1.选题与要求

个人图书馆

> 需求：
> （1）实现图书的增删改查；图书的信息包括：id, 书名、出版社、分类号、ISBN号、出版日期、价格、购买日期；可以按照书名、出版社等检索分页展示图书列表
> （2）实现简单的借出、归还管理。（图书表可以增加是否在架字段）
> 同时要有借还日志(增加一个借还历史表，每次借和还占用一条记录，字段有：图书的id，借出时间、归还时间、借阅人)
> （3）借还历史日志的分页查询展示

考核内容

> 每个学生选择一个题目，不分组
> 1、学会编写软件需求分析文档、软件详细设计文档
> 2、掌握后端软件开发方法，熟悉 java 和 IDEA 开发环境
> 3、掌握前端软件开发方法，熟悉 vscode 环境，熟悉 typescript 语言和 react 框架
> 4、熟练使用git版本管理工具
>
> 提交内容：
> 软件需求分析文档、软件详细设计文档、源代码、答辩PPT

2.后端学习

基本步骤

- MySQL设计表
- generatorConfig.xml添加table信息
- run maven
- service中添加interface
- impl.java中添加java类
- 添加对应controller 类

分页查询

- 浏览器按F12进入开发者模式
- 前端向后端发送的请求格式

```
{"current":1,"pageSize":20,"departmentName":"通信"}
```

表示请求当前页面，页面大小和部门名称

- 后端向前端发送的JSON数据格式

```
{
  code: 200,
  data: {
    current: 1,
```

```
      pageSize: 20,
      total: 2,
      list: [
        {
          id: 1,
          departmentName: "通信所3444",
          contact: "张三",
          contactPhone: "ddd",
          description: "这是一条备注",
          createdAt: "2019-12-03 17:31:28",
          updatedAt: "2023-03-03 10:36:23",
          createdBy: 1,
          createdByDesc: "管理员",
        },
        {
          id: 2,
          departmentName: "通信所",
          contact: "张三",
          contactPhone: "1532384234234",
          description: "这是一条备注",
          createdAt: "2019-12-03 17:48:06",
          updatedAt: "2023-03-03 10:36:29",
          createdBy: 1,
          createdByDesc: "管理员",
        },
      ],
    },
    message: null,
    success: true,
  };
```

- 后端中controller中的

```java
  Page<DepartmentVO>
```

分页数据，返回模板类/泛化类的数据，Page类表示返回的json数据格式中的内容

- 前端向后端发送的查询内容由 DepartmentQueryDTO 类接收

```
  listDepartment(@RequestBody DepartmentQueryDTO queryDTO)
```

- 模糊查询

```
  queryDTO.setDepartmentName(FormatUtils.makeFuzzySearchTerm(queryDTO.getDepartm
entName()));
```

注意MyBatis 自动生成的Mapper类

Mapper.xml里，要添加count的SQL定义

- MySQL查询记录语句

```
select count(*) from department;
select count(*) from department where department_name like '%通信%';
```

- Mybatis

```
<where>
    <if test="departmentName != null">
        department_name like #{departmentName}
    </if>
</where>
```

不是NULL，则有输入过滤条件。

#{departmentName}会将departmentName内容替换到MySQL语句中，若有多个逻辑条件，则用and逻辑与

- 利用myBatis到数据库中查询数据，以分页的方式

```
/**
* 根据查询条件获取部门列表
*
* @param queryDTO 查询条件
* @param offset   开始位置
* @param limit    记录数量
* @return 部门列表
*/
List<Department> list(
    @Param("queryDTO") DepartmentQueryDTO queryDTO,
    @Param("offset") Integer offset,
    @Param("limit") Integer limit
);
```

- 将Department对象转VO对象

```
    /**
     * 将实体对象转换为VO对象
     *
     * @param department 实体对象
     * @param nameMap
     * @return VO对象
     */
    public static DepartmentVO convertToVO(Department department, Map<Integer,
String> nameMap) {
        DepartmentVO departmentVO = new DepartmentVO();
        //把第一个对象的字段，复制到第二个对象，前提要求是字段必须对应
        BeanUtils.copyProperties(department, departmentVO);

        departmentVO.setCreatedByDesc(nameMap.get(department.getCreatedBy()));
        return departmentVO;
    }
}
```

- IDEA F8单步执行Run over F7 Run into

避免重复创建部门

1. dao中创建Mapper类

```
Department getByName(@Param("name")String name);
```

2. xml中添加

```
    <!-- 获取名字 -->
        <select id="getByName" resultMap="BaseResultMap">
            select  *
            from department
            where department_name = *{name}
        </select>
```

3. Impl中添加Assert

```
Assert.isNull(departmentMapper.getByName(departmentDTO.getDepartmentName()),"部门已经存在");
```

3.前端学习

只看java基本语法，能够在项目中写代码即可

基本知识

1. 箭头函数

> c: int add(int a, int b);
> ts: function add(a: number, b: number);
> ts: 数据类型：number(数字类型，3, 5.2), string, boolean, ...
>
> 箭头函数是没有名字的
> Typescript => javascript => 在浏览器中运行
> chrome V8 js引擎，单进程单线程里run的
> 异步函数： 非阻塞

2. 定义一个回调函数

```
//定义一个Delete的回调函数
const handleDelete = async () => {
    if (!selectedRowKeys?.length) return;
    openConfirm(`您确定删除${selectedRowKeys.length}条记录吗`, async () => {
        // await deleteBook(selectedRowKeys);
        refAction.current?.reload();
    });
};
```

3. 声明变量与方法

```
//声明两个常量，变量名为selectedRowKeys，改变变量值的方法为selectRow
const [selectedRowKeys, selectRow] = useState<number[]>([]);
//声明页面级状态变量，将变量传递给对话框，<>内表示是什么类型的数据，()为空表示不对其初始化
const [book, setBook] = useState<API.BookVO>();
```

4. 以异步非阻塞方式运行

```
myfunc();
```

5. 回调式

```
myfunc().then((data)=>{
    console.log(data);
    myFunc2().then(d=>{
       //...
    });
})
```

6. 异步函数的执行方式

注意只有外部是async异步函数的时候才可以在内部使用await

```
async function myfff(){
    await myfunc();

   ///....
}
```

7. 分页展示数据结果集的antd pro

```
<ProTable<API.DepartmentVO>
```

8. 刷新操作

```
const refAction = useRef<ActionType>(null);
```

```
refAction.current?.reload();
```

9. ProTable存储数据

```
//params为分页信息，protable自动将params传递给listBook，
//数据由convertPageData转换为antd protable的数据格式
request={async (params = {}) => {
    return convertPageData(await listBook(params));
}}
```

10. 模态对话框显示数据

```
//设置book为当前record，并且使对话框可视化
render: (dom, record) => {
    return (
        <a
        onClick={() => {
        setBook(record);
        setVisible(true);
    }}
    >
        {dom}
</a>
);
},

    <InputDialog
//接收book数据
detailData={book}
onClose={(result) => {
    setVisible(false);
    result && refAction.current?.reload();
}}
visible={visible}
/>
```

注意

> 页面级状态发生变化的时候
> 重新渲染
>
> 页面被加载的时候，第一次执行你的渲染函数
> React巧妙的技术
>
> useEffect开始会执行一次
>
> 当数组中的变量变化时，useEffect会被调用
>
> useModel: 应用级状态，所有页面模块可共享。页面刷新或关闭浏览器后消失 umi useModal

```
useEffect(() => {
    waitTime().then(() => {
      if (props.detailData) {
        form?.current?.setFieldsValue(props.detailData);
      } else {
        form?.current?.resetFields();
      }
    });
  }, [    变量    ]);
```

npm安装

1. 更换淘宝源之后npm install还会报错
2. 即使使用清除命令也会报错

```
清理缓存命令：
npm cache clean --force
```

3. 手动删除缓存目录中的全部内容

```
C:\Users\bsgbs\AppData\Local\npm-cache
```

4. 再次运行npm install即可完成安装

OpenAPI运行出错

装好依赖之后发现运行OpenAPI会报错

解决方案：

1. 删除安装目录

```
E:\Files\Projects\CXCollege\software\frontend\web-frontend-master\web-frontend\node_modules
```

2. 重装npn install
3. 运行后端之后，再次尝试npm run openapi成功运行

---

1. 使用Navicat新建book表格

 1

2.generatorConfig.xml添加table信息

```
<table tableName="book" domainObjectName="Book" enableCountByExample="false"
               enableUpdateByExample="false"
               enableDeleteByExample="false" enableSelectByExample="false"
selectByExampleQueryId="false"></table>
```

3.run maven

4.service中添加interface

```java
package redlib.backend.service;

import redlib.backend.dto.TestDTO;

public interface BookService {
    void insert(BookDTO bookDTO);
}
```

4.impl中添加java类

```java
package redlib.backend.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import redlib.backend.dao.BookMapper;
import redlib.backend.dto.BookDTO;
import redlib.backend.model.Book;
import redlib.backend.service.BookService;

@Service
public class BookServiceImpl implements BookService {
    @Autowired
    private BookMapper bookMapper;

    @Override
    public void insert(BookDTO bookDTO) {
        Book book = new Book();
        book.setBookName(bookDTO.getBook_name());
        book.setIsbn(bookDTO.getISBN());
        book.setBookAuthor(bookDTO.getBook_author());
        book.setBookPublish(bookDTO.getBook_publish());
        book.setBookPrice(bookDTO.getBook_price());
        book.setBookType(bookDTO.getBook_type());
        book.setBookLocation(bookDTO.getBook_location());
        bookMapper.insert(book);
    }
}
```

5.添加对应controller 类

```java
package redlib.backend.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import redlib.backend.annotation.NeedNoPrivilege;
import redlib.backend.dto.BookDTO;
import redlib.backend.service.BookService;
@RestController
@RequestMapping("/api/book")
public class BookController {
```

```
    @Autowired
    private BookService bookService;
    @PostMapping("addRecord")
    @NeedNoPrivilege
    public void addRecord(@RequestBody BookDTO bookDTO){
        bookService.insert(bookDTO);
    }
}
```

6.添加DTO类

```
package redlib.backend.dto;

import lombok.Data;

import java.math.BigDecimal;

@Data
public class BookDTO {
    private String Book_name;
    private String ISBN;
    private String Book_author;
    private String Book_publish;
    private BigDecimal Book_price;
    private String Book_type;
    private String Book_location;
}
```

常用注解

```
@Autowired：引用其他组件
@Data的使用： lombok，自动生成getter和setter
@Slf4j的使用：日志。可以直接用log.debug, log.error等
@Service:  业务服务组件。这个组件可以访问数据库
@RestController:  控制器组件
@GetMapping和@PostMapping:  控制器方法注解，指定URL路径
@NeedNoToken:  不需要登录即可访问的方法
```

