

JSON API - работаем по спецификации

Алексей Авдеев, Neuron.Digital

FC







2018

**Frontend
Conf**

Профессиональная
конференция
фронтенд-разработчиков



0 себе

1.  Алексей Авдеев (<https://github.com/avdeev>)
2.  Из Нижнего Новгорода, работаю в Москве
3.  [Neuron.Digital](https://neuron.digital)
4.  Веду команду из 8 фронтенд-разработчиков
5.  Програмирую с 2002 года
6.  Сейчас активно использую React

История

1. **1981 RPC**

2. **2000 REST**

RPC

RPC Пример

1. **Клиент:** вызови функцию *"hello"* с параметром *"FrontendConf"*
2. **Сервер:** ответ - *"Hello FrontendConf!"*

XML-RPC (from 1998)

01. <?xml version="1.0"?>

02. <methodCall>

03. <methodName>examples.getStateName</methodName>

04. <params>

05. <param><value><i4>40</i4></value></param>

06. </params>

07. </methodCall>

Simple Object Access Protocol (SOAP) (2007)

```
01. <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/...
02.     <soap:Body>
03.         <getProductDetails xmlns="http://warehouse.example...
04.             <productID>12345</productID>
05.         </getProductDetails>
06.     </soap:Body>
07. </soap:Envelope>
```


JSON-RPC (2013)

01. {

02. "jsonrpc": "2.0", "method": "subtract",

03. "params": [42, 23], "id": 1

04. }

{"jsonrpc": "2.0", "result": 19, "id": 1}

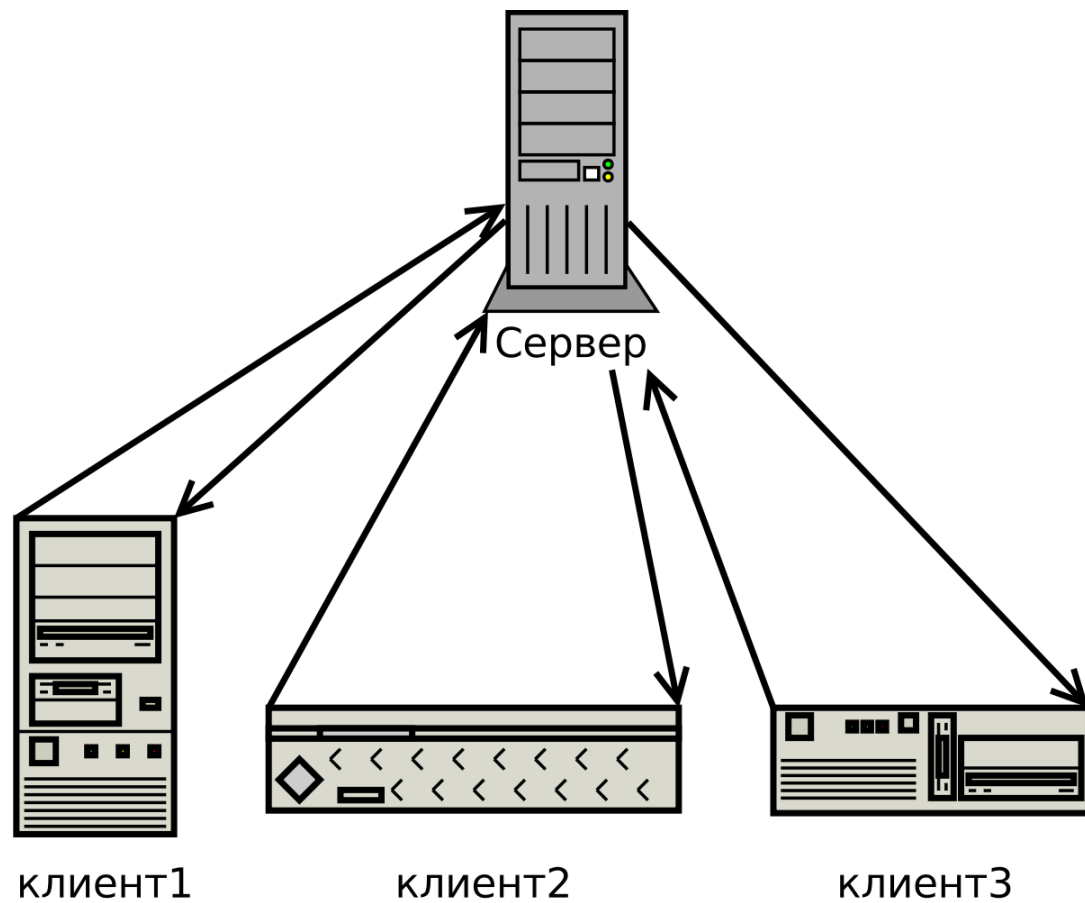
GraphQL (2016)

```
01. POST /graphql HTTP/1.1
02. Content-Type: application/json
03. {
04.   "query": "...",
05.   "operationName": "...",
06.   "variables": { "myVariable": "someValue", ... }
07. }
```

