基于微信小程序的在线书城

摘要

微书是一款基于微信小城序的在线书城，书城数据库使用mongodb存储，书城数据使用nodejs爬虫从网络上爬取，并使用loopback作为后端接口框架。功能上，微信实现以下5个模块，分别是我的书架、书城、个人心中、H5阅读器、登录注册。这些模块中又存在许多子模块，互相连接和配合，为用户提供便捷和舒适的阅读体验，让用户能更简单并且不付费的读到自己想读的书籍。除此之外微书还支持书评以及查看书籍排行榜，帮助用户从浩瀚的书海中找到自己喜欢的书。本论文将介绍所设计的微书，并对本系统进行数据分析和设计。

**关键字:** 微信小程序, H5阅读器, 网络爬虫, Loopback, mongodb

Abstract

FllowHeart is an online book stores based on Wehcat Small Program.All books is crawled from the website for publishing books. We use mongodb to storage the data of all books and user, and use loopback as the back-end interface framework. Functionally, FllowHeart is made of the following 5 modules, namely, my bookshelf, book stores, personal center, H5 reader, login and registe. There are many sub modules in these modules, width the connection and cooperation with each other, FllowHeart becomes to be able to provide users with a convenient and comfortable reading experience, so that users can be more simple and do not pay to read their own books. In addition, FllowHeart also supports reviewing and ranking of books, which can help users to find their liking books from the vast sea of books. This paper will introduce the design of the micro book, and the data analysis and design of the system.

**Keywords**: The Small WeChat program, H5 reader, Internet crawler, Loopback, mongodb

目录

[摘要 1](#_Toc14178)

[1 前言 3](#_Toc24884)

[1.1 项目介绍 3](#_Toc23710)

[1.2 项目背景 3](#_Toc3310)

[2 微书书城系统分析 3](#_Toc11467)

[2.1 需求分析 3](#_Toc12665)

[2.2 功能分析 3](#_Toc21133)

[2.3 系统用例图设计 3](#_Toc3625)

[2.4 系统流程图设计 3](#_Toc14910)

[2.5 系统开发环境 3](#_Toc9974)

[3 数据库分析与设计 4](#_Toc9427)

[3.1 数据库分析 4](#_Toc23135)

[3.2 数据库概念设计 4](#_Toc31432)

[3.3 数据逻辑结构设计 4](#_Toc11029)

[3.4 各表之间的联系图 4](#_Toc10118)

[4 微书设计和功能实现 4](#_Toc1111)

[4.1 系统登录注册模块 4](#_Toc18661)

[4.1.1 登录注册界面设计 4](#_Toc13806)

[4.1.2 登录注册逻辑实现 4](#_Toc7082)

[4.2 我的书架模块 4](#_Toc3356)

[4.2.1 我的书架界面设计 4](#_Toc5445)

[4.2.2 我的书架逻辑实现 4](#_Toc23043)

[4.3 H5阅读器模块 4](#_Toc23565)

[4.3.1 阅读器界面设计 4](#_Toc27984)

[4.3.2 分页算法 4](#_Toc6863)

[4.3.3 左右滑动翻页实现 4](#_Toc14381)

[4.3.4 阅读器风格切换、字体设置、查看目录 4](#_Toc20848)

[4.4 书籍详情模块 4](#_Toc18627)

[4.3.1 书籍详情页界面设计 5](#_Toc5095)

[4.3.2 书评和点赞功能实现 5](#_Toc28630)

[4.4 书城模块 5](#_Toc29159)

[4.4.1 书籍详情页界面设计 5](#_Toc68)

[4.4.2 书评和点赞功能实现 5](#_Toc21740)

[4.5 书城模块 5](#_Toc24425)

[4.5.1 个人中心界面设计 5](#_Toc20355)

[4.5.2 个人信息 5](#_Toc5559)

[4.5.3 个人设置 5](#_Toc28677)

[4.5.4 我的消息 5](#_Toc18695)

[4 微书测试 5](#_Toc31277)

[4.1 测试项目 5](#_Toc5736)

[4.2 测试用例 5](#_Toc17950)

[参考文档 5](#_Toc12156)