4.个人图书馆

图书信息表 *book*

字段名称	数据类型	索引	样例	说明
book_id	int(10)	Primary Key	1234	书籍ID
book_name	char(20)		马原	书籍名称
isbn	char(13)		1234567891234	ISBN
book_author	char(10)		作者	书籍作者
book_publish	char(20)		山大出版社	书籍出版社
book_price	double(20,2)		30.23	书籍价格
book_type	char(20)			书籍类型
book_boutime	datetime			购买时间
book_pubtime	datetime			出版时间

基本页面框架

后端

- **增加接口**

```
package redlib.backend.service;

import redlib.backend.dto.BookDTO;
import redlib.backend.dto.query.BookQueryDTO;
import redlib.backend.model.Page;
import redlib.backend.vo.BookVO;

public interface BookService {
    void insert(BookDTO bookDTO);
    Page<BookVO> listByPage(BookQueryDTO queryDTO);
}
```

- **Impl中实现Service**

```
package redlib.backend.service.impl;

import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.util.Assert;
import redlib.backend.dao.BookMapper;
import redlib.backend.dto.BookDTO;
```

```java
import redlib.backend.dto.query.BookQueryDTO;
import redlib.backend.model.Book;
import redlib.backend.model.Page;
import redlib.backend.service.AdminService;
import redlib.backend.service.BookService;
import redlib.backend.service.utils.BookUtils;
import redlib.backend.utils.FormatUtils;
import redlib.backend.utils.PageUtils;
import redlib.backend.vo.BookVO;

import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.stream.Collectors;

@Service
public class BookServiceImpl implements BookService {
    @Autowired
    private BookMapper bookMapper;

    @Autowired
    private AdminService adminService;

    @Override
    public void insert(BookDTO bookDTO) {
        Book book = new Book();
        book.setBookName(bookDTO.getBookName());
        book.setIsbn(bookDTO.getISBN());
        book.setBookAuthor(bookDTO.getBookAuthor());
        book.setBookPublish(bookDTO.getBookPublish());
        book.setBookPrice(bookDTO.getBookPrice());
        book.setBookType(bookDTO.getBookType());
        book.setBookBoutime(bookDTO.getBookBoutime());
        book.setBookPubtime(bookDTO.getBookPubtime());
        bookMapper.insert(book);
    }


    @Override
    public Page<BookVO> listByPage(BookQueryDTO queryDTO) {
        if (queryDTO == null) {
            queryDTO = new BookQueryDTO();
        }

        //模糊查询

 queryDTO.setBookName(FormatUtils.makeFuzzySearchTerm(queryDTO.getBookName()));
        //将查询条件发送给dao接口，先查询命中个数
        Integer size = bookMapper.count(queryDTO);
        PageUtils pageUtils = new PageUtils(queryDTO.getCurrent(),
queryDTO.getPageSize(), size);

        if (size == 0) {
            // 没有命中，则返回空数据。
```

```
            return pageUtils.getNullPage();
        }
        List<Book> list = bookMapper.list(queryDTO, pageUtils.getOffset(),
pageUtils.getLimit());
        Set<String> adminIds =
list.stream().map(Book::getBookName).collect(Collectors.toSet());

 adminIds.addAll(list.stream().map(Book::getBookAuthor).collect(Collectors.toSet
()));
        //Map<String, String> nameMap = adminService.getNameMap(adminIds);

        List<BookVO> voList = new ArrayList<>();
        for (Book book : list) {
            BookVO vo = BookUtils.convertToVO(book);
            voList.add(vo);
        }

        //按照前端支持的格式返回给前端
        return new Page<>(pageUtils.getCurrent(), pageUtils.getPageSize(),
pageUtils.getTotal(), voList);
    }
}
```

- **新增BookUtils.java封装各种功能**

```
package redlib.backend.service.utils;

import org.springframework.beans.BeanUtils;
import org.springframework.util.Assert;
import redlib.backend.dto.BookDTO;
import redlib.backend.model.Book;
import redlib.backend.utils.FormatUtils;
import redlib.backend.vo.BookVO;


public class BookUtils {
    /**
     * 规范并校验bookDTO
     *
     * @param bookDTO 实体对象
     */
    public static void validateBook(BookDTO bookDTO) {
        FormatUtils.trimFieldToNull(bookDTO);
        Assert.notNull(bookDTO, "书籍数据不能为空");
        Assert.hasText(bookDTO.getBookName(), "书名不能为空");
    }

    public static BookVO convertToVO(Book book) {
        BookVO bookVO = new BookVO();
        BeanUtils.copyProperties(book, bookVO);
        return bookVO;
    }
}
```

- **在BookMapper.java中添加**

```
    List<Book> list(@Param("queryDTO") BookQueryDTO queryDTO, @Param("offset")
Integer offset, @Param("limit") Integer limit);

    Integer count(BookQueryDTO queryDTO);
```

- **xml中添加**

```
  <!-- 获取名字个数(分页) -->
  <select id="count" resultType="integer">
    select count(*)
    from book
    <where>
      <if test="bookName != null">
        book_name like #{bookName}
      </if>
    </where>
  </select>

  <!-- 获取部门(分页) -->
  <select id="list" resultMap="BaseResultMap">
    select
    <include refid="Base_Column_List"/>
    from book
    <where>
      <if test="queryDTO.bookName != null">
        Book_name like #{queryDTO.bookName}
      </if>
    </where>
    limit #{offset}, #{limit}
  </select>
```

- **BookController.java中添加**

```
@Autowired
private BookService bookService;

@PostMapping("listBook")
@Privilege("page")
public Page<BookVO> listBook(@RequestBody BookQueryDTO queryDTO) {
    return bookService.listByPage(queryDTO);
}
```

- **添加BookQueryDTO.java模糊查询**

```
package redlib.backend.dto.query;

import lombok.Data;

@Data
public class BookQueryDTO extends PageQueryDTO{
    /**
     * 书籍名称，模糊匹配
     */
    private String bookName;
}
```

前端

- **routes.ts中添加页面路由**

```
{
        path: '/base/book',
        name: 'book',
        component: './base/book',
        access: 'hasPrivilege'
      },
```

- **在\src\locales\zh-CN\menu.ts修改维护菜单中文名称**

```
'menu.base.book': '书籍管理',
```

- **创建 \src\pages\base\book\index.tsx**

- **运行Openapi**

```
npm run openapi
```

- **基本框架代码**

```
import { deleteBook, listBook } from '@/services/api/book';
import { convertPageData } from '@/utils/request';
import { openConfirm } from '@/utils/ui';
import { PlusOutlined, DeleteOutlined } from '@ant-design/icons';
import { ActionType, PageContainer, ProColumns, ProTable } from '@ant-
design/pro-components';
import { Button } from 'antd';
import { useRef, useState } from 'react';
import InputDialog from './InputDialog';

export default () => {
  const refAction = useRef<ActionType>(null);
  //声明两个常量，变量名为selectedRowKeys，改变变量值的方法为selectRow
```

```
const [selectedRowKeys, selectRow] = useState<number[]>([]);
//声明页面级状态变量，将变量传递给对话框，<>内表示是什么类型的数据，()为空表示不对其初始化
const [book, setBook] = useState<API.BookVO>();
const [visible, setVisible] = useState(false);
//定义一个Delete的回调函数
const handleDelete = async () => {
  if (!selectedRowKeys?.length) return;
  openConfirm(`您确定删除${selectedRowKeys.length}条记录吗`, async () => {
    // await deleteBook(selectedRowKeys);
    refAction.current?.reload();
  });
};
const columns: ProColumns<API.BookVO>[] = [
  {
    title: 'ID',
    dataIndex: 'bookId',
    width: 100,
    search: false,
  },
  {
    title: '书名',
    dataIndex: 'bookName',
    width: 100,
    render: (dom, record) => {
      return (
        <a
          onClick={() => {
            setBook(record);
            setVisible(true);
          }}
        >
          {dom}
        </a>
      );
    },
  },
  {
    title: 'ISBN',
    dataIndex: 'isbn',
    width: 100,
    search: false,
  },
  {
    title: '分类号',
    dataIndex: 'bookType',
    width: 100,
    search: false,
  },
  {
    title: '作者',
    dataIndex: 'bookAuthor',
    width: 100,
    search: false,
  },
  {
```

```
      title: '出版社',
      dataIndex: 'bookPublish',
      width: 100,
      search: false,
    },
    {
      title: '出版时间',
      dataIndex: 'bookPubtime',
      width: 100,
      search: false,
    },
    {
      title: '价格',
      dataIndex: 'bookPrice',
      width: 100,
      search: false,
    },
    {
      title: '购买时间',
      dataIndex: 'bookBoutime',
      width: 100,
      search: false,
    },
  ];



  return (
    <PageContainer>
      <ProTable<API.BookVO>
        actionRef={refAction}
        rowKey="id"
        //params为分页信息，protable自动将params传递给listBook，
        //数据由convertPageData转换为antd protable的数据格式
        request={async (params = {}) => {
          return convertPageData(await listBook(params));
        }}
        toolBarRender={() => [
          <Button
            type="primary"
            key="primary"
            onClick={() => {
              setBook(undefined);
              setVisible(true);
            }}
          >
            <PlusOutlined /> 新建
          </Button>,
          <Button
            type="primary"
            key="primary"
            danger
            onClick={handleDelete}
            disabled={!selectedRowKeys?.length}
          >
```

```
            <DeleteOutlined /> 删除
          </Button>,
        ]}
        columns={columns}
        rowSelection={{
          onChange: (rowKeys) => {
            selectRow(rowKeys as number[]);
          },
        }}
      />
      //通过visible控制状态框是否显示
      <InputDialog
        detailData={book}
        onClose={(result) => {
          setVisible(false);
          result && refAction.current?.reload();
        }}
        visible={visible}
      />
    </PageContainer>
  );
};
```

- **完成基本页面显示**

为Book增加功能

后端

1.addBook

- **增加接口**

```
Integer addBook(BookDTO bookDTO);
```

- **BookController.java中添加**

```
@PostMapping("addBook")
@NeedNoPrivilege
public void addBook(@RequestBody BookDTO bookDTO){
    bookService.addBook(bookDTO);
}
```

- **添加BookQueryDTO.java**

```
package redlib.backend.dto.query;

import lombok.Data;

@Data
public class BookQueryDTO extends PageQueryDTO{
    /**
         * 书籍名称，模糊匹配
         */
    private String bookName;
}
```

- **完善Impl实现**

```
@Override
public Integer addBook(BookDTO bookDTO){
    // 校验输入数据正确性
    BookUtils.validateBook(bookDTO);

    /*
             * 此处后续可以增加条件
             *    相同名称的书籍可能可以共存（不同ISBN）
             * */
    //         Assert.isNull(bookMapper.getByName(bookDTO.getBook_name()),"书籍已经存在");

    // 创建实体对象，用以保存到数据库
    Book book = new Book();
    // 将输入的字段全部复制到实体对象中
    BeanUtils.copyProperties(bookDTO, book);
    // 调用DAO方法保存到数据库表
    bookMapper.insert(book);
    return book.getBookId();
}
```