REST

Требования к архитектуре REST

1. Модель клиент-сервер
2. Отсутствие состояния
3. Кэширование
4. **Единообразие интерфейса**
5. Слои
6. Код по требованию (необязательное ограничение)



Начинаем проектировать



RESTful блог

CRUD

Единообразный интерфейс

1. Создать статью
2. Получить список статей
3. Получить статью
4. Обновить статью
5. Удалить статью

Единообразный интерфейс

Метод	Путь	Действие
POST	/articles	Создать статью
GET	/articles	Получить список статей
GET	/articles/1	Получить статью
PATCH	/articles/1	Обновить статью
DELETE	/articles/1	Удалить статью

Content-Type, Accept

Ошибка

01. DELETE /articles/1 HTTP/1.1

02. Accept: application/json

01. HTTP/1.1 200 OK

02. Content-Type: application/json

03.



MIME-типы

- text/plain
- application/octet-stream
- application/pdf
- image/png
- application/json
- application/xml
- application/vnd.ms-excel

Документация

“

*- Документация? Мой API
самодокументируемый.*

Начинающий бекендер

Полезная нагрузка

“

- JSON, в котором лежит HTML и XML

обернутый в другой XML

Начинающий бекендер

Аутентификация

“

*- Напишем свою собственную систему
безопасности*

Начинающий бекендер

Авторизация

“

- Передаем пароль GET-параметром

Начинающий бекендер

Ошибки

“

*- Всегда возвращаем 200 ИЛИ HTML-страницу
с ошибкой*

Начинающий бекендер

Кэш

“

*- Ничего не кэшируется. Максимум, ответы
от БД на сервере.*

Начинающий бекендер



Создание

01. POST /articles HTTP/1.1

02. Content-Type: application/json

03. { "id": 1, "title": "Про JSON API" }

01. HTTP/1.1 422 Unprocessable Entity

02. HTTP/1.1 403 Forbidden

03. HTTP/1.1 500 Internal Server Error

Возвращайте ошибки

👉 После

01. HTTP/1.1 422 Unprocessable Entity

02. Content-Type: application/json

03.

04. { "errors": [{

05. "status": "422",

06. "title": "Title already exist",

07.]}]}

**Добавим
паджинацию**

< PREV 1 2 3 4 ... 335 336 NEXT >

Запрос списка

01. GET /articles HTTP/1.1

02. Content-Type: application/json

01. HTTP/1.1 200 OK

02. [

03. { "id": 1, "title": "Про JSON API"},

04. { "id": 2, "title": "Про XML-RPC"}

05.]

После

01. GET /articles?page[size]=30&page[number]=2

02. Content-Type: application/json

01. HTTP/1.1 200 OK

02. {

03. "data": [{ "id": 1, "title": "JSON API"}, ...],

04. "meta": { "count": 10080 }

05. }

Или так

01. GET /articles?page[offset]=30&page[limit]=30

02. Content-Type: application/json

01. HTTP/1.1 200 OK

02. {

03. "data": [{ "id": 1, "title": "JSON API"}, ...],

04. "meta": { "count": 10080 }

05. }

Или так

01. GET /articles?page[published_at]=1538332156

02. Content-Type: application/json

01. HTTP/1.1 200 OK

02. {

03. "data": [{ "id": 1, "title": "JSON API"}, ...],

04. "meta": { "count": 10080 }

05. }

Проблема N + 1

Выведем 10 статей с указанием автора

1. 1 запрос на получение статей
2. 10 запросов для получения авторов каждой статьи

Итого: 11 запросов

Добавляем связи

Запрос списка со связями

01. GET /articles?include=author

02. Content-Type: application/json



Решение: запрос списка со связями

01. HTTP/1.1 200 OK

02. { "data": [{

03. { attributes: { "id": 1, "title": "JSON API" },

04. { relationships: {

05. "author": { "id": 1, "name": "Avdeev" } }

06. }, ...

07.]}]}

Проблема дублирования данных

Выведем 10 статей с указанием автора, у всех статей один автор

Итого: один автор включен в ответ 10 раз

👉 Решение: нормализация данных

01. HTTP/1.1 200 OK

02. { "data": [{

03. "id": "1", "type": "article",

04. "attributes": { "title": "JSON API" },

05. "relationships": { ... }

06. }, ...]

07. }



Решение: нормализация данных

01. HTTP/1.1 200 OK

02. { "data": [{

03. ...