# 1 前言

## 1.1 项目介绍

小程序是一种不需要下载安装即可使用的应用，它实现了应用「触手可及」的梦想，用户扫一扫或者搜一下即可打开应用。借助小程序，应用将无处不在，随时可用，但又无需安装卸载。另一方面，面对阅读收费化，广告化的趋势，我们希望给用户提供免费、舒适、纯粹的阅读体验，让微读成为用户的移动图书馆，用深度的内容去对抗浮躁的世界。结合以上两方面，我们使用微信小程序作为前端基础，使用nodejs以及数据库mongodb作为后台支撑，搭建了一个移动端阅读应用--微书。

## 1.2 项目背景

随着微信小程序的推出，越来越多的开发者将自己的产品往小程序迁移，希望借助微信巨大流量以及信息传播的便捷性，让自己的产品广为人知。同时，互联网已经步入大数据时代，在资源有限，用户较少的情况下，很多产品很难去聚集大量数据，此时爬虫就可以作为一个暂时的数据来源。微书的开发正是基于以上两点，使用nodejs爬虫作为数据来源，微信小程序作为产品展示方式，loopback作为后端支撑，构建起了一款免费的在线书城。使用微书，你可以读到许多以前需要付费才能读到的书籍，同时精心设计的阅读器也能带给你不同于网页阅读的舒适的阅读体验。

# 2 微书书城系统分析

## 2.1 需求分析

在网络普及的时代，网上阅读已经走进人们的生活。相比传统的书本，网上阅读能让用户更方便快捷的选择自己喜爱的书籍，拥有的庞大的书籍库森罗万象，能满足用户各方面的需求。同时网络阅读器能将自己喜欢的书籍都存放在一部小小的手机中，同时能保存用户书签，想法这些数据，方便用户随时查看。一个标准的在线书城应该能帮助用户查找和选择喜欢的书籍，并且提供诸如书单列表、书籍更新状态、书签等等这些书籍管理功能，以及用来在线阅读的H5阅读器。

在线书城在架构上应该包含前台和后台，前台主要是提供美观和人性化的界面，利用后台提供的接口展示数据，给用户提供良好的阅读环境。后台包括三方面，一是数据库设计，提供快速数据查询和存储，二是数据来源，在前期没有用户基础的前提下，可以使用网络爬虫从书籍发布网站上爬取已有的数据，三是api接口，为前端提供获取数据的入口。

在线书城应该具有如下几个重要的功能：

1. 对于书城中的每本书，用户可以点击查看书籍详情，然后决定是否加入个人书架。
2. 用户查看书籍详情的时候可以看到别的用户对这本书的评价，自己也可以对这本写书评。
3. 有一个个人书架页面方便用户管理自己的书籍，最好能提供一个基本的搜索功能，以便用户能快速定位到自己要找的书籍。另外需要有导向书城页的链接，用于用户添加新的书籍。
4. 书城上可以放置一些推荐专题、分类、和排行榜之类的信息，引导用户快速选择自己喜欢的书，此外最好提供一个全局搜索的功能，应对用户直接通过书名定位书籍的需求。
5. 书城应该有基本的用户管理功能，包括登录注册、微信绑定、查看和更新个人信息、阅读设置、我的消息等等。