描述

接口地址：book/addBook

请求方法: POST

入参：

参数名	参数描述	数据类型	必填	示例/备注
bookName	名称	String	M	书名
isbn	ISBN	String	M	1234567891234
bookAuthor	作者	String	M	作者
bookPublish	出版社	String	M	山大

参数名rice	参数描述	数据类型	必填	示例/备注
bookPubtime	出版时间	String	O	2021-08-09
bookBoutime	购买时间	String	O	2021-08-09
bookType	分类号	String	O	GV943

出参：无

示例：

```
{"code":200,"data":null,"message":null,"success":true}
```

接口描述：

1、 校验必填项不能为空

2、 将入参保存至book表各对应字段

测试

成功新增信号与系统

 12

 13

2.updateBook

- **增加接口**

```
Integer updateBook(BookDTO bookDTO);
```

- **BookController.java中添加**

```
@PostMapping("updateBook")
@Privilege("update")
public Integer updateBook(@RequestBody BookDTO bookDTO) {
    return bookService.updateBook(bookDTO);
}
```

- **完善Impl实现**

```
@Override
public Integer updateBook(BookDTO bookDTO) {
    // 校验输入数据正确性
    Token token = ThreadContextHolder.getToken();
    BookUtils.validateBook(bookDTO);
    Assert.notNull(bookDTO.getBookId(), "书籍id不能为空");
    Book book = bookMapper.selectByPrimaryKey(bookDTO.getBookId());
    Assert.notNull(book, "没有找到书籍，Id为：" + bookDTO.getBookId());

    BeanUtils.copyProperties(bookDTO, book);
    bookMapper.updateByPrimaryKey(book);
    return book.getBookId();
}
```

描述

接口地址：book/updateBook

请求方法: POST

入参：书籍id

参数名	参数描述	数据类型	必填	示例/备注
bookName	名称	String	M	书名
isbn	ISBN	String	M	1234567891234
bookAuthor	作者	String	M	作者
bookPublish	出版社	String	M	山大
bookPrice	价格	Double	O	23.13
bookPubtime	出版时间	String	O	2021-08-09
bookBoutime	购买时间	String	O	2021-08-09
bookType	分类号	String	O	GV943

出参：无

示例：

```
{"code":200,"data":1,"message":null,"success":true}
```

接口描述：

1、 校验必填项不能为空

测试

山大--->科学出版社，成功修改

 9

 10

 11

3.deleteByCodes

- **增加接口**

```
void deleteByCodes(List<Integer> ids);
```

- **BookController.java中添加**

```
@PostMapping("deleteBook")
@Privilege("delete")
public void deleteBook(@RequestBody List<Integer> ids) {
    bookService.deleteByCodes(ids);
}
```

- **完善Impl实现**

```
@Override
public Integer updateBook(BookDTO bookDTO) {
    // 校验输入数据正确性
    Token token = ThreadContextHolder.getToken();
    BookUtils.validateBook(bookDTO);
    Assert.notNull(bookDTO.getBookId(), "书籍id不能为空");
    Book book = bookMapper.selectByPrimaryKey(bookDTO.getBookId());
    Assert.notNull(book, "没有找到书籍，Id为：" + bookDTO.getBookId());

    BeanUtils.copyProperties(bookDTO, book);
    bookMapper.updateByPrimaryKey(book);
    return book.getBookId();
}
```

- **添加BookMapper.java**

```
void deleteByCodes(@Param("codeList") List<Integer> codeList);
```

- **添加BookMapper.xml**

```
<!-- 批量删除 -->
  <update id="deleteByCodes">
    delete from book
    where book_id in
    <foreach item="item" index="index" collection="codeList" open="("
separator="," close=")">
      #{item}
    </foreach>
```

描述

接口地址：book/deleteBook

请求方法: POST

入参:

参数名	参数描述	数据类型	必填	示例/备注
0	书籍ID	Integer	M	0: 57

出参：无

示例:

```
{"code":200,"data":null,"message":null,"success":true}
```

接口描述：暂无

测试

成功删除《马原》

 14

 15

4.多种模糊查询

- **BookQueryDTO.java**

```
private String isbn;
private String author;
private String publisher;
private String category;
private String status;
```

- **完善Impl实现**

```
queryDTO.setName(FormatUtils.makeFuzzySearchTerm(queryDTO.getName()));
queryDTO.setIsbn(FormatUtils.makeFuzzySearchTerm(queryDTO.getIsbn()));
queryDTO.setAuthor(FormatUtils.makeFuzzySearchTerm(queryDTO.getAuthor()));
queryDTO.setPublisher(FormatUtils.makeFuzzySearchTerm(queryDTO.getPublisher()));
queryDTO.setCategory(FormatUtils.makeFuzzySearchTerm(queryDTO.getCategory()));
queryDTO.setStatus(FormatUtils.makeFuzzySearchTerm(queryDTO.getStatus()));
```

- **添加BookMapper.xml**

```
<select id="list" resultMap="BaseResultMap">
    select
    <include refid="Base_Column_List"/>
    from book
    <where>
      <if test="queryDTO.name != null">
        name like #{queryDTO.name}
      </if>
      <if test="queryDTO.isbn != null">
        isbn like #{queryDTO.isbn}
      </if>
      <if test="queryDTO.author != null">
        author like #{queryDTO.author}
      </if>
      <if test="queryDTO.publisher != null">
        publisher like #{queryDTO.publisher}
      </if>
      <if test="queryDTO.category != null">
        category like #{queryDTO.category}
      </if>
      <if test="queryDTO.status != null">
        status like #{queryDTO.status}
      </if>
    </where>
    limit #{offset}, #{limit}
  </select>
```

前端

修改 src\pages\base\book\InputDialog\index.tsx 文件

```
import { ModalForm, ProForm, ProFormInstance, ProFormText, ProFormDatePicker }
from '@ant-design/pro-components';
import { message } from 'antd';
import { useEffect, useRef } from 'react';
import { waitTime } from '@/utils/request';
import { addBook, updateBook } from '@/services/api/book';
import moment from 'dayjs';

interface InputDialogProps {
  detailData?: API.BookDTO;
```

```
  visible: boolean;
  onClose: (result: boolean) => void;
}

export default function InputDialog(props: InputDialogProps) {
  const form = useRef<ProFormInstance>(null);

  useEffect(() => {
    waitTime().then(() => {
      if (props.detailData) {
        form?.current?.setFieldsValue(props.detailData);
      } else {
        form?.current?.resetFields();
      }
    });
  }, [props.detailData, props.visible]);

  const onFinish = async (values: any) => {
    const { bookName, isbn, bookAuthor, bookPublish, bookPrice, bookType,
bookBoutime, bookPubtime } = values;
    const data: API.BookDTO = {
      bookId: props.detailData?.bookId,
      bookName,
      isbn,
      bookAuthor,
      bookPublish,
      bookPrice,
      bookType,
      bookBoutime,
      bookPubtime,

    };
    if (props.detailData) {
      await updateBook(data);
    } else {
      await addBook(data);
    }
    message.success('保存成功');
    props.onClose(true);
    return true;
  };

  return (
    //返回一个模态对话框
    <ModalForm
      width={600}
      onFinish={onFinish}
      formRef={form}
      modalProps={{
        destroyOnClose: true,
        onCancel: () => props.onClose(false),
      }}
      title={props.detailData ? '修改书籍' : '新增书籍'}
      open={props.visible}
    >
```

```
      <ProForm.Group>
        <ProFormText
          name="bookName"
          label="书籍名称"
          rules={[
            {
              required: true,
              message: '请输入书籍名称！',
            },
          ]}
        />
        <ProFormText
          name="isbn"
          label="ISBN"
          rules={[
            {
              required: true,
              message: '请输入ISBN号！',
            },
          ]}
        />
      </ProForm.Group>
      <ProForm.Group>
        <ProFormText
          name="bookPublish"
          label="出版社"
          rules={[
            {
              required: true,
              message: '请输入出版社名称！',
            },
          ]}
        />
        <ProFormText
          name="bookAuthor"
          label="作者"
          rules={[
            {
              required: true,
              message: '请输入作者名称！',
            },
          ]}
        />
      </ProForm.Group>
      <ProForm.Group>
        <ProFormText
          name="bookPrice"
          label="书籍价格"
        />
        <ProFormText
          name="bookType"
          label="分类号"
        />
      </ProForm.Group>
```

```
      <ProForm.Group>
        <ProFormDatePicker
          name="bookBoutime"
          label="购买时间"
          width={200}
          initialValue={moment('2021-08-09')}
        />
        <ProFormDatePicker
          name="bookPubtime"
          label="出版时间"
          width={200}
          initialValue={moment('2021-08-09')}
        />
      </ProForm.Group>
    </ModalForm>
  );
}
```

删除 search=false 实现多种模糊查询

借阅日志

借阅日志表

借阅日志表 *book_log*

字段名称	数据类型	索引	样例	说明
id	int(11)	Primary Key	1234	自增id主键
reader_name	char(255)		张三	借阅人
book_id	int(11)		23	书籍ID
book_name	char(255)		信号与系统	书名
book_isbn	char(255)		1234567891234	书籍ISBN，必须为13位
borrowed_time	date		2023-04-01	借阅时间
return_time	date		TX001	还书时间，可以为NULL

接口设计

1.展示页面

接口地址：bookLog/listBookLog

请求方法: POST

入参

参数名	参数描述	数据类型	必填	示例/备注
current	当前页面	Integer	M	1
pageSize	页面大小	Integer	M	1

出参:

参数名	参数描述	数据类型	必填	示例/备注
data	返回数据	List of Object	M	
├─ current	当前页面	Integer	M	过滤条件
├─ list	返回列表	List of Object	O	
├── bookId	书籍id	Integer	O	
├── bookIsbn	书籍isbn	String	O	
├── bookName	书名	String	O	
├── borrowedTime	借阅时间	date	O	
├── id	id	Integer	O	主键
├── readerName	借阅人姓名	String	O	
├── returnTime	归还时间	date	O	
├─ pageSize	页面大小	Integer	M	过滤条件
├─ total	总命中数	Integer	M	

示例:


```
{
  "code": 200,
  "data": {
    "current": 1,
    "pageSize": 1,
    "total": 5,
    "list": [
      {
        "id": 1,
        "name": "string",
        "isbn": "1234314234323",
        "author": "string",
        "publisher": "string",
```

```
        "price": 321,
        "category": "string",
        "boughtTime": "2023-04-01",
        "publishedTime": "2023-04-01",
        "status": "借出",
        "classification": "漫画"
      }
    ]
  },
  "message": null,
  "success": true
}
```


