Оглавление

Предисловие от издательства

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте **www.dmkpress.com**, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Manning Publications Co. очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты **dmkpress@gmail.com** со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую предоставлять вам качественные материалы.

Вступление

Благодарим вас за выбор «Разработка веб-приложения GraphQL с React, Node.js и Neo4j». Цель этой книги – показать, как можно использовать GraphQL, React, Apollo и базу данных Neo4j (так называемый стек GRAND) для создания сложных приложений, интенсивно применяющих данные. Возможно, вам интересно, почему мы выбрали именно эту комбинацию технологий. Я надеюсь, что в процессе чтения вы оцените продуктивность, производительность и интуитивно понятные преимущества использования графовой модели данных на всем протяжении – от базы данных до API и в коде, выбирающем данные на стороне клиента.

Я мечтал найти такую книгу, когда, будучи молодым инженером, получил работу в небольшом стартапе, занимающемся созданием полнофункционального веб-приложения. Мы потратили месяцы на оценку стека технологий и изучение способов их комбинирования друг с другом. В конце концов мы приступили к работе с применением технологий, которые нас устраивали, но на их выбор потребовалось много итераций.

GraphQL – это технология, коренным образом изменившая подходы к разработке веб-приложений. Эта книга посвящена GraphQL; однако одного лишь понимания, как создавать серверы и писать операции GraphQL, недостаточно для реализации приложений полного цикла. Нужно также подумать о том, как организовать выборку данных из GraphQL и управление состоянием внешнего приложения, как защитить API, как развернуть приложение, и учесть массу других соображений. Вот почему эта книга не только о GraphQL; она рассказывает об использовании GraphQL в целом, показывая, как разные части сочетаются друг с другом. Если перед вами стоит задача создать приложение полного цикла с использованием GraphQL, то эта книга для вас!

Благодарности

Работа над книгой – длительный процесс, требующий помощи и поддержки многих людей. Невозможно отметить всех, кто помог в создании этой книги, не упустив никого. Конечно, эта книга была бы невозможна без всех участвовавших в создании удивительных технологий, о которых мы рассказываем.

Спасибо Майклу Стивенсу (Michael Stephens) за предложение написать книгу о GraphQL и помощь в развитии идеи полного стека GraphQL, Карен Миллер (Karen Miller) за рецензирование ранних версий каждой главы, а также всем сотрудникам издательства Manning, принявшим участие в проекте: Дугу (Doug), Александару (Aleksandar), Энди (Andy), Кристиану (Christian), Мелоди (Melody), Ниеку (Niek), Гордану (Gordan) и Марии (Marija). Спасибо моей семье, что терпели меня во время работы над этой книгой. Особое спасибо сообществу за помощь в проверке идей, изложенных в этой книге, а также за подробные отзывы и вклад в развитие библиотеки Neo4j GraphQL.

Дорогие мои рецензенты: Андрес Сакко (Andres Sacco), Брендон Фрайер (Brandon Friar), Кристофер Хаупт (Christopher Haupt), Дамиан Эстебан (Damian Esteban), Данило Зекович (Danilo Zekovic), Дениз Вехби (Deniz Vehbi), Ферит Топку (Ferit Topcu), Франс Оилинки (Frans Oilinki), Густаво Гомес (Gustavo Gomes), Харш Раваль (Harsh Raval), Иво Санчес Чека Кросато (Ivo Sánchez Checa Crosato), Хосе Антонио Эрнандес Ороско (Jose Antonio Hernandez Orozco), Хосе Сан Леандро (Jose San Leandro), Кевин Реди (Kevin Ready), Константинос Леймонис (Konstantinos Leimonis), Кшиштоф Камычек (Krzysztof Kamyczek), Микеле Аддуци (Michele Adduci), Мигель Исидоро (Miguel Isidoro), Ричард Мейнсен (Richard Meinsen), Ричард Воган (Richard Vaughan), Роб Лейси (Rob Lacey), Рональд Борман (Ronald Borman), Райан Хубер (Ryan Huber), Сейт Кумар Саху (Satej Kumar Sahu), Симеон Лейзерзон (Simeon Leyzerzon), Стефан Туральски (Stefan Turalski), Таня Вилке (Tanya Wilke), Теофанис Деспоудис (Theofanis Despoudis) и Владимир Пасман (Vladimir Pasman), спасибо вам! Ваши предложения помогли сделать эту книгу лучше.

