《31 讲带你搞懂 SkyWalking》

徐郡明 资深技术专家

— 拉勾教育出品 —

第27讲：实战入门 GraphQL
如何将 REST API 换成 GraphQL

GraphQL 简介

拉勾教育
— 互联网人实战大学 —

GraphQL

是一个用于 API 的查询语言，是一个使用基于类型系统来执行查询的服务端运行时

并没有和任何特定数据库或者存储引擎绑定

对服务端 API 中的数据提供了一套易于理解的完整描述

使得客户端能够准确地获得它需要的数据，而且没有任何冗余

也让 API 更容易地随着时间推移而演进，还能用于构建强大的开发者工具

GraphQL 简介

- **可描述**

  使用 GraphQL，你获取的都是你想要的数据，不多也不会少

- **分级**

  GraphQL 天然遵循了对象间的关系，通过一个简单的请求，可以获取到一个对象及其相关的对象

- **强类型**

  使用 GraphQL 的类型系统，能够清晰、准确的描述数据

  这样就能确保从服务器获取的数据和我们查询的一致

GraphQL 简介

- **跨语言**

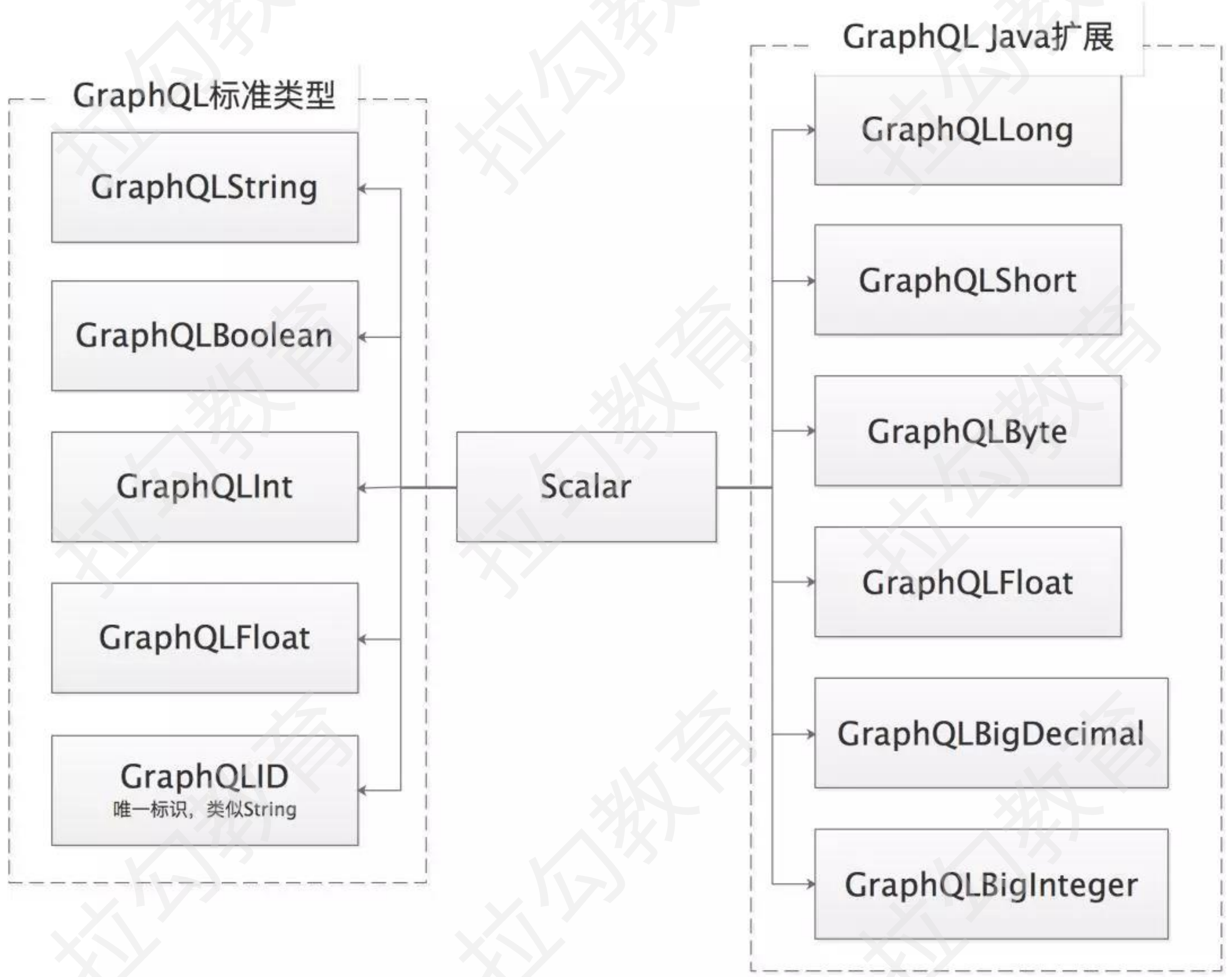
  GraphQL 并不绑定于某一特定的语言

- **兼容性**

  GraphQL 不限于某一特定存储平台

  GraphQL 可以方便地接入已有的存储、代码、甚至可以连接第三方的 API

GraphQL 类型系统



GraphQL 类型系统

```
type Book {
  id: ID # 编号
  name: String #书名
  pageCount: Int #页数
  author: Author # 作者
}
```

GraphQL 类型系统

```
interface ComicCharacter {
  name: String;
}
```

GraphQL 类型系统

```
type Cat {
  name: String;
  lives: Int;
}
type Dog {
  name: String;
  bonesOwned: int;
}
union Pet = Cat | Dog
```

