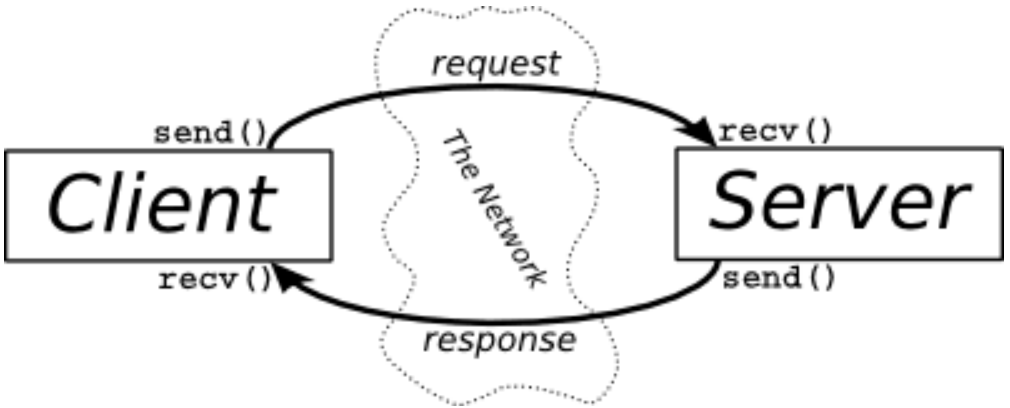
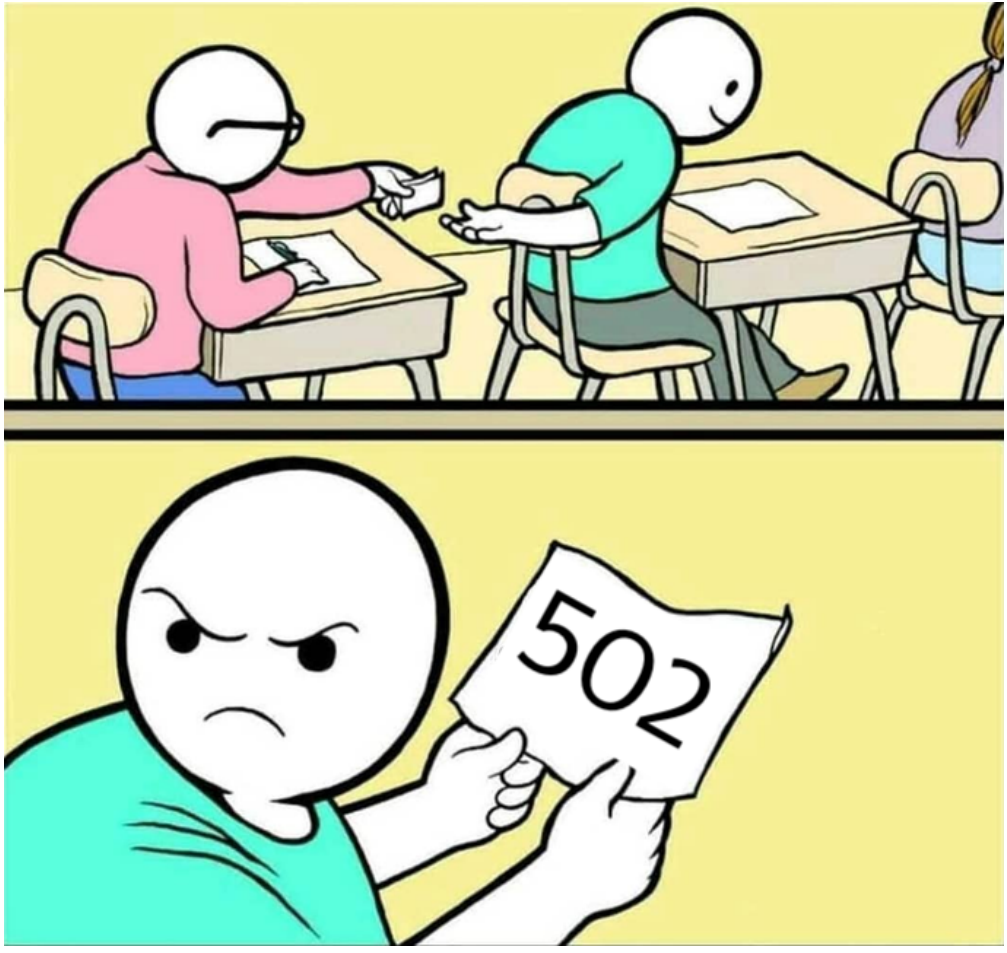
Клиент-Сервер

