互联网应用开发技术

Web Application Development

第8课 WEB后端-SPRING DATA JDBC

Episode Eight

Access to RDBMS Using JDBC with Spring

陈昊鹏

chen-hp@sjtu.edu.cn

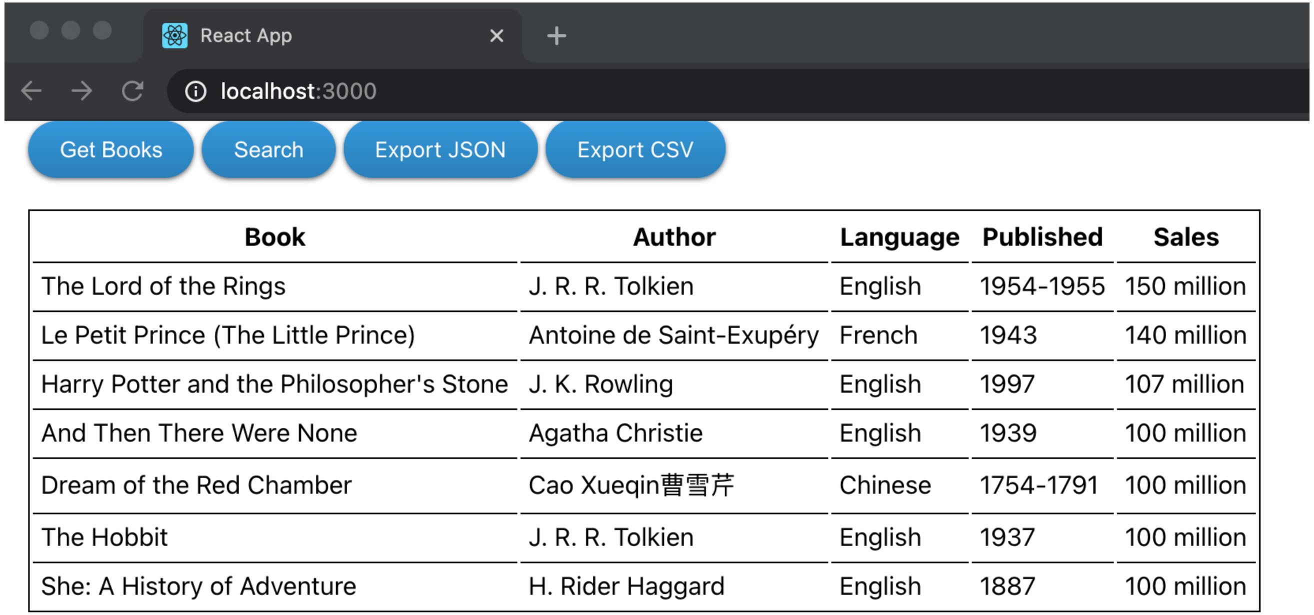
Web Application Development

Spring Data JDBC

REliable, INtelligent & Scalable Systems

- We build an application that uses Spring's JdbcTemplate to access data stored in a relational database.

