中南大学

程序设计训练

实验报告

物联网 1802 白明强 王云鹏

2020-7-21

基于 Spring Boot 与 Vue 的 Kindle 图书推送系统

一、问题场景

Kindle 是 Amazon 公司推出的、目前市占率最大的电子书阅读器，以其方便、灵敏的操作和类似纸质书籍的阅读体验而受到广大消费者的广泛欢迎。为了满足消费者对自己的个人文档、其他地方购买的电子书籍的阅读需求，Amazon 提供了这样的功能：每台 Kindle 设备可以绑定一个 Amazon 账户，此 Amazon 账户具有一个个人邮箱地址，当以附件的形式向此邮箱发送电子书籍文件时，如果发件人的邮箱地址在该账户设置的白名单中，Kindle 设备在下次联网的时候就会自动下载此附件并添加到此设备的书架中，由此我们可以开发一个电子书商城或者说推送系统，用户只需注册时填好 Amazon 邮箱地址，然后便可以浏览系统数据库中存在的书籍，遇到喜欢的书可以点击推送按钮，将电子书传递到 Kindle 设备，方便了用户。

二、程序大纲

1. 简介

系统采用 B-S 模式，由 Java 语言开发后台服务程序，使用 Spring Boot 框架进行接口数据的处理和返回，使用 Mybatis 框架读取后台 Mysql 数据库，使用保存在 Redis 内存数据库中 Cookie 来验证登录状态。前端采用 Vue 进行响应式处理，采用 axios 库对后端的 API 接口进行异步 ajax 请求，采用 element-UI 进行界面的美化。

2. 服务端 API 设计

服务端只负责提供信息查询接口，全部采用 json 格式，不进行任何的页面渲染，html 的数据填充和渲染由前端完成。下面是推送系统的 API 设计：

/api/book: POST

接口	功能	参数	返回值

/search	负责进行数据库的查询和返回。	bookname 字段表示按书名模糊查找
		author 字段表示按作者模糊查找
		两者同时存在时，按与关系进行
			当结果含多项时，返回一个 JSON 列表。
			[
			{
			"bookid": 3,
			"bookname": "aaa",
			"author": "bbb",
			"description": "ccc",
			"covername": "ddd",
			"date": "eee",
			"path": ""
			}
			]

/item	负责进行某一本书的查找	bookid 书籍的数据库编号	同上格式的 JSON，单一项
/send	负责进	bookid 书籍	成功： 状态码 200

	行邮件的推送	的数据库编号
		另外还需要浏览器 cookie 中的 token 字段验证通过，表示用户已经登录，否则报错，拒绝发送
			{
			"statusCode": 200,
			"info": "发送成功，请查收您的 Kindle 个人资料库！"
			}
			失败：状态码 400
			JSON 格式同上，只是 info 的具体信息发生变化

/api/user: POST

接口	功能	参数	返回值

/login	负责进行用户的登录验证。	username:用户名
		userpwd:用户密码
			{
			"statusCode": 200,
			"info": "登录成功！"
			}

/checkLogin	负责验证每次网页请求时用户是否登录	无 JSON 需求，要求 Cookie 中的 token 字段	无 JSON 返回，以状态码区分，token 通过为 200，不通过为 400

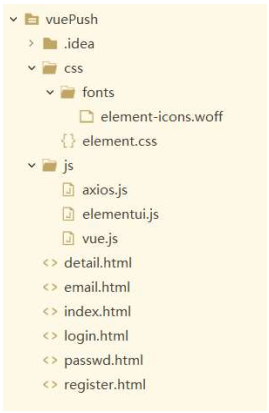
/email	负责更改用户注册邮箱地址	email：新邮箱地址
		以及 Cookie 中的 token 字段
			成功： 状态码 200
			{
			"statusCode": 200,
			"info": "注册邮箱设置成功！"
			}
			失败：状态码 400
			JSON 格式同上，只是 info 的具体信息发生变化

/logout	负责清空登录状态	无 JSON 需求，要求 Cookie 中的 token 字段	空返回体，状态为 200 OK

/passwd	负责修改登录密码	oldpasswd 旧密码
		newpasswd 新密码
		capcha 验证码
		以及 Cookie 中的 token 字段
			成功时：
			{
			"statusCode": 200,
			"info": "新密码设置成功！"
			}

/checklogin	负责返回登录状态和用户的信息	Cookie 中的 token 字段	查找到用户则返回用户信息，否则返回 400 错误

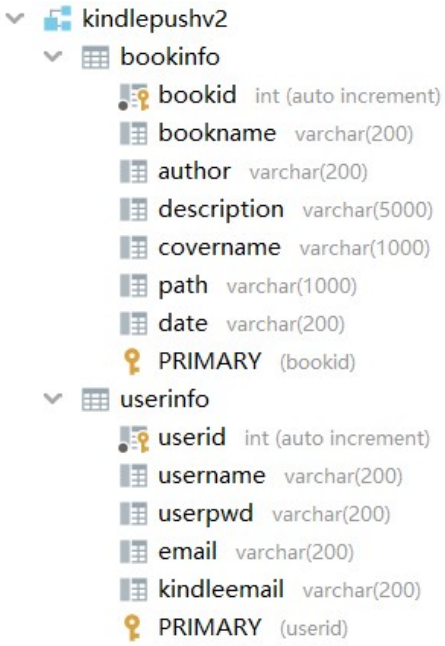
3. 前端页面设计



index.html：首页
detail.html：详细信息页
login.html：登录页
passwd.html：修改密码页
register.html：注册页
email.html：修改邮箱页

js 目录下为引入的 js 库文件
css 目录下为 elementUI 的 CSS(层叠样式表)文件和图标文件

4. 数据库结构设计



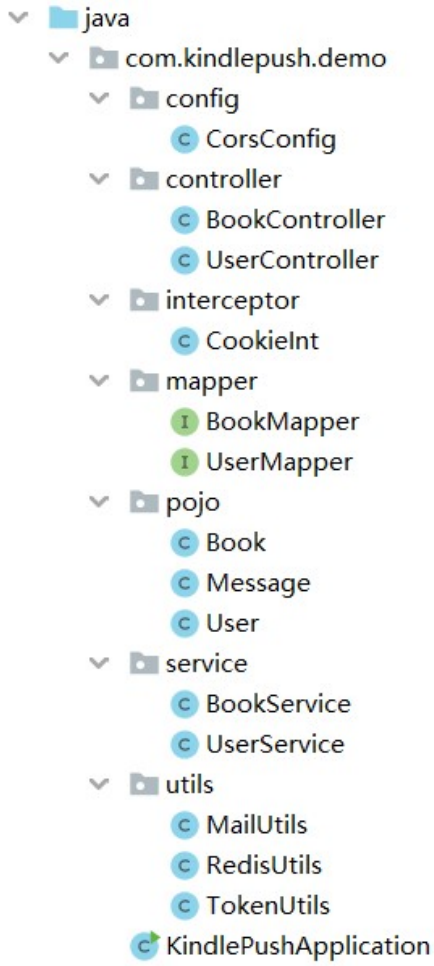
表结构如左图所示，SQL 代码如下：

```
drop database if exists KindlePushv2;
create database KindlePushv2;
drop table if exists bookinfo;
create table bookinfo(
                        bookid int
auto_increment primary key,
                        bookname varchar(200),
                        author varchar(200),
                        description
varchar(5000),
                        covername
varchar(1000),
                        path varchar(1000),
                        date varchar(200)
);

drop table if exists userinfo;
create table userinfo(
                        userid int
auto_increment primary key,
                        username varchar(200),
                        userpwd varchar(200),
                        email varchar(200),
                        kindle_email
varchar(200)
);
```

三、程序实现

1. 服务端：

包结构如图所示，下面结合代码介绍每个文件的作用。



1.1 POJO 包

1.1.1 Book.java

提供对数据库 bookinfo 表的映射，@TableName 注解提示 mybatis-plus 要到数据库哪个表去建立映射。@TableId 注解提示此变量为主键。

```
package com.kindlepush.demo.pojo;

import com.baomidou.mybatisplus.annotation.IdType;
import com.baomidou.mybatisplus.annotation.TableId;
import com.baomidou.mybatisplus.annotation.TableName;

@TableName("bookinfo")
public class Book {
```

```
    @TableId(type= IdType.AUTO)
    private Integer bookid;

    private String bookname;
    private String author;
    private String description;
    private String covername;
    private String date;
    private String path;

        //getter、setter、toString 方法省略
    }
```