GraphQL 类型系统

拉勾教育
— 互 联 网 人 实 战 大 学 —

InputObject

主要用于封装方法参数，GraphQL Schema 中的定义与 Object 类似

主要区别是将 type 关键字换成 input 关键字

GraphQL Java 中对应的类型是 GraphQLInputObjectType

Enum

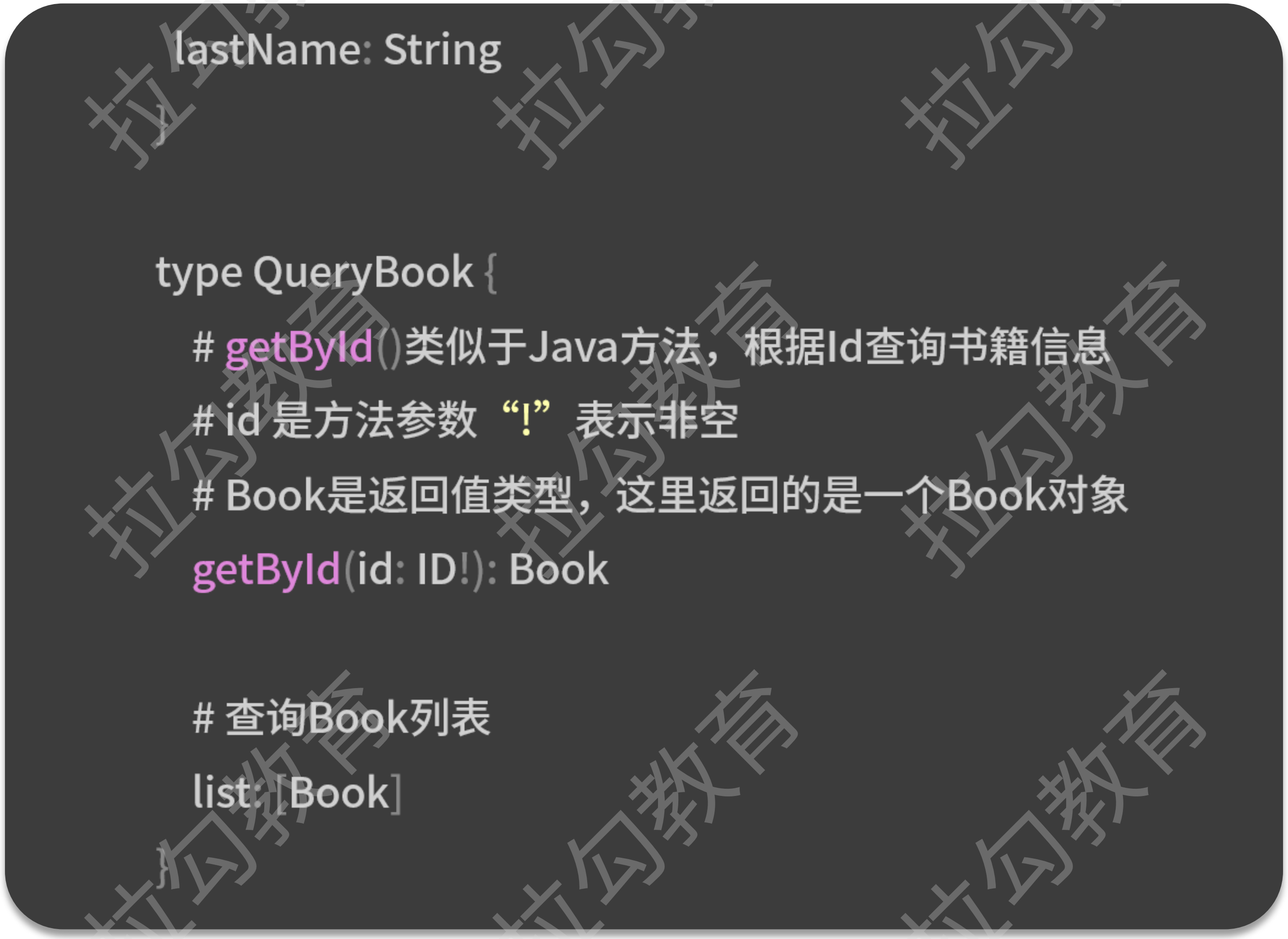
类似于 Java 中的枚举

GraphQL Java基础入门

```
type Book {
  id: ID # 编号
  name: String #书名
  pageCount: Int #页数
  author: Author # 作者
}

type Author {
 id: ID # 作者编号
 firstName: String # 作者书名
 lastName: String
}
```

GraphQL Java基础入门

拉勾教育
— 互 联 网 人 实 战 大 学 —



```
    lastName: String
}


type QueryBook {
    # getById()类似于Java方法，根据Id查询书籍信息
    # id 是方法参数“!”表示非空
    # Book是返回值类型，这里返回的是一个Book对象
    getById(id: ID!): Book

    # 查询Book列表
    list: [Book]
}
```

加载 GraphQL Schema 文件

拉勾教育

— 互 联 网 人 实 战 大 学 —

```java
@Component
public class GraphQLProvider {
  private GraphQL graphQL;
  @Bean
  public GraphQL graphQL() {
    return graphQL;
  }

  @PostConstruct
  public void init() throws IOException {
    //读取 GraphQL Schema文件并创建 GraphQL实例，
    //该GraphQL实例会通过上面的 graphQL()方法暴露给Spring，
    //默认情况下，请求到"/graphql"这个path上的请求都会由该GraphQL实例处理
    URL url = Resources.getResource("book.graphqls");
    String sdl = Resources.toString(url, Charsets.UTF_8);
    GraphQLSchema graphQLSchema = buildSchema(sdl);
    this.graphQL = GraphQL.newGraphQL(graphQLSchema).build();
  }
```

加载 GraphQL Schema 文件

```
    URL url = Resources.getResource("book.graphqls");
    String sdl = Resources.toString(url, Charsets.UTF_8);
    GraphQLSchema graphQLSchema = buildSchema(sdl);
    this.graphQL = GraphQL.newGraphQL(graphQLSchema).build();
  }

  private GraphQLSchema buildSchema(String sdl) {
    // GraphQL Schema文件被解析之后，就是这里的 TypeDefinitionRegistry对象
    TypeDefinitionRegistry typeRegistry = new SchemaParser().parse(sdl);
    //注册DataFetcher，DataFetcher的介绍以及buildWiring()方法实现在后面马上会进行介绍
    RuntimeWiring runtimeWiring = buildWiring();
    SchemaGenerator schemaGenerator = new SchemaGenerator();
    //将GraphQL Schema中定义的与 DataFetcher关联起来
    return schemaGenerator.makeExecutableSchema(typeRegistry, runtimeWiring);
  }

  //这哪是省略buildWiring()方法
}
```

加载 GraphQL Schema 文件

```
    URL url = Resources.getResource("book.graphqls");
    String sdl = Resources.toString(url, Charsets.UTF_8);
    GraphQLSchema graphQLSchema = buildSchema(sdl);
    this.graphQL = GraphQL.newGraphQL(graphQLSchema).build();
  }

  private GraphQLSchema buildSchema(String sdl) {
    // GraphQL Schema文件被解析之后，就是这里的 TypeDefinitionRegistry对象
    TypeDefinitionRegistry typeRegistry = new SchemaParser().parse(sdl);
    //注册DataFetcher，DataFetcher的介绍以及buildWiring()方法实现在后面马上会进行介绍
    RuntimeWiring runtimeWiring = buildWiring();
    SchemaGenerator schemaGenerator = new SchemaGenerator();
    //将GraphQL Schema中定义的与 DataFetcher关联起来
    return schemaGenerator.makeExecutableSchema(typeRegistry, runtimeWiring);
  }

  //这哪是省略buildWiring()方法
}
```

