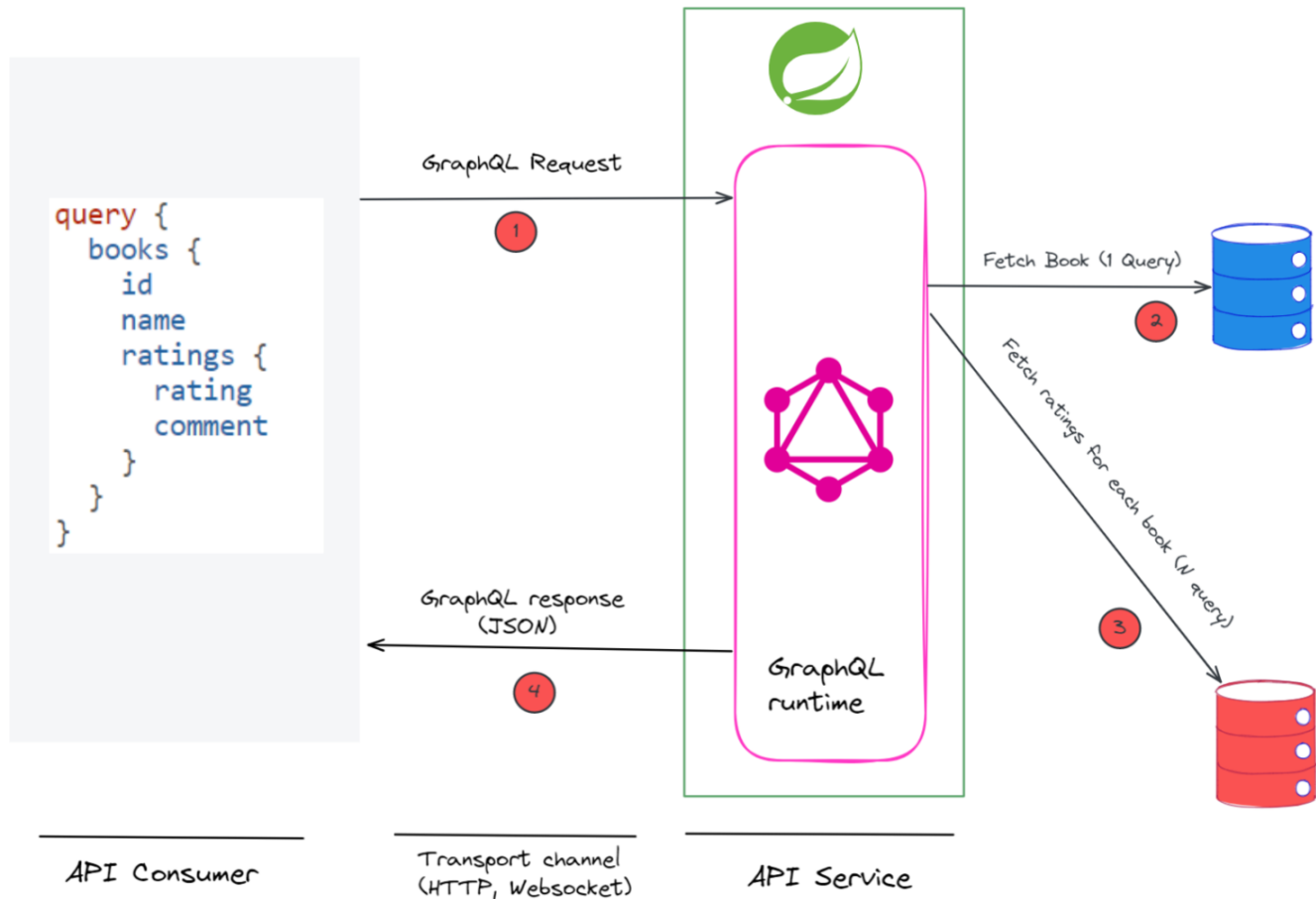




Spring for GraphQL: How to solve the N+1 Problem?

by **Pankaj** — August 7, 2022 in **GraphQL** Reading Time: 11 mins read

0 0 0

0
SHARES 2.3k
VIEWS

f Share on Facebook

t Share on Twitter

in Share on LinkedIn



[GraphQL promises to solve some major shortcomings associated with REST API](#), namely over-fetching and under-fetching. But, GraphQL is also prone to one problem, commonly known as **N + 1 problem**. In short, if the GraphQL service is not implemented correctly, it can cause significant latency in the API calls.