1.1.2 User.java

结构同 Book.java，对应数据库中的 userinfo 表。字段如下：

```
private Long userid;
private String username;
private String userpwd;
private String email;
private String kindleemail;
```

1.1.3 Message.java

API 返回信息的对象封装，不对应数据库表。字段如下：

```
int statusCode;
String info;
```

1.2 Mapper 包

1.2.1 BookMapper.java

DAO（数据操作层）的封装，正常情况下此处编写对数据库的具体操作函数，我们使用的 mybatis-plus 插件已经封装好常用的数据操作选项，我们的项目没有复杂的数据操作，封装的函数已经满足需要，所以我们在这里直接继承 BaseMapper 接口即可。

添加@Repository 注解，Spring 会帮我们自动生成 BookMapper 类对象并添加到 Spring 的 IOC 容器，然后在 Service 层中我们需要的时候注入即可。

```
package com.kindlepush.demo.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.kindlepush.demo.pojo.Book;
import org.springframework.stereotype.Repository;

@Repository
public interface BookMapper extends BaseMapper<Book> {

}
```

1.2.2 UserMapper.java

同 BookMapper.java，继承 BaseMapper 对象，由插件自动生成 UserMapper 对象。

```
package com.kindlepush.demo.mapper;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.kindlepush.demo.pojo.User;
import org.springframework.stereotype.Repository;

@Repository
public interface UserMapper extends BaseMapper<User> {

}
```

1.3 Service 包

1.3.1 BookService.java

服务层一般存放业务逻辑处理，也是一些关于数据库处理的操作，但不是直接和数据库打交道，具体的数据库操作由 mapper 完成。这里，类似 Mapper 中的操作，我们没有额外的数据处理需求，所以也可以使用插件提供的 service 通用封装，继承 ServiceImpl 类，并提供 mapper 和 pojo 的泛型即可。

这里使用@Service 注解，交给 Spring 进行管理。

```
package com.kindlepush.demo.service;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.kindlepush.demo.mapper.BookMapper;
import com.kindlepush.demo.pojo.Book;
import org.springframework.stereotype.Service;

@Service
public class BookService extends ServiceImpl<BookMapper,Book> {

}
```

1.3.2 UserService.java

与 BookService.java 进行同样的处理即可。

```
package com.kindlepush.demo.service;

import com.baomidou.mybatisplus.extension.service.impl.ServiceImpl;
import com.kindlepush.demo.mapper.UserMapper;
import com.kindlepush.demo.pojo.User;
import org.springframework.stereotype.Service;

@Service
public class UserService extends ServiceImpl<UserMapper,User> {
}
```

1.4 Controller 包

Controller（控制器）层编写直接的业务逻辑，通过接收前端传过来的参数进行业务操

作。包下的两个文件比较长，我们按函数来解释。

1.4.1 BookController.java

类定义和类成员如下所示，@Controller 注解表明这是一个控制器，Spring 会直接调用它来处理业务。类前面的@RequestMapper 注解设置访问路径的前缀。

类成员有四个，BookService 对象和 UserService 对象用来对数据库进行操作，MailUtils 是随后介绍的自定义的工具类，负责发送邮件，这三个类成员采取@Autowired 注解由 Spring 进行注入。Jedis 对象由自定义工具类 RedisUtils 中的静态方法获取，负责对 redis 内存数据库进行操作。

```
@Controller
@RequestMapping("/api/book")
public class BookController {

    @Autowired
    private BookService bookService;
    @Autowired
    private UserService userService;
    @Autowired
    public MailUtils mailUtils;
    public Jedis jedis = RedisUtils.initPool().getResource();

    //以下的方法暂时省略
}
```

下面是 search 接口的定义，其中方法前的@PostMapping 注解设置接口路径，@RequestBody 注解表示此路径接收 JSON，并将键值对保存在 Map 类型的 searchBy 对象中。

在前端未搜索，直接访问 index.html 时，也会访问这个接口获取列表，这时是没有 JSON 的，所以要先判断一下 searchBy 是否为 null，是的话要进行初始化，不判断的话在进行 get 操作时会报空指针异常。然后根据获取到的 bookname 和 author 信息进行对应的数据库查找获取一个 List<Book>对象。要注意 book 里面的 path 是敏感信息，需要在返回前清空它。设置 path 字段是用于 send 接口中查找本地图书文件的位置并发送。

返回对象是 Spring MVC 提供的 ResponseEntity 类型，它接收一个泛型参数，作用是将泛型类序列化为 JSON 返回给前端。

```
@PostMapping("/search")
public ResponseEntity<List<Book>> search(@RequestBody(required = false)
Map<String,String> searchby){
```

```
    //初始化 如果未传参数 则默认随机排序
    List<Book> books=bookService.list();

    //获取数据,前台无参数POST 时，searchby 是null，所以必须校验
    if(searchby==null){
        searchby = new HashMap<>();
    }
    String bookname = searchby.get("bookname");
    String author = searchby.get("author");


    //有书名没作者
    if(bookname!=null&&author==null) {
        System.out.println(searchby);
        books = bookService.list(Wrappers.<Book>query().like("bookname",
searchby.get("bookname")));
    }
    //有作者没书名
    if(bookname==null&&author!=null) {
        System.out.println(searchby);
        books =
bookService.list(Wrappers.<Book>query().like("author",searchby.get("author"
)));
    }
    //有作者也有书名
    if(searchby.get("bookname")!=null&&searchby.get("author")!=null) {
        System.out.println(searchby);
         books = bookService.list(Wrappers.<Book>query().like("bookname",
searchby.get("bookname"))
         .like("author",searchby.get("author")));
    }
    //清除路径信息
    books.forEach(x->x.setPath(""));

    return ResponseEntity.ok(books);
}
```

然后是 item 接口，它按照请求 JSON 中的 ID 查找书籍并返回。异常时返回 400 状态码。

```
//点进详情页面时 展示书籍具体信息
@PostMapping("/item")
public ResponseEntity<Book> findById(@RequestBody Map<String,Integer>
idMap){
```

```
    Integer bookid = idMap.get("bookid");
    Book book = bookService.getOne(Wrappers.<Book>query().eq("bookid",
bookid));
    if(book!=null) {
        return ResponseEntity.ok(book);
    }else{
        return ResponseEntity.status(400).build();
    }
}
```

最后是 send 接口，它只需要 JSON 有一个数字的 bookid，所以 Map 的泛型为 String 到 Integer，首先用 jedis 对象获取内存数据库中的 token 字段（在后面将要解释的 login 函数中，会将发给用户的 token 和 token 对应的 username 以 username:token 和 token:username 的形式存入 redis），如果当前前端发来的 token 在 redis 中可以获取到 username，则用户已登录。然后通过 bookid 获取 resultBook，找不到则返回 400。若通过 token 找不到 username 也返回 400。找到了则获取 kindle 邮箱，读取本地书籍文件，用 MailUtils 发送到这个 Kindle 邮箱。

```
@PostMapping("/send")
public ResponseEntity<Message> sendBook(@RequestBody Map<String,Integer>
info, @CookieValue(name = "token",required = false,defaultValue = "")String
token){

    Book resultBook;
    String path;
    String kindleemail;
    String username =  jedis.get(token);

    try{
        resultBook = bookService.getById(info.get("bookid"));
        path = resultBook.getPath();
    }catch(Exception e){
        return ResponseEntity.status(400).body(
                new Message(400,"未找到此书籍！")
        );
    }


    if(username==null){
        return ResponseEntity.status(400).body(
                new Message(400,"登录已过期，请重新登录！")
        );
    }else{
        //用户已经登录 并且得到了用户名  日后可在此处写扣费等逻辑
```

```
        User user = userService.getOne(Wrappers.<User>query().eq("username",
username));
        kindleemail = user.getKindleemail();
    }


    Boolean sendFileSuccess = mailUtils.sendMailWithAttachment(path,
kindleemail);
    if(sendFileSuccess){
        return ResponseEntity.status(200).body(
                new Message(200,"发送成功，请查收您的 Kindle 个人资料库！")
        );
    }else{
        return ResponseEntity.status(400).body(
                new Message(400,"发送失败，请稍后重试！")
        );
    }
}
```