接口描述：

查询MySQL中book_log表数据。

1.实现基础功能，分页，更新，删除、增加

后端

BookLogServiceImpl.java

```
package redlib.backend.service.impl;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.util.Assert;
import redlib.backend.dao.BookLogMapper;
import redlib.backend.dao.BookMapper;
import redlib.backend.dto.BookDTO;
import redlib.backend.dto.BookLogDTO;
import redlib.backend.dto.query.BookLogQueryDTO;
import redlib.backend.dto.query.BookQueryDTO;
import redlib.backend.model.Book;
import redlib.backend.model.BookLog;
import redlib.backend.model.Page;
import redlib.backend.service.AdminService;
import redlib.backend.service.BookLogService;
import redlib.backend.service.utils.BookLogUtils;
import redlib.backend.service.utils.BookUtils;
import redlib.backend.utils.FormatUtils;
import redlib.backend.utils.PageUtils;
import redlib.backend.vo.BookLogVO;
import redlib.backend.vo.BookVO;
import java.util.ArrayList;
import java.util.List;
@Service
public class BookLogServiceImpl implements BookLogService {
    @Autowired
    private BookLogMapper bookLogMapper;

    @Autowired
    private AdminService adminService;
```

```
    @Override
    public Integer addBookLog(BookLogDTO bookLogDTO){
        // 校验输入数据正确性
        BookLogUtils.validateBookLog(bookLogDTO);
        // 创建实体对象，用以保存到数据库
        BookLog bookLog = new BookLog();
        // 将输入的字段全部复制到实体对象中
        BeanUtils.copyProperties(bookLogDTO, bookLog);
        // 调用DAO方法保存到数据库表
        bookLogMapper.insert(bookLog);
        return bookLog.getId();
    }

    @Override
    public Integer updateBookLog(BookLogDTO bookLogDTO) {
        // 校验输入数据正确性
        BookLogUtils.validateBookLog(bookLogDTO);
        Assert.notNull(bookLogDTO.getBookId(), "书籍id不能为空");
        BookLog bookLog =
bookLogMapper.selectByPrimaryKey(bookLogDTO.getBookId());
        Assert.notNull(bookLog, "没有找到书籍，Id为：" + bookLogDTO.getBookId());

        BeanUtils.copyProperties(bookLogDTO, bookLog);
        bookLogMapper.updateByPrimaryKey(bookLog);
        return bookLog.getId();
    }

    @Override
    public Page<BookLogVO> listByPage(BookLogQueryDTO queryDTO) {
        if (queryDTO == null) {
            queryDTO = new BookLogQueryDTO();
        }

        //模糊查询

 queryDTO.setBookName(FormatUtils.makeFuzzySearchTerm(queryDTO.getBookName()));
        queryDTO.setOrderBy(FormatUtils.formatOrderBy(queryDTO.getOrderBy()));
        //将查询条件发送给dao接口，先查询命中个数
        Integer size = bookLogMapper.count(queryDTO);
        PageUtils pageUtils = new PageUtils(queryDTO.getCurrent(),
queryDTO.getPageSize(), size);

        if (size == 0) {
            // 没有命中，则返回空数据。
            return pageUtils.getNullPage();
        }

        List<BookLog> list = bookLogMapper.list(queryDTO, pageUtils.getOffset(),
pageUtils.getLimit());

        List<BookLogVO> voList = new ArrayList<>();
        for (BookLog bookLog : list) {
            BookLogVO vo = BookLogUtils.convertToVO(bookLog);
            voList.add(vo);
```

```
        }
        return new Page<>(pageUtils.getCurrent(), pageUtils.getPageSize(),
pageUtils.getTotal(), voList);
    }

    @Override
    public void deleteByCodes(List<Integer> ids) {
        Assert.notEmpty(ids, "书籍id列表不能为空");
        bookLogMapper.deleteByCodes(ids);
    }


}
```

BookLogService.java

```
package redlib.backend.service;

import redlib.backend.dto.BookLogDTO;
import redlib.backend.dto.query.BookLogQueryDTO;
import redlib.backend.model.Page;
import redlib.backend.vo.BookLogVO;

import java.util.List;

public interface BookLogService {
    Integer addBookLog(BookLogDTO bookLogDTO);

    Page<BookLogVO> listByPage(BookLogQueryDTO queryDTO);

    /**
     * 更新部门数据
     *
     * @param bookLogDTO 部门输入对象
     * @return 部门编码
     */
    Integer updateBookLog(BookLogDTO bookLogDTO);

    /**
     * 根据编码列表，批量删除书籍
     *
     * @param ids 编码列表
     */
    void deleteByCodes(List<Integer> ids);

}
```

BookLogMapper.xml添加

```
  <!-- 获取名字个数(分页) -->
  <select id="count" resultType="integer">
    select count(*)
    from book_log
```

```
    <where>
      <if test="bookName != null">
        book_name like #{bookName}
      </if>
    </where>
  </select>

  <!-- 获取书籍(分页) -->
  <select id="list" resultMap="BaseResultMap">
    select
    <include refid="Base_Column_List"/>
    from book_log
    <where>
      <if test="queryDTO.readerName != null">
        and reader_name like #{queryDTO.readerName}
      </if>
      <if test="queryDTO.bookIsbn != null">
        and book_isbn like #{queryDTO.bookIsbn}
      </if>
      <if test="queryDTO.bookId != null">
        and book_id like #{queryDTO.bookId}
      </if>
      <if test="queryDTO.bookName != null">
        and book_name like #{queryDTO.bookName}
      </if>
    </where>
    <if test="queryDTO.orderBy != null">
      order by ${queryDTO.orderBy}
    </if>
    <if test="queryDTO.orderBy == null">
      order by id desc
    </if>
    limit #{offset}, #{limit}
  </select>

  <!-- 批量删除 -->
  <update id="deleteByCodes">
    delete from book_log
    where id in
    <foreach item="item" index="index" collection="codeList" open="("
separator="," close=")">
      #{item}
    </foreach>
  </update>
```

BookLogMapper.java

```
package redlib.backend.dao;

import org.apache.ibatis.annotations.Param;
import redlib.backend.dto.query.BookLogQueryDTO;
import redlib.backend.model.BookLog;

import java.util.List;
```

```
public interface BookLogMapper {
    int deleteByPrimaryKey(Integer id);

    int insert(BookLog record);

    int insertSelective(BookLog record);

    BookLog selectByPrimaryKey(Integer id);

    int updateByPrimaryKeySelective(BookLog record);

    int updateByPrimaryKey(BookLog record);



    List<BookLog> list(@Param("queryDTO") BookLogQueryDTO queryDTO,
                       @Param("offset") Integer offset,
                       @Param("limit") Integer limit);

    Integer count(BookLogQueryDTO queryDTO);

    /**
     * 根据代码列表批量删除书籍
     *
     * @param codeList id列表
     */
    void deleteByCodes(@Param("codeList") List<Integer> codeList);

}
```

BookLogController.java

```
package redlib.backend.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import redlib.backend.annotation.BackendModule;
import redlib.backend.annotation.NeedNoPrivilege;
import redlib.backend.annotation.Privilege;
import redlib.backend.dto.BookDTO;
import redlib.backend.dto.BookLogDTO;
import redlib.backend.dto.query.BookLogQueryDTO;
import redlib.backend.dto.query.BookQueryDTO;
import redlib.backend.model.Page;
import redlib.backend.service.BookLogService;
import redlib.backend.service.BookService;
import redlib.backend.vo.BookLogVO;
import redlib.backend.vo.BookVO;

import java.util.List;

@RestController
```

```
@RequestMapping("/api/bookLog")
@BackendModule({"page:页面"})
public class BookLogController {
    @Autowired
    private BookLogService bookLogService;
    @PostMapping("addBookLog")
    @NeedNoPrivilege
    public void addBookLog(@RequestBody BookLogDTO bookLogDTO){
        bookLogService.addBookLog(bookLogDTO);
    }

    @PostMapping("listBookLog")
//    @Privilege("page")
    @NeedNoPrivilege
    public Page<BookLogVO> listBookLog(@RequestBody BookLogQueryDTO queryDTO) {
        return bookLogService.listByPage(queryDTO);
    }

    @PostMapping("updateBookLog")
    @Privilege("update")
    public Integer updateBookLog(@RequestBody BookLogDTO bookLogDTO) {
        return bookLogService.updateBookLog(bookLogDTO);
    }

    @PostMapping("deleteBookLog")
    @Privilege("delete")
    public void deleteBookLog(@RequestBody List<Integer> ids) {
        bookLogService.deleteByCodes(ids);
    }
}
```

BookLogQueryDTO.java

```
package redlib.backend.dto.query;

import lombok.Data;

/**
 * @author bsgbs
 */
@Data
public class BookLogQueryDTO extends PageQueryDTO{
    private Integer readerId;
    private String readerName;
    private Integer bookId;
    private String bookIsbn;
    private String bookName;
    private String orderBy;

}
```

BookLogDTO.java

```
package redlib.backend.dto;
```

```java
import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.Data;

import java.util.Date;

@Data
public class BookLogDTO {
    private String readerName;
    private Integer bookId;
    private String bookName;
    private String bookIsbn;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private Date borrowedTime;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private Date returnTime;

}
```

BookLogVO.java

```java
package redlib.backend.vo;


import lombok.Data;
import redlib.backend.model.BookLog;

/**
 * @author bsgbs
 */
@Data
public class BookLogVO extends BookLog {
}
```