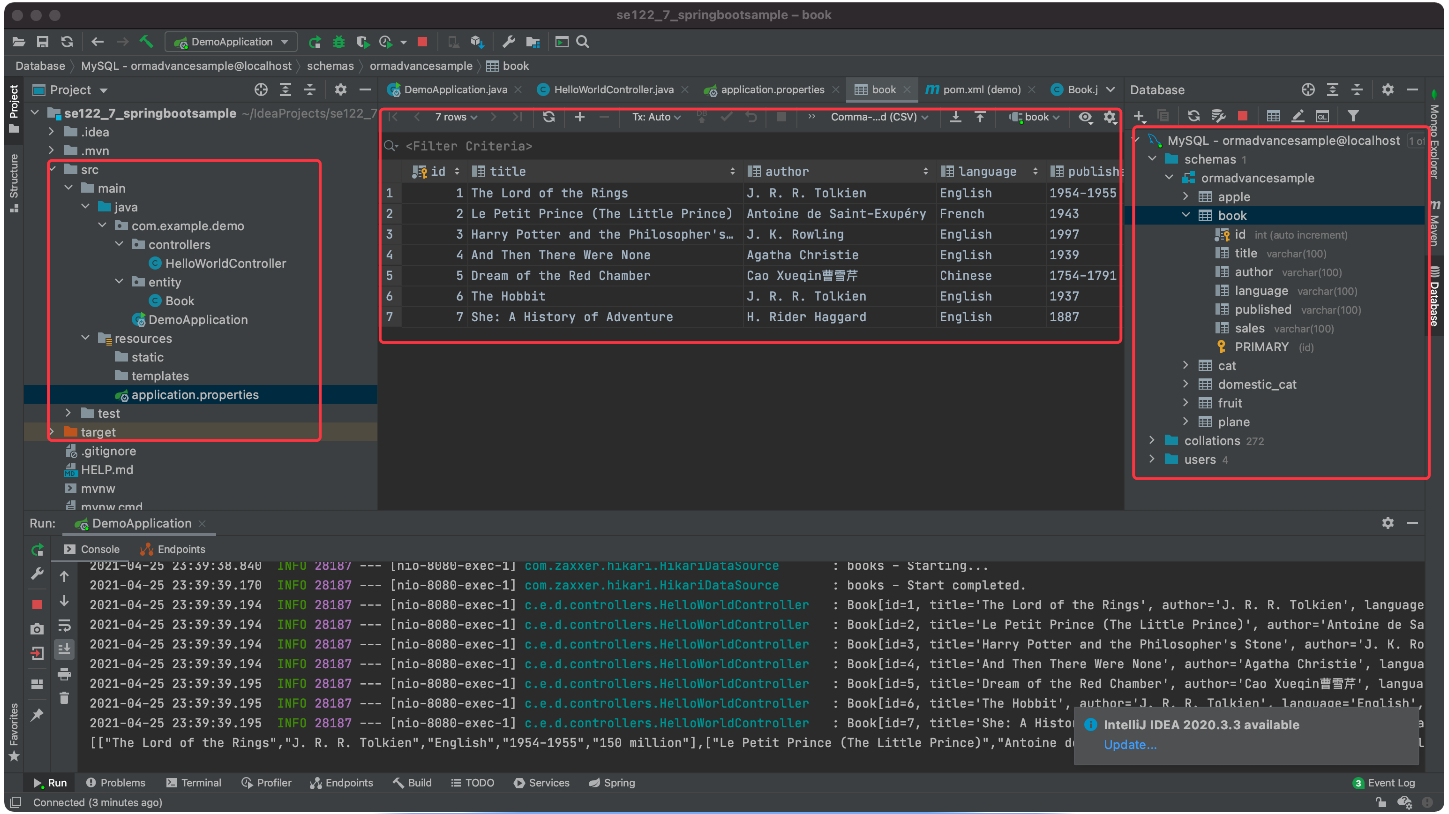
是对JDBC的一种封装



Book	Author	Language	Published	Sales
The Lord of the Rings	J. R. R. Tolkien	English	1954-1955	150 million
Le Petit Prince (The Little Prince)	Antoine de Saint-Exupéry	French	1943	140 million
Harry Potter and the Philosopher's Stone	J. K. Rowling	English	1997	107 million
And Then There Were None	Agatha Christie	English	1939	100 million
Dream of the Red Chamber	Cao Xueqin曹雪芹	Chinese	1754-1791	100 million
The Hobbit	J. R. R. Tolkien	English	1937	100 million
She: A History of Adventure	H. Rider Haggard	English	1887	100 million

Spring Data JDBC

REliable, INtelligent & Scalable Systems



se122_7_springbootsample – book

Database 〉 MySQL - ormadvancesample@localhost 〉 schemas 〉 ormadvancesample 〉 book

```
se122_7_springbootsample ~/IdeaProjects/se122_7
  .idea
  .mvn
  src
    main
      java
        com.example.demo
          controllers
            HelloWorldController
          entity
            Book
          DemoApplication
      resources
        static
        templates
        application.properties
    test
  target
  .gitignore
  HELP.md
  mvnw
  mvnw.cmd
```