1.3.2 UserService 包

UserService 包用来处理和用户相关的请求。如注册、改密、登出等。

类定义和类成员如下，设定路径前缀为/api/user，需要 UserService 对象操作数据库、MailUtils 对象发送注册验证码、Jedis 对象操作 Redis。

```
@Controller
@RequestMapping("/api/user")
public class UserController {

    @Autowired
    public UserService userService;
    @Autowired
    public MailUtils mailUtils;
    public Jedis jedis = RedisUtils.initPool().getResource();
    //方法省略
}
```

login 登录方法，@RequestBody 注解可以接受一个 POJO 对象，Spring MVC 会将 JSON 中对应的字段值映射到该 POJO 对象中，效率要比 Map 高。方法还接收一个 `HttpServletResponse` 对象，它是 Servlet 技术中最基本的返回对象，会由 Spring MVC 传入，这里用它来设置 Cookie。

方法首先判断 username 字段是否为空，空则返回 400 错误。然后试图在数据库中查找对应的 User，找不到则返回 400 并说明用户不存在。存在则判断输入的密码和数据库中保存的密码是否相同，相同则登录成功。然后调用 TokenUtils 类的方法生成一个随机字符串 token 设置为 Cookie，并将 username->token 和 token->username 两个键值对保存到 redis。

这里另外还需要一个 Cookie 字段 SameSite 设置为 None，是与浏览器安全策略有关的一个字段，当发生跨域时，Chrome 浏览器的内核要求此字段为 None，否则浏览器拒绝设置 Cookie。

```
    @PostMapping("/login")
    @ResponseBody
    public  ResponseEntity<Message> login(@RequestBody User user,
HttpServletResponse response){

        if(user.getUsername()==null){
            return ResponseEntity.status(400).body(
                    new Message(400,"请输入用户名！")
            );
        }
        User dbUser =
userService.getOne(Wrappers.<User>query().eq("username",
user.getUsername()));

        if(dbUser==null){
            return ResponseEntity.status(404).body(
                    new Message(404,"用户名不存在！")
            );
        }

        if(dbUser.getUserpwd().equals(user.getUserpwd())){

            //保存临时 token
            //设置超时时间
            SetParams setParams = new SetParams();
            setParams.ex(7200);

            String token = TokenUtils.generate(24);

            jedis.set(user.getUsername(), token,setParams);
//            //也将 value 作为 key，便于通过 token 反查用户名
            jedis.set(token,user.getUsername(),setParams);

            Cookie cookie = new Cookie("token",token);
            cookie.setMaxAge(7200);
            cookie.setDomain("www.leyou.com");
            cookie.setPath("/");
```

```
            Cookie cookie1 = new Cookie("SameSite","None");
            cookie1.setDomain("www.leyou.com");
            cookie1.setPath("/");

            response.addCookie(cookie);
            response.addCookie(cookie1);

            return ResponseEntity.status(200).body(
                  new Message(200,"登录成功！")
            );
        }
        return ResponseEntity.status(400).body(
                new Message(400,"密码错误，请重新输入！")
        );
    }
```

checkLogin 方法，使用用户 Cookie 中的 token 查找 redis，判断用户是否登录，并且使前端获取到登录用户的各种信息。此方法存在主要是为了控制前端某些元素的显示，如“登录”按钮只能在未登录的状态下存在。

```
@PostMapping("/checklogin")
@ResponseBody
public  ResponseEntity<User> checklogin(@CookieValue(value =
"token",required = false,defaultValue = " ")String token){

    String username = jedis.get(token);
    User user;
    if(username!=null){
        user =
userService.getOne(Wrappers.<User>query().eq("username",username));
        return ResponseEntity.ok(user);
    }else{
        return ResponseEntity.status(400).build();
    }
}
```

email 方法，用来更改用户注册邮箱。方法通过 jedis 获取 redis 中的 username，无则置空字符串。然后用获取到的 username 查找 mysql，当 username 为空串时返回的 User 对象必定为 null，这时方法返回 400 未登录。若字段 email 为空，也返回 400 提示。至此 User 和 email 都已经获取，利用 userService 进行更新即可。

```
@PostMapping("/email")
public ResponseEntity<Message> setEmail(@RequestBody Map<String,String>
```

```
info,@CookieValue(name = "token",required = false,defaultValue = "")String
token){

    String redisUsername = jedis.get(token);
    String username = (redisUsername==null? redisUsername:"");

    User user =
userService.getOne(Wrappers.<User>query().eq("username",username));

    if(user==null){
        return ResponseEntity.badRequest().body(
                new Message(400,"未登录！")
        );
    }else if(info.get("email")==null){
        return ResponseEntity.badRequest().body(
                new Message(400,"请输入要更改的新注册邮箱地址！")
        );
    }
    //user 和 email 地址都获取成功
    user.setEmail(info.get("email"));
    boolean success = userService.updateById(user);
    if(success){
        return ResponseEntity.ok().body(
                new Message(200,"注册邮箱设置成功！")
        );
    }else{
        return ResponseEntity.status(500).body(
                new Message(500,"注册邮箱设置失败，请稍后重试。")
        );
    }
}
```

---

logout 方法，接收 token，负责将 redis 中的两个相关键值对删除，即清除了登录状态，总是成功的，返回 200 OK。

```
@PostMapping("logout")
public ResponseEntity<Message> logout(@CookieValue(name = "token",required
= false,defaultValue = "")String token){
    String username = jedis.get(token);
    if(username!=null){
```

```
        jedis.del(username);
        jedis.del(token);
    }
    return ResponseEntity.ok().build();
}
```

setPasswd 方法，需要旧密码、新密码、token、验证码，如果第一个 try 块中的 requireNonNull 不通过，则返回 400 错误要求完整输入。通过完整性检查则使用 token 获取用户名，然后从 mysql 读取 User 对象，验证验证码是否填写正确，正确则继续验证旧密码是否填写正确，有一个出错则返回错误，都通过则修改密码并返回成功。

```
  @PostMapping("/passwd")
public ResponseEntity<Message> setPasswd(@RequestBody Map<String,String>
info, @CookieValue(name = "token",required = false,defaultValue = "")String
token){
    //发过来的token 一定是对的
    String username,oldPasswd,newPasswd,capcha;

    try {
        oldPasswd = Objects.requireNonNull(info.get("oldpasswd"));
        newPasswd = Objects.requireNonNull(info.get("newpasswd"));
        capcha = Objects.requireNonNull(info.get("capcha"));
    }catch(Exception e){
        return ResponseEntity.badRequest().body(
                new Message(400,"请输入完整的旧密码和新密码！")
        );
    }
    username = jedis.get(token); //当前用户真实的username

    if(!capcha.equals(jedis.get(username+"_capcha"))){
        return ResponseEntity.badRequest().body(
                new Message(400,"验证码有误！")
        );
    }
    User trueUser =
userService.getOne(Wrappers.<User>query().eq("username",username));
    String trueOldPasswd =trueUser.getUserpwd();
    if(!oldPasswd.equals(trueOldPasswd)){
        return ResponseEntity.badRequest().body(
                new Message(400,"旧密码有误！")
        );
    }else{
        trueUser.setUserpwd(newPasswd);
        boolean success=false;
```

```
        try {
             success = userService.updateById(trueUser);
        }catch(Exception e){
            e.printStackTrace();
        }
        if(success){
            return ResponseEntity.ok().body(
                    new Message(200,"新密码设置成功！")
            );
        }else{
            return ResponseEntity.status(500).body(
                    new Message(500,"新密码设置失败，请稍后重试。")
            );
        }
    }
}
```

getCapcha 方法，用于生成验证码，并调用 MailUtils 向用户的注册邮箱发送。它接收用户名、邮箱地址两个参数和 token 这个 Cookie。先验证两个字段是否为空，空则返回 400 错误，通过则随机生成验证码，发送给用户邮箱，并将验证码存入 redis，等候其他 API 验证。

```
@PostMapping("/getCapcha")
public ResponseEntity<Message> getCapcha(@RequestBody Map<String,String>
info,@CookieValue(name = "token",required = false,defaultValue = "")String
token){
    String username = info.get("username");
    String email = info.get("email");
    if(username==null||email==null){
        return ResponseEntity.badRequest().body(
                new Message(400,"请输入接收验证码的邮箱！")
        );
    }
    String subject = "来自 KindlePush 的验证码,五分钟内有效";
    String capcha = TokenUtils.generate(6);
    Boolean sendSuccess = mailUtils.sendMail(subject, capcha, email);
    if(sendSuccess){
        //存入 redis
        SetParams params = new SetParams();
        params.ex(600);
        jedis.set(username+"_capcha",capcha,params);
        return ResponseEntity.ok().body(
                new Message(200,"发送成功！")
        );
```

```
    }else{
        return ResponseEntity.status(500).body(
                new Message(500,"发送失败，请稍后重试。")
        );
    }
}
```