О книге

Цель книги «Разработка веб-приложения GraphQL с React, Node.js и Neo4j» – показать, как разные части стека GraphQL сочетаются друг с другом в полномасштабных приложениях и как разработчики могут использовать онлайн-службы для поддержки разработки и развертывания. С этой целью в каждой главе будут представляться новые идеи и понятия и применяться для создания и развертывания полномасштабного приложения.

Кому адресована эта книга

Эта книга предназначена для веб-разработчиков полного цикла, заинтересованных в технологии GraphQL и имеющих хотя бы базовый уровень понимания Node.js API и особенностей клиентских приложений на JavaScript, использующих этот API. Прочитав эту книгу, читатель получит базовое представление о Node.js и клиентском JavaScript, но, что особенно важно, приобретет мотивацию для освоения приемов создания служб и приложений с использованием GraphQL.

Организация книги

Эта книга состоит из девяти глав, разделенных на три части. В каждой главе обсуждаются новые концепции и технологии в контексте создания полномасштабных приложений.

В первой части вы познакомитесь с GraphQL – графовой базой данных для Neo4j – и собственно с понятием графов:

- в главе 1 обсуждаются компоненты полномасштабных приложений GraphQL, включая введение во все конкретные технологии, используемые в этой книге (GraphQL, React, Apollo и база данных Neo4j);
- глава 2 знакомит с GraphQL и основами создания GraphQL API (определения типов и функции распознавания);
- глава 3 знакомит с графовой базой данных Neo4j, моделью графа свойств и языком запросов Cypher;
- глава 4 демонстрирует возможности GraphQL при работе с графовой базой данных Neo4j посредством библиотеки Neo4j GraphQL.

Во второй части мы сосредоточимся на разработке клиентского приложения с использованием React:

- глава 5 знакомит с инфраструктурой библиотеки React и концепциями ее применения, которые пригодятся, когда мы приступим к созданию примера клиентского приложения;
- глава 6 показывает, как организовать выборку данных и управление состоянием клиента с помощью React и GraphQL при работе с GraphQL API, созданным в предыдущих главах.

В третьей части мы займемся защитой приложения и его развертыванием с использованием облачных служб:

- глава 7 показывает, как защитить приложение, используя GraphQL и Auth0;
- глава 8 знакомит с облачными службами, обычно используемыми для развертывания баз данных, GraphQL API и приложений React;
- глава 9 завершает книгу обсуждением абстрактных типов в GraphQL, разбиением наборов данных на страницы с помощью курсоров и обработки свойств отношений в GraphQL.

Эту книгу следует читать от начала до конца, потому что каждая следующая глава основывается на предыдущих, и все они описывают процесс создания полномасштабного приложения. Читатели могут сосредоточиться на отдельных главах, погрузившись в интересующие их темы, но при этом желательно прочитать предыдущие главы, чтобы узнать, как и почему созданы те или иные части приложения.

О примерах программного кода

Эта книга также содержит множество примеров программного кода как в пронумерованных листингах, так и в виде включений в обычный текст. В обоих случаях исходный код оформлен `шрифтом фиксированной ширины`, чтобы вам было проще отличать его от основного текста. Иногда вам будет встречаться код, оформленный **`жирным моноширинным шрифтом`**, чтобы выделить изменившиеся фрагменты, по сравнению с предыдущими шагами, например когда в существующую строку кода добавляется что-то новое.

Во многих случаях исходный код переформатирован, чтобы уместить его по ширине книжной страницы. В частности, мы добавили разрывы строк и отступы. В редких случаях даже этого было недостаточно, и мы добавили маркеры продолжения строки (➥). Кроме того, мы удалили комментарии из листингов, которые подробно описываются в тексте. Многие листинги сопровождаются дополнительными примечаниями, описывающими важные понятия.

Получить выполняемые фрагменты кода можно из электронной версии книги по адресу **https://livebook.manning.com/book/fullstack-graphql-applications**. Все примеры, что приводятся в книге, доступны для загрузки на веб-сайте издательства Manning (**www.manning.com**) и в репозитории GitHub по адресу **https://github.com/johnymontana/fullstack-graphql-book**.

