращаться к тексту и переводить его в верхний регистр.

```
re.sub(r'(\w)\1', lambda r: r.group(0).upper(), text)
```

Что делать, если мы хотим найти все слова, в которых содержатся двойные буквы и выделить их уже целиком, а не просто найти сами двойные буквы? Что такое слова с двойными буквами? Это слово, которое начинается с границы слова потом может идти любое число букв до тех пор, пока не встретится двойная, потом идет это самая буква, а потом она ещё раз повторяется, и после неё идёт ещё сколько-то букв, и затем опять граница слова. Вот так можно найти слова с двойными буквами. Т.к. нам нужно всё слово целиком, мы его тоже заключаем в группу. Получается вот такой регулярное выражение.

```
    re.sub(r'\b(\w*(\w)\2\w*)\b', r'[\1]', text)
```

В метод sub мы можем передать группу, которую мы захватили ранее, в частности группа номер один – это у нас всё слово.