04. "relationships": {

05. "author": { "id": 1, "type": "people" } }

06. }

07. }, ...]

08. }

46



Решение: нормализация данных

01. HTTP/1.1 200 OK

02. {

03. "data": [...],

04. "included": [{

05. "id": 1, "type": "people",

06. "attributes": { "name": "Avdeev" }

07. }]

08. }

47

**Нужны не все
поля ресурса**

Решение

GET /articles/1?fields[articles]=title HTTP/1.1

01. HTTP/1.1 200 OK

02. { "data": [{

03. "id": "1", "type": "article",

04. "attributes": { "title": "Про JSON API" },

05. }, ...]

06. }

Оформим и зарегистрируем Media Type

“

(registered 2013-07-21, last updated 2013-07-21)

Name : Steve Klabnik

Email : steve&steveklabnik.com

MIME media type name : Application

MIME subtype name : Vendor Tree - vnd.api+json

<http://www.iana.org/assignments/media-types/application/vnd.api+json>



JSON API готов

{json:api}

Не упомянул про

1. Гипермедиа (links)
2. Фильтрация (filters)
3. Сортировка
4. Коды ответов

JSON API (2015)

01. GET /articles HTTP/1.1

02. Accept: application/vnd.api+json

JSON API (2015)

01. HTTP/1.1 200 OK

02. Content-Type: application/vnd.api+json

03. {

04. "links": { "self": "http://example.com/articles },

05. "data": [{

06. "type": "articles", "id": "1",

07. "attributes": {

08. "title": "JSON API paints my bikeshed!"

09. }

10. }]

11. }

Экосистема JSON API

Список реализаций спецификации - <http://jsonapi.org/implementations/>

1. 170 различных реализаций
2. для 32 языков программирования
3. и это только добавленные в каталог
4. PR Welcome

Плюсы JSON API

1. Общее соглашение для всех
2. Меньше споров внутри команды
3. Высокая производительность разработки
4. Популярные проблемы уже решены
5. Прост для понимания
6. Лаконичен
7. Open Source



Минусы JSON API

1. **Фронтенд**: надо парсить ответы
2. **Бекенд**: контроль вложенности
3. **Бекенд**: сложность запросов к БД
4. **Бекенд**: безопасность
5. Спека сложно читается (не все смогли)
6. Не все либы реализуют спеку хорошо

Подводные камни JSON API

1. Количество relationships в выдаче неограничено
2. Сложно работать с полиморфными связями
3. Одно и то же можно запросить по-разному (неоднозначность)
4. Сложно писать Swagger (OpenAPI)

OpenAPI

здорового человека



POST

/pet Add a new pet to the store



PUT

/pet Update an existing pet



GET

/pet/findByStatus Finds Pets by status



GET

/pet/findByTags Finds Pets by tags



GET

/pet/{petId} Find pet by ID



POST

/pet/{petId} Updates a pet in the store with form data



DELETE

/pet/{petId} Deletes a pet



POST

/pet/{petId}/uploadImage uploads an image



OpenAPI для JSON API

```

  {
    data
      MatchWithRelationships {
        id          string($uuid)
        type        string
        attributes  > {...}
        relationships
          {
            group > {...}
          }
      }
    included
      {
        oneOf ->
          Group > {...}
          Team > {...}
          Stage > {...}
          Match > {...}
          Stadium > {...}
      }
  }

```

OData (2015)

0Data (2015)

- 01. GET `http://services.odata.org/v4/TripRW/People` HTTP/1.1
- 02. OData-Version: 4.0
- 03. OData-MaxVersion: 4.0

OData (2015)

```
01. HTTP/1.1 200 OK
02. Content-Type: application/json; odata.metadata=minimal
03. OData-Version: 4.0
04. {
05.   '@odata.context': 'http://services.odata.org/V4/...
06.   '@odata.nextLink': 'http://services.odata.org/V4/...
07.   'value': [{
08.     '@odata.id': 'http://services.odata.org/V4/...
09.     '@odata.etag': 'W/'08D1D5BD423E5158'',
10.     '@odata.editLink': 'http://services.odata.org/V4/...
11.     'UserName': 'russellwhyte',
12.     ...
```

Что с GraphQL?

Высокий порог входа



Эффект большого взрыва

1. Нет никакого API

2. 

3. GraphQL

Или

1. 🦎 Простейший RESTful
2. 🦎 Прогрессивное улучшение
3. 🦎 JSON API



Ад на бекенде

1. Контроль вложенности
2. Рекурсия
3. Ограничение частоты
4. Контроль доступа

За подробностями

1. <http://jsonapi.org>
2. <http://www.odata.org>
3. <https://graphql.org>
4. <http://xmlrpc.scripting.com>
5. <https://www.jsonrpc.org>

Вопросы ?