加载 GraphQL Schema 文件

```
public interface DataFetcher<T> {
    // DataFetchingEnvironment中记录了很多信息，例如:
    //该DataFetcher对应的字段以及类型、查询的外层对象以及根对象、当前上下文信息等等一系列信息
    T get(DataFetchingEnvironment dataFetchingEnvironment) throws Exception;
}
```

加载 GraphQL Schema 文件

```java
@Autowired
GraphQLDataFetchers graphQLDataFetchers;

private RuntimeWiring buildWiring() {
    return RuntimeWiring.newRuntimeWiring()
            //将Query.getById与getBookByIdDataFetcher()方法返回的DataFetcher实现关联
            .type(newTypeWiring("Query").dataFetcher("getById",
                        graphQLDataFetchers.getBookByIdDataFetcher())
                        .dataFetcher("list", graphQLDataFetchers.listDataFetcher()))
            //将Book.author字段与getBookByIdDataFetcher()方法返回的DataFetcher实现关联
            .type(newTypeWiring("Book").dataFetcher("author",
                        graphQLDataFetchers.getAuthorDataFetcher()))
            .build();
}
```

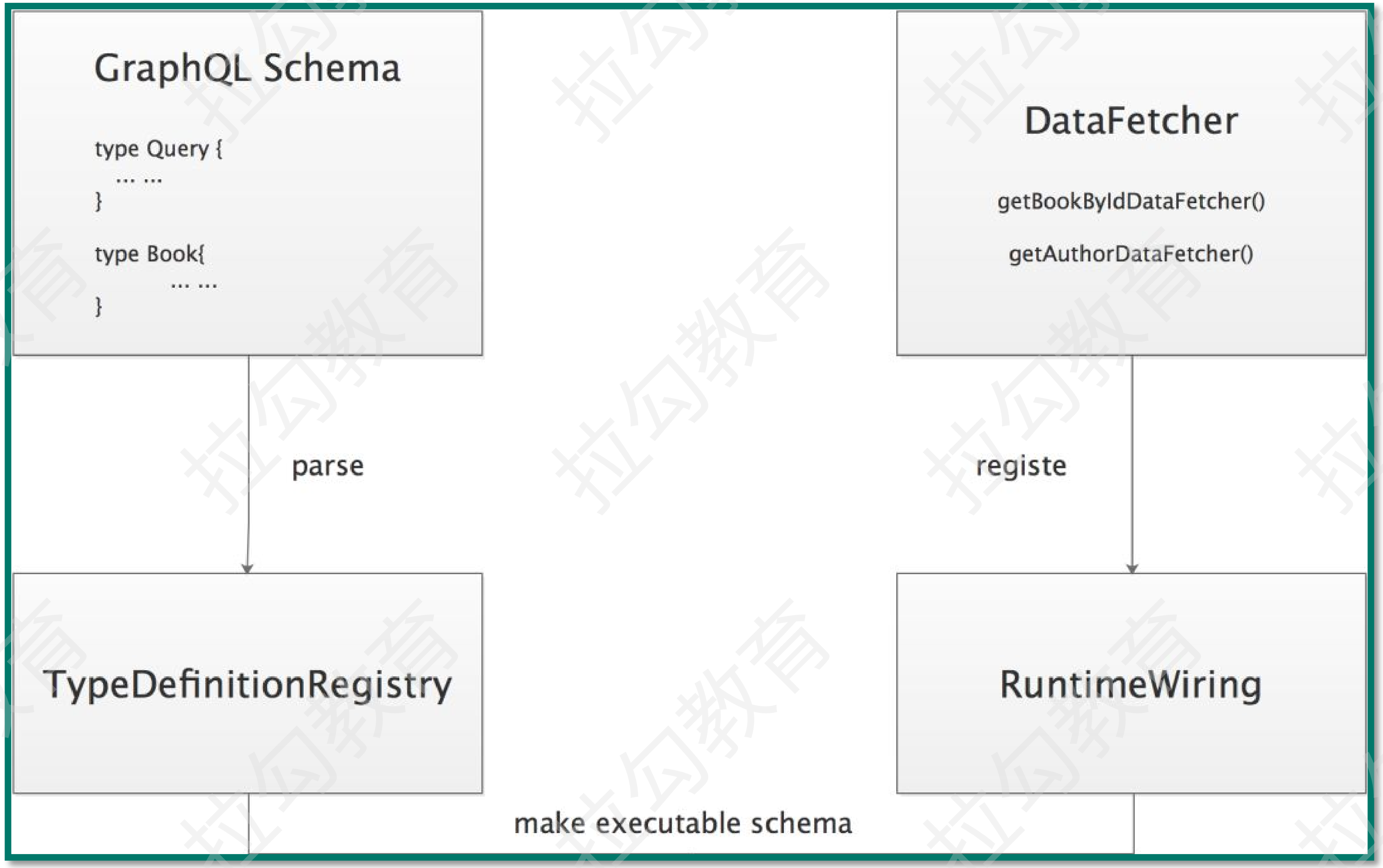
加载 GraphQL Schema 文件

```
@Component
public class GraphQLDataFetchers {

    private static List<ImmutableMap<String, String>> books = Arrays.asList(
            ImmutableMap.of("id", "book-1""name", "Harry Potter and the Philosopher's
Stone""pageCount", "223""authorId", "author-1"),
    );
    private static List<ImmutableMap<String, String>> authors = Arrays.asList(
            ImmutableMap.of("id", "author-1""firstName", "Joanne""lastName", "Rowling"),
    );

    public DataFetcher getBookByIdDataFetcher() {
        return dataFetchingEnvironment -> {
            //获取 id参数，然后根据id查找 books集合并返回相应的 Book信息
            String bookId = dataFetchingEnvironment.getArgument("id");
            return books.stream().filter(book -> book.get("id").equals(bookId))
                .findFirst().orElse(null);
        };
    }
```