@dimastark


request
send()
recv()
Client
The Network
Server
recv()
send()
response


502

Клиент-серверное общение TCP/IP

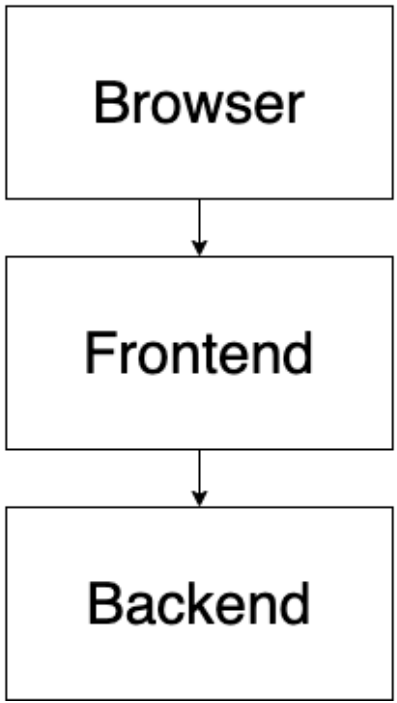
Ethernet (Link)

IP (Network)

TCP/UDP (Transport)

HTTP (Application)

Современная Web архитектура



HTTP (RFC-2616)

HyperText Transfer Protocol

Ресурс

```
http://localhost:5000/notes?limit=10&sortBy=name
↑       ↑           ↑    ↑      ↑

scheme host       port path  query
```

Запрос

```
POST /notes HTTP/1.1
Accept: application/json

Accept-Encoding: gzip, deflate
Content-Type: application/json; charset=utf-8
Host: localhost:8080
User-Agent: HTTPie/0.9.3

{
    "name": "films",
    "text": "Films to watch"
}
```

Ответ

```
HTTP/1.1 200 OK
Content-Length: 67

Content-Type: application/json; charset=utf-8
Date: Wed, 16 Mar 2016 14:32:18 GMT
X-Powered-By: Express

{
    "createdAt": 1458138738899,
    "name": "films",
    "text": "Films to watch"
}
```

Метод

GET	получение ресурса
POST	создание ресурса
DELETE	удаление ресурса

Код ответа

200 Ok
201 Created
204 No content

400 Bad request
401 Unauthorized
403 Forbidden
404 Not found

301 Moved Permanently
304 Not modified

500 Internal Server Error
504 Gateway Timeout

Stateless

Сам не хранит состояние клиента между запросами, всё состояние целиком описывается в каждом запросе

Взаимодействие приложений

```
Notes.find(name)
      ↑     ↑
 method     arguments
```

Remote Procedure Call

Запрос

```
{
    "jsonrpc": "2.0",
    "id": 1,
    "method": "findNote",
    "params": ["films"]
}
```

Ответ

```
{
    "jsonrpc": "2.0",
    "id": 1,
    "result": {
        "name": "films",
        "text": "..."
    }
}
```

REpresentational State Transfer

Запрос

```
GET /notes/films HTTP/1.1
```

Ответ

```
HTTP/1.1 200 Ok
Content-Type: application/json

{
    "name": "films",
    "text": "..."
}
```

gRPC

```
service NotesService {
    rpc Find (NoteIdRequest) returns (Note) {}

}

message Note {
    string name = 1;
    string text = 2;
}

message NoteIdRequest {
    string name = 1;
}
```

gRPC

```
const { notes as NotesService } = grpc.load('notes.proto');


const client = new NotesService('localhost:50051');

client.find({ name: 'films' }, (error, note) => {});
```