前端

index主页面

```
import {  deleteBookLog,listBookLog } from '@/services/api/bookLog';
import { convertPageData,orderBy } from '@/utils/request';
import { openConfirm } from '@/utils/ui';
import { PlusOutlined, DeleteOutlined } from '@ant-design/icons';
import { ActionType, PageContainer, ProColumns, ProTable } from '@ant-
design/pro-components';
import { Button } from 'antd';
import { useRef, useState } from 'react';
import InputDialog from './InputDialog';

export default () => {
  const refAction = useRef<ActionType>(null);
  //声明两个常量，变量名为selectedRowKeys，改变变量值的方法为selectRow
  const [selectedRowKeys, selectRow] = useState<number[]>([]);
  //声明页面级状态变量，将变量传递给对话框，<>内表示是什么类型的数据，()为空表示不对其初始化
  const [book, setBook] = useState<API.BookLogVO>();
  const [visible, setVisible] = useState(false);
```

```
const columns: ProColumns<API.BookLogVO>[] = [
  {
    title: '书籍ID',
    dataIndex: 'bookId',
    sorter: true,
    fixed: true,
    width: 50,
    search: false,
  },
  {
    title: '书名',
    dataIndex: 'bookName',
    width: 100,
  },
  {
    title: '借阅人',
    dataIndex: 'readerName',
    fixed: true,
    width: 100,
  },
  {
    title: 'ISBN',
    dataIndex: 'bookIsbn',
    width: 100,
    search: false,
  },
  {
    title: '借阅日期',
    width: 100,
    sorter: true,
    dataIndex: 'borrowedTime',
    valueType: 'date',
    search: false,
  },
  {
    title: '还书日期',
    dataIndex: 'returnTime',
    key: 'returnTime',
    valueType: 'date',
    width: 100,
    search: false,
  },
];


return (
  <PageContainer>
    <ProTable<API.BookLogVO>
      actionRef={refAction}
      rowKey="id"
      pagination={{
        defaultPageSize: 10,
      }}
      search={{
```

```
          labelWidth: 120,
        }}
        scroll={{ x: 100 }}
        request={async (params = {}, sort) => {
          console.log(sort);
          return convertPageData(await listBookLog({ ...params, orderBy:
orderBy(sort) }));
        }}
        columns={columns}
      />
    </PageContainer>
  );
};
```

inputdialog页面

```
import { ModalForm, ProForm, ProFormInstance, ProFormSelect, ProFormText,
ProFormDatePicker } from '@ant-design/pro-components';
import { message, Switch, Tooltip } from 'antd';
import { useEffect, useRef, useState } from 'react';
import { waitTime } from '@/utils/request';
import { addBook, updateBook } from '@/services/api/book';
import moment from 'dayjs';

interface InputDialogProps {
  detailData?: API.BookDTO;
  visible: boolean;
  onClose: (result: boolean) => void;
}

export default function InputDialog(props: InputDialogProps) {
  const form = useRef<ProFormInstance>(null);
  const [readonly, setReadonly] = useState(false);

  useEffect(() => {
    waitTime().then(() => {
      if (props.detailData) {
        form?.current?.setFieldsValue(props.detailData);
      } else {
        form?.current?.resetFields();
      }
    });
  }, [props.detailData, props.visible]);

  const onFinish = async (values: any) => {
    const { name, isbn, author, publisher, price, category, boughtTime,
publishedTime, status } = values;
    const data: API.BookDTO = {
      id: props.detailData?.id,
      name,
      isbn,
      author,
      publisher,
      price,
      category,
```

```
      boughtTime,
      publishedTime,
      status,
    };

    if (props.detailData) {
      await updateBook(data);
    } else {
      await addBook(data);
    }
    message.success('保存成功');
    props.onClose(true);
    return true;
  };


  return (
    //返回一个模态对话框
    <ModalForm
      width={600}
      onFinish={onFinish}
      formRef={form}
      modalProps={{
        destroyOnClose: true,
        onCancel: () => props.onClose(false),
      }}
      title={props.detailData ? '修改书籍' : '新增书籍'}
      open={props.visible}
    >

      <ProForm.Group>
        <ProFormText
          name="name"
          label="书籍名称"
          rules={[
            {
              required: true,
              message: '请输入书籍名称! ',
            },
          ]}
        />

        <ProFormText
          name="isbn"
          label="ISBN"
          tooltip="输入必须为13位"
          rules={[
            {
              required: true,
              message: '请输入ISBN号! ',
            },
            {
              // 此处设为false可以解决填写后退格导致两种报错的问题
              required: false,
              message: 'ISBN格式不符合要求! ',
```

```
            max: 13,
            min: 13
          },
        ]}
      />
    </ProForm.Group>
    <ProForm.Group>
      <ProFormText
        name="publisher"
        label="出版社"
        rules={[
          {
            required: true,
            message: '请输入出版社名称! ',
          },
        ]}
      />
      <ProFormText
        name="author"
        label="作者"
        rules={[
          {
            required: true,
            message: '请输入作者名称! ',
          },
        ]}
      />
    </ProForm.Group>
    <ProForm.Group>
      <ProFormText
        name="price"
        label="书籍价格"
      />
      <ProFormText
        name="category"
        label="分类号"
      />
    </ProForm.Group>

    <ProForm.Group>
      <ProFormDatePicker
        name="boughtTime"
        label="购买时间"
        initialValue={moment()}
      />
      <ProFormDatePicker
        name="publishedTime"
        label="出版时间"
        initialValue={moment()}
      />
      <ProFormSelect
        name="status"
        label="状态"
        options={[
          { label: '在架', value: '在架' },
```

```
            { label: '借出', value: '借出' },
          ]}
        />
      </ProForm.Group>
    </ModalForm >
  );
}
```

2.序号及借阅日期排序

前端

return中添加

```
<ProTable<API.BookLogVO>
        actionRef={refAction}
        rowKey="id"
        pagination={{
          defaultPageSize: 10,
        }}
        search={{
          labelWidth: 120,
        }}
        scroll={{ x: 100 }}
        request={async (params = {}, sort) => {
          console.log(sort);
          return convertPageData(await listBookLog({ ...params, orderBy:
orderBy(sort) }));
        }}
        columns={columns}
      />
```

后端

BookLogServiceImpl.java中添加

```
queryDTO.setOrderBy(FormatUtils.formatOrderBy(queryDTO.getOrderBy()));
```

BookLogQueryDTO.java中添加

```
private String orderBy;
```

xml中后添加

```
<if test="queryDTO.orderBy != null">
      order by ${queryDTO.orderBy}
    </if>
    <if test="queryDTO.orderBy == null">
      order by id desc
    </if>
```

测试

点击书籍ID排序如下

 19

 20

借书

接口设计

借书

接口地址：book/lendBook

请求方法: POST

入参

参数名	参数描述	数据类型	必填	示例/备注
bookDTO	BookDTO数据	List of Object	M	
├─ id	书籍id	Integer	M	主键
├─ name	书名	List of Object	O	
├─ isbn	书籍isbn	String	O	
├─ author	书籍作者	String	O	
├─ publisher	出版社	String	O	
├─ price	书籍价格	Double	O	
├─ category	书籍分类号	Integer	O	
├─ boughtTime	书籍购买时间	date	O	
├─ publishedTime	书籍出版时间	date	O	
├─ status	书籍状态	String	O	过滤条件
├─ classification	书籍分类	String	O	

参数名	参数描述	数据类型	必填	示例/备注
bookLogDTO	LogDTO数据	List of Object	M	
├─ readerName	当前页面	String	M	
├─ borrowedTime	返回列表	date	O	

出参：无

示例：


```
{"code":200,"data":null,"message":null,"success":true}
```


接口描述：

1.前端向后端发送选中书籍信息和借阅信息，后端将借阅日志所需的信息从书籍信息中复制进来。

2.后端根据书籍id检索，自动更新书籍状态为“借出”，并且增加一条借阅日志。

1.实现借书功能

后端

参考Department的写法，之前在添加部门中使用insert函数添加，理论上只要添加适当的函数即可在借还书籍时同时为log中添加记录

 16

添加BookAndLogDTO类用于接收前端传递的参数

```
@Data
public class BookAndLogDTO {
    private BookDTO bookDTO;
    private BookLogDTO bookLogDTO;
}
```

BookService.java添加接口

```
Integer lendBook(BookAndLogDTO bookAndLogDTO);
```

Impl.java中完善

```
@Override
public Integer lendBook(BookAndLogDTO bookAndLogDTO){
    // 校验输入数据正确性
    BookLogUtils.validateBookLog(bookAndLogDTO.getBookLogDTO());
    BookUtils.validateBook(bookAndLogDTO.getBookDTO());

    // 创建实体对象，用以保存到数据库
    BookLog log = new BookLog();
    Book book = new Book();

    //修改书籍状态为借出
```

```
    bookAndLogDTO.getBookDTO().setStatus("借出");

    // 将输入的字段全部复制到实体对象中
    //将两个DTO中的数据传递到log中，便于使用bookLogMapper中的insert函数
    BeanUtils.copyProperties(BookLogUtils.convertToDTO(bookAndLogDTO),log);
    //将bookAndLogDTO类中的BookDTO()数据传入book中用于更新状态为借出
    BeanUtils.copyProperties(bookAndLogDTO.getBookDTO(), book);

    // 调用DAO方法保存到数据库表
    bookLogMapper.insert(log);
    // 更新书籍状态
    bookMapper.updateByPrimaryKey(book);

    return log.getId();
}
```

BookController.java

```
@PostMapping("lendBook")
@Privilege("lend")
public Integer lendBook(@RequestBody BookAndLogDTO bookAndLogDTO) {
    return bookService.lendBook(bookAndLogDTO);
}
```

BookLogUtils.java中添加格式转换函数

```
/**
    将两个DTO中的数据传递到log中，便于使用bookLogMapper中的insert函数
    接收BookAndLogDTO类型，复制内容并输出为bookLogDTO类型
    */
public static BookLogDTO convertToDTO(BookAndLogDTO bookAndLogDTO) {
    BookLogDTO bookLogDTO = new BookLogDTO();
    //此处无法直接Copy，因为需要把两个不同的DTO中的数据传递到bookLogDTO中
    bookLogDTO.setBookId(bookAndLogDTO.getBookDTO().getId());
    bookLogDTO.setBookName(bookAndLogDTO.getBookDTO().getName());
    bookLogDTO.setBookIsbn(bookAndLogDTO.getBookDTO().getIsbn());

    bookLogDTO.setBorrowedTime(bookAndLogDTO.getBookLogDTO().getBorrowedTime());
    bookLogDTO.setReturnTime(bookAndLogDTO.getBookLogDTO().getReturnTime());
    bookLogDTO.setReaderName(bookAndLogDTO.getBookLogDTO().getReaderName());
    return bookLogDTO;
}
```

前端

主页面添加借书对话框

导入对话框组件

```
import InputDialog2 from './InputDialog2';
```

定义变量

```
const [bookLog, setBookLog] = useState<API.BookVO>();
const [lendVisible, setLendVisible] = useState(false);
```

实现对话框交互

```
<InputDialog2
detailData={bookLog}
onClose={(result) => {
    setLendVisible(false);
    result && refAction.current?.reload();
}}
visible={lendVisible}
/>
```

对话框实现按钮

```
<Button
type="primary"
key="primary"
onClick={() => {
    setLendVisible(true);
}}
disabled={!selectedRowKeys?.length}
>
    <LogoutOutlined /> 借出
</Button>,
```

借书思路

> - 选择某一行，可以点击“借阅”按钮，实现借书
> - 点击借书，对话框内显示书籍资料，手动填入借阅人信息
> - 后续考虑扩展为选择多行书籍，同时借阅

通过查询选择行的代码，发现**onChange()**函数，可以自带两个参数

```
    onChange?: (selectedRowKeys: Key[], selectedRows: T[], info: {
        type: RowSelectMethod;
    }) => void;
```