Not surprisingly, Spring for GraphQL provides an elegant solution to solve N + 1 problem.

So, what exactly is N + 1 problem in GraphQL?

- [What is the N + 1 problem?](#)
- [How to solve N + 1 problem?](#)
- [Another approach to solve the 'N + 1' problem](#)
- [Code example](#)
- [Summary](#)

What is the N + 1 problem?

Firstly, it's not as dreaded as it sounds 😊. Secondly, it has nothing to do with Big O, used to describe the algorithm's performance. Thirdly, the solution to the N + 1 problem is surprisingly straightforward.

But, before we discuss the solution, let's understand the problem first.

Let's try to understand this problem with one example.

Assuming, you have defined the following GraphQL Apis.

```
type Query {  
  books: [Book]  
}  
  
type Book {  
  id : ID  
  name : String  
  author: String  
  price: Float  
  ratings: [Rating]  
}  
  
type Rating {  
  id: ID  
  rating: Int  
  comment: String  
  user: String  
}
```

And on the service side, it's implemented as:

```
@QueryMapping
public Collection<Book> books() {
    return bookCatalogService.getBooks();
}

@SchemaMapping
public List<Rating> ratings(Book book) {
    return bookCatalogService.ratings(book);
}
```

In the above code, we have defined two **Data Fetcher** s

1. **books()** for field books of the GraphQL object type **Query** .
2. **ratings(..)** for field ratings of the type **Book** .

The important point to note here is if you don't specify a Data Fetcher for a field then the GraphQL assigns a default **PropertyDataFetcher** which, in turn, looks for the **public XXX getXXX()** method in the POJO object defined by the **Type** .

Therefore, in the above example, GraphQL resolves the field **name** from **public String getName()** method of the **Book** object.

Also, when the GraphQL service executes a query, it calls a Data Fetcher for every **field** .

So, if a GraphQL client requests the following data,

```
query {
  books {
    id
    name
    ratings {
      rating
      comment
    }
  }
}
```