GraphQL

```
type Note {
    name: String!

    text: String
}

type Query {
    note(name: String!): Note
}
```

GraphQL

```
POST /graphql
Content-Type: application/json


{
    "query": "query { note(name: 'films') { name, text }"
}

HTTP/1.1 200 Ok
Content-Type: application/json

{
    "data": { "note": {"name": "films", "text": "..."} },
    "errors": [ ... ]
}
```

WebSockets

```
const socket = new WebSocket('ws://localhost:8080');


socket.send(JSON.stringify({
    id: '1',
    method: 'findNote',
    params: ['films']
});

socket.onmessage = message => {
    const { id, method, params } = JSON.parse(message);

    // ...
    socket.send({ id, result });
});
```

REST

REST определяет
понятия (ресурсы и их представление)
требования к взаимодействию

GET

Получает состояние ресурса в одном из представлений (JSON, XML, HTML)

```
GET /notes
GET /notes/films
GET /notes/films/pinned
GET /notes?limit=10

200 Ok

404 Not found
400 Bad request /notes?limit=muahahaha
```

POST

Создаёт новый ресурс с начальным состоянием, когда мы не знаем его ID

```
POST /notes

201 Created

409 Conflict
```

PUT

Создаёт новый ресурс с начальным состоянием, когда мы знаем его ID

```
PUT /notes/films
PUT /notes/films/pinned

200 Ok
204 No content
```

PUT

Обновляет состояние существующего ресурса целиком

```
PUT /notes/films
PUT /notes/films/pinned

200 Ok
204 No content

404 Not found
```

DELETE

Удаляет существующий ресурс

```
DELETE /notes/films
DELETE /notes/films/pinned

200 Ok
204 No content

404 Not found
```

PATCH

Обновляет состояние существующего ресурса частично

```
PATCH /notes/films

200 Ok
204 No content

404 Not found
```

HEAD

Запрашивает заголовки, чтобы проверить существование ресурса

```
HEAD /notes/films

200 Ok

404 Not found
```

OPTIONS

Запрашивает правила взаимодействия, например, доступные методы

```
OPTIONS /search

204 No content
Allow: OPTIONS, GET, HEAD

POST /search

405 Method not allowed
```

Идемпотентность

Один и тот же запрос приводит к одному и тому же состоянию

```
GET       – да (не модифицирующий)
OPTIONS   – да (не модифицирующий)
HEAD      – да (не модифицирующий)
POST      – нет
PUT       – да
DELETE    – да
PATCH     – нет
```

XMLHttpRequest

~~XMLHttpRequest~~

Fetch

Fetch

```
const promise = fetch(url[, options]);
```

options

```
{
    method: 'POST',

    headers: {
        'Accept': 'application/json'
    },
    body: JSON.stringify({
        id:   1,
        name: 'films'
    })
}
```

Promise

```
fetch('/notes')
    .then(res => {

        res.headers.get('Content-Type'); // application/json
        res.status; // 200

        return res.json();
    })
    .then(notes => {
        console.info(notes);
    })
    .catch(error => {
        console.error(error);
    });
```

Отмена запроса

```
const controller = new AbortController();

const signal = controller.signal;
fetch('/notes', { signal }); // pending
signal.abort();
```

REST

Большое число запросов

Сложности в проектировании при росте зависимостей между сущностями

Лишние данные в ответе от сервера

Всегда необходимо помнить об обратной совместимости

Нет удобных инструментов для разработки

GraphQL

GraphQL

Язык запросов к API, а так же среда исполнения для этих запросов

GraphQL

Строгая типизация

Получаем только то, что действительно необходимо

Возможность получить все необходимые данные за один запрос

Отсутствие проблем с обратной совместимостью и расширением

Удобные инструменты для разработки

Реализации на всех популярных языках

GraphQL

```
query {
    note(name: "Books") {
        name
        text
        comments {
            text
            author {
                name
            }
        }
    }
}
```