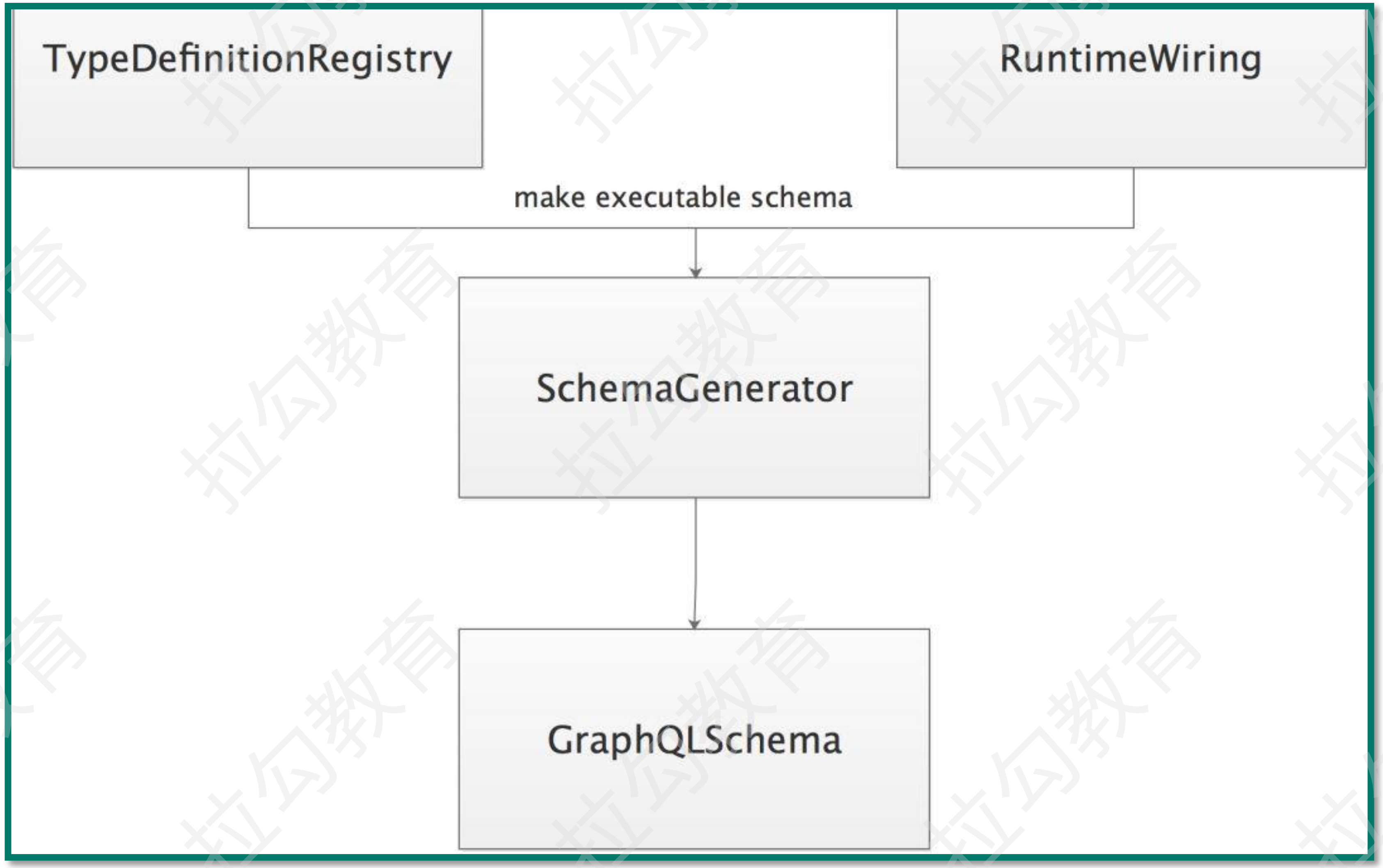
加载 GraphQL Schema 文件

```
            .findFirst().orElse(null);
    };
}

public DataFetcher getAuthorDataFetcher() {
    return dataFetchingEnvironment -> {
        // DataFetcher 会按照 GraphQL Schema定义从外层向内层调用
        //这里可以直接通过 DataFetchingEnvironment获取外层 DataFetcher查找到的数据(即关联的Book)
        Map<String, String> book = dataFetchingEnvironment.getSource();
        String authorId = book.get("authorId"); //根据 authorId查找作者信息
        return authors.stream().filter(author -> author.get("id").equals(authorId))
            .findFirst().orElse(null);
    };
}

public DataFetcher listDataFetcher() {
    return dataFetchingEnvironment -> books;
}
}
```