## 2.2 功能分析

根据以上需求分析，我将微书划分成五个主要的模块，分别是我的书架、书籍详情页、H5阅读器、个人中心、登录注册。各个功能模块的具体说明如下：

我的书架：我的书架用来展示用户已经加入书架的书籍，并提供书籍更新提醒、书籍按照时间分类、以及搜索功能。

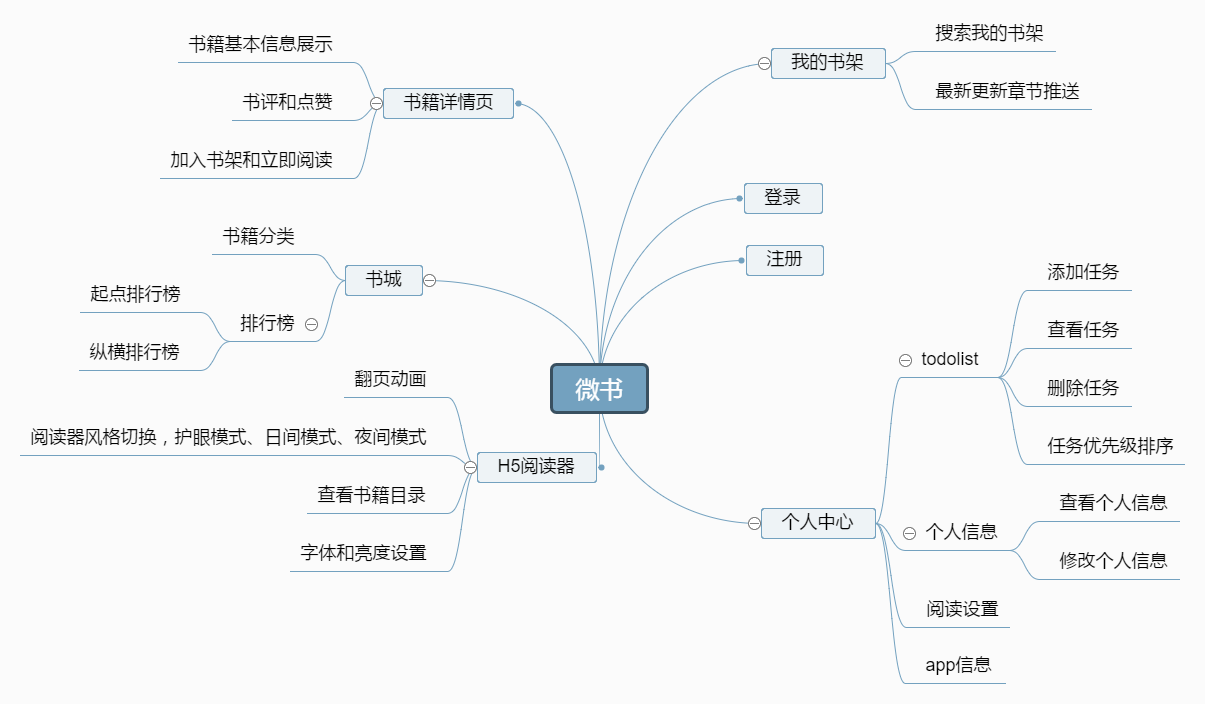
书籍详情页：书籍详情页会展示关于这本书籍的诸如书籍名字、封面、作者、简介等这些信息，并提供给用户“添加至书架”和“立即阅读”的按钮。书评也是详情页的一大功能，能让用户发表一些对这本书的看法，也可以帮助后来的用户更好的选择自己喜欢的书籍，并提供一个用户交流的平台。

H5阅读器：阅读器是给用户提供一个在线阅读的工具，具有左右滑动的翻页效果、查看书籍目录、切换阅读器风格、更改字体这些功能。

个人中心：在个人中心，用户可以查看自己的个人信息，并选择是否更改它们，另外可以设置阅读器风格、字体，或者对软件做一些设定（比如更新提示、阅读久了提示休息等等）。还有用户可以查看自己发表的书评和其他用户回复自己的信息。

登录注册：微书有自己的用户管理，在小程序中可以获取到用户的一些基本信息，但是这些远远不够，所以我们需要基于微信小程序将用户注册为微书的用户，并使用自己的服务器单独维护用户的登录状态。

下图就是微书的功能模块图：



## 2.3 系统用例图设计

微书使用者这里分了三种---小程序用户、产品运营、产品运维，小程序用户就是通过扫描微书二维码进入并使用微书的人，产品运营即是负责编辑书城推荐专题、新书栏目、以及使用邮件通知用户。产品运维主要维护后端接口、数据库、以及网络爬虫的正常执行。下面是基于这三类使用者绘制的用例图：

## userCase

## 2.4 系统流程图设计

用户扫码进入微书，首先需要验证是否绑定微信，如果未绑定微信，跳转至微信注册页面，然后执行登录得到后端返回的登录凭证，进入我的书架页面。

书架页面则会判断是否已经拥有书籍，以及所拥有的书籍是否更新，如果已经有在架书籍，可以点击进入H5阅读器，否则会被导向到书城页面去添加书籍。

在书籍详情页用户可以选择将这本书添加至自己的书籍，也可以将这本书从书架中移除。此外可以添加这本书的书评，或者评论其他用户的书评。

如果用户进入H5阅读器开始阅读了，系统会记录用户已阅读时间，并在离开阅读器的时候更新已阅读时间。在阅读器中用户可以切换阅读器风格为护眼模式，日渐模式、或者夜间模式，也可以改变阅读器亮度和字体，还可以查看这本书籍的所有章节。