GraphiQL

kilogram-api

Types

```
ID, Int, Float, String, Boolean
```

```
type User {
    id: ID
    name: String
}
```

```
type Comment {
    text: String
    author: User
}
```

```
type Note {
    name: String
    text: String
    comments: [Comment]
}
```

```
type Query {
    note(name: String!): Note
    notes: [Note]
}
```

Unions

```
type Admin {
    id: ID
    name: String
    accessLevel: String
}

type NormalUser {
    id: ID
    name: String
    age: Int
}

union User = Admin | NormalUser
```

Queries

```
query {
    note(name: "Books") {
        name
        text
        comments {
            text
        }
    }
}
```

```
{
    "data": {
        "note": {
            "name": "Books",
            "text": "Books to read",
            "comments": [
                { "text": "Очень круто!" },
                { "text": "А мне не очень понравил
                { "text": "Peter, объяснишь почему
            ]
        }
    }
}
```

Queries

```
query {
    note(name: "Books") {
        text
    }

    user(id: 1) {
        name
    }
}
```

```
{
    "data": {
        "note": {
            "text": "Books to read"
        },
        "user": {
            "name": "Max"
        }
    }
}
```

Aliases

```
query {
    firstNote: note(name: "Books") {
        text
    }

    secondNote: note(name: "Films") {
        text
    }
}
```

```
{
    "data": {
        "firstNote": {
            "text": "Books to read"
        },
        "secondNote": {
            "text": "Films to watch"
        }
    }
}
```

Named Queries

```
query NotesQuery {
    firstNote: note(name: "Books") {
        text
    }

    secondNote: note(name: "Films") {
        text
    }
}
```

Fragments

```
fragment NoteFields on Note {
    id
    name
    text
}
```

Fragments

```
query {
    firstNote: note(name: "Books") {
        ...NoteFields
    }

    secondNote: note(name: "Films") {
        ...NoteFields
    }
}
```

Inline fragments

```
query {
    users {
        __typename

        ... on Admin {
            name
            accessLevel
        }

        ... on NormalUser {
            name
            age
        }
    }
}
```

Variables

```
query NoteQuery($name: String!) {
    note(name: $name) {
        name
        text
    }
}

// "variables"

{ "name": "Books" }
```

Directives

```
query NoteQuery($name: String!, $withComments: Boolean!) {
    note(name: $name) {
        name
        text
        comments @include(if: $withComments) {
            text
        }
    }
}
```

Directives

```
query NoteQuery($name: String!, $withoutComments: Boolean!) {
    note(name: $name) {
        name
        text
        comments @skip(if: $withoutComments) {
            text
        }
    }
}
```

Mutations

```
mutation CreateNote($name: String!, $text: String!) {
    createNote(name: $name, text: $text) {
        name
        text
    }
}
```

Errors

```
query {
    note(name: "Books") {
        name
        createdAt
    }
}
```

```
{
    "errors": [
        {
            "message": "Cannot query field \"createdAt\" on typ
            "locations": [
                {
                    "line": 4,
                    "column": 5
                }
            ]
        }
    ]
}
```

GraphQL

Новая технология

Мало паттернов

Сложности при работе с SQL базами данных

GraphQL

~~Новая технология~~

~~Мало паттернов~~

Сложности при работе с SQL базами данных

GraphQL Клиенты

Lokka	Максимально простой в использовании. Базовая поддержка запросов и мутаций. Простейшее кэширование
Apollo	Более гибкий. Хороший баланс между функциональностью и сложностью использования
Relay	Наиболее функциональный, из-за чего наиболее сложный в использовании. Много внимания уделено производительности (особенно на мобильных).

GraphQL

GraphQL Specification

GraphQL.js

GraphQL Best Practices

GraphQL Clients

Lokka

Apollo

Relay