其中第一个参数为选择行的主键（即ID），第二个参数为选择行的列表**（此特性也为后续选择多行同时借阅提供可能）**

因此，实现借出一本书只需要传递records[0]到对话框即可，代码如下

```
rowSelection={{
    onChange: (rowKeys, records) => {
        selectRow(rowKeys as number[]);
        setBookLog(records[0]);
  },
}}
```

inputdialog对话框

```
import { ModalForm, ProForm, ProFormInstance, ProFormSelect, ProFormText,
ProFormDatePicker } from '@ant-design/pro-components';
import { message, Switch, Tooltip } from 'antd';
import { useEffect, useRef, useState } from 'react';
import { waitTime } from '@/utils/request';
import { addBook, updateBook, lendBook } from '@/services/api/book';
import moment from 'dayjs';
import { addBookLog, updateBookLog } from '@/services/api/bookLog';



interface InputDialogProps {
  detailData?: API.BookDTO;
  visible: boolean;
  onClose: (result: boolean) => void;
}

export default function InputDialog(props: InputDialogProps) {
  const form = useRef<ProFormInstance>(null);
  const [readonly, setReadonly] = useState(false);

  useEffect(() => {
    waitTime().then(() => {
      if (props.detailData) {
        form?.current?.setFieldsValue(props.detailData);
      } else {
        form?.current?.resetFields();
      }
    });
  }, [props.detailData, props.visible]);

  const onFinish = async (values: any) => {
    const { name, isbn, author, publisher, price, category, boughtTime,
publishedTime, status, classification,
      readerName, bookId, bookName, bookIsbn, borrowedTime, returnTime } =
values;
    const data: API.BookAndLogDTO = {
      bookDTO: {
        id: props.detailData?.id,
        name,
        isbn,
        author,
        publisher,
        price,
        category,
        boughtTime,
        publishedTime,
        status,
        classification
      },
      bookLogDTO: {
        readerName,
        bookId,
```

```
        bookName,
        bookIsbn,
        borrowedTime,
        returnTime
      }
    };


    if (props.detailData) {
      try {
        await lendBook(data);
      }
      catch (error) {
        message.success('借出成功');
      }
    } else {
    }

    props.onClose(true);
    return true;
  };


  return (
    //返回一个模态对话框
    <ModalForm
      width={600}
      onFinish={onFinish}
      formRef={form}
      modalProps={{
        destroyOnClose: true,
        onCancel: () => props.onClose(false),
      }}
      title={props.detailData ? '借出' : '借出'}
      open={props.visible}
    >
      <ProFormText
        name="category"
        label="category"
        hidden
      />
      <ProFormText
        name="boughtTime"
        label="boughtTime"
        hidden
      />
      <ProFormText
        name="publishedTime"
        label="publishedTime"
        hidden
      />
      <ProFormText
        name="classification"
        label="classification"
        hidden
```

```
      />

      <ProForm.Group>
        <ProFormText
          name="name"
          label="书名"
        />

        <ProFormText
          name="isbn"
          label="ISBN"
        />
      </ProForm.Group>
      <ProForm.Group>
        <ProFormText
          name="publisher"
          label="出版社"
        />
        <ProFormText
          name="author"
          label="作者"
        />
      </ProForm.Group>
      <ProForm.Group>
        <ProFormText
          name="price"
          label="书籍价格"
        />
        <ProFormText
          name="category"
          label="分类号"
        />
      </ProForm.Group>

      <ProForm.Group>
        <ProFormSelect
          name="status"
          label="状态"
          options={[
            { label: '在架', value: '在架' },
            { label: '借出', value: '借出' },
          ]}
        />
      </ProForm.Group>

      <ProForm.Group>
        <ProFormText
          name="readerName"
          label="借阅人"
        />
        <ProFormDatePicker
          name="borrowedTime"
          label="借出日期"
          initialValue={moment()}
        />
```

```
      </ProForm.Group>

    </ModalForm >
  );
}
```

测试

id为1的书籍目前在架

 21

选择借出，并添加借阅人

 22

书籍状态变为借出，并且借阅记录表格中新增一条记录

 23

 24

2.判断借书不合理的操作

后端

BookServiceImpl.java中的LendBook()函数开头添加

```
// 校验输入数据正确性
BookLogUtils.validateBookLog(bookAndLogDTO.getBookLogDTO());
//书籍状态不是“在架”，则不能借出此书
Assert.isTrue(bookAndLogDTO.getBookDTO().getStatus()!="在架", "书籍不在架");
BookUtils.validateBook(bookAndLogDTO.getBookDTO());
```

BookLogUtils.java

```
public static void validateBookLog(BookLogDTO bookLogDTO) {
    FormatUtils.trimFieldToNull(bookLogDTO);
    Assert.hasText(bookLogDTO.getReaderName(), "借阅人不能为空");
}
```

前端

前端若不做处理，虽然后端会assert断言错误，但是前端仍然会显示“借出成功”。此时，需要将其做以下修改

```
if (props.detailData) {
    try {
        await lendBook(data);
    }
    catch (error) {
        message.success('借出成功');
    }
} else {
}
```

3.一次借多本书

前端

主页面修改之前数组的第一个元素为所有数组

```
setBookLog(records);
```

```
import { ModalForm, ProForm, ProFormInstance, ProFormSelect, ProFormText,
ProFormDatePicker, ProTable } from '@ant-design/pro-components';
import { message, Switch, Tooltip } from 'antd';
import { useEffect, useRef, useState } from 'react';
import { waitTime } from '@/utils/request';
import { lendBooks } from '@/services/api/book';
import moment from 'dayjs';
import React from 'react';

interface InputDialogProps {
  detailData?: API.BookDTO[];
  visible: boolean;
  onClose: (result: boolean) => void;
}

export default function InputDialog(props: InputDialogProps) {
  const form = useRef<ProFormInstance>(null);

  // 222222222注意两dialog中有两种类型
  // 注意不同位置注释语法不同，不要被坑了
  const [formData, setFormData] = useState<API.BookDTO[]>([]);
  console.log("formData")
  console.log(formData)

  useEffect(() => {
    waitTime().then(() => {
      if (props.detailData) {
        form?.current?.setFieldsValue(props.detailData);
        setFormData(props.detailData);
      } else {
        form?.current?.resetFields();
      }
    });
  }, [props.detailData, props.visible]);


  const onFinish = async (values: any) => {
    const { readerName,borrowedTime,returnTime } = values;
    const bookDTOArray: API.BookDTO[] = [];
    const bookLogDTOArray: API.BookLogDTO[] = [];

    /**
     * 使用forEach()方法遍历formData数组，将每个元素转换为一个包含书籍信息的DTO对象
（bookDTO），
     * 并将其添加到bookDTOArray数组中。
```

```
     * 还将每个元素的借阅日志信息转换为一个包含借阅日志的DTO对象（bookLogDTO），
     * 并将其添加到bookLogDTOArray数组中。
     */
    formData.forEach((item) => {
      //将一个包含书籍信息的DTO对象添加到bookDTOArray数组
      bookDTOArray.push({
        id: item.id,
        name: item.name,
        isbn: item.isbn,
        author: item.author,
        publisher: item.publisher,
        price: item.price,
        category: item.category,
        boughtTime: item.boughtTime,
        publishedTime: item.publishedTime,
        status: item.status,
        classification: item.classification
      });

      bookLogDTOArray.push({
        readerName,
        bookId: item.id,
        bookName: item.name,
        bookIsbn: item.isbn,
        borrowedTime:borrowedTime,
        returnTime: returnTime||null
      });
    });

    /**
     * bookDTOArray和bookLogDTOArray数组中的元素一一对应，
     * 可以通过它们的索引值将它们组合成一个新的DTO对象（bookAndLogDTO）并返回给后端。
     */
    const data: API.BookAndLogDTO[] = bookDTOArray.map((item, index) => ({
      bookDTO: item,
      bookLogDTO: bookLogDTOArray[index]
    }));

    console.log("data")
    console.log(data)
    // 1111111111setFormData(props.detailData);

    if (props.detailData) {
      try {
        await lendBooks(data);
        // setFormData([]);
      }
      catch (error) {
        message.error('借出失败');
      }
    } else {
    }
    props.onClose(true);
    return true;
  };
```

```
  return (
    <ModalForm
      width={700}
      onFinish={onFinish}
      formRef={form}
      modalProps={{
        destroyOnClose: true,
        onCancel: () => props.onClose(false),
      }}
      title={props.detailData ? '借出' : '借出'}
      open={props.visible}
    >
      <ProForm.Group >
        <ProFormText
          name={`readerName`}
          label="借阅人"
          rules={[
            {
              required: true,
              message: '请输入借阅人姓名！',
            },
          ]}
        />
        <ProFormDatePicker
          name={`borrowedTime`}
          label="借出日期"
          initialValue={moment()}
        />
      </ProForm.Group>

      <ProTable
        dataSource={formData}
        columns={[
          { title: '书名', dataIndex: 'name' },
          { title: '出版社', dataIndex: 'publisher' },
          { title: '作者', dataIndex: 'author' },
          { title: 'ISBN', dataIndex: 'isbn' },
        ]}
        options={false}
        search={false}
        cardBordered={false}
      />
    </ModalForm>
  );

}
```

后端

修改lendBooks传入参数为一个数组，通过for循环遍历一次还书

```
@Override
public void lendBooks(BookAndLogDTO[] bookAndLogDTO){
    for(int i=0;i<bookAndLogDTO.length;i++){
```

```
        BookAndLogDTO bookLogDTO = new BookAndLogDTO();
        BeanUtils.copyProperties(bookAndLogDTO[i], bookLogDTO);

        //书籍状态不是“在架”，报错
        Assert.isTrue(bookLogDTO.getBookDTO().getStatus().equals("在架"),"书籍不在
架");

        // 创建实体对象，用以保存到数据库
        BookLog log = new BookLog();

        // 将输入的字段全部复制到实体对象中
        //将两个DTO中的数据传递到log中，便于使用bookLogMapper中的insert函数
        BeanUtils.copyProperties(BookLogUtils.convertToDTO(bookLogDTO),log);

        // 调用DAO方法保存到数据库表
        bookLogMapper.insert(log);

        //修改书籍状态为借出
        bookLogDTO.getBookDTO().setStatus("借出");

        // 更新书籍状态
        updateBook(bookLogDTO.getBookDTO());
    }

}
```

还书

实现了一次还多本书

前端

主页面index

```
//定义一个一次还多本书的的回调函数
const returnBooks = async (bookLog: any) => {
    for (let i = 0; i < bookLog.length; i++) {
        returnBook(bookLog[i]);
    }
    selectRow([]);
}

//定义一个还书的回调函数
const handleReturn = async (bookLog: any) => {
    if (!selectedRowKeys?.length) return;
    openConfirm(`您确定归还${selectedRowKeys.length}本书吗`, async () => {
        await returnBooks(bookLog);
        refAction.current?.reload();
    });
};
```

```
<Button
type="primary"
key="primary"
onClick={() => {
    handleReturn(bookLog);
}}
disabled={!selectedRowKeys?.length}
>
    <LoginOutlined />  还书
</Button>,
```

```
rowSelection={{
              onChange: (rowKeys, records) => {
                  console.log("records");
                  console.log(records);
                  setBookLog(records);
                  selectRow(rowKeys as number[]);
              },
}}
```

