Оглавление

1 Добро пожаловать в GraphQL

Прежде чем королева Англии посвятила Тима Бернерса-Ли (Tim Berners-Lee) в рыцари, он был программистом. Работал в ЦЕРН, Европейской лаборатории физики элементарных частиц в Швейцарии, и был окружен рядами талантливых исследователей. Бернерс-Ли хотел помочь коллегам делиться их идеями, поэтому решил создать сеть, в которой ученые могли бы публиковать и обновлять информацию. В итоге проект стал первым веб-сервером и первым веб-клиентом, а браузер WorldWideWeb (позже переименованный в Nexus) был выпущен в ЦЕРН (www.w3.org/People/Berners-Lee/Longer.html) в декабре 1990 года.

Благодаря своему проекту Бернерс-Ли позволил исследователям просматривать и обновлять веб-контент на своих компьютерах. WorldWideWeb — это HTML, URL-адреса, браузер и интерфейс WYSIWYG («что видишь, то и получаешь») для обновления контента.

Сегодня Интернет не просто HTML в браузере. Интернет — это ноутбуки. Это умные часы. Это смартфоны. Это микросхема радиочастотной идентификации (RFID) в вашем гаджете. Это робот, который накладывает еду вашим кошкам, пока вы находитесь на работе.

Клиенты сегодня намного производительнее, но мы стремимся все к тому же: загружать данные как можно быстрее. Нам нужно, чтобы наши приложения были эффективными, потому что наши пользователи придерживаются высоких стандартов. Они ожидают, что наши приложения будут работать хорошо при любых условиях: от 2G на смартфонах до мощного оптоволоконного Интернета на компьютерах с большим экраном. Быстрые приложения упрощают

общение с нашим контентом. Быстрые приложения делают пользователей счастливыми. И да, быстрые приложения зарабатывают нам деньги.

Получение данных с сервера клиентом быстро и предсказуемо — это история Сети, ее прошлое, настоящее и будущее. Хотя в книге мы часто ссылаемся на минувшие годы, мы здесь для того, чтобы поговорить о современных решениях. Мы здесь, чтобы поговорить о будущем. Мы здесь, чтобы поговорить о GraphQL.

Что такое GraphQL

GraphQL (www.graphql.org) — это язык запросов для ваших API. А также среда выполнения для запросов с вашими данными. Сервис GraphQL является транспортно независимым, но обычно обслуживается через HTTP.

Чтобы продемонстрировать GraphQL-запрос и его ответ, взглянем на SWAPI (graphql.org/swapi-graphql/), API Star Wars. SWAPI является интерфейсом представления состояний (REST API), который был преобразован с помощью GraphQL. Мы можем использовать его для отправки запросов и получения данных.

GraphQL-запрос касается только необходимых данных. Рисунок 1.1, *слева*, представляет пример GraphQL-запроса. Мы запрашиваем данные для персоны принцессы Леи. И получаем запись, соответствующую Лее Органе, потому что указали, что нужны данные пятой персоны (`personID: 5`). Затем мы запрашиваем три поля данных: `name`, `birthYear` и `created`. *Справа* — полученный ответ: данные JSON отформатированы в соответствии с формой нашего запроса. Этот ответ содержит только те данные, которые нам нужны.

Затем мы можем настроить запрос, поскольку запросы интерактивны. Можно изменить его и увидеть новый результат. Если мы добавим поле `FilmConnection`, сможем запросить название каждого из фильмов с Леей, как показано на рис. 1.2.

Запрос вложен, и, когда он выполняется, можно перемещать связанные объекты. Благодаря этому требуется сделать один HTTP-запрос для двух типов данных. Нам не нужно совершать несколько

```
1  query {
2    person(personID:5) {
3      name
4      birthYear
5      created
6    }
7  }
```

```
{
  "data": {
    "person": {
      "name": "Leia Organa",
      "birthYear": "19BBY",
      "created": "2014-12-10T15:20:09.791000Z"
    }
  }
}
```

Рис. 1.1. Пользовательский запрос к API Star Wars

```
1  query {
2    person(personID:5) {
3      name
4      birthYear
5      created
6      filmConnection {
7        films {
8          title
9        }
10     }
11   }
12 }
```

```
{
  "data": {
    "person": {
      "name": "Leia Organa",
      "birthYear": "19BBY",
      "created": "2014-12-10T15:20:09.791000Z",
      "filmConnection": {
        "films": [
          {
            "title": "A New Hope"
          },
          {
            "title": "The Empire Strikes Back"
          },
          {
            "title": "Return of the Jedi"
          },
          {
            "title": "Revenge of the Sith"
          },
          {
            "title": "The Force Awakens"
          }
        ]
      }
    }
  }
}
```

Рис. 1.2. Запрос фильмов

подходов, чтобы развернуть несколько объектов. Мы не получаем дополнительных нежелательных данных об этих типах. С помощью GraphQL наши клиенты могут получить все необходимые данные по одному запросу.