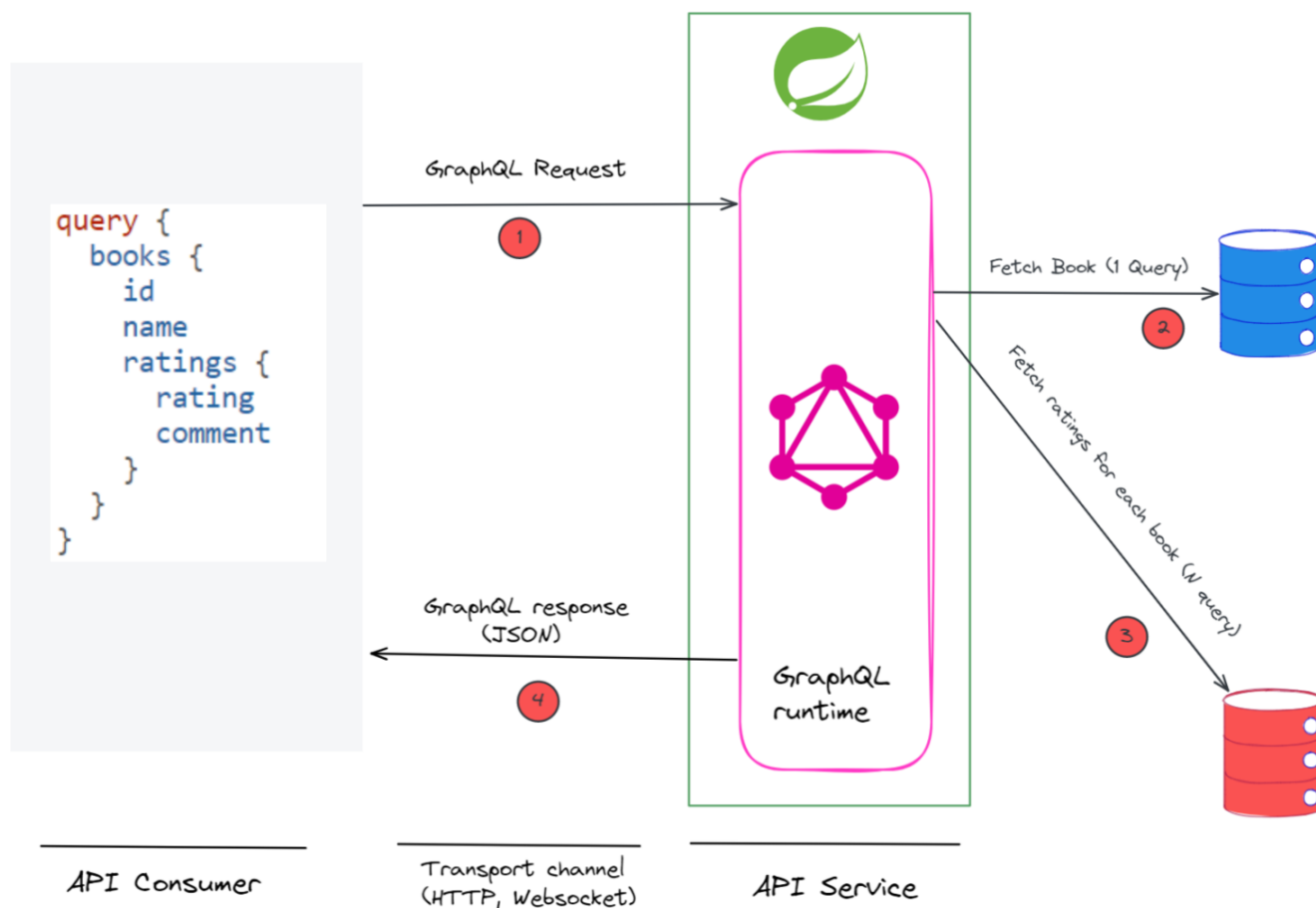
then the GraphQL runtime engine does the following.

1. Parses the request and validates the request against the schema.

2. Then it calls the `book` Data Fetcher (handler method `books()`) to fetch book information once.
3. And, then it calls the `ratings` Data Fetcher for each book.

Now, it's very much possible that `Books` and `Ratings` are stored in different databases, or books and ratings are different microservices altogether. In any case, this results in $1 + N$ network calls.

. . .



GraphQL n + 1 problem

. . .



The n+1 problem means that the GraphQL server executes many unnecessary round trips to fetch nested data.

If you check the service log then you will see that `rating` Data Fetcher is called sequentially for each and every call to book.

```
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching all books..  
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching rating for book, id 1  
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching rating for book, id 2  
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching rating for book, id 3  
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching rating for book, id 4  
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching rating for book, id 5
```

Log

How to solve N + 1 problem?

In Spring for GraphQL, you can solve this problem using `@BatchMapping` annotation.

You need to declare `@BatchMapping` annotation on a handler method that takes `List<Book>` and returns a `Map` containing `Book` and `List<Rating>` as:

```
@BatchMapping (field = "ratings", typeName = "Book")  
public Map<Book, List<Rating>> ratings(List<Book> books) {  
    log.info("Fetching ratings for all books");  
    return ratingService.ratingsForBooks(books);  
}
```



In `@Batchmapping`, Spring batches the call to the `rating Data Fetcher` .

You can also leave out the parameters `field` and `typeName` as the `field` name defaults to the method name, while the type name defaults to the simple class name of the input `List` element type.

```
@BatchMapping
public Map<Book, List<Rating>> ratings(List<Book> books) {
    log.info("Fetching ratings for all books");
    return ratingService.ratingsForBooks(books);
}
```

If you run the above query in GraphQL at <http://localhost:8080/graphql?path=/graphql>, you will see that the GraphQL engine batches the call to the `ratings` field.

```
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching all books..
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching ratings for books [Book[id=1, r
```

Service log

Another approach to solve the ‘N + 1’ problem

There is another way to solve N+1 in Spring for GraphQL – using low-level [GraphQL Java Data Loader](#) API. A `Data Loader` is a utility class that allows batch loading of data identified by a set of unique keys. It's a pure Java port of [Facebook DataLoader](#).