register 方法，用于注册新用户。接收用户名、密码、注册邮箱、Kindle 邮箱等必要的参数，进行完整性检验，然后检查验证码，均验证通过则进行数据库增加一条用户的操作并返回 200OK。否则按照具体的情况返回错误信息。

```
@PostMapping("/register")
public ResponseEntity<Message> register(@RequestBody Map<String,String>
info){
    String username;
    String userpwd;
    String email;
    String capcha;
    String kindleemail;
    try {
        //info 里要有 username userpwd email capcha
        username = Objects.requireNonNull(info.get("username"));
        userpwd = Objects.requireNonNull(info.get("userpwd"));
        email = Objects.requireNonNull(info.get("email"));
        capcha = Objects.requireNonNull(info.get("capcha"));
        kindleemail = Objects.requireNonNull(info.get("kindleemail"));
    }catch(NullPointerException e){
        return ResponseEntity.badRequest().body(
                new Message(400,"输入信息有误，请重新登录！"));
    }
    //验证码通过
    if(jedis.get(username+"_capcha").equals(capcha)){
        boolean save = userService.save(
                new User(null, username, userpwd, email,kindleemail)
        );
        if(save){
            //使验证码失效
            jedis.del(username+"_capcha");
            return ResponseEntity.ok().body(
                    new Message(200,"注册成功！")
            );
        }else{
            return ResponseEntity.status(500).body(
```

```
                    new Message(500,"注册失败，请稍后重试。")
            );
        }
    }else{
        return ResponseEntity.status(500).body(
                new Message(500,"验证码有误！")
        );
    }
}
```

1.5 Utils 包

此包下主要是一些工具类。

1.5.1 MailUtils.java

用于发送邮件的工具类。使用了 javamail 的一些 API。需要手动设置发件邮箱的 SMTP 信息等等。

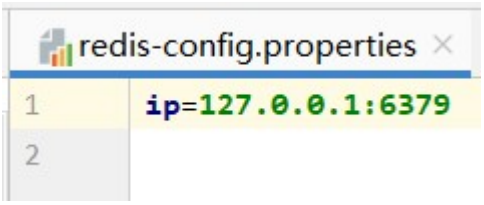
```
package com.kindlepush.demo.utils;

import org.springframework.core.io.FileSystemResource;
import org.springframework.mail.SimpleMailMessage;
import org.springframework.mail.javamail.JavaMailSenderImpl;
import org.springframework.mail.javamail.MimeMessageHelper;
import org.springframework.stereotype.Component;

import javax.mail.MessagingException;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeUtility;
import java.io.UnsupportedEncodingException;

@Component
public class MailUtils {

    private JavaMailSenderImpl jms;

    public MailUtils() {
```

```
        jms = new JavaMailSenderImpl();
        jms.setHost("smtp.qq.com");
        jms.setProtocol("smtp");
        jms.setUsername("535834197@qq.com");
        jms.setPassword("qovjkvpolnexbgfd");
        jms.setDefaultEncoding("UTF-8");
    }

    public Boolean sendMailWithAttachment(String path, String receiver){
        MimeMessage message = jms.createMimeMessage();
        try {

            MimeMessageHelper helper = new
MimeMessageHelper(message,true,"utf-8");
            helper.setFrom("535834197@qq.com");
            helper.setTo(receiver);
            helper.setSubject("书籍发送");
            helper.setText("这是您订阅的书籍");

            FileSystemResource file = new FileSystemResource(path);

            String filename =
path.substring(path.lastIndexOf('\\')).substring(1);

helper.addAttachment(MimeUtility.encodeWord(file.getFilename()),file);
            System.out.println(file.getFilename());
            jms.send(message);
            return Boolean.TRUE;//success
        } catch (MessagingException | UnsupportedEncodingException e) {
            e.printStackTrace();
            return Boolean.FALSE;//fail
        }
    }

    public Boolean sendMail(String subject,String text,String receiver){
        SimpleMailMessage message = new SimpleMailMessage();
        message.setFrom("535834197@qq.com");
        message.setTo(receiver);
        message.setSubject(subject);
        message.setText(text);
        try {
            jms.send(message);
            return Boolean.TRUE;
        }catch(Exception e){
```

```
            e.printStackTrace();
            return Boolean.FALSE;
        }
    }
}
```

1.5.2 RedisUtils.java

包含一个静态方法 initPool，返回 redis 的一个连接池。连接验证数据从 classpath 下的 **redis-config.properties** 文件读取。



此文件只有一条信息。

```
package com.kindlepush.demo.utils;

import redis.clients.jedis.JedisPool;

import java.io.IOException;
import java.io.InputStream;
import java.util.Properties;

public class RedisUtils {


    public  static JedisPool initPool() throws IOException {
        InputStream inputStream =
RedisUtils.class.getResourceAsStream("/redis-config.properties");
        Properties properties = new Properties();
        properties.load(inputStream);
        String[] address = properties.getProperty("ip").split(":");
        JedisPool pool = new JedisPool(address[0]);
        return pool;
    }
}
```

1.5.3 TokenUtils.java

包含一个生成任意位数随机字符串的 generate 方法。

```
public class TokenUtils {

    public static String generate(int length){
        String
str="abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
        Random random=new Random();
        StringBuffer sb=new StringBuffer();
        for(int i=0;i<length;i++){
            int number=random.nextInt(62);
            sb.append(str.charAt(number));
        }
        return sb.toString();
    }
}
```

1.6 Config 包

包含一个用于对 Spring MVC 进行设置的 Configuration 类。实现 WebMvcConfigurer 类可以达到修改 Spring Boot 某些默认配置的功能。这里覆盖了 CorsConfig 方法，是为了解决前端页面的域名和 API 请求的域名不一致所产生的跨域问题。在开发的过程中我用了两台机器，通过改 HOSTS 文件的方式使得前端和后端不在同一个域名上。实际部署后，可以通过 tomcat 的某些设置将静态前端文件和动态 API 请求部署在同一个渔域名下，就不再需要 CORS 配置了。

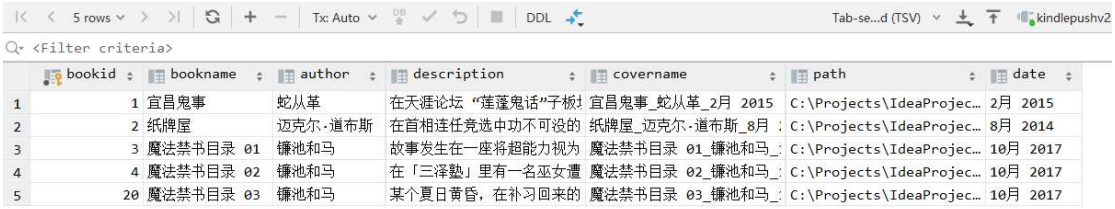
```
import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins("http://www.bmq.com")
                .allowCredentials(true)
                .allowedMethods("GET", "POST", "DELETE", "PUT","OPTIONS")
                .maxAge(3600);
    }
```