加载 GraphQL Schema 文件

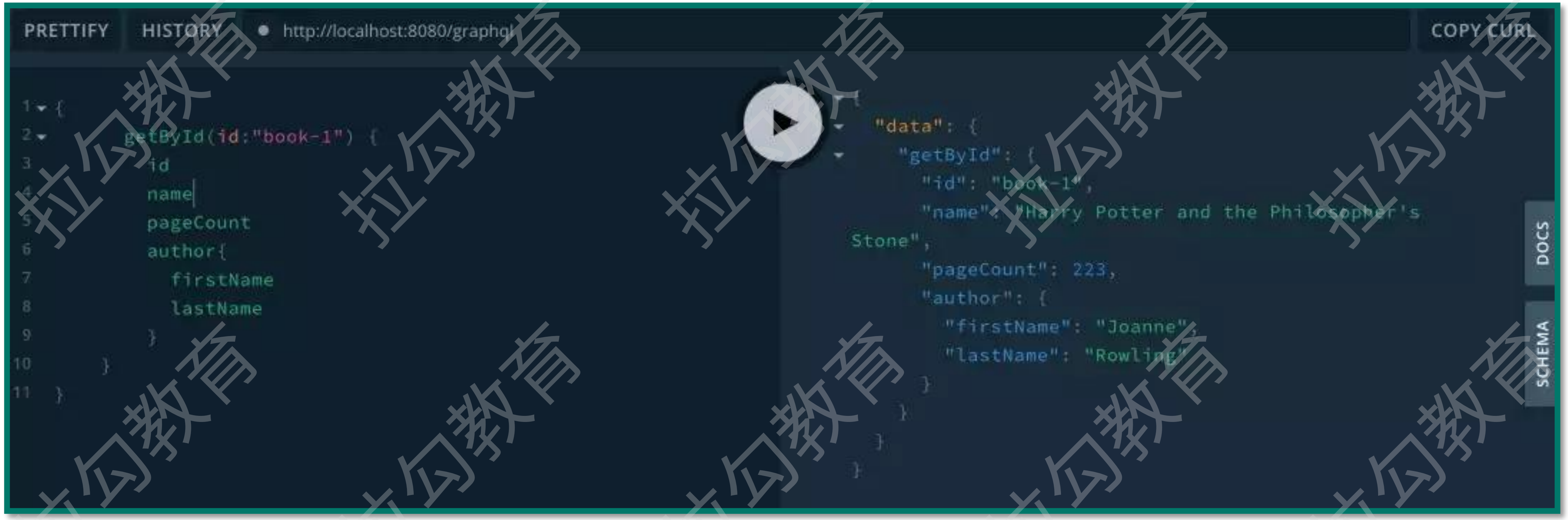


加载 GraphQL Schema 文件



加载 GraphQL Schema 文件

启动该 Spring 项目之后，可以使用 GraphQL Playground 访问"/graphql"
并传入查询 Book 的请求



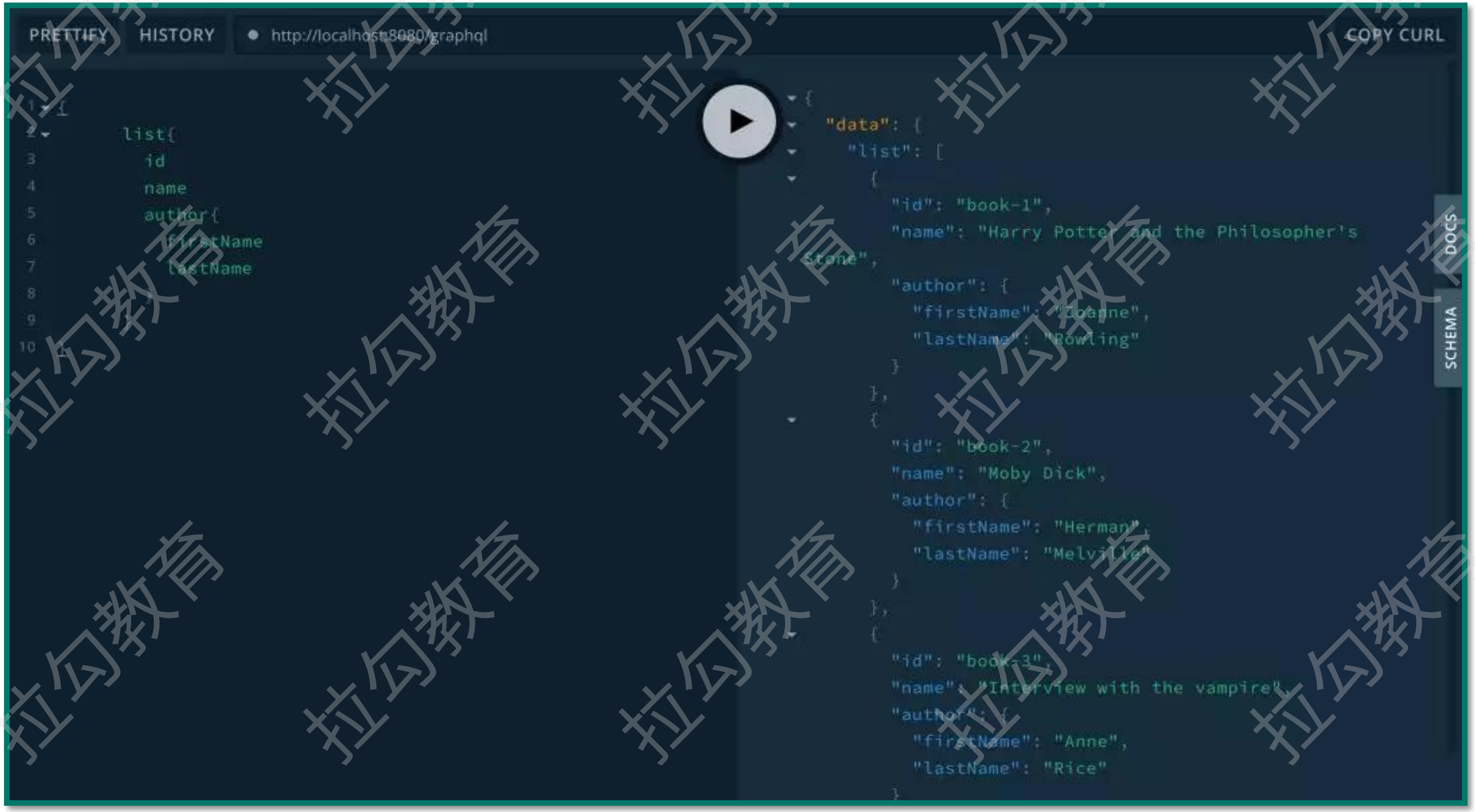
```
{
    getById(id:"book-1") {
      id
      name
      pageCount
      author{
        firstName
        lastName
      }
    }
}
```

```
{
  "data": {
    "getById": {
      "id": "book-1",
      "name": "Harry Potter and the Philosopher's
Stone",
      "pageCount": 223,
      "author": {
        "firstName": "Joanne",
        "lastName": "Rowling"
      }
    }
  }
}
```

加载 GraphQL Schema 文件

```
curl 'http://localhost:8080/graphql' -H 'Content-Type:
application/json' --data-binary '{"query":"\n{\n
getById(id:\"book-1\") {\n    id\n      name\n
pageCount\n      author{\n      \tfirstName\n
\tlastName\n      }\n   }\n}"}' --compressed
```

加载 GraphQL Schema 文件



```
{
    list{
      id
      name
      author{
        firstName
        lastName
      }
    }
}
```

```
{
  "data": {
    "list": [
      {
        "id": "book-1",
        "name": "Harry Potter and the Philosopher's Stone",
        "author": {
          "firstName": "Joanne",
          "lastName": "Rowling"
        }
      },
      {
        "id": "book-2",
        "name": "Moby Dick",
        "author": {
          "firstName": "Herman",
          "lastName": "Melville"
        }
      },
      {
        "id": "book-3",
        "name": "Interview with the vampire",
        "author": {
          "firstName": "Anne",
          "lastName": "Rice"
        }
```

GraphQL Java Tools 入门

GraphQL Java Tools 可以帮助我们屏蔽底层的 GraphQL Java 中的复杂概念和重复代码

能够从 GraphQL Schema 定义（即 .graphqls 文件）中构建出相应的 Java 的 POJO 类型对象

将读取 classpath 下所有以 .graphqls 为后缀名的文件，然后创建 GraphQL Schema 对象

GraphQL Java Tools 也是依赖于 GraphQL Java 实现的

GraphQL Java Tools 入门

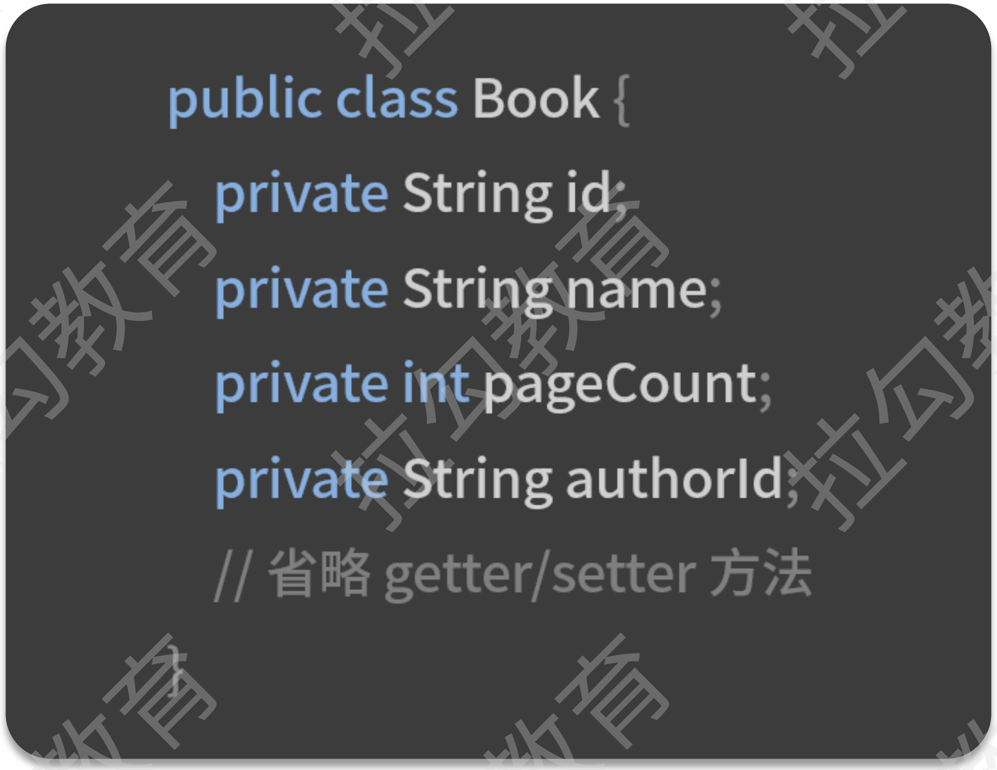
```
extend type Query{ # 扩展 Query
  getAuthorById(id: ID!): Author # 根据 id 查询作者信息
}
```