This is how it works.

1. Define a Data Loader that takes a set of unique keys and returns results.
2. Register the Data Loader in the `DataLoaderRegistry`.
3. Pass Data Loader to `@SchemaMapping` handler method.

A `DataLoader` defers loading by returning a future so it can be done in a batch. A `DataLoader` maintains a per request cache of loaded entities that can further improve performance.

Since `BatchLoaderRegistry` is available as a bean, you can inject that anywhere (in our case we have injected it in the `Controller` class for convenience).

```
batchLoaderRegistry
    .forTypePair(Book.class, List.class)
    .registerMappedBatchLoader(
        (books, env) -> {
            log.info("Calling loader for books {}", books);
            Map bookListMap = ratingService.ratingsForBooks(List.copyOf(books));
            return Mono.just(bookListMap);
        });
```

And then define a handler method that takes Data Loader as an argument.

```
@SchemaMapping
public CompletableFuture<List<Rating>> ratings(Book book, DataLoader<Book, List<Rating>>
    log.info("Fetching rating for book, id {}", book.id());
    return loader.load(book);
}
```

If you run the above query in GraphiQL at <http://localhost:8080/graphiql?path=/graphql>, you will see that even though **rating** Data Fetcher is called for every **book** but GraphQL engine batches call to **Data Loader** as a single call.

. . .

```
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching all books..
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching rating for book, id 1
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching rating for book, id 2
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching rating for book, id 3
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching rating for book, id 4
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Fetching rating for book, id 5
[ctor-http-nio-2] d.t.g.controller.BooksCatalogController : Calling loader for books [Book[id=1,
```

Service log

. . .

Code example

The working code example of this article is listed on [GitHub](#) . To run the example, clone the repository, and import **graphql-spring-batch** as a project in your favourite IDE as a **Gradle** project.

Previous Post

Next Post

Summary
Spring for GraphQL : @SchemaMapping and @QueryMapping

Spring for GraphQL: Mutation

A GraphQL service can be susceptible to N + 1 problem if not implemented correctly. The n+1 problem means that the GraphQL server executes many unnecessary round trips to fetch nested data. In Spring for GraphQL, we can solve this problem by defining a handler method with **@BatchMapping** annotation.

Tags: [graphql](#) [spring-boot](#)

**Pankaj**

Software Architect @ Schlumberger "" Cloud | Microservices | Programming | Kubernetes | Architecture | Machine Learning | Java | Python ""