}

2.前端

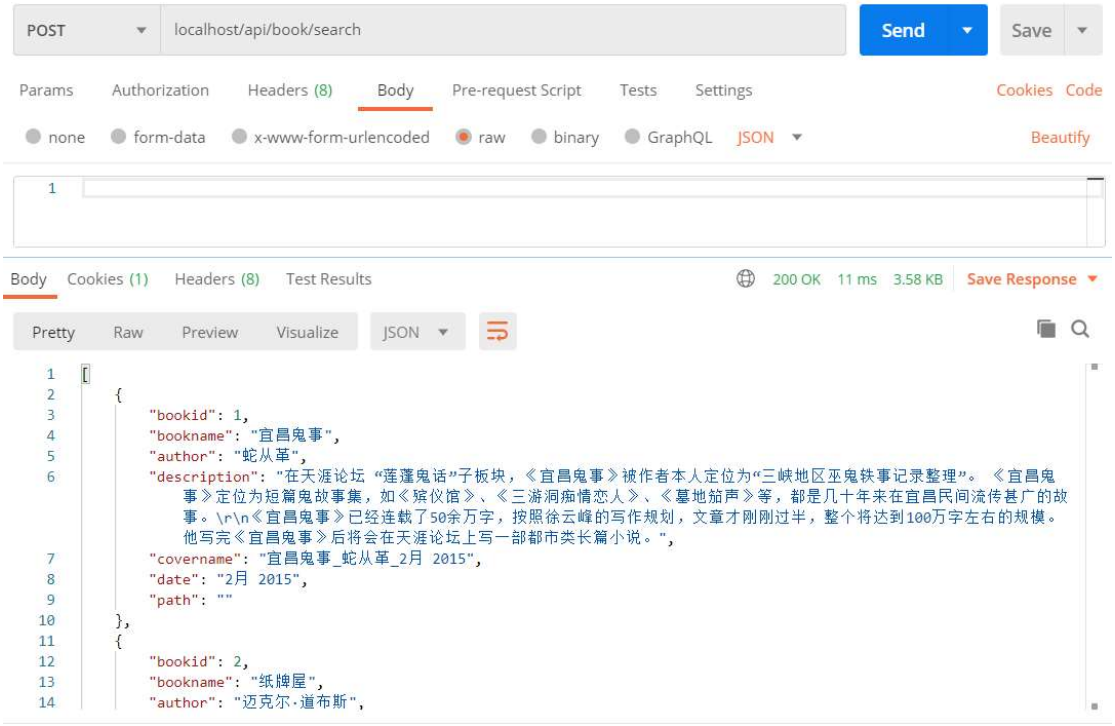
四、效果演示

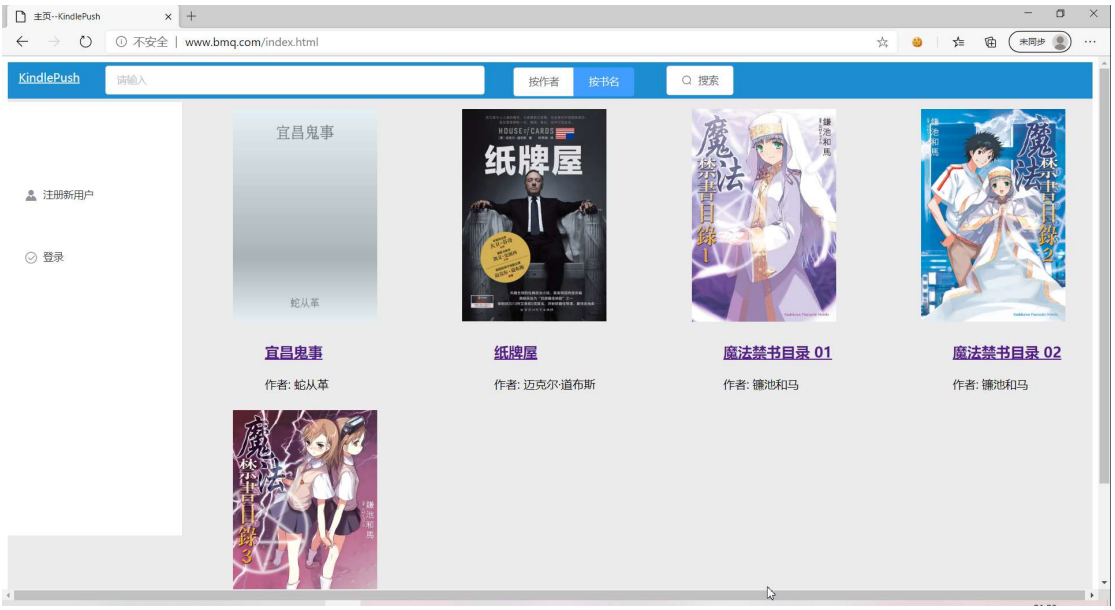
事先在数据库插入了几条数据。如下所示：



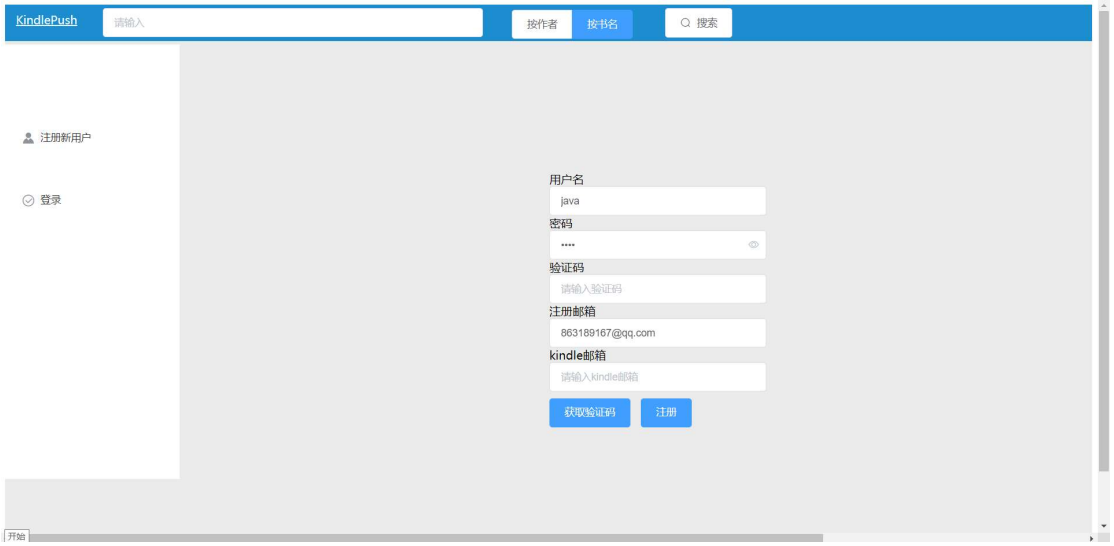
	bookid	bookname	author	description	covername	path	date
1	1	宜昌鬼事	蛇从革	在天涯论坛 “莲蓬鬼话”子板块	宜昌鬼事_蛇从革_2月 2015	C:\Projects\IdeaProjec…	2月 2015
2	2	纸牌屋	迈克尔·道布斯	在首相连任竞选中功不可没的	纸牌屋_迈克尔·道布斯_8月	C:\Projects\IdeaProjec…	8月 2014
3	3	魔法禁书目录 01	镰池和马	故事发生在一座将超能力视为	魔法禁书目录 01_镰池和马_	C:\Projects\IdeaProjec…	10月 2017
4	4	魔法禁书目录 02	镰池和马	在「三泽塾」里有一名巫女遭	魔法禁书目录 02_镰池和马_	C:\Projects\IdeaProjec…	10月 2017
5	20	魔法禁书目录 03	镰池和马	某个夏日黄昏，在补习回来的	魔法禁书目录 03_镰池和马_	C:\Projects\IdeaProjec…	10月 2017