GraphQL Java Tools 入门

```
type Mutation {
   createBook(input : BookInput!) : Book!
   createAuthor(firstName:String!, lastName:String!) : ID!
}

input BookInput {  # input 表示入参
   name : String!
   pageCount : String!
   authorId: String!
}
```

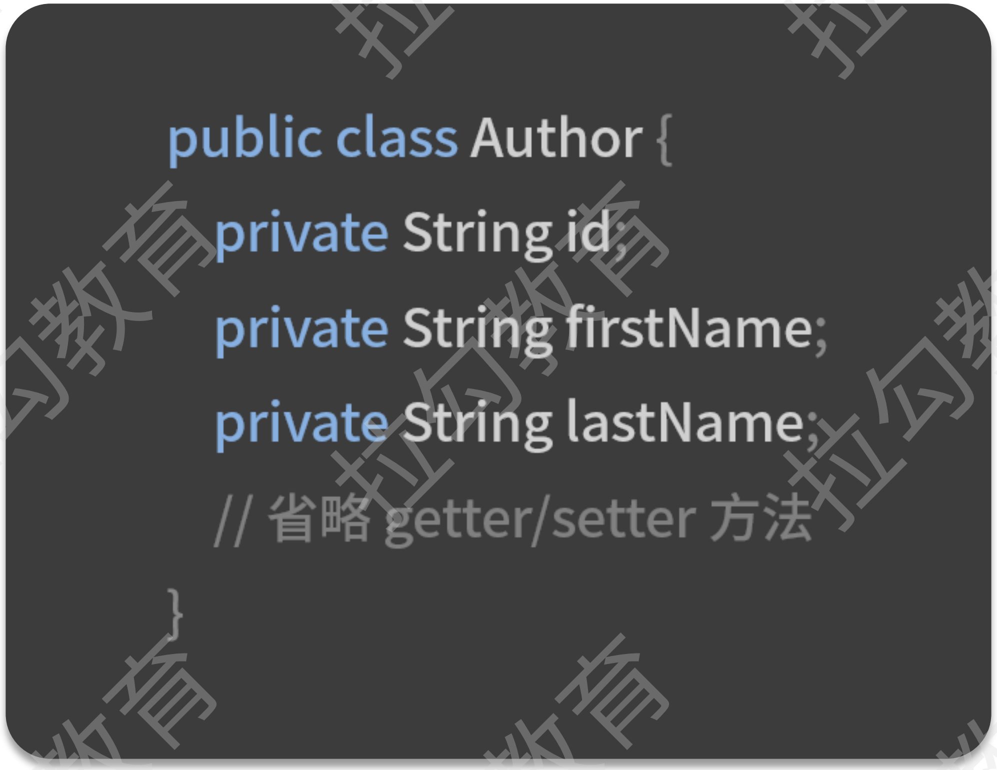
GraphQL Java Tools 入门

拉勾教育
— 互联网人实战大学 —


```
public class Book {
    private String id;
    private String name;
    private int pageCount;
    private String authorId;
    // 省略 getter/setter 方法
}
```

GraphQL Java Tools 入门

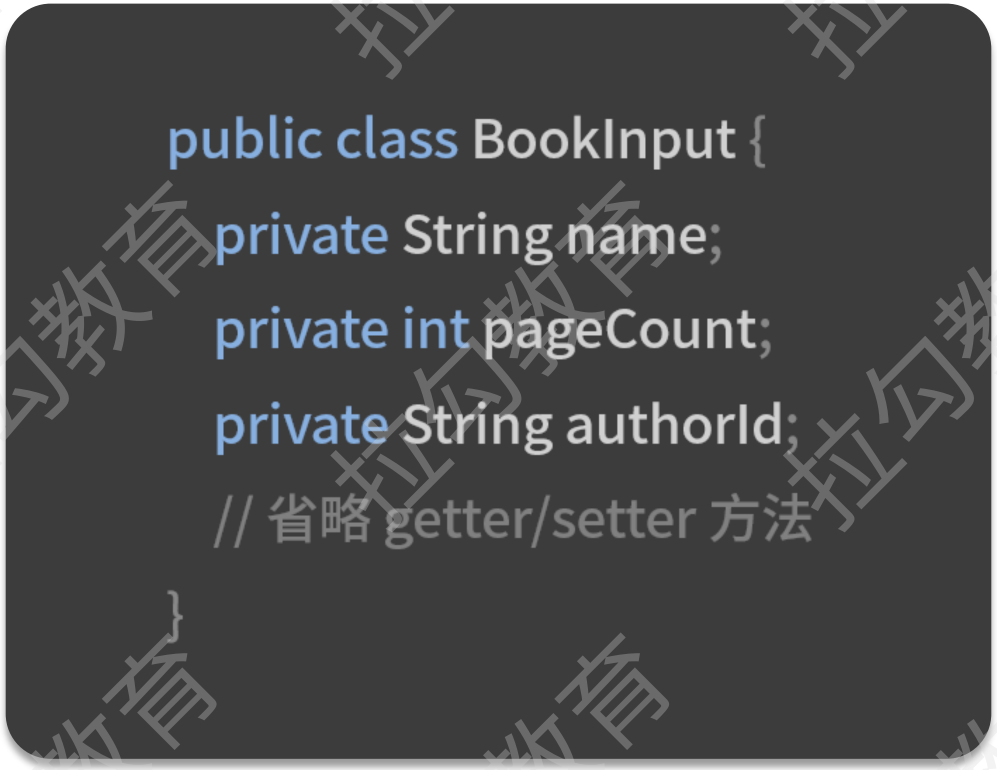
拉勾教育
— 互 联 网 人 实 战 大 学 —



```
public class Author {
  private String id;
  private String firstName;
  private String lastName;
  // 省略 getter/setter 方法
}
```