	id	title	author	language	publish
1	1	The Lord of the Rings	J. R. R. Tolkien	English	1954-1955
2	2	Le Petit Prince (The Little Prince)	Antoine de Saint-Exupéry	French	1943
3	3	Harry Potter and the Philosopher's…	J. K. Rowling	English	1997
4	4	And Then There Were None	Agatha Christie	English	1939
5	5	Dream of the Red Chamber	Cao Xueqin曹雪芹	Chinese	1754-1791
6	6	The Hobbit	J. R. R. Tolkien	English	1937
7	7	She: A History of Adventure	H. Rider Haggard	English	1887

```
MySQL - ormadvancesample@localhost
  schemas 1
    ormadvancesample
      apple
      book
        id int (auto increment)
        title varchar(100)
        author varchar(100)
        language varchar(100)
        published varchar(100)
        sales varchar(100)
        PRIMARY (id)
      cat
      domestic_cat
      fruit
      plane
  collations 272
  users 4
```

```
2021-04-25 23:39:38.840  INFO 28187 --- [nio-8080-exec-1] com.zaxxer.hikari.HikariDataSource       : books - Starting...
2021-04-25 23:39:39.170  INFO 28187 --- [nio-8080-exec-1] com.zaxxer.hikari.HikariDataSource       : books - Start completed.
2021-04-25 23:39:39.194  INFO 28187 --- [nio-8080-exec-1] c.e.d.controllers.HelloWorldController   : Book[id=1, title='The Lord of the Rings', author='J. R. R. Tolkien', language
2021-04-25 23:39:39.194  INFO 28187 --- [nio-8080-exec-1] c.e.d.controllers.HelloWorldController   : Book[id=2, title='Le Petit Prince (The Little Prince)', author='Antoine de Sa
2021-04-25 23:39:39.194  INFO 28187 --- [nio-8080-exec-1] c.e.d.controllers.HelloWorldController   : Book[id=3, title='Harry Potter and the Philosopher's Stone', author='J. K. Ro
2021-04-25 23:39:39.194  INFO 28187 --- [nio-8080-exec-1] c.e.d.controllers.HelloWorldController   : Book[id=4, title='And Then There Were None', author='Agatha Christie', langua
2021-04-25 23:39:39.195  INFO 28187 --- [nio-8080-exec-1] c.e.d.controllers.HelloWorldController   : Book[id=5, title='Dream of the Red Chamber', author='Cao Xueqin曹雪芹', langua
2021-04-25 23:39:39.195  INFO 28187 --- [nio-8080-exec-1] c.e.d.controllers.HelloWorldController   : Book[id=6, title='The Hobbit', author='J. R. R. Tolkien', language='English',
2021-04-25 23:39:39.195  INFO 28187 --- [nio-8080-exec-1] c.e.d.controllers.HelloWorldController   : Book[id=7, title='She: A Histo
[["The Lord of the Rings","J. R. R. Tolkien","English","1954-1955","150 million"],["Le Petit Prince (The Little Prince)","Antoine d
```

IntelliJ IDEA 2020.3.3 available
Update...

Connected (3 minutes ago)

pom.xml

```xml
  <dependencies>
   <dependency>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-jdbc</artifactId>
   </dependency>
   <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <scope>runtime</scope>
   </dependency>
   <!-- https://mvnrepository.com/artifact/com.alibaba/fastjson -->
   <dependency>
      <groupId>com.alibaba</groupId>
      <artifactId>fastjson</artifactId>
      <version>1.2.76</version>
   </dependency>
</dependencies>
```

application.properties

```
spring.datasource.url=jdbc:mysql://localhost:3306/ormadvancesample
            ?useUnicode=true&characterEncoding=UTF-8&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=reins2011!
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.tomcat.max-active=20
```

Book.java

```java
public class Book {
   private Long id;
   private String title;
   private String author;
   private String language;
   private String published;
   private String sales;

   public Book(Long id, String title, String author, String language, String published, String sales) {
      this.id = id;
      this.title = title;
      this.author = author;
      this.language = language;
      this.published = published;
      this.sales = sales;
   }

   @Override
   public String toString() {
      return String.format(
            "Book[id=%d, title='%s', author='%s', language='%s', published='%s', sales='%s']",
            id, title, author, language, published, sales);
   }

}
```

给定输出格式，主要是规定返回格式，方便前端的读取。

HelloWorldController.java

```
@RestController
public class HelloWorldController {

    @Autowired （注入）
    JdbcTemplate jdbcTemplate;

    @CrossOrigin （跨域）
    @RequestMapping(“/”)
    public String home() {
        final Logger log = LoggerFactory.getLogger(HelloWorldController.class);
        List<Book> result = new ArrayList<Book>();

        log.info(“Querying Books”);
        result = jdbcTemplate.query(
                “SELECT * FROM book”,
                (rs, rowNum) -> new Book(rs.getLong(“id”),
                        rs.getString(“title”),
                        rs.getString(“author”),
                        rs.getString(“language”),
                        rs.getString(“published”),
                        rs.getString(“sales”))
        );
        Iterator<Book> it = result.iterator();
```

