# 示例项目： book组 图书的查找功能，先分支

## 1.目录

```
library_system/
├── frontend/
│   └── /src/modules/
│       └── book/    (◍ 图书小组前端)
│           ├── pages/           ← BookSearchPage.vue（图书搜索页面）
│           ├── components/      ← BookCard.vue / BookSearchBar.vue（展示与输入）
│           ├── api.js           ← 封装接口请求：getBooks(query)
│           └── index.js
│
├── backend/
│   ├── Controllers/
│   │   └── Book/
│   │       └── BookController.cs   (◍ 图书小组后端控制器，接收前端请求)
│   ├── Services/
│   │   └── Book/
│   │       └── BookService.cs          (◍ 查询逻辑封装：模糊搜索、分页、分类)
│   ├── DTOs/
│   │   └── Book/
│   │       └── BookDetailDto.cs    (◍ 查询结果结构定义)
│   ├── Repositories/
│   │   └── Book/
│   │       └── BookRepository.cs       (◍ 编写 SQL 查询数据库)
│   ├── Models/
│   │   └── Book.cs                 (◍ 映射数据库)
│   ├── Program.cs
│   ├── Startup.cs
│   └── appsettings.json
│
├── database/
│   ├── views/
│   │   └── book/
│   │       └── book_detail_view.sql ← ◍ 公共视图（联合Book + BookInfo + 分类 + 位置）
│   └── init.sql  (包含上述视图)
```

## 2.sql层

### 1.book_detail_view.sql

```
CREATE OR REPLACE VIEW book_detail_view AS
SELECT
    b.BookID,
    b.Status,
    b.ShelfID,
    b.BuildingID,
    b.ISBN,
    bi.Title,
    bi.Author,
    bi.Stock
FROM Book b
JOIN BookInfo bi ON b.ISBN = bi.ISBN;
```

### 2.在init.sql中包含上述视图

```
@views/book/book_detail_view.sql
```

### 3.运行方法

- 开发时：book_detail_view.sql拖拽到sql developer中，点击运行脚本

  退出sql developer时，若选择提交更改，本次在sql developer中执行的文件都会被服务器永久保存

  若选择回退更改，本次在sql developer中执行的文件不会被保存，Oracle回退到你本次所有操作之前
- 移植到其他项目时：在服务器一键执行执行init.sql脚本，注册所有sql语句

### 4.我已经插入部分数据，数据先不要自行insert，过几天统一导入

## 3.后端层

### 总览

```
前端请求
↓
[Controller] BookController.cs         ← 接收请求，做参数处理
↓
[Service] BookService.cs               ← 实现业务逻辑/组合数据
↓
[Repository] BookRepository.cs         ← 直接操作数据库
↓
数据库 (Oracle)                        ← 查询 book_detail_view 视图
↓
返回 DTO 列表 BookDetailDto.cs         ← 每一项是一本书的数据结构
↑
JSON 返回到前端
```

### 1.BookController.cs

```
using Microsoft.AspNetCore.Mvc;
[ApiController]
[Route("api/[controller]")]
public class BookController : ControllerBase
{
    private readonly BookService _service;

    public BookController(BookService service)
    {
        _service = service;
    }

    [HttpGet("search")]
    public async Task<IEnumerable<BookDetailDto>> Search(string keyword)
    {
        return await _service.SearchBooksAsync(keyword ?? "");
    }
}
```

2.BookService.cs

```
public class BookService
{
    private readonly BookRepository _repository;

    public BookService(BookRepository repository)
    {
        _repository = repository;
    }

    public Task<IEnumerable<BookDetailDto>> SearchBooksAsync(string keyword)
    {
        return _repository.SearchBooksAsync(keyword);
    }
}
```

3.BookRepository.cs

```
public class BookRepository
{
    private readonly string _connectionString;

    public BookRepository(string connectionString)
    {
        _connectionString = connectionString;
    }

    public async Task<IEnumerable<BookDetailDto>> SearchBooksAsync(string keyword)
    {
        var sql = @"
            SELECT BookID, ISBN, Title, Author, Status
            FROM book_detail_view
            WHERE LOWER(Title) LIKE :keyword OR LOWER(Author) LIKE :keyword";

        using var connection = new
Oracle.ManagedDataAccess.Client.OracleConnection(_connectionString);
        await connection.OpenAsync();

        return await Dapper.SqlMapper.QueryAsync<BookDetailDto>(
            connection, sql, new { keyword = $"%{keyword.ToLower()}%" });
    }
}
```

4.DTO BookDetailDto.cs

```
public class BookDetailDto
{
    public string BookID { get; set; }
    public string ISBN { get; set; }
    public string Title { get; set; }
    public string Author { get; set; }
    public string Status { get; set; }
}
```