Всякий раз, когда запрос выполняется на сервере GraphQL, он проверяется на наличие системы типов. Каждый сервис GraphQL определяет типы в схеме GraphQL. Вы можете представить систему

типов как схему данных вашего API, подкрепленную списком объектов, которые вы определяете. Например, запрос персоны поддерживается объектом `Person`:

```
type Person {
    id: ID!
    name: String
    birthYear: String
    eyeColor: String
    gender: String
    hairColor: String
    height: Int
    mass: Float
    skinColor: String
    homeworld: Planet
    species: Species
    filmConnection: PersonFilmsConnection
    starshipConnection: PersonStarshipConnection
    vehicleConnection: PersonVehiclesConnection
    created: String
    edited: String
}
```

Тип `Person` определяет все поля вместе с их типами, касающиеся принцессы Леи и доступные для запроса. В главе 3 мы подробно рассмотрим схему и систему типов GraphQL.

GraphQL часто упоминается как *декларативный* язык для извлечения данных. Под этим мы подразумеваем, что разработчики будут перечислять свои требования к данным, говоря о том, *какие* данные им нужны, и не уточняя, *как* они собираются их получить. Серверные библиотеки GraphQL существуют на разных языках, включая C#, Clojure, Elixir, Erlang, Go, Groovy, Java, JavaScript, .NET, PHP, Python, Scala и Ruby[1].

В этой книге мы сосредоточимся на том, как создавать сервисы GraphQL с помощью JavaScript. Все методы, которые мы обсуждаем в этой книге, можно применить к GraphQL, используя любой язык.

---

[1] См. GraphQL Server Libraries на странице graphql.org/code/.

Спецификация GraphQL

GraphQL — спецификация для клиент-серверного взаимодействия. Что делает спецификация? Она описывает возможности и характеристики языка. Мы пользуемся преимуществами языковых спецификаций, поскольку они обеспечивают наличие общего словаря и лучших практик использования языка сообществом.

Достойным внимания примером спецификации программного обеспечения является спецификация ECMAScript. Время от времени представители компаний — разработчики браузеров, технических компаний и сообщества в целом — собираются вместе и разрабатывают то, что должно быть включено в спецификацию ECMAScript (и исключено из нее). То же самое верно для GraphQL. Несколько людей собрались вместе и написали, что должно быть включено (и не включено) в язык. Теперь это служит руководством для всех реализаций GraphQL.

Когда спецификация была выпущена, создатели GraphQL также поделились ссылочной реализацией сервера GraphQL на JavaScript — `graphql.js` (github.com/graphql/graphql-js). Это полезный проект, но цель данной эталонной реализации заключается не в том, чтобы указать, какой язык вы используете для реализации своего сервиса. Это просто руководство. После того как вы поймете язык запросов и систему типов, вы сможете создать свой сервер на любом языке, который предпочитаете.

Если спецификация и реализация различны, то что на самом деле приведено в спецификации? Спецификация описывает язык и грамматику, которые вы должны применять при написании запросов. Она также устанавливает систему типов, механизмы ее выполнения и проверки. За исключением указанного, спецификация не особенно важна. GraphQL не определяет, какой язык использовать, как хранить данные или каких клиентов поддерживать. Язык запросов — это рекомендации, но фактический дизайн вашего проекта зависит от вас. (Если вы хотите подробнее разобраться во всем этом, можете изучить документацию по адресу facebook.github.io/graphql/.)

Принципы проектирования GraphQL

Несмотря на то что GraphQL не контролирует, как вы создаете свой API, спецификация предлагает некоторые рекомендации[1].

- *Иерархичность*. GraphQL-запрос является иерархическим. Поля встраиваются в другие поля, и запрос формируется подобно данным, которые он возвращает.
- *Ориентированность на продукт*. GraphQL управляется потребностями данных клиента, а также языком и временем выполнения, поддерживаемыми клиентом.
- *Строгая типизация*. Сервер GraphQL поддерживается системой GraphQL. В схеме каждая точка данных имеет определенный тип, насчет которого она будет проверена.
- *Запросы, определенные клиентом*. Сервер GraphQL предоставляет возможности, которые клиенты могут использовать.
- *Интроспектива*. Язык GraphQL может запрашивать серверную систему типов GraphQL.

Теперь, когда у вас есть базовое представление о спецификации GraphQL, поговорим о том, почему она была создана.

Происхождение GraphQL

В 2012 году сотрудники компании Facebook решили, что необходимо перестроить собственные мобильные приложения. Приложения iOS и Android компании были просто тонкими обертками представлений мобильного сайта. Компания Facebook использовала сервер RESTful и таблицы данных FQL (Facebook для SQL). Производительность была сомнительной, и приложения часто аварийно завершали работу. В этот момент инженеры поняли, что

---

[1] См. GraphQL Spec, июнь 2018 (http://facebook.github.io/graphql/June2018/#sec-Overview).

им необходимо улучшить способ передачи данных в клиентские приложения[1].

Ли Байрон (Lee Byron), Ник Шрок (Nick Schrock) и Дэн Шафер (Dan Schafer) намеревались пересмотреть данные со стороны клиента. Они решили создать GraphQL, язык запросов, который будет описывать возможности и требования моделей данных для клиентских/серверных приложений компании.