在个人中心页，首先会展示用户的已阅读时间、累计阅读天数、和已拥有的书籍。点击个人头像可以查看自己的个人信息，选择修改即可修改自己的信息。在设置栏，用户可以对阅读器做一些自己喜欢的设定，下次打开阅读器会默认加载这些设置。在个人消息栏会展示其他用户回复自己的消息，用可以也可以直接回复别人。

书城会加载最新的推荐专题、分类、以及书籍排行榜，帮助用户更好的选择自己喜欢的书籍。下面是微书的系统流程图：

Untitled Diagram

## 2.5 系统开发环境

微书开发所需环境以及所用工具如下：

1. 开发使用的操作系统：Windows 10 企业版
2. 开发使用的数据库以及工具：Mongodb、Robomongo
3. 开发使用的文本编辑器：Sublime Text3
4. Nodejs调试工具：Webstorm 2017.3.1
5. 小程序开发工具：微信官方开发者工具

# 3 数据库分析与设计

## 3.1 数据库分析

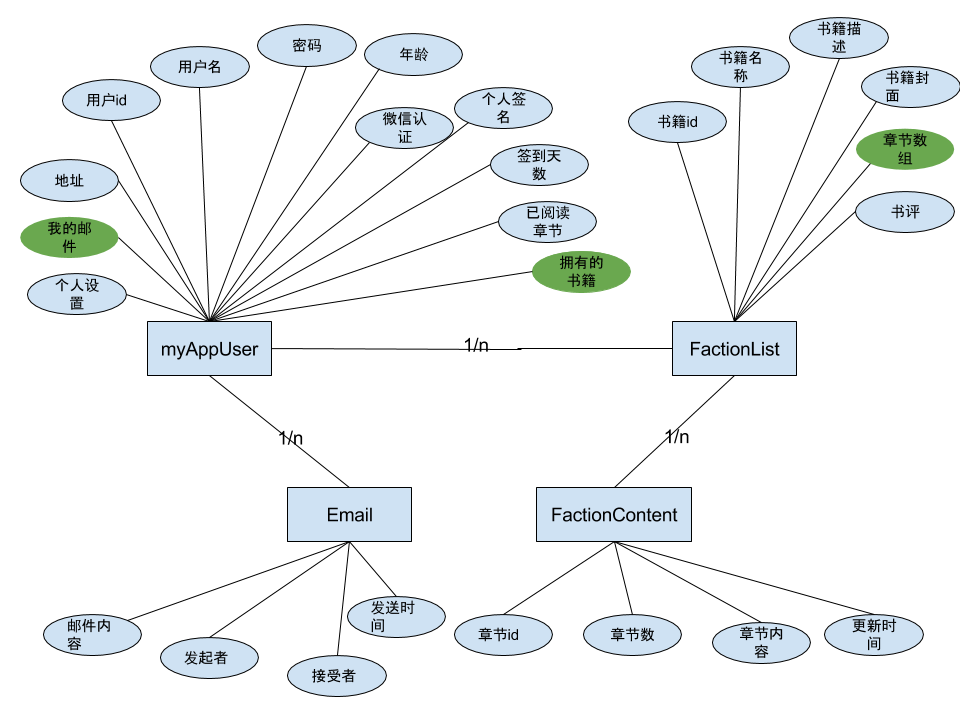
微书主要是存储一些文本，而mongo在处理文本数据上速度比较快，并且mongo和所用的主要开发语言nodejs结合比较紧密。另外mongo有类似mysql的父子文档的概念，也可以轻松的实现不同文档之间的关系模型以及联合查询。所以在开发之前将mongodb作为数据库，用以保存书籍基本信息、章节信息、用户信息、和排行榜信息。

## 3.2 数据库概念设计

微书规划除了5个实体，分别为书籍基本信息实体、章节信息实体、用户信息实体、排行榜信息实体、以及邮件信息实体。这里考虑到章节信息数据量比较大，所以没有将所有章节信息和书籍基本信息存储在一个文档，而是利用mongo父子文档的模型定义两个表分开存储两者的信息。