Требования к программному обеспечению

Для следования за примерами в книге необходимо установить последнюю версию Node.js. Все примеры я опробовал с версией v16. По своему опыту рекомендую использовать инструмент `nvm` для установки и управления версиями Node.js. Инструкции по установке и использованию `nvm` можно найти по адресу **https://github.com/nvm-sh/nvm**.

Мы также будем применять несколько (бесплатных) онлайн-служб для развертывания. Доступ к большинству из них можно получить с помощью учетной запи-

си GitHub, поэтому обязательно зарегистрируйтесь на GitHub (**https://github.com/**), если вы этого еще не сделали.

Живое обсуждение книги

Приобретая книгу «Разработка веб-приложения GraphQL с React, Node.js и Neo4j», вы получаете бесплатный доступ к онлайн-платформе liveBook для чтения книг издательства Manning. Благодаря эксклюзивным возможностям этой платформы вы можете оставлять свои комментарии к книге как в целом, так и к определенным разделам или абзацам, добавлять заметки для себя, задавать технические вопросы и отвечать на них, а также получать помощь от автора и других пользователей. Чтобы получить доступ к форуму и зарегистрироваться на нем, откройте в веб-браузере страницу **https://livebook.manning.com/book/fullstack-graphql-applications/discussion**. Узнать больше о форумах Manning и познакомиться с правилами поведения можно по адресу **https://livebook.manning.com/discussion**.

Издательство Manning обязуется предоставить своим читателям место встречи, где может состояться содержательный диалог между отдельными читателями и между читателями и автором. Но со стороны авторов отсутствуют какие-либо обязательства уделять форуму какое-то определенное внимание – их присутствие на форуме остается добровольным (и неоплачиваемым). Мы предлагаем задавать авторам стимулирующие вопросы, чтобы их интерес не угасал! Форум и архив с предыдущими обсуждениями остаются доступны на сайте издательства, пока книга продолжает издаваться.

Другие онлайн-ресурсы

Вам обязательно пригодится документация к библиотеке Neo4j GraphQL, доступная по адресу **https://neo4j.com/docs/graphql-manual/current/**. В числе других полезных онлайн-ресурсов можно назвать бесплатные онлайн-курсы на GraphAcademy (**https://graphacademy.neo4j.com/**), сайт сообщества Neo4j (**https://community.neo4j.com/**).

Об авторе

Уильям Лион (William Lyon) – консультант в Neo4j, где он помогает разработчикам успешно создавать приложения с графовыми базами данных. До прихода в Neo4j работал инженером-программистом в стартапах, занимающихся созданием финансовых систем, мобильных приложений для индустрии недвижимости и прогнозными API. Имеет степень магистра информатики, полученную в университете штата Монтана, и ведет блог на **lyonwj.com**.

Об иллюстрации на обложке

Рисунок на обложке книги называется «Dame de l'Isle de Tinne» (леди с острова Тинне) из коллекции Жака Грассе де Сен-Совер (Jacques Grasset de Saint-Sauveur), опубликованной в 1797 году. Все иллюстрации в этой коллекции тщательно прорисованы и раскрашены вручную.

В те дни по одежде было легко определить, где живет человек, чем занимается и какое положение занимает в обществе. Мы в издательстве Manning славим изобретательность, предприимчивость и радость компьютерного бизнеса обложками книг, изображающими богатство региональных различий многовековой давности, оживших благодаря иллюстрациям, таким как эта.

Часть I

Введение в стек GraphQL

Прежде чем начать путешествие в стек GraphQL, рассмотрим технологии, которые будут использоваться, и мощную концепцию графового мышления. Этот раздел посвящен серверной части нашего приложения и, в частности, базе данных и GraphQL API.

В главе 1 вы познакомитесь с компонентами приложения GraphQL полного цикла и с конкретными технологиями, которые будут использоваться на протяжении всей книги: GraphQL, React, Apollo и БД Neo4j. В главе 2 мы с головой погрузимся в GraphQL и основы создания GraphQL API. В главе 3 исследуем графовую базу данных Neo4j, модель данных графа свойств и язык запросов Cypher. В главе 4 посмотрим, как использовать интеграцию базы данных для поддержки GraphQL и, в частности, библиотеку Neo4j GraphQL для создания GraphQL API, поддерживаемых графовой базой данных. По завершении первой части книги у вас будет готовая к экспериментам база данных и начальное приложение GraphQL API, после чего вы сможете перейти ко второй части книги и приступить к созданию внешнего интерфейса.