InputDialog.tsx修改为如下

```
import { ModalForm, ProForm, ProFormInstance, ProFormSelect, ProFormText,
ProFormDatePicker, ProTable } from '@ant-design/pro-components';
import { message, Switch, Tooltip } from 'antd';
import { useEffect, useRef, useState } from 'react';
import { waitTime } from '@/utils/request';
import { lendBook, lendBooks } from '@/services/api/book';
import moment from 'dayjs';
import React from 'react';

interface InputDialogProps {
  detailData?: API.BookDTO[];
  visible: boolean;
  onClose: (result: boolean) => void;
}

export default function InputDialog(props: InputDialogProps) {
  const form = useRef<ProFormInstance>(null);
  const [readonly, setReadonly] = useState(false);

  // 222222222注意两dialog中有两种类型
  // 注意不同位置注释语法不同，不要被坑了
  const [formData, setFormData] = useState<API.BookDTO[]>([]);
  console.log("formData")
  console.log(formData)

  useEffect(() => {
    waitTime().then(() => {
      if (props.detailData) {
        form?.current?.setFieldsValue(props.detailData);
        setFormData(props.detailData);
      } else {
        form?.current?.resetFields();
```

```
      }
    });
  }, [props.detailData, props.visible]);


  const onFinish = async (values: any) => {
    const { readerName,borrowedTime,returnTime } = values;
    const bookDTOArray: API.BookDTO[] = [];
    const bookLogDTOArray: API.BookLogDTO[] = [];

    /**
     * 使用forEach()方法遍历formData数组，将每个元素转换为一个包含书籍信息的DTO对象（bookDTO），
     * 并将其添加到bookDTOArray数组中。
     * 还将每个元素的借阅日志信息转换为一个包含借阅日志的DTO对象（bookLogDTO），
     * 并将其添加到bookLogDTOArray数组中。
     */
    formData.forEach((item) => {
      //将一个包含书籍信息的DTO对象添加到bookDTOArray数组
      bookDTOArray.push({
        id: item.id,
        name: item.name,
        isbn: item.isbn,
        author: item.author,
        publisher: item.publisher,
        price: item.price,
        category: item.category,
        boughtTime: item.boughtTime,
        publishedTime: item.publishedTime,
        status: item.status,
        classification: item.classification
      });

      bookLogDTOArray.push({
        readerName,
        bookId: item.id,
        bookName: item.name,
        bookIsbn: item.isbn,
        borrowedTime:borrowedTime,
        returnTime: returnTime||null
      });
    });

    /**
     * bookDTOArray和bookLogDTOArray数组中的元素一一对应，
     * 可以通过它们的索引值将它们组合成一个新的DTO对象（bookAndLogDTO）并返回给后端。
     */
    const data: API.BookAndLogDTO[] = bookDTOArray.map((item, index) => ({
      bookDTO: item,
      bookLogDTO: bookLogDTOArray[index]
    }));

    console.log("data")
    console.log(data)
    // 1111111111setFormData(props.detailData);
```

```
  if (props.detailData) {
    try {
      await lendBooks(data);
      // setFormData([]);
    }
    catch (error) {
      message.error('借出失败');
    }
  } else {
  }
  props.onClose(true);
  return true;
};

return (
  <ModalForm
    width={700}
    onFinish={onFinish}
    formRef={form}
    modalProps={{
      destroyOnClose: true,
      onCancel: () => props.onClose(false),
    }}
    title={props.detailData ? '借出' : '借出'}
    open={props.visible}
  >
    <ProForm.Group >
      <ProFormText
        name={`readerName`}
        label="借阅人"
        rules={[
          {
            required: true,
            message: '请输入借阅人姓名！',
          },
        ]}
      />
      <ProFormDatePicker
        name={`borrowedTime`}
        label="借出日期"
        initialValue={moment()}
      />
    </ProForm.Group>

    <ProTable
      dataSource={formData}
      columns={[
        { title: '书名', dataIndex: 'name' },
        { title: '出版社', dataIndex: 'publisher' },
        { title: '作者', dataIndex: 'author' },
        { title: 'ISBN', dataIndex: 'isbn' },
      ]}
      options={false}
      search={false}
```

```
        cardBordered={false}
      />
    </ModalForm>
  );

}
```

后端

BookService.java添加

```
void lendBook(BookAndLogDTO bookAndLogDTO);
```

Impl.java

```
@Override
public void returnBook(BookDTO bookDTO){
    //只有书籍状态不是“借出”，则报错
    Assert.isTrue(bookDTO.getStatus().equals("借出"), "书籍未借出");
    //修改书籍状态为借出
    bookDTO.setStatus("在架");
    //更新书籍状态
    updateBook(bookDTO);
    //补充对应借阅日志的还书日期
    bookLogMapper.insertTime(bookDTO);
}
```

BookController.java添加

```
@PostMapping("returnBook")
@Privilege("return")
public void returnBook(@RequestBody BookDTO bookDTO) {
    bookService.returnBook(bookDTO);
}
```

BookLogMapper.xml中添加对应语句，实现自动为匹配的最新的对应书籍添加还书日期

```
  <!-- 获取对应借出书籍的日志记录 -->
  <update id="insertTime" parameterType="java.lang.Integer">
    update book_log
    set return_time = now()
    where book_id = #{id,jdbcType=INTEGER}
    order by id desc
    limit 1
  </update>
```

BookLogMapper.java中添加

```
/**
     * 更新部门数据
     *
     * @param bookDTO 部门输入对象
     * @return 部门编码
     */
Integer insertTime(BookDTO bookDTO);
```

图表展示

ts直接连接mysql表格

5.问题

1.分页查询后端xml修改错误

照着课程视频跑分页查询的时候出现了如图报错，后端刚开始就会停止执行

```
<!-- 获取名字 -->
    <select id="getByName" resultType="BaseResultMap">
        select  *
        from department
        where department_name = *{name}
    </select>
```

解决方案

- **使用reultMap 时可以用 "BaseResultMap"**

```
reultMap ="BaseResultMap"
```

- **或使用 resultType 要写类的全名（包含包名）**

```
resultType="redlib.backend.model.Book"
```

2.时间格式不正确

使用以下方法添加注解的时候，Swagger中会出现多条时间的json信息

```
@JsonFormat(pattern = "yyyy-MM-dd")
private Date BookBoutime;
```

此外，在前端生成DTO类时也发现会生成多条时间

原因

后端DTO命名不规范

解决方案

将变量命名使用驼峰结构，与Book.java同步之后解决问题（以上所有代码适当微调）

```
package redlib.backend.dto;

import com.fasterxml.jackson.annotation.JsonFormat;
import lombok.Data;

import java.math.BigDecimal;
import java.util.Date;
@Data
public class BookDTO {
    private Integer id;
    private String name;
    private String isbn;
    private String author;
    private String publisher;
    private Double price;
    private String category;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private Date boughtTime;
    @JsonFormat(pattern = "yyyy-MM-dd")
    private Date publishedTime;
}
```

3.~~selectedRowKeys删除问题~~

测试前端时发现删除某一行时会选中所有图书

原因

由于在 InputDialog\index.tsx 中修改为

```
const data: API.BookDTO = {
    bookId: props.detailData?.bookId,
};
```

解决方案

需要修改 rowKey 如下

```
actionRef={refAction}
rowKey="bookId"
```

4.价格使用Decimal无法显示

书籍价格使用Decimal类型的时候，前端新增正常，但是对某一本书修改参数时，无法显示价格

解决方案

原因未知，有两种修改方法

1.MySQL，前后端同时修改为Integer，尝试设置小数位数为2，可以解决显示问题，但无法显示小数。

2.MySQL，前后端同时修改为Double，同时设置小数位数为2，测试发现可以解决显示问题，并解决价格无法显示小数问题。

5.~~Mysql设置主键自增，删除数据后，主键id依然从删除位置增加~~

不是问题，无需解决

6.前端无法实现查询书籍

原因

修改MySQL表格时，xml文件忘记同步更改，修改后成功实现查询

解决方案

```
<!-- 获取名字个数(分页) -->
 <select id="count" resultType="integer">
   select count(*)
   from book
   <where>
     <if test="name != null">
```

```
        name like #{name}
      </if>
    </where>
  </select>

  <!-- 获取书籍(分页) -->
  <select id="list" resultMap="BaseResultMap">
    select
    <include refid="Base_Column_List"/>
    from book
    <where>
      <if test="queryDTO.name != null">
        name like #{queryDTO.name}
      </if>
    </where>
    limit #{offset}, #{limit}
  </select>
```

7.BookLog中list方法执行时报错

```
There is no getter for property named 'book_id' in 'class
redlib.backend.dto.query.BookLogQueryDTO
```

原因

MySQL中分页查询参数设置错误

```
  <!-- 获取名字个数(分页) -->
  <select id="count" resultType="integer">
    select count(*)
    from book_log
    <where>
      <if test="id != null">
        id like #{id}
      </if>
    </where>
  </select>
```

解决方案

修改为book_log列表中的id改为bookName字段

```
  <!-- 获取名字个数(分页) -->
  <select id="count" resultType="integer">
    select count(*)
    from book_log
    <where>
      <if test="bookName != null">
        book_name like #{bookName}
      </if>
    </where>
  </select>
```

8.swagger中测试删除booklog时报错

```
Invalid bound statement (not found):
redlib.backend.dao.BookLogMapper.deleteByCodes
```

原因

xml文件中忘记添加MySQL语句

解决方案

```
  <!-- 批量删除 -->
  <update id="deleteByCodes">
    delete from book
    where id in
    <foreach item="item" index="index" collection="codeList" open="("
separator="," close=")">
      #{item}
    </foreach>
  </update>
```

注释

此函数在日志记录中不太需要，后续考虑删除

9.分页查询，多种条件同时查找时没找到对象会报错

原因

sql语句中，当两个if 都不等于null的时候，两个sql语句由于没有东西连接起来，会产生报错

解决方案

在每个检索的 sql 语句前添加使用 and 连接

```
  <select id="list" resultMap="BaseResultMap">
    select
    <include refid="Base_Column_List"/>
    from book_log
```

```
    <where>
      <if test="queryDTO.readerName != null">
        and reader_name like #{queryDTO.readerName}
      </if>
      <if test="queryDTO.bookIsbn != null">
        and book_isbn like #{queryDTO.bookIsbn}
      </if>
      <if test="queryDTO.bookId != null">
        and book_id like #{queryDTO.bookId}
      </if>
      <if test="queryDTO.bookName != null">
        and book_name like #{queryDTO.bookName}
      </if>
    </where>
    <if test="queryDTO.orderBy != null">
      order by ${queryDTO.orderBy}
    </if>
    <if test="queryDTO.orderBy == null">
      order by id desc
    </if>
    limit #{offset}, #{limit}
  </select>
```