GraphQL Java Tools 入门

拉勾教育
— 互联网人实战大学 —



```
public class BookInput {
  private String name;
  private int pageCount;
  private String authorId;
  // 省略 getter/setter 方法
}
```

GraphQL Java Tools 入门

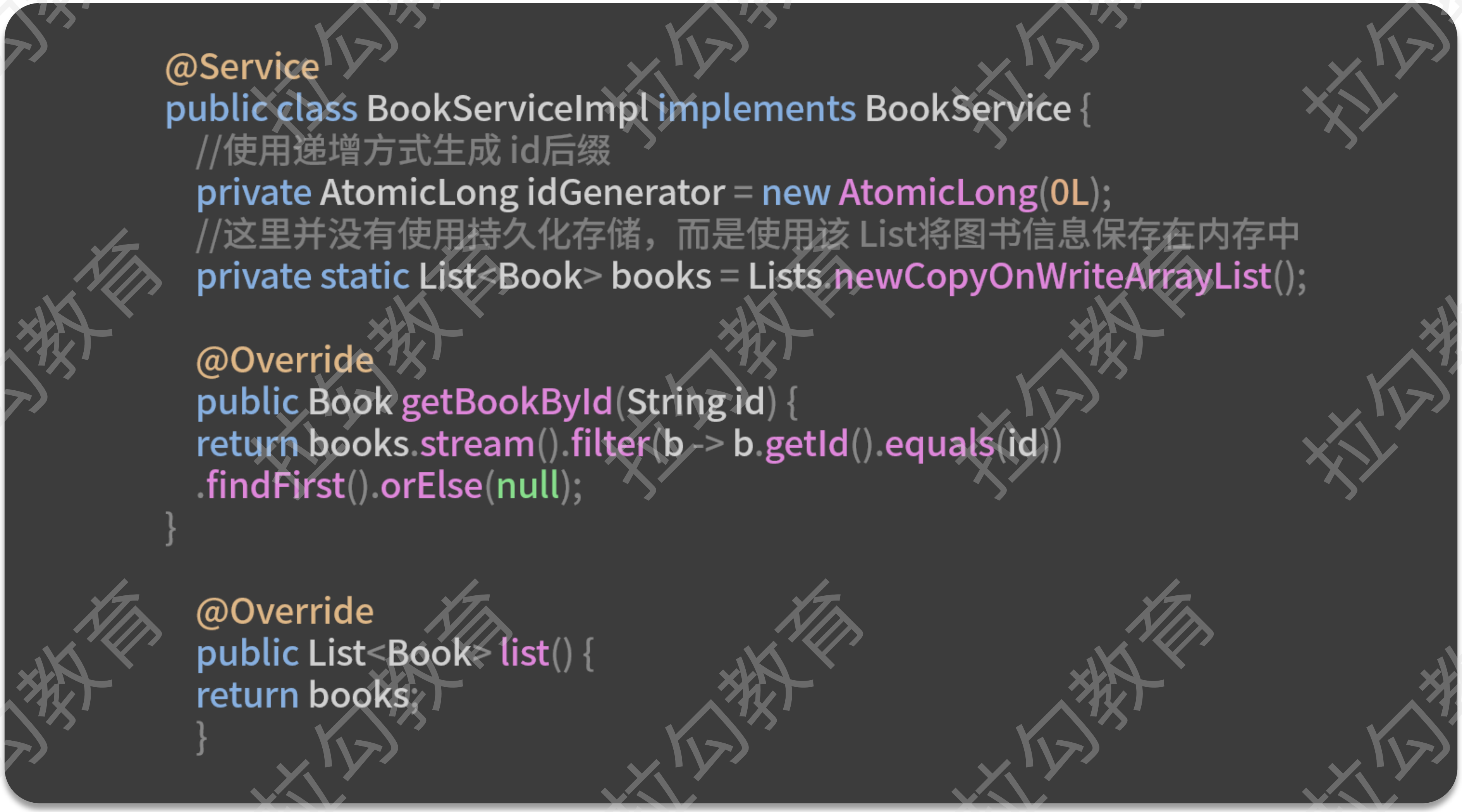
```
@Component
class BookResolver implements GraphQLResolver<Book> {
@Autowired
private AuthorService authorService;
  public Author author(Book book) {
    return authorService.getAuthorById(book.getAuthorId());
  }
}
```

GraphQL Java Tools 入门

拉勾教育
— 互 联 网 人 实 战 大 学 —

```
public interface BookService extends GraphQLQueryResolver,
GraphQLMutationResolver {
   Book getBookById(String id);
   List<Book> list();
   Book createBook(BookInput input);
}

public interface AuthorService extends GraphQLQueryResolver,
GraphQLMutationResolver {
   String createAuthor(String firstName, String lastName);
   Author getAuthorById(String id);
}
```

GraphQL Java Tools 入门



```
@Service
public class BookServiceImpl implements BookService {
  //使用递增方式生成 id后缀
  private AtomicLong idGenerator = new AtomicLong(0L);
  //这里并没有使用持久化存储，而是使用该 List将图书信息保存在内存中
  private static List<Book> books = Lists.newCopyOnWriteArrayList();

  @Override
  public Book getBookById(String id) {
  return books.stream().filter(b -> b.getId().equals(id))
  .findFirst().orElse(null);
}

  @Override
  public List<Book> list() {
  return books;
  }
```