Глава 1

Что такое стек GraphQL?

В этой главе:

- компоненты, составляющие типичное приложение GraphQL полного цикла;
- технологии, используемые в книге (GraphQL, React, Apollo и БД Neo4j), и их сочетание в контексте приложения полного цикла;
- требования к приложению, которое будет создаваться на протяжении всей книги.

1.1. Обзор стека GraphQL

В этой главе мы познакомимся с технологиями, которые будут использоваться на протяжении всей книги:

- GraphQL – для создания API;
- React – для создания пользовательского интерфейса и клиентского веб-приложения на JavaScript;
- Apollo – инструменты для работы с GraphQL как на сервере, так и на клиенте;
- Neo4j – база данных, которую мы используем для хранения данных приложения и управления ими.

Создание приложения GraphQL полного цикла предполагает работу с многоуровневой архитектурой, широко известной как *трехуровневое приложение*, которая состоит из внешнего интерфейса, уровня API и базы данных. На рис. 1.1 можно видеть отдельные компоненты приложения GraphQL полного цикла и их взаимодействие друг с другом.

На протяжении всей книги мы будем использовать эти технологии и компоненты для создания простого приложения, подробно обсуждая каждый в процессе реализации. А основные требования к приложению будут перечислены в последнем разделе этой главы.

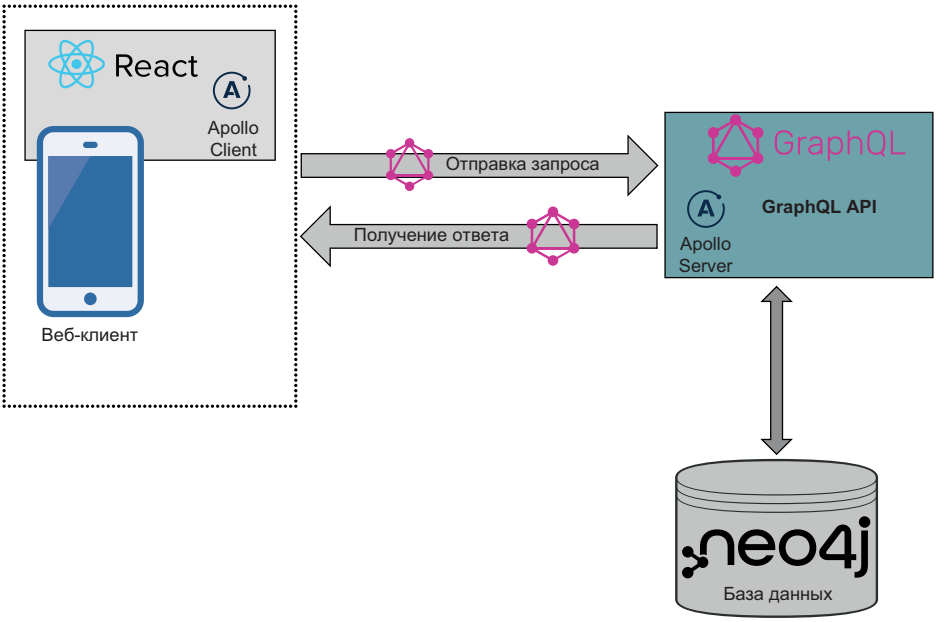

Рис. 1.1. Компоненты приложения GraphQL полного цикла: GraphQL, React, Apollo и база данных Neo4j

Основное внимание в данной книге уделяется изучению приемов создания приложений GraphQL, поэтому рассматривать GraphQL мы будем на примере приложения полного цикла в комплексе с другими технологиями, включая разработку схемы, интеграцию с базой данных, создание веб-интерфейса, обращающегося к GraphQL API, добавление аутентификации и т. д. Учитывая все это, книга предполагает наличие у читателя некоторых базовых знаний об особенностях создания веб-приложений, но не требует опыта работы с каждой конкретной технологией. Чтобы добиться успеха, читатель должен иметь базовые навыки программирования на JavaScript как на стороне клиента, так и в Node.js, а также владеть такими понятиями, как API (Application Programming Interface – прикладной программный интерфейс) и базы данных. Для опробования примеров должен быть установлен пакет `node` и желательно уметь пользоваться инструментом командной строки `npm` (или `yarn`) для создания проектов Node.js и установки зависимостей. Мы будем использовать последнюю LTS-версию Node.js (на момент написания этих строк – версия 16.14.2), которую можно получить по адресу **https://nodejs.org/**. Для управления версиями Node.js можно использовать диспетчер версий `nvm`. Дополнительную информацию вы найдете по адресу **https://github.com/nvm-sh/nvm**.