1. 书籍基本信息实体包含书籍的基本信息，其中包括书籍id、书籍名称、书籍描述、书籍封面图片地址、书籍字数、书籍章节数组（包含这本书所有章节的id的数组）、书籍最新章节（用于判断书籍是否更新）、书籍更新时间。
2. 章节信息实体是书籍基本信息实体的子文档，它的id被包含在基本信息实体的章节数组中。其中包括章节id，章节数（第几章）、章节标题，章节内容、爬取来源、以及更新时间。
3. 用户信息实体，包含许多用户信息，比如用户id年龄、生日、头像、个人签名、昵称等等，其中有一些连接其他实体的字段，比如我的书评、我的消息、我的设置、我的邮件、我的书籍，这些都会在用户管理的过程中被添加或者更新。
4. 排行榜信息实体，用来记录爬虫从起点网和纵横网排序到的书籍排行榜信息。其中包括分类名字，起点排行榜数组，纵横排行榜数组，以及更新时间
5. 邮件信息实体，用来记录所发邮件，其中包括发送人、接收人、发送时间、邮件标题、邮件内容等。

下面是这些实体的E-R图：



## 3.3 数据逻辑结构设计

书籍基本信息实体：

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 描述 |
| \_id | ObjectId | 书籍id，mongodb自动生成 |
| factionName | String | 书籍名称 |
| des | String | 书籍简介 |
| headerImage | String | 书籍封面图片地址 |
| author | String | 书籍作者 |
| updateTime | Date | 最新更新时间 |
| sectionArray | Arra | 章节id数组 |
| newest | Int32 | 最新章节数 |
| comments | Array | 评论数组 |

章节信息实体：

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 描述 |
| \_id | ObjectId | 章节id，mongodb自动生成 |
| sectionNum | Int32 | 章节序号，第几章 |
| sectionTitle | String | 章节标题 |
| sectionContent | String | 章节内容 |
| sectionResource | String | 章节来源 |
| recentUpdateTime | Date | 最新更新时间 |
| des | String | 额外描述 |

排行榜实体：

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 描述 |
| \_id | ObjectId | 榜单id，mongodb自动生成 |
| standard | String | 分类统一名字 |
| engName | String | 分类英文名字 |
| qidian | String | 起点分类的名字 |
| zongheng | String | 纵横分类的名字 |
| qd\_url | String | 起点排行榜地址 |
| updateTime | Date | 更新时间 |
| zhRank | Array | 纵横排行数组 |
| qdRank | Array | 起点排行数组 |

用户实体：

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 描述 |
| \_id | ObjectId | 用户id，mongodb自动生成 |
| age | Int32 | 年龄 |
| nickName | String | 昵称 |
| signature | String | 个人签名 |
| myBooks | Array | 我的书籍数组 |
| auth | String | 微信认证字段 |
| avatar | String | 个人头像图片地址 |
| hasReadTime | Int32 | 纵横排行数组 |
| continueReadDay | Int32 | 起点排行数组 |
| address | String | 地址 |
| gender | Int32 | 性别，0表示女，1表示男 |
| realm | String | 真实姓名 |
| username | String | 用户名 |
| password | String | 密码 |
| email | String | 邮箱 |
| emailVerified | Boolean | 是否已经通过邮箱验证 |
| setting | Array | 个人设置数组 |
| birthday | String | 生日 |

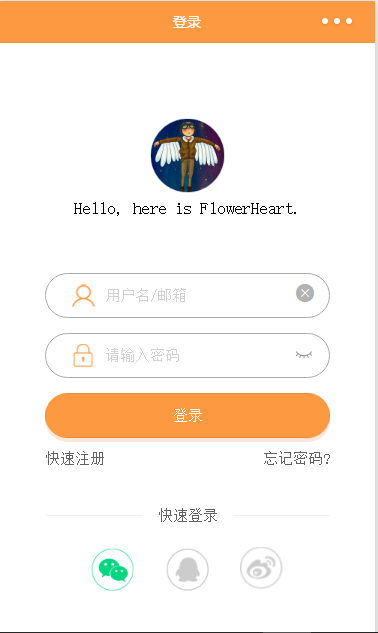
邮箱实体：

|  |  |  |
| --- | --- | --- |
| 字段名 | 数据类型 | 描述 |
| \_id | ObjectId | 榜单id，mongodb自动生成 |
| from | String | 发送人 |
| to | String | 接收人 |
| title | String | 邮件标题 |
| content | String | 邮件内容 |
| time | String | 邮件发送时间 |