GraphQL Java Tools 入门

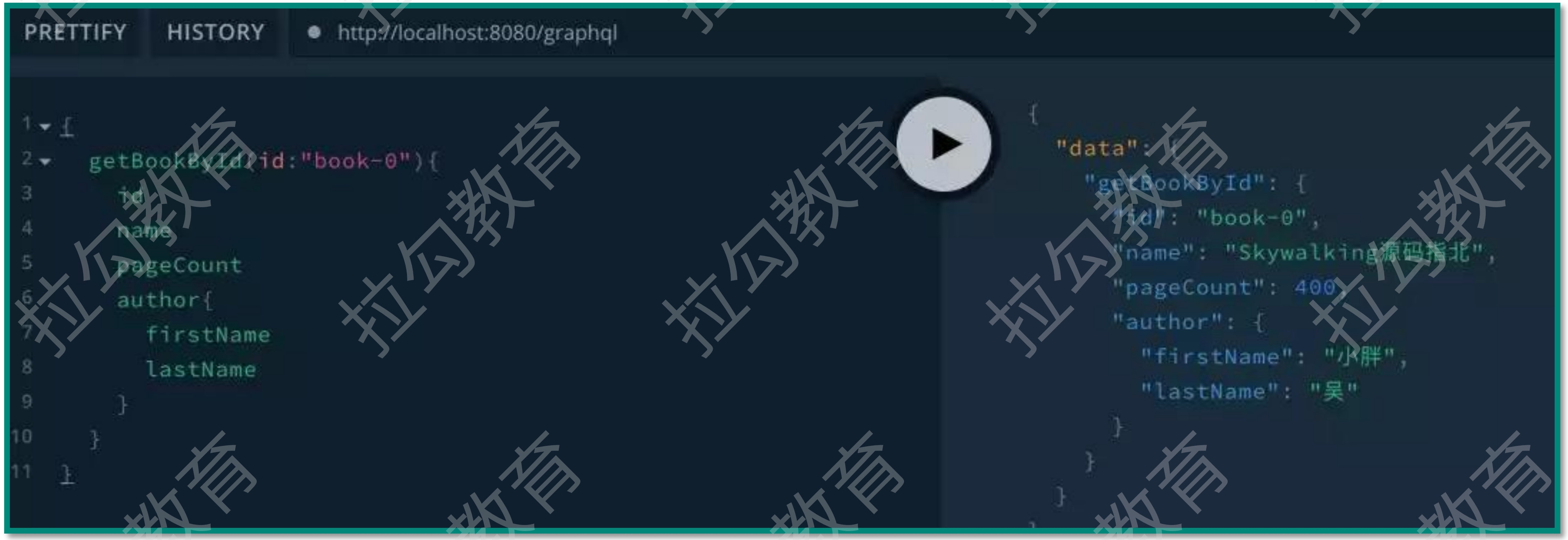
```java
    @Override
    public Book createBook(BookInput input) {
        String id = "book-" + idGenerator.getAndIncrement();
        Book book = new Book();
        book.setId(id);
        book.setName(input.getName());
        book.setPageCount(input.getPageCount());
        book.setAuthorId(input.getAuthorId());
        books.add(book);
        return book;
    }
}

@Component
public class AuthorServiceImpl implements AuthorService {
    private AtomicLong idGenerator = new AtomicLong(0L);
    private static List<Author> authors = Lists.newCopyOnWriteArrayList();
```

GraphQL Java Tools 入门

```java
@Override
public String createAuthor(String firstName, String lastName) {
    String id = "author-" + idGenerator.getAndIncrement();
    Author author = new Author();
    author.setId(id);
    author.setFirstName(firstName);
    author.setLastName(lastName);
    authors.add(author);
    return id;
}

@Override
public Author getAuthorById(String id) {
    return authors.stream().filter(a -> a.getId().equals(id))
    .findFirst().orElse(null);
    }
}
```

GraphQL Java Tools 入门

拉勾教育
— 互 联 网 人 实 战 大 学 —

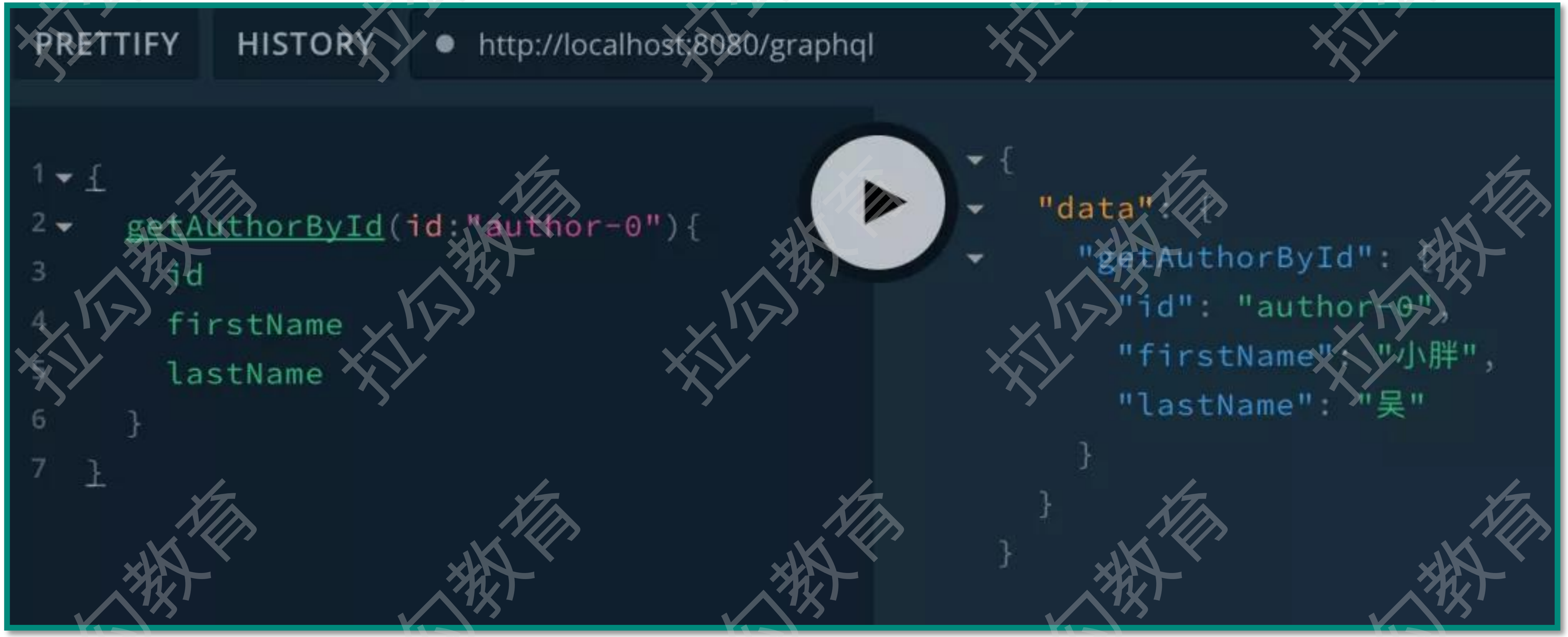


```
{
  getBookById(id:"book-0"){
    id
    name
    pageCount
    author{
      firstName
      lastName
    }
  }
}
```

```
{
  "data": {
    "getBookById": {
      "id": "book-0",
      "name": "Skywalking源码指北",
      "pageCount": 400,
      "author": {
        "firstName": "小胖",
        "lastName": "吴"
      }
    }
  }
}
```

GraphQL Java Tools 入门

拉勾教育
— 互 联 网 人 实 战 大 学 —



```
{
  getAuthorById(id:"author-0"){
    id
    firstName
    lastName
  }
}
```

```
{
  "data": {
    "getAuthorById": {
      "id": "author-0",
      "firstName": "小胖",
      "lastName": "吴"
    }
  }
}
```

Next：第28讲《深入 query-graphql 插件，SW Rocketbot 背后的英雄》