HelloWorldController.java

REliable, INtelligent & Scalable Systems

```
    ArrayList<JSONArray> booksJson = new ArrayList<JSONArray>();
    while (it.hasNext()) {
      Book book = (Book) it.next();
      ArrayList<String> arrayList = new ArrayList<String>();
      arrayList.add(book.getTitle());
      arrayList.add(book.getAuthor());
      arrayList.add(book.getLanguage());
      arrayList.add(book.getPublished());
      arrayList.add(book.getSales());
      booksJson.add((JSONArray) JSONArray.toJSON(arrayList));
    }
    String  booksString = JSON.toJSONString(booksJson, SerializerFeature.BrowserCompatible);

    System.out.println(booksString);

    return booksString;
  }
}
```

这段是在拼装用于返回的JSON类型字符串

DemoApplication.java

```
package com.example.demo;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@SpringBootApplication
public class DemoApplication {
   public static void main(String[] args) {
      SpringApplication.run(DemoApplication.class, args);
   }
}
```

book table

```
create table book
(
   id        int auto_increment  primary key,
   title      varchar(100) null,
   author    varchar(100) null,
   language  varchar(100) null,
   published varchar(100) null,
   sales     varchar(100) null
);
```

App.js

```
getBooks = () => {
  fetch("http://localhost:8080/")
    .then(response => response.json())
    .then(data => {
      alert("data:" + data);
      this.setState({
        data: data,
      });
    }).catch(function (ex) {
    console.log('parsing failed', ex)
  })
}
```

App.js

```
renderToolbar = () => {
  return (
    <div className="toolbar">
      <button onClick={this.getBooks}>Get Books</button>
      <button onClick={this.toggleSearch}>Search</button>
      <a onClick={this.download.bind(this, 'json')}
        href="data.json">Export JSON</a>
      <a onClick={this.download.bind(this, 'csv')}
        href="data.csv">Export CSV</a>
    </div>
  );
};
```

JdbcTemplate

- **This is the central class in the JDBC core package.**
  - It simplifies the use of JDBC and helps to avoid common errors.
  - It executes core JDBC workflow, leaving application code to provide SQL and extract results.
  - This class executes SQL queries or updates, initiating iteration over ResultSets and catching JDBC exceptions and translating them to the generic, more informative exception hierarchy defined in the org.springframework.dao package.Code using this class need only implement callback interfaces, giving them a clearly defined contract.
  - Can be used within a service implementation via direct instantiation with a DataSource reference, or get prepared in an application context and given to services as bean reference.
  - Note: The DataSource should always be configured as a bean in the application context, in the first case given to the service directly, in the second case to the prepared template.

JdbcTemplate

```
jdbcTemplate.execute("DROP TABLE customers IF EXISTS");
jdbcTemplate.execute("CREATE TABLE customers("
                        + "id SERIAL, first_name VARCHAR(255), last_name VARCHAR(255))");
// Split up the array of whole names into an array of first/last names
List<Object[]> splitUpNames = Arrays.asList("John Woo", "Jeff Dean",
                "Josh Bloch", "Josh Long").stream()
                   .map(name -> name.split(" "))
                   .collect(Collectors.toList());

// Use a Java 8 stream to print out each tuple of the list
splitUpNames.forEach(name ->
      log.info(String.format("Inserting customer record for %s %s", name[0], name[1])));

// Uses JdbcTemplate's batchUpdate operation to bulk load data
jdbcTemplate.batchUpdate("INSERT INTO customers(first_name, last_name) VALUES (?,?)",
                                             splitUpNames);
log.info("Querying for customer records where first_name = 'Josh':");
jdbcTemplate.query( "SELECT id, first_name, last_name FROM customers
                   WHERE first_name = ?", new Object[] { "Josh" },
                   (rs, rowNum) -> new Customer(rs.getLong("id"), rs.getString("first_name"),
                                                        rs.getString("last_name"))
                   ).forEach(customer -> log.info(customer.toString()));
```

JdbcTemplate

- Methods of `JdbcTemplate`:
  - `execute`
    - Executes any SQL statements. In general, it is used for DDL
  - `update & batchUpdate`
    - `update` executes insert, update and delete statements
    - `batchUpdate` executes batch statements
  - `query & queryForXXX`
    - Execute queries
  - `call`
    - Executes Callable Statements

References

- Accessing Relational Data using JDBC with Spring
  - https://spring.io/guides/gs/relational-data-access/
- Accessing data with MySQL
  - https://spring.io/guides/gs/accessing-data-mysql/
- JdbcTemplate Javadoc
  - https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/jdbc/core/JdbcTemplate.html

- *Web开发技术*
- *Web Application Development*

Thank You!