POSTMAN 主页 API 测试，浏览器正常显示：

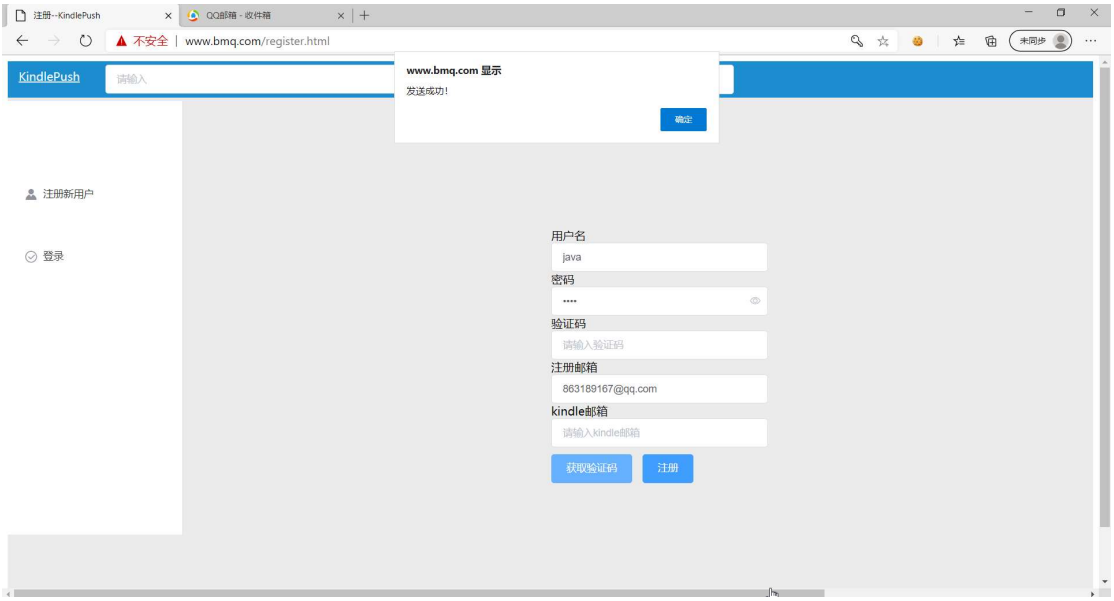




下面进行新用户的注册:

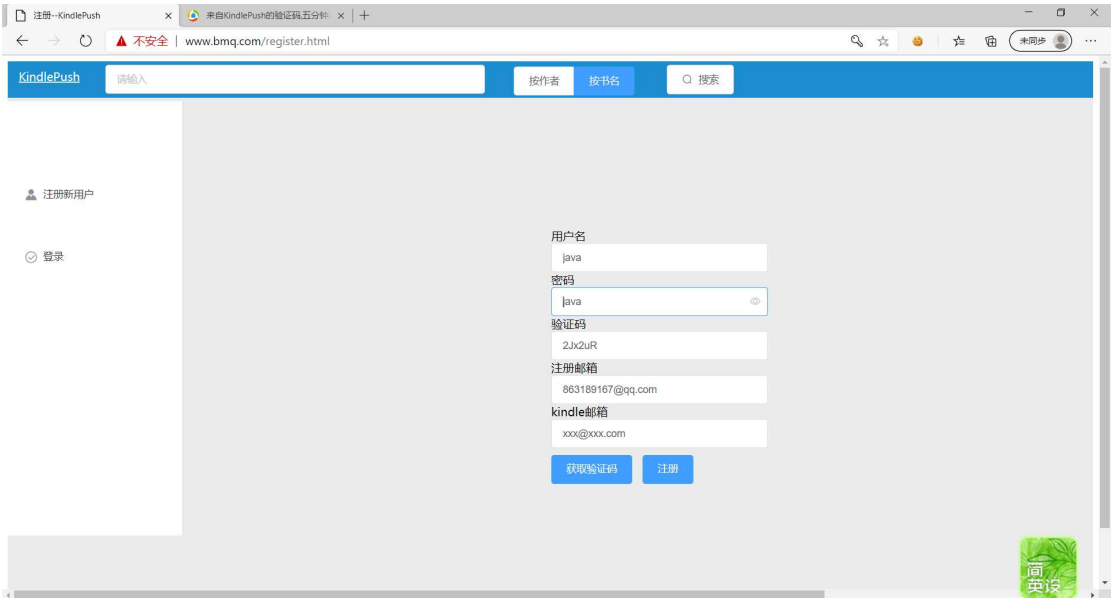


单击**获取验证码**，弹出 发送成功 消息框**:**

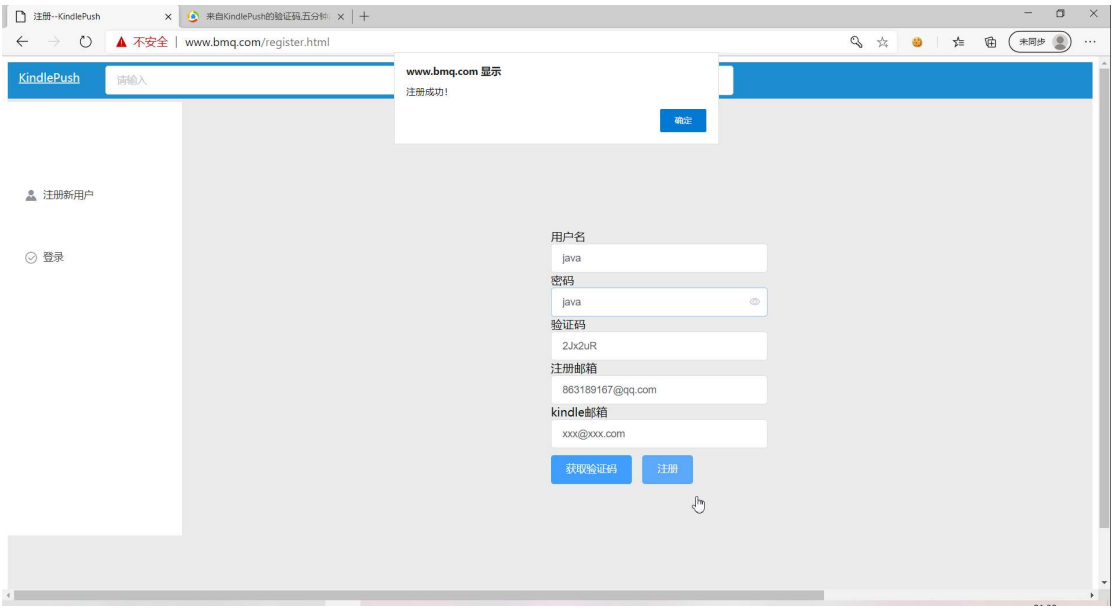


查收邮箱，收到验证码邮件：

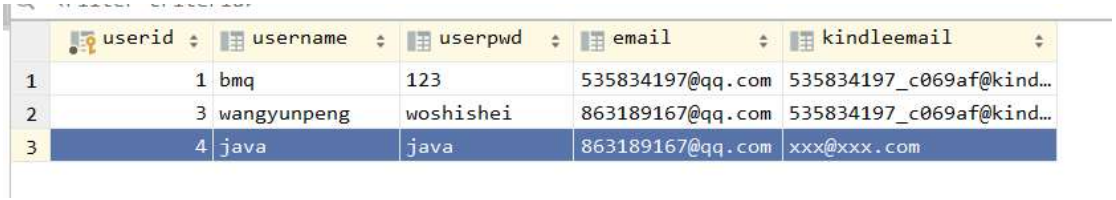
填写并注册：



注册成功：

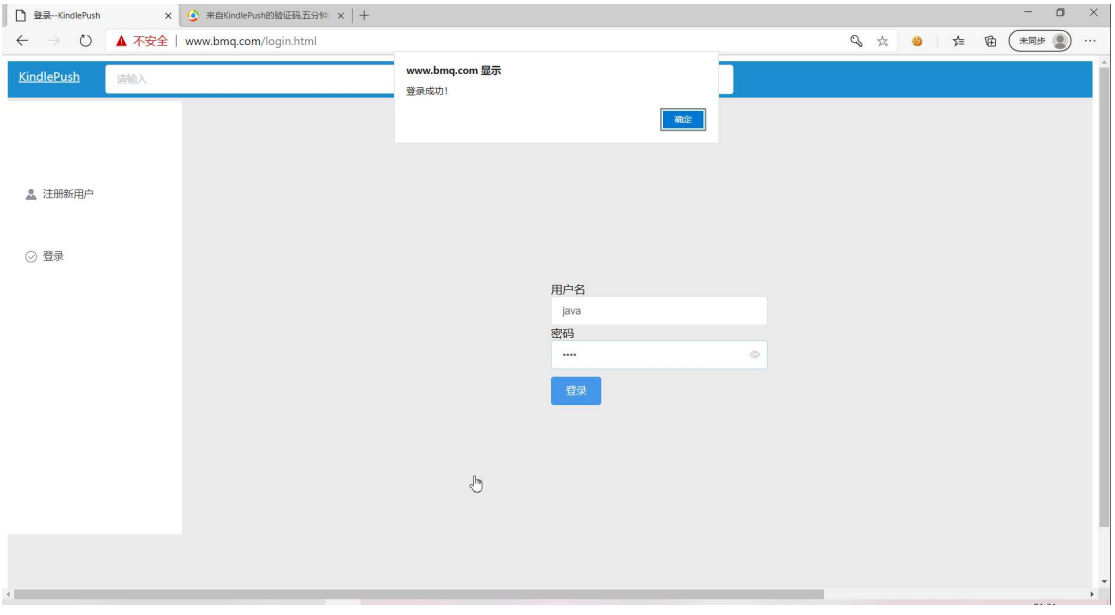


查看服务器数据库，成功写入：

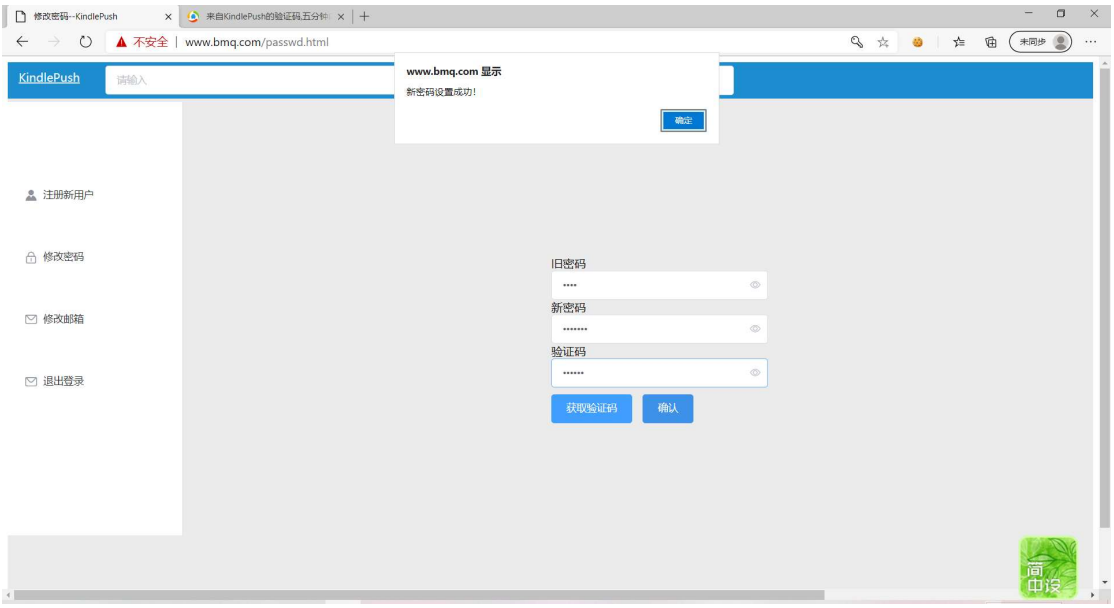


	userid	username	userpwd	email	kindleemail
1	1	bmq	123	535834197@qq.com	535834197_c069af@kind…
2	3	wangyunpeng	woshishei	863189167@qq.com	535834197_c069af@kind…
3	4	java	java	863189167@qq.com	xxx@xxx.com

测试登录，登录成功：



测试修改密码，再次获取了一下验证码（可以看到登录成功后左栏发生了变化）：

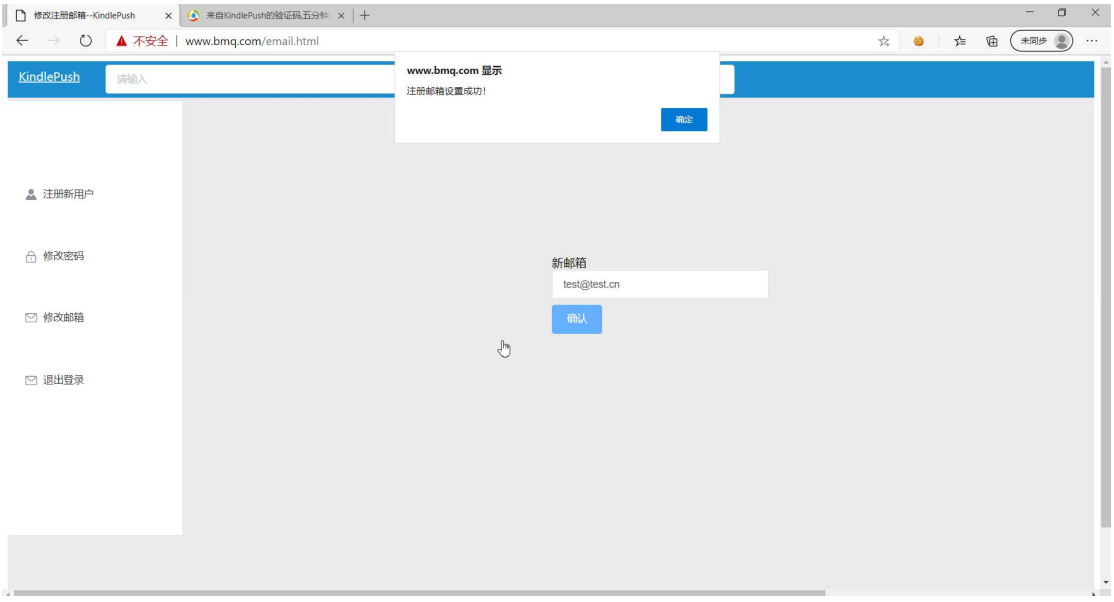


查看服务器数据库，修改成功：

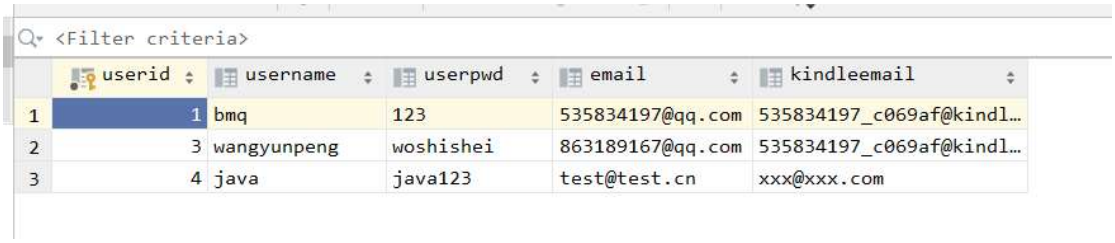
<Filter criteria>

	userid	username	userpwd	email	kindleemail
1	1	bmq	123	535834197@qq.com	535834197_c069af@kind…
2	3	wangyunpeng	woshishei	863189167@qq.com	535834197_c069af@kind…
3	4	java	java123	863189167@qq.com	xxx@xxx.com