Перед обсуждением каждой технологии дается краткое введение и по мере необходимости предлагаются дополнительные источники более подробной информации. Также в процессе обсуждения конкретных технологий, используемых вместе с GraphQL, будут перечисляться другие аналогичные технологии (технология создания веб-интерфейса Vue, которую можно использовать вместо React). В конечном счете цель этой книги – показать, как эти технологии сочетаются друг с другом, и помочь читателю составить полную картину стека технологий для создания приложений на основе GraphQL.

1.2. GraphQL

GraphQL – это спецификация для создания API. Она описывает язык запросов к API и способ выполнения этих запросов. При создании GraphQL API разработчик описывает доступные данные, используя строгую систему типов. Эти описания, также определяющие точки входа в API, становятся спецификацией, основываясь на которой, клиент может запросить необходимые ему данные.

GraphQL обычно рассматривают как альтернативу REST – парадигме разработки API, наверняка знакомой вам. Это верное суждение, но лишь в некоторых случаях, потому что GraphQL также может обертывать существующие REST API или другие источники данных. Это обусловлено независимостью GraphQL от хранилища данных, благодаря которой GraphQL можно использовать с любыми источниками данных.

> *GraphQL – это язык запросов для API и среда выполнения этих запросов. GraphQL предоставляет полное и понятное описание данных, доступных в API, дает клиентам возможность запрашивать именно то, что им нужно, и ничего больше, тем самым упрощая развитие API с течением времени и давая разработчику мощные инструменты.*
>
> – graphql.org (**https://graphql.org/**)

А теперь более конкретно рассмотрим некоторые аспекты GraphQL.

1.2.1. *Определения типов в GraphQL*

GraphQL API организован не вокруг конечных точек, соответствующих ресурсам (как в REST), а вокруг определений, описывающих типы данных, поля и связи между ними. Эти определения типов становятся схемой API, который обслуживается одной конечной точкой.

Поскольку службы GraphQL могут быть реализованы на любом языке, для описания типов GraphQL используется свой универсальный язык определения схем GraphQL Schema Definition Language (SDL). Рассмотрим пример на рис. 1.2 – простое приложение для работы с фильмотекой. Представьте, что вас наняли для создания веб-сайта, позволяющего пользователям выполнять поиск сведений о фильмах в каталоге по их названиям, именам актеров и описаниям, а также показывать похожие фильмы, которые могут быть интересны пользователям.

Начнем с создания нескольких простых определений типов GraphQL (листинг 1.1), определяющих предметную область приложения.

Листинг 1.1. Простые определения типов для GraphQL API фильмотеки

```
type Movie {
  movieId: ID!
  title: String
  actors: [Actor]
}
```

`Movie` – **это тип объекта GraphQL, содержащего одно или несколько полей**

`title` – **это поле типа** `String`

Поля могут ссылаться на другие типы, например в данном случае на список объектов типа `Actor`

```
type Actor {
  actorId: ID!
  name: String
  movies: [Movie]
}

type Query {
  allActors: [Actor]
  allMovies: [Movie]
  movieSearch(searchString: String!): [Movie]
  moviesByTitle(title: String!): [Movie]
}
```

`ActorId` – **обязательное (или непустое), на что указывает символ** `!`**, поле типа** `Actor`

`Query` – **специальный тип в GraphQL, определяющий точки входа в API**

Поля также могут иметь аргументы; в этом случае поле `movieSearch` **принимает обязательный строковый аргумент** `searchString`