10.书籍点借阅之后确定，发现两个表格中都会有字段丢失

原因

前端代码底层为将选中行中的数据传递给对话框，然后对话框再传递给后端进行更新

解决方案

主页面向对话框中传递数据时传递全部数据，后端使用hidden属性，隐藏不需要的表格，即可实现传递数据又不显示

```
<ProFormText
name="classification"
label="classification"
hidden
/>
```

11.借书时无法向对话框传递参数

原因

一开始将typescripts列表的起始下标与MySQL记混，以为也是从1开始，代码如下

```
setBookLog(records[1]);
```

解决方案

typescripts列表的起始下标从0开始，修改records下标为0（首个元素）

```
setBookLog(records[0]);
```

12.~~多余接口删除（暂未解决，后需考虑）~~

由于BookLog后端框架为Book复制修改，许多没必要存在的接口后续考虑删除（此处就暂时未写接口文档）

13.lendBook函数报错

报错如下

```
org.springframework.validation.BeanPropertyBindingResult: 2 errors Field error
in object 'bookDTO' on field 'boughtTime': rejected value [2023-04-01]; codes
[typeMismatch.bookDTO.boughtTime,typeMismatch.boughtTime,typeMismatch.java.util.
Date,typeMismatch]; arguments
[org.springframework.context.support.DefaultMessageSourceResolvable: codes
[bookDTO.boughtTime,boughtTime]; arguments []; default message [boughtTime]];
default message [Failed to convert property value of type 'java.lang.String' to
required type 'java.util.Date' for property 'boughtTime'; Failed to convert from
type [java.lang.String] to type [@com.fasterxml.jackson.annotation.JsonFormat
java.util.Date] for value '2023-04-01'] Field error in object 'bookDTO' on field
'publishedTime': rejected value [2023-04-01]; codes
[typeMismatch.bookDTO.publishedTime,typeMismatch.publishedTime,typeMismatch.java
.util.Date,typeMismatch]; arguments
[org.springframework.context.support.DefaultMessageSourceResolvable: codes
[bookDTO.publishedTime,publishedTime]; arguments []; default message
[publishedTime]]; default message [Failed to convert property value of type
'java.lang.String' to required type 'java.util.Date' for property
'publishedTime'; Failed to convert from type [java.lang.String] to type
[@com.fasterxml.jackson.annotation.JsonFormat java.util.Date] for value '2023-
04-01']
```

原因

controller对应函数中忘记添加注解@RequestBody

解决方案

```
public void returnBook(@RequestBody BookDTO bookDTO) {
    bookService.returnBook(bookDTO);
}
```

14.借出、归还Assert仍然不符合逻辑

借出的书本，仍然可以借出。在在架的书本，仍然可以归还。

> 点还书按钮的时候，前端把书籍信息传递给后端，后端接收后首先用Assert判断书籍状态是否符合逻辑，然后修改为书籍状态为“在架”。但是我尝试发现不管怎么修改Assert语法（尝试过==和！=等），但发现都达不到想要的效果

原因

java比较字符串相等，不能用==

```
Assert.isTrue(bookDTO.getStatus()!="借出", "书籍在架！");
```

解决方案

```
//只有书籍状态不是“借出”，则报错
//即，前表达式不成立时报错
Assert.isTrue(bookDTO.getStatus().equals("借出"), "书籍在架");
```

15.借出成功后没有提示

原因

由于后端返回值为void，await lendBooks(data)的返回值可能一直为undefined，导致无法判断

可能原因

根据提供的代码，似乎 `try-catch` 块只捕获在 `await lendBooks(data)` 调用期间发生的错误。任何在后端代码中发生的错误，例如 `Assert` 错误，都不会被这个 `try-catch` 块捕获。

如果后端代码发生错误导致请求失败，那么 `await lendBooks(data)` 调用可能会拒绝并抛出一个错误，导致 `catch` 块执行并显示 `'借出失败'` 消息。但是，如果后端代码返回成功的响应，但是包含一个错误消息表示操作失败，那么 `await lendBooks(data)` 调用仍可能成功解析，并显示 `'借出成功'` 消息。

为确保后端代码中的错误得到正确处理，重要的是在后端代码中也包括错误处理，并确保它返回适当的响应以指示成功或失败。此外，前端代码应更新以正确处理这些响应，并向用户显示适当的消息。

后端使用assert即使success为false前端也会显示正确，即无论如何都会执行到，都会弹出此框

```
try {
    await lendBooks(data);
    message.success('借出成功');
} catch (error) {
    message.error('借出失败');
}
```

解决方案

1.第一种解决方案（实际采用此种方案实现显示的效果，但不规范）

前端通过遍历借还多本书，并且判断书籍状态（即通过前端模拟后端Assert功能）

借书

```
if (props.detailData) {
    let sum = 0;
    for (let i = 0; i < data.length; i++) {
        if(data[i].bookDTO?.status==="在架"){
            await lendBooks(data[i]);
        }
        else{
            sum = sum+1;
        }
    }
    if(sum!==0){
        message.error(`共${sum}本书借出失败`);
    }else{
        message.success(`借出成功`);
    }
} else {
}
```

还书

```
  //定义一个一次还多本书的的回调函数
  const returnBooks = async (bookLog: any) => {
    let sum = 0;
    for (let i = 0; i < bookLog.length; i++) {
      if(bookLog[i].status ==="借出"){
        await returnBook(bookLog[i]);
      }
      else{
        sum = sum+1;
      }
    }

    if(sum!==0){
      message.error(`共${sum}本书归还失败,${bookLog.length-sum}本书归还成功`);
    }else{
      message.success(`归还成功`);
    }
    selectRow([]);
  }

  //定义一个还书的回调函数
  const handleReturn = async (bookLog: any) => {
    if (!selectedRowKeys?.length) return;
    openConfirm(`您确定归还${selectedRowKeys.length}本书吗`, async () => {
      await returnBooks(bookLog);
      refAction.current?.reload();
    });
```

```
  };
```

2.第二种解决方案（更加推荐，规范，但不知道怎么实现显示效果）

```
try{
    if(props.detailsData){
        await updateDepartment(date,{throwError:true})
    }else{
        pass
    }catch(ex){
        return true;//表示报错不执行任何操作
    }
}

//以下为不报错时顺序执行的代码
```

16.对话框传递参数不同步的问题

借出，还书的对话框显示状态与主页面相对有延时，两者不同步，如何修改?

即选中一栏之后，首次借出正常，若不取消勾选，第一次成功之后仍然可以成功借书（状态并未更新到借书状态框中），只有取消勾选，再次勾选，才能够在借书对话框中刷新状态。

原因

初步判断为rowselect中的列表参数首个元素添加参数未改变的问题

 27

测试发现，书籍状态在选择行时向对话框传递过一次，但是在借出完书之后，选择行并未改变，并未重新写入，此时records中保存的仍然是修改之前的消息

解决方案

通过在借书、还书成功之后添加，取消勾选

```
selectRow([]);
```

17.还多本书时，log页面显示参数缺失

原因

一开始以为是后端copy数据出现问题，通过F12查看发现是前端向后端传递的数据缺失

解决方案

values中添加borrowedTime，returnTime变量，bookDTOArray中添加id: item.id，修改bookLogDTOArray中的最后两行为

```
borrowedTime:borrowedTime,
returnTime: returnTime||null
```

```
const { readerName,borrowedTime,returnTime } = values;

formData.forEach((item) => {
    //将一个包含书籍信息的DTO对象添加到bookDTOArray数组
    bookDTOArray.push({
        id: item.id,
        name: item.name,
        isbn: item.isbn,
        author: item.author,
        publisher: item.publisher,
        price: item.price,
        category: item.category,
        boughtTime: item.boughtTime,
        publishedTime: item.publishedTime,
        status: item.status,
        classification: item.classification
    });

    bookLogDTOArray.push({
        readerName,
        bookId: item.id,
        bookName: item.name,
        bookIsbn: item.isbn,
        borrowedTime:borrowedTime,
        returnTime: returnTime||null
    });
});
```

18.无法将主页面向对话框传递的数据暂存给formData

原因

setFormData函数调用位置错误

之前在data下方调用此函数，网页控制台查看发现其一直为空

解决方案

在钩子函数中调用setFormData(props.detailData);

```
useEffect(() => {
    waitTime().then(() => {
        if (props.detailData) {
            form?.current?.setFieldsValue(props.detailData);
            setFormData(props.detailData);
        } else {
            form?.current?.resetFields();
        }
    });
}, [props.detailData, props.visible]);
```

19.图像显示不正常

bug21修改成功后此处也解决

20.Navicat与后端执行MySQL语句结果不一致

原因

1.resultMap与resultType用错

2.语句放错xml文件导致BaseResultMap不同无法对应

解决方案

1.修改resultMap为BaseResultMap

BaseResultMap将取得的数据库字段与VO类做映射

2.将mysql语句从移动到借还记录的xml文件中

21.时间乱序

原因

MySQL表格中测试数据忘记删除，使用数组的contact函数导致乱序

其余可能原因

> 归还时间为null的问题
> contact的问题（可能性不大）

解决方案

删除多余测试数据

22.ts中的引号

以下为正确用法

```
message.error(`共${sum}本书借出失败`);
```

以下为错误用法

```
message.error('共${sum}本书借出失败');
```

23. setXXX异步更新问题

> 它会将新的状态值加入到一个队列中，并在稍后的时间点异步地批量更新组件的状态。这意味着，调用 setXXX 并不会立即更新组件的状态值，而是等待 React 系统在适当的时机进行批量>更新。

解决方案

使用useEffect钩子函数，在检测到变量被，更新后再设置innerHTML

```
  useEffect(() => {
    async function fetchJinrishici() {
      const jinrishici = require('jinrishici');
      await jinrishici.load(result => {
        console.log("诗词内容");
        console.log(result.data.content);
        setPoem(result.data.content);
        console.log("使用setPoem之后,poem变量");
        console.log(poem);
        setInfo("--" + result.data.origin.author + "《" +
result.data.origin.title + "》");
      });

    }

    fetchJinrishici();
  }, []);

  useEffect(() => {
    if (document.getElementById("poem")) {
      console.log("1123")
      document.getElementById("poem").innerHTML = poem;
    }
    if (document.getElementById("info")) {
      document.getElementById("info").innerHTML = info;
    }
  }, [poem, info]);
```

24.注释问题

typescripts不同位置注释格式不同的问题，使用ctrl+？能更方便的注释，避免错误