测试修改邮箱：

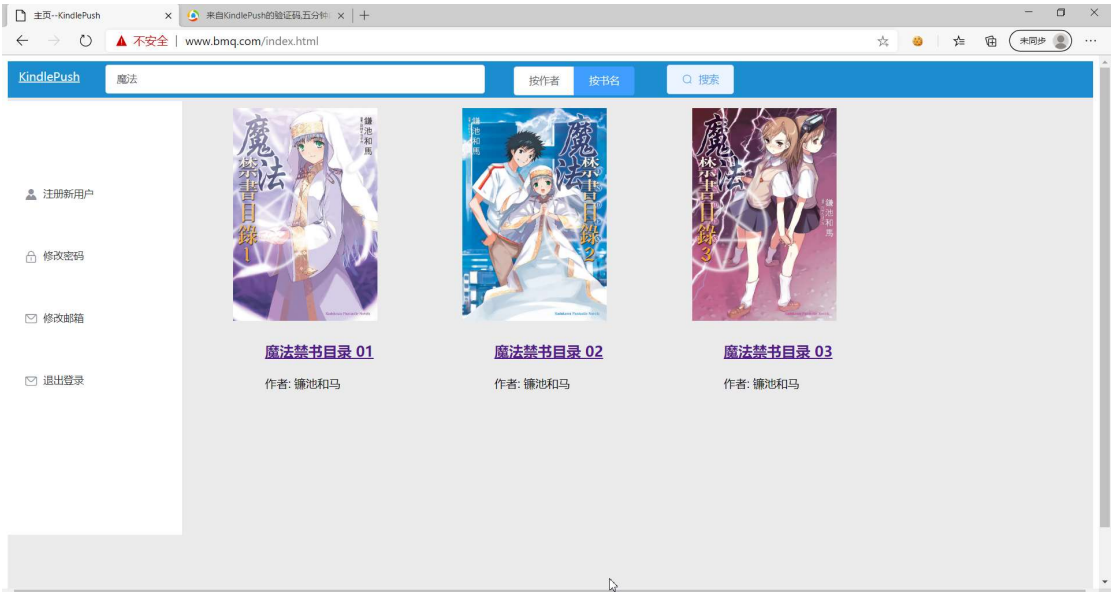


查看服务器数据库，修改成功，test@test.cn：



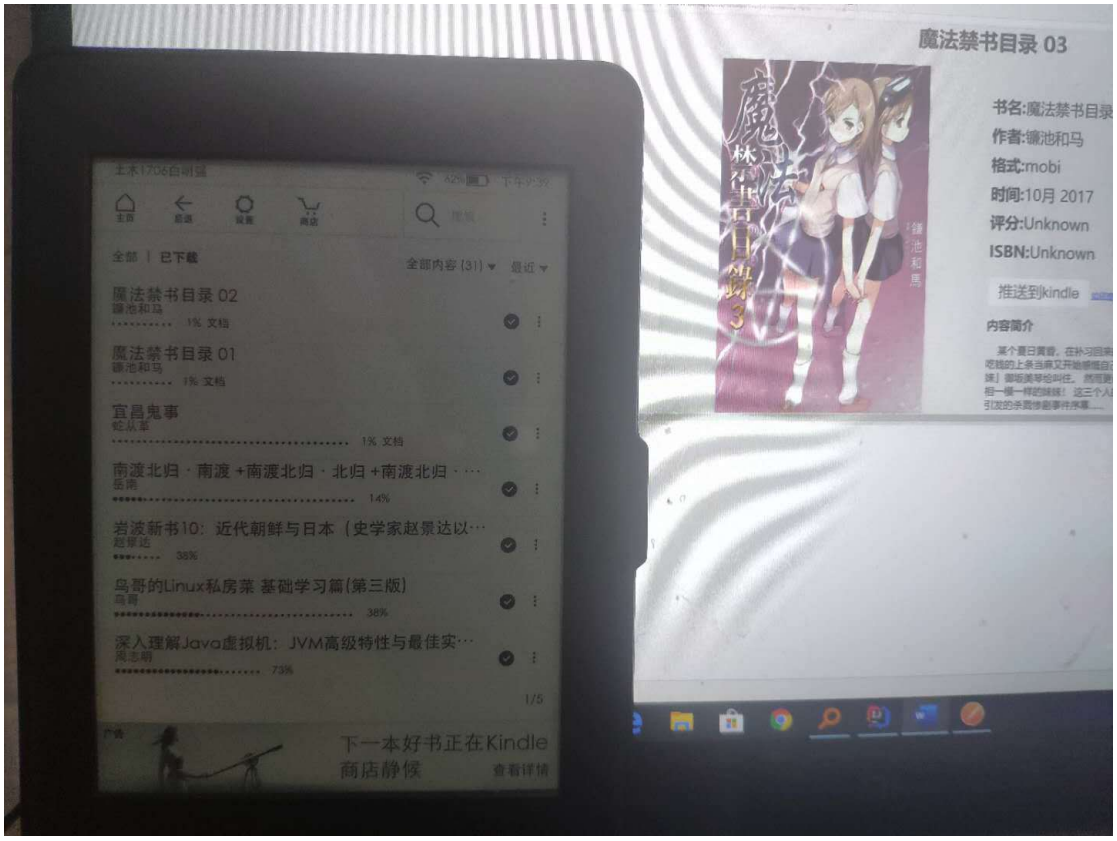
	userid	username	userpwd	email	kindleemail
1	1	bmq	123	535834197@qq.com	535834197_c069af@kindl…
2	3	wangyunpeng	woshishei	863189167@qq.com	535834197_c069af@kindl…
3	4	java	java123	test@test.cn	xxx@xxx.com

测试搜索，结果正确：



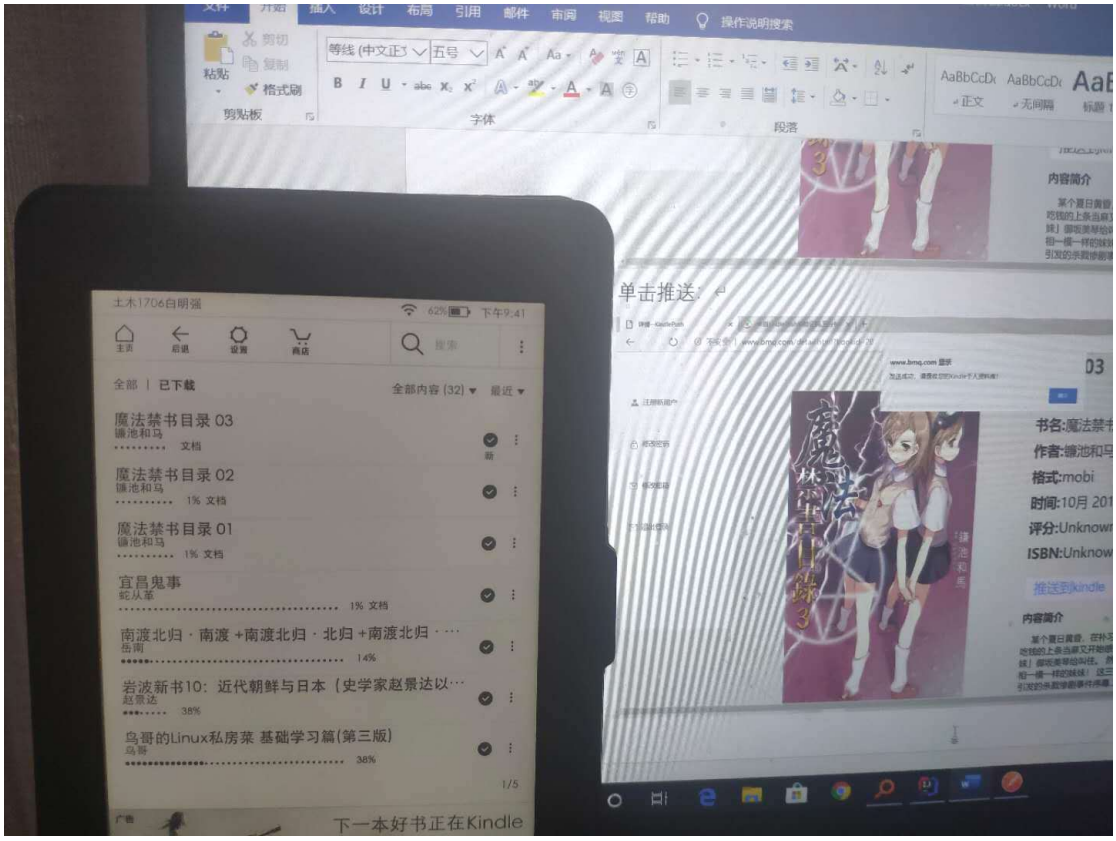
点击某一本图书：

原始 Kindle 状态：



单击推送：

稍等后可以看到推送成功，魔法禁书目录 03 出现在 Kindle 的列表中。



登录前 redis 状态，可以看到 java 用户存在：

```
redis 127.0.0.1:6379> keys *
1) "1voBR7rlcVrY1WsT2AXpez7D"
2) "java"
3) "yxJnjjtTyMN00zWPIgKs3kGu"
4) "3ga5KhitOJAmqBTqBHZDfGrE"
```

单击退出登录：

再次查看 redis，发现 java 用户已经消失，故登出成功。此处以 token 为键的数据项难以编程删除，所以最开始我们设置 token 为 2 小时的有效期，等过期时 redis 会自动将其删除。

```
redis 127.0.0.1:6379> keys *
1) "1voBR7rlcVrY1WsT2AXpez7D"
2) "java"
3) "yxJnjjtTyMN00zWPIgKs3kGu"
4) "3ga5KhitOJAmqBTqBHZDfGrE"
redis 127.0.0.1:6379> keys *
1) "1voBR7rlcVrY1WsT2AXpez7D"
2) "3ga5KhitOJAmqBTqBHZDfGrE"
```

至此全部功能测试完毕。

五、问题与解决

that 和 this 的问题
equals 的问题
CORS 的问题
静态资源的问题
SameSite 的问题

六、收获