# 4 微书设计和功能实现

## 4.1 系统登录注册模块

### 4.1.1 登录注册界面设计

### 4.1.2 登录注册逻辑实现

1. 登录

常见的小程序登录方式有三种，分别是自有账号登录、使用第三方平台账号登录、使用微信账号登录，下面会对每种登录方式的实现原理做描述。

使用微信账号注册，是官方推荐的登录方式，毕竟小程序是基于微信，这种登录方式在安全性也比其他的要高（微信对数据做了数据签名和加密）。但是由于小程序没有Cookie机制，故实现小程序登录以及登录状态的维护和以往的登录不一样，大致的流程如下：

1. 调用小程序API的wx.login()获取到微信登录凭证。

2. 利用登录凭证调用小程序API的wx.getUserInfo()获取到基本用户信息（如头像，昵称等），和一些敏感用户信息，以及数据签名（将会用于数据验证）。

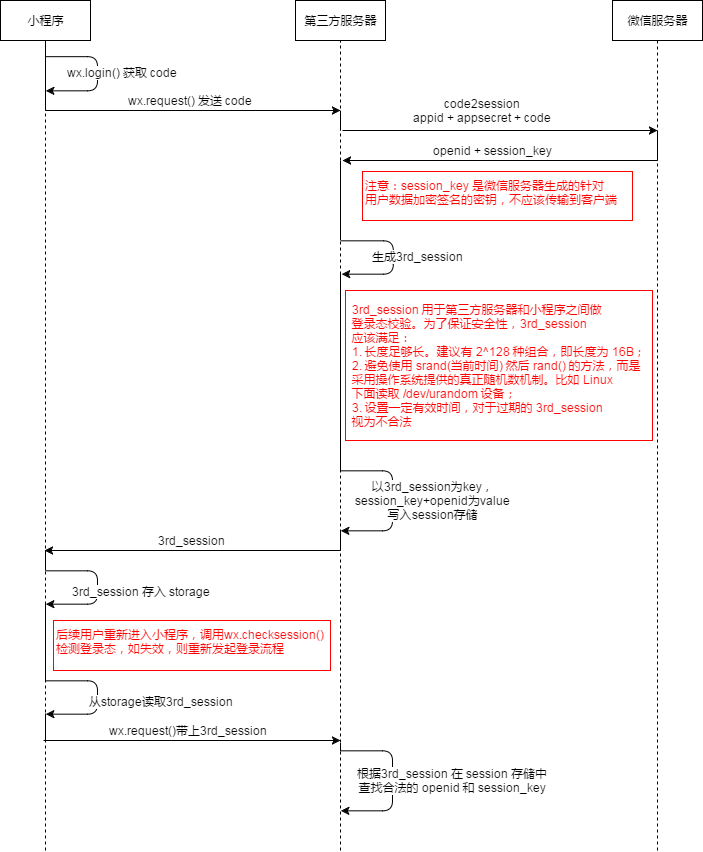
3. 将凭证wxcode通过自己写的接口发送给后端（第三方服务器），后端向微信的服务器发送code2Session的请求，使用code换取用户唯一openid以及section\_key。

4.将上一步拿到的openid和section\_key作为键值，并使用随机算法生成一个唯一的id作为键值名sessionid，然后将键值名和键值存入redis中，并设置过期时间为7天。并提供一个sessionid的检查是否过期的方法，用来判断登录状态是否有效。

5. 将生成的键值名sessionid返回给前端，前端将sessionid存储到小程序缓存中。

6. 在用户离开小程序并再次登录时，首先调用后端接口判断sessionid是否过期，如果没有过期就跳转到我的书单，如果已经过期就重新执行一遍上面的操作。

下图详细的介绍了小程序如何实现登录，并维护登录状态的。



使用自有的账号登录，这个和常见的登录操作一直，查询数据库中的myAppUser表，核对用户名和密码，登录状态的维护和使用微信登录一致

使用第三方账号登录，这里值得注意的是小程序不支持Html页面，那些需要使用重定向来进行登录的第三方API就需要改造下，或者不能用了。

1. 注册

微书的注册不同于普通的注册，多了注册前信息初始化这一步。由于微书是基于小程序，所以我们希望在用户填写注册信息的时候，对于微信账户中已有的信息，诸如昵称、头像、城市这些能有一个初始化的值，这样既减少了用户注册的时候输入，也显得微书和微信结合更加紧密些。下面介绍下微书的注册流程：