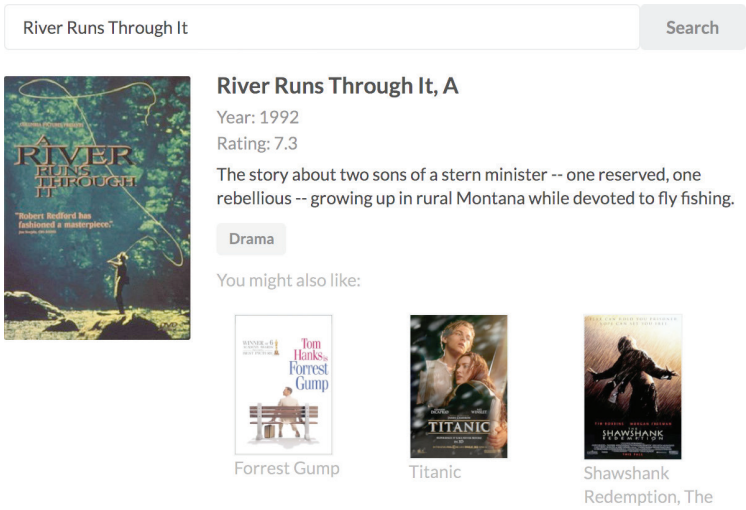

Рис. 1.2. Простое веб-приложение фильмотеки

Наши определения объявляют типы GraphQL, используемые в API, их поля и связи между ними. При определении типа объекта (например, `Movie`) также указываются все поля, доступные в объекте, и их типы (позже можно добавить дополнительные поля, используя ключевое слово `extend`). В этом примере поле `title` определяется со скалярным типом `String`, т. е. поле может содержать только одно значение. В отличие от него, поля объектных типов могут содержать несколько полей и ссылок на другие типы. В данном примере `actors` – это поле типа `[Actor]`, оно может содержать массив объектов `Actor` и определяет связь между типами `Movie` и `Actor` (такие связи образуют «граф»).

Поля могут быть необязательными или обязательными. Поле `actorId` в типе `Actor` является обязательным (т. е. оно не может быть пустым). Это означает, что каждый объект `Actor` должен иметь значение в поле `actorId`. Поля без восклицательного знака (`!`) в определении могут иметь значение `NULL`, т. е. они – необязательные.

Поля в типе `Query` определяют точки входа в службу GraphQL. Схемы GraphQL также могут содержать тип `Mutation`, определяющий точки входа для операций

записи в API. Третий особый тип, связанный с точками входа, – тип `Subscription`. Он определяет события, на которые клиент может подписаться.

> **ПРИМЕЧАНИЕ.** Здесь мы опускаем многие важные концепции GraphQL, такие как операции изменения, типы интерфейсов и объединений и т. д., но не волнуйтесь; мы только начинаем и скоро доберемся до них!

На этом этапе вам может быть интересно, где хранится граф GraphQL. Определяя типы GraphQL, мы фактически определяем граф. Граф – это структура данных, состоящая из узлов (сущностей или объектов) и отношений, соединяющих узлы. Именно такую структуру мы определили в описаниях типов на языке SDL. Определения выше задают простой граф со структурой, изображенной на рис. 1.3.

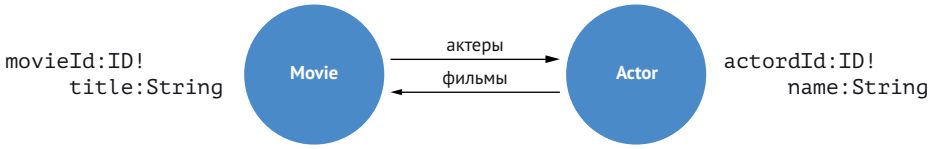

Рис. 1.3. Определения типов GraphQL для веб-приложения фильмотеки в виде графа

Графы предназначены для описания связанных данных, и здесь мы определили, как фильмы и актеры связаны между собой. GraphQL позволяет моделировать данные в виде графа и перемещаться по графу с помощью операций GraphQL.

Когда служба GraphQL получает запрос на выполнение операции, она проверяет и выполняет эту операцию в соответствии со схемой определений типов. Давайте рассмотрим пример запроса, который служба GraphQL может выполнить, руководствуясь заданными выше определениями типов.