Related Posts

GRAPHQL

GraphQL Directive

[🕒](#) SEPTEMBER 29, 2022

GRAPHQL

Spring for GraphQL: Mutation

[🕒](#) AUGUST 14, 2022

GRAPHQL

Spring for GraphQL : @SchemaMapping and @QueryMapping

[🕒](#) JULY 29, 2022

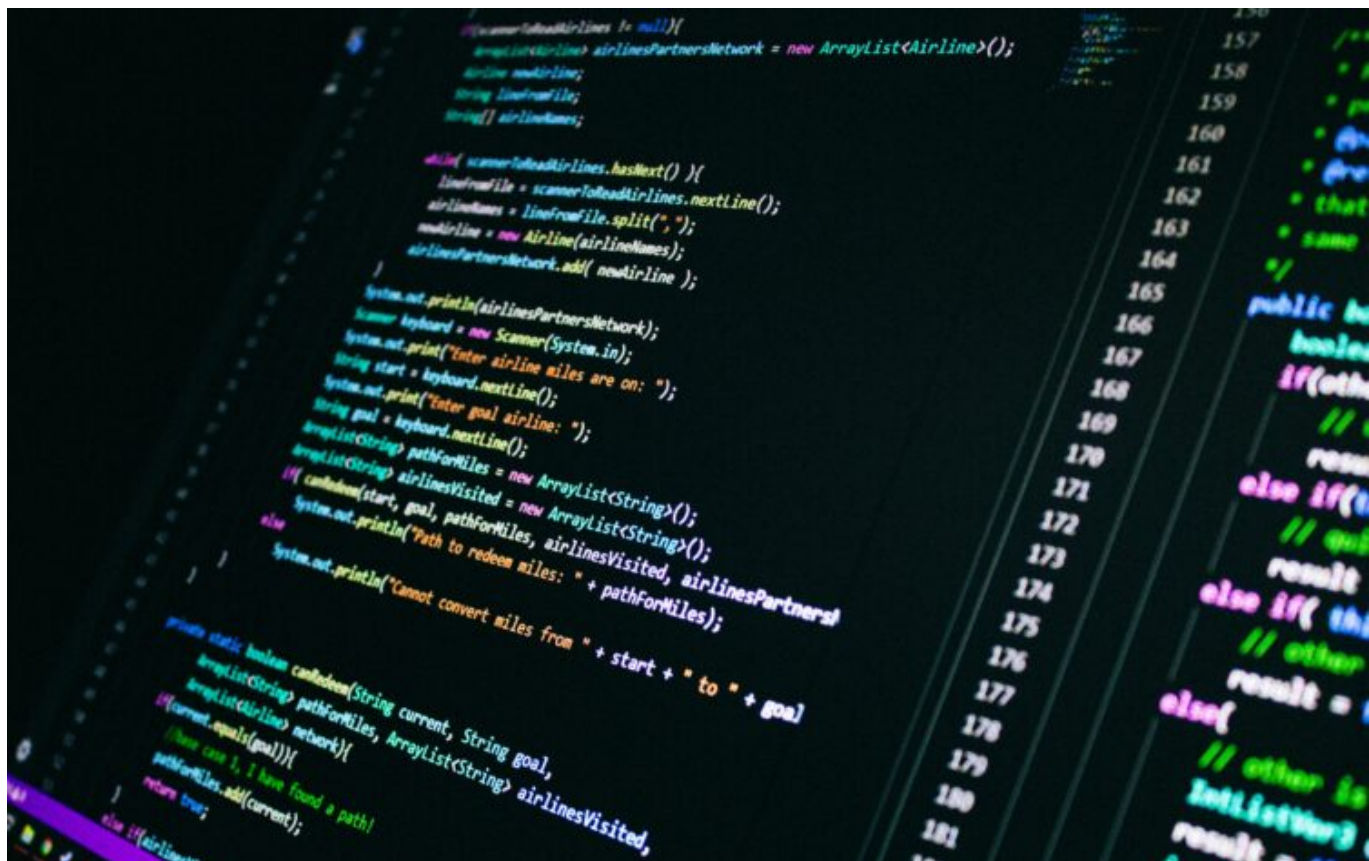
GRAPHQL

Getting started with Spring Boot GraphQL service

[🕒](#) JULY 23, 2022

Discussion about this post

Recent Articles



gRPC Bidirectional Streaming with Code Example

© FEBRUARY 17, 2023




gRPC Client Streaming

© JANUARY 20, 2023

Distributed Transactions in Microservices: Implementing Saga with Temporal

© NOVEMBER 8, 2022

Workflow Orchestration with Temporal and Spring Boot

 OCTOBER 29, 2022

Trending	Comments	Latest
----------	----------	--------

Deploying a RESTful Spring Boot Microservice on Kubernetes

🕒 AUGUST 18, 2021

Workflow Orchestration with Temporal and Spring Boot

🕒 OCTOBER 29, 2022

gRPC Interceptor: unary interceptor with code example

🕒 APRIL 30, 2022

gRPC for microservices communication

🕒 AUGUST 29, 2021



FACEBOOK



TWITTER



PINTEREST

TECHDOZO

Simplifying modern tech stack!

Browse by Category

Bitesize

Java

Spring Boot

GraphQL

Kubernetes

gRPC

Microservices

Recent Articles

gRPC Bidirectional Streaming with Code Example

🕒 FEBRUARY 17, 2023

gRPC Client Streaming

🕒 JANUARY 20, 2023