1. 首先还是使用wx.login()获取到登录凭证，然后调用wx.getUserInfo()获取到用户在微信中的用户信息

2. 利用这些用户信息作为注册表单的初始化信息，当然用户在注册的时候依旧可以改变这些初始化信息

3. 在用户点击提交按钮的时候对用户填写的信息做正则校验，对于校验不通过的注册项给予用户提示。

4. 如果所有注册项均通过正则校验，则调用后端的注册接口将用户提交的信息发送至后端，后端将这些数据存入数据库即完成了注册。

这里提及下实现上传个人头像的功能实现原理。利用小程序的wx.chooseImage的方法可以选择一张本地的图片获取拍摄一张照片，在这之后此方法会返回一个临时的图片地址。微书使用的第三方图片存储服务器是七牛云，七牛云首先需要拿到上传凭证uploadToken，所以这里自己实现了一个getUploadToken的接口专门用来返回上传凭证字段。此后，只需要将上传凭证和临时图片地址使用wx.request发送给七牛云就上传成功了，成功之后会返回一个七牛云的在线地址。把这个在线地址更新作为用户新的图像地址就可以了。

## 4.2 我的书架模块

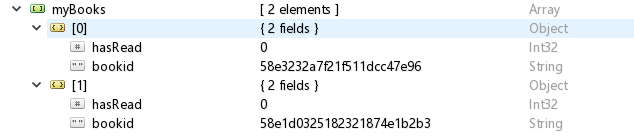
### 4.2.1 我的书架界面设计



### 4.2.2 我的书架逻辑实现

我的书架页主要实现了书单列表和搜索功能，下面分别介绍这两个功能的实现原理。

书单列表页展示的是用户加入书架的书籍，所以在数据库的myAppuser表下就有一个myBooks的字段，用来记录用户的书籍以及每本书的以阅读章数。



后端根据前端传过来的userid获取到用户的书单数组。然后根据数组中的bookid查看书籍基本信息表得到，书籍的封面图片地址、书名、以及最新章节，结合之前的书籍id以及已阅读章节，一起返回给前端。最后前端利用循环将这些信息展示至页面上。



书单搜索则是使用了正则表达式去匹配用户输入的搜索值，会对书籍名称以及书籍描述做检索。如果存在与之匹配的值，就将此书籍显出来，否则如果没有与之匹配的值，将其设置为隐藏。此外书单搜索还加入了高亮显示匹配项的功能，实现的原理即在使用js正则表达式的exec方法的时候会得到每个匹配项的位置值，根据这个位置值在字符串的前后动态添加<code>和</code>标签，然后在css中设置这种标签的样式。

|  |
| --- |
| while (regExp.exec(tempStr) != null) {  console.log(++count);  lastIndex = regExp.lastIndex + 13 \* (count - 1);  //每次循环notChageStr并非不变，而是多了<code></code>共计13个字符，所以为了保证后续循环中lastindex的正确性应该将lastindex自增13  leftStr = notChageStr.substring(0, lastIndex - searchString.length);  rightStr = notChageStr.substring(lastIndex);  notChageStr = leftStr + '<code>' + searchString + '</code>' + rightStr;  } |