В июле 2015 года программисты выпустили первоначальную спецификацию GraphQL и эталонную реализацию GraphQL на JavaScript под названием `graphql.js`. В сентябре 2016 года спецификация GraphQL прошла стадию «технического предварительного просмотра». Это означало, что она была официально готова к выпуску, хотя уже много лет до этого применялась в Facebook. Сегодня GraphQL задействует почти все, что касается извлечения данных в компании Facebook, и используется в компаниях IBM, Intuit, Airbnb и др.

История транспортировки данных

GraphQL реализует некоторые новомодные идеи, но все это следует воспринимать в историческом контексте передачи данных. Задумываясь о передаче данных, мы пытаемся понять, как передавать их между компьютерами. Мы запрашиваем некоторые данные из удаленной системы и ожидаем ответа.

Удаленный вызов процедур

В 1960-х годах был изобретен удаленный вызов процедур (remote procedure call, RPC). Клиент инициировал RPC, который отправлял сообщение с запросом на удаленный компьютер, чтобы что-то сделать. Удаленный компьютер отправлял ответ клиенту. Эти компьютеры

---

[1] См. видео «Выборка данных для приложений React» Дэна Шафера (Dan Schafer) и Джинга Чена (Jing Chen), www.youtube.com/watch?v=9sc8Pyc51uU.

отличались от клиентов и серверов, которые мы используем сегодня, но процесс был в основном одинаковым: запрос некоторых данных у клиента, получение ответа от сервера.

Простой протокол доступа к объектам

В конце 1990-х годов в Microsoft был разработан протокол простого доступа к объектам (Simple Object Access Protocol, SOAP). SOAP использовал язык XML для кодирования сообщений и протокол HTTP для транспортировки. В протоколе SOAP также применялась система типов и была представлена концепция ресурсоориентированных вызовов для данных. SOAP давал довольно хорошие результаты, но вызвал разочарование, поскольку его реализации были довольно сложными.

Архитектура REST

Парадигма API, с которой вы, вероятно, наиболее знакомы сегодня, — REST. Архитектура REST была описана в 2000 году в докторской диссертации (bit.ly/2j4SIKI) Роя Филдинга (Roy Fielding) в Калифорнийском университете в Ирвайне. Филдинг описал ресурсоориентированную архитектуру, в которой пользователи будут перемещаться по веб-ресурсам, выполняя такие операции, как GET, PUT, POST и DELETE. Сеть ресурсов можно рассматривать как *виртуальную машину состояний*, а действия (GET, PUT, POST, DELETE) — как изменения состояния в машине. Сегодня мы можем считать такое само собой разумеющимся, но тогда это было довольно сложно. (Да, и Филдинг получил степень доктора наук.)

В архитектуре RESTful маршруты представляют информацию. Например, запрос информации от каждого из этих маршрутов даст конкретный ответ:

```
/api/food/hot-dog
/api/sport/skiing
/api/city/Lisbon
```

Архитектура REST позволяет создать модель данных с множеством конечных точек, это гораздо более простой подход, чем в предыдущих архитектурах. Архитектура предоставила новый способ обработки

данных во все более сложной сети, но не обеспечивала соблюдение конкретного формата ответов данных. Первоначально архитектура REST использовалась с XML. *AJAX* сперва было аббревиатурой, расшифровываемой как *асинхронный JavaScript и XML*, поскольку данные ответа из запроса Ajax были отформатированы на языке XML (теперь это автономное слово, записываемое как *Ajax*). Это был болезненный шаг для веб-разработчиков: приходилось анализировать ответы XML до того, как данные могли быть применены в JavaScript.

Вскоре после этого Дугласом Крокфордом (Douglas Crockford) была разработана и стандартизирована технология JavaScript Object Notation (JSON). JSON обеспечивает элегантный формат данных, который могут анализировать и использовать многие языки. Крокфорд написал книгу *JavaScript: The Good Parts*[1] (bit.ly/js-good-parts) (O'Reilly, 2008), в которой сообщил, что JSON — одна из отличных технологий.

Влияние архитектуры REST неоспоримо. Она используется для создания бесчисленных API. Разработчики различного звена только выиграли от ее применения. Существуют даже настолько проникшиеся форматом люди, яростно спорящие о том, что есть и чем не является RESTful, что они получили название *рестафарианцы*. Итак, если все так хорошо, почему Байрон, Шрок и Шафер опять собираются разрабатывать что-то новое? Ответ кроется в определенных недостатках архитектуры REST.

Недостатки архитектуры REST

Когда спецификация GraphQL увидела свет, некоторые позиционировали ее как замену REST. «REST мертва!» — воскликнули ее ранние последователи, а затем призвали всех нас бросить лопату в багажник и отвезти наши ничего не подозревающие API REST в лес. Это было отличным поводом для возбуждения интереса к тематическим блогам и начала диспутов на конференциях, но позиционирование GraphQL как убийцы REST — явное упрощение. Тонкий подход заключается в том, что по мере развития Всемирной паутины REST при определенных условиях проявляет признаки деформации. Язык GraphQL создавался для облегчения такой деформации.

---

[1] *Крокфорд Д.* JavaScript: сильные стороны. — СПб.: Питер, 2013.