5.Program.cs

```
var builder = WebApplication.CreateBuilder(args);

// 输出当前环境 (Development / Production)
Console.WriteLine($"当前运行环境: {builder.Environment.EnvironmentName}");

// 添加控制器服务
builder.Services.AddControllers()
    .AddJsonOptions(options =>
    {
        options.JsonSerializerOptions.PropertyNamingPolicy = null;
    });

// 添加 CORS 支持 (便于前端访问)
builder.Services.AddCors(options =>
{
    options.AddDefaultPolicy(policy =>
    {
        policy
            .AllowAnyOrigin()      // 生产环境可替换为具体域名
            .AllowAnyHeader()
            .AllowAnyMethod();
    });
});

// 日志输出到控制台（调试用）
builder.Logging.ClearProviders();
builder.Logging.AddConsole();

// 读取连接字符串（根据环境自动读取 appsettings.Development.json 或
appsettings.Production.json)
var connectionString = builder.Configuration.GetConnectionString("OracleDB")
                    ?? throw new InvalidOperationException("缺少 OracleDB 连接字符
串配置");
// 注册服务依赖（Repository 使用 Singleton，Service 使用 Transient）
builder.Services.AddSingleton(new BookRepository(connectionString));
builder.Services.AddTransient<BookService>();

var app = builder.Build();

// 使用 CORS（顺序要在 MapControllers 之前）
app.UseCors();

// 启用控制器路由
app.UseRouting();
app.MapControllers();

// 启动应用
app.Run();
```

## 6.运行方法

- 开发时：运行在本地/backend根目录命令行启动方法

```
set ASPNETCORE_ENVIRONMENT=Development
dotnet run
```

- 答辩前运行在服务器

```
export ASPNETCORE_ENVIRONMENT=Production
dotnet run
```

## 7.测试方法

```
http://localhost:5000/api/book/search?keyword=操作系统原理
```

## 4.前端层( 无需重复 npm install axios)

### 1.api.js

```
import http from '@/services/http.js'

export function getBooks(keyword) {
return http.get('/book/search', {
    params: { keyword }
})
}
```

### 2./component/BookSearchComponent.vue

```
<!-- BookSearch.vue -->
<script setup>
import { ref } from 'vue'
import { getBooks } from '../api.js'  // 根据你的模块路径，调整为相对路径

const keyword = ref('')
const books = ref([])

const search = async () => {
const res = await getBooks(keyword.value)
books.value = res.data
}
</script>

<template>
<div class="book-search">
    <input v-model="keyword" placeholder="请输入书名或作者" />
    <button @click="search">搜索</button>

    <div v-for="book in books" :key="book.BookID">
    <p>{{ book.Title }} - {{ book.Author }}</p>
    </div>
</div>
</template>

<style scoped>
.book-search {
padding: 1rem;
}
```

```
input {
margin-right: 0.5rem;
}
</style>
```

### 3.调用组件

```
import BookSearch from '@/modules/book/components/BookSearch.vue'


<BookSearch />
```

### 4.运行方法

- 开发时

```
npm run dev
```

- 答辩前：运行在服务器上

```
npm run build
```

## 5.调用关系

```
┌───────────────────────────┐
│         Frontend 模块      │
│    ┌────────────────────┐  │
│    │ modules/reader/    │  │← 读者组前端模块（页面、组件）
│    ├────────────────────┤  │
│    │ modules/book/      │  │← 图书组前端模块
│    ├────────────────────┤  │
│    │ modules/admin/     │  │← 管理员组前端模块
└────┴────────────────────┴──┘
              │
              ▼ (通过 API 请求)
┌───────────────────────────┐
│          Backend 模块      │
│    ┌────────────────────┐  │
│    │ Controllers/reader/│  │← ReaderController.cs 接收前端请求
│    │ Services/reader/   │  │← ReaderService.cs 实现业务逻辑
│    │ DTOs/reader/       │  │← ReaderLoginDto.cs 数据结构定义
│    ├────────────────────┤  │
│    │ Controllers/book/  │  │← 图书组控制器
│   │ Services/book/      │   │← 图书组业务逻辑
│   │ DTOs/book/          │   │← 图书组数据结构
│   ├─────────────────────┤   │
│   │ Controllers/admin/  │   │← 管理员组控制器
│   │ Services/admin/     │   │← 管理员业务逻辑
│   │ DTOs/admin/         │   │← 管理员数据结构
└───┴─────────────────────┴───┘
             │
             ▼ (通过 DbContext、Dapper 等)
┌─────────────────────────────┐
│         Database 层          │
│ (Oracle, 单实例 + 单用户) │
│                             │
│  所有组共用一个用户 schema │
│   final_owner@ORCLPDB1      │
│                             │
│  ┌─────────────────────┐  │
│  │ schema.sql          │   │← 所有表结构（共享）
│  ├─────────────────────┤  │
│  │ functions/reader/   │   │← 读者组函数
│  │ views/reader/       │   │← 读者组视图
│  │ triggers/reader/    │   │← 读者组触发器
│  ├─────────────────────┤  │
│  │ functions/book/     │   │← 图书组函数
│  │ views/book/         │   │← 图书组视图
│  │ triggers/book/      │   │← 图书组触发器
│  ├─────────────────────┤  │
│  │ functions/admin/    │   │← 管理员组函数
│  │ views/admin/        │   │← 管理员组视图
│  │ triggers/admin/     │   │← 管理员组触发器
└───┴─────────────────────┴───┘
```