1.2.2. *Запросы GraphQL*

Запросы GraphQL определяют порядок обхода графа данных в соответствии с определениями типов и запрашивают подмножество полей для возврата в ответе – это называется *выборкой множества*. Следующий запрос начинает обход графа с точки входа, заданной в поле запроса `allMovies`, и отыскивает актеров, связанных с каждым фильмом (листинг 1.2). Затем для каждого актера выполняется поиск других фильмов, в которых они снимались.

Листинг 1.2. Запрос GraphQL для поиска актеров и фильмов

```
query FetchSomeMovies {
  allMovies {
    title
    actors {
      name
      movies {
        title
      }
    }
  }
```

Необязательное имя операции. По умолчанию используется имя `query`, и его можно опустить. Имя операции – в данном случае `FetchSomeMovies` – тоже необязательное и может быть опущено

Здесь указывается точка входа, поле в типе `Query` или `Mutation`. В этом случае точкой входа для запроса является поле `allMovies` в типе `Query`

Выборка множества определяет поля, которые должны быть возвращены в ответ на запрос

Если предполагается вернуть поле объектного типа, следует определить вложенную выборку множества, описывающую возвращаемые поля

Для возврата полей объектов `Movie` необходимо определить вложенную выборку множества

```
  }
}
```

Обратите внимание, что наш запрос является вложенным и описывает порядок обхода графа связанных объектов (в данном случае фильмов и актеров). Этот обход и его результаты можно представить в виде графа данных визуально (рис. 1.4).

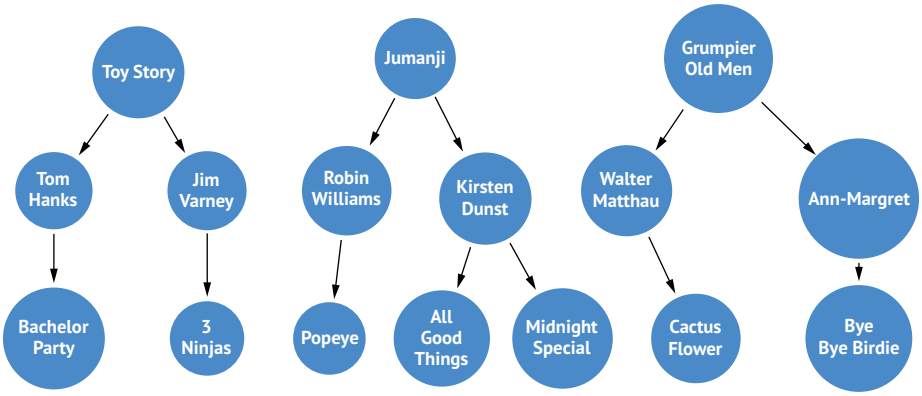

Рис. 1.4. Порядок обхода графа данных при выполнении запроса GraphQL

Обход графа данных можно представить визуально, но типичным результатом запроса GraphQL является документ JSON, как показано в листинге 1.3.

Листинг 1.3. Результаты запроса в формате JSON

```
"data": {
  "allMovies": [
    {
      "title": "Toy Story",
      "actors": [
        {
          "name": "Tom Hanks",
          "movies": [
            {
              "title": "Bachelor Party"
            }
          ]
        },
        {
          "name": " Jim Varney",
          "movies": [
            {
              "title": "3 Ninjas: High Noon On Mega Mountain"
            }
          ]
        }
      ]
    },
    {
```

```
      "title": "Jumanji",
      "actors": [
        {
          "name": "Robin Williams",
          "movies": [
            {
              "title": "Popeye"
            }
          ]
        },
        {
          "name": "Kirsten Dunst",
          "movies": [
            {
              "title": "Midnight Special"
            },
            {
              "title": "All Good Things"
            }
          ]
        }
      ]
    },
    {
      "title": "Grumpier Old Men",
      "actors": [
        {
          "name": "Walter Matthau",
          "movies": [
            {
              "title": "Cactus Flower"
            }
          ]
        },
        {
          "name": " Ann-Margret",
          "movies": [
            {
              "title": "Bye Bye Birdie"
            }
          ]
        }
      ]
    }
  ]
}
```


Как можно заметить, ответ соответствует форме полученной выборке множества – возвращаются только поля, указанные в запросе. Но откуда берутся данные? Логика получения данных GraphQL API определяется так называемыми *функциями разрешения* (resolver functions), которые определяют, какие данные со-