## 4.3 H5阅读器模块

### 4.3.1 阅读器界面设计

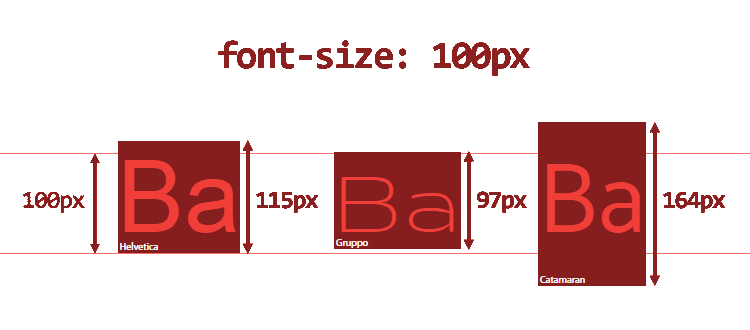
### 4.3.2 阅读器分页

使用前端做阅读器分页需要注意三个问题：

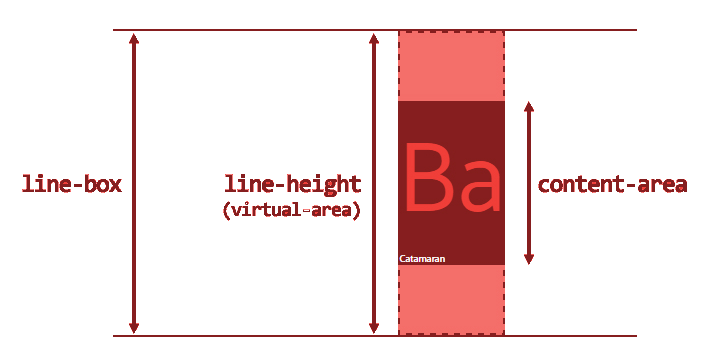
字符分页

行高（line-height）、字体大小（font-size）、以及字体类型(font-family)是影响一个页面到底可以容纳多个字符（包括文字，符号，空格，换行等等）的主要因素。我们来看下三者的定义以及他们之间的关系。

**字体大小**（font-size），它设置的是字体中字符框的高度，在一个页面中同样大小的不同字体表现出来的实际高度不尽相同。

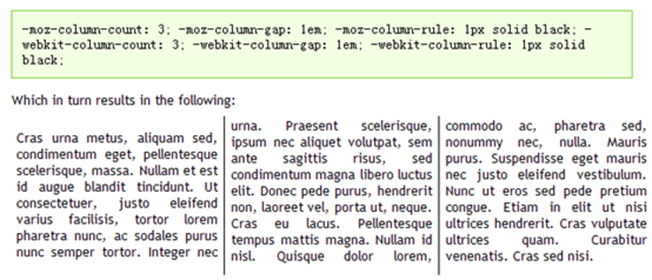


**行高**（line-heigth）会影响行框的布局。在应用到一个块级元素时，它定义了该元素中基线之间的最小距离而不是最大距离。line-height 与 font-size 的计算值之差分为两半，分别加到一个文本行内容的顶部和底部。可以包含这些内容的最小框就是行框（content-area），当line-height设置为1即100%的时候，line-height等于content area。另外，行高并非字体基线（baseline）之间的距离。



**字体类型**（font-family），不同的字体行高的默认值不一样，一般都是1~1.2之间，在更改页面的字体类型之后，需要重新执行分页算法。

以上就是影响分页的三个主要因素，下面介绍如何根据后端传过来的章节内容精准地分页。我们先来讲下分页的实现原理，分页其实是利用了css3的一个分栏属性---columns，分栏能够在不同栏之间实现文本的拆分，如果我们把每一栏设置成刚好一页，就能初步实现分页的效果了。故实现分页其实就是根据后端传过来来的章节内容的字数，计算最多分出的页数（最后一页未满也算作一页），然后动态设置承载文本内容的html元素的column-count属性值为最大分页数。



根据章节内容计算最大分页数，就是先将章节内容使用正则按照空行符\n分割一段一段的文字，然后调用wx.getSystemInfo获取到设备的高度和宽度信息，根据屏幕宽度分别计算每一段文字的最多可以占用的行数。然后使用这个行数乘以line-height就得到每一段文字实际高度，将这些实际高度加起来，然后加上虑空行的高度就得到了这一章的总高度，将这个高度除以屏幕高度取整加一就得到了最大分页数。下面是算法的流程图：

Untitled Diagram (1)

下面贴出部分源码：

|  |
| --- |
| function countPageNum(str, fontSize, lineHeight, windowW, windowH, pixelRatio){  var returnNum = 0;  fontSize = fontSize/pixelRatio;  lineHeight = lineHeight/pixelRatio;  //将str根据’\n‘截成数组  var strArray = str.split(/\n+/);  var splitArray = [];  var reg = new RegExp('\n+', 'igm');  var result = '';  //这里写一个for循环去记录每处分隔符的\n的个数，这将会影响到计算换行的高度  while ((result = reg.exec(str)) != null) {  splitArray.push(result.toString().match(/\n/img).length);  }  //spliArray比strArray少一，这里加一项使之数量一样  splitArray.push(0);  var totalHeight = 0;  strArray.forEach(function(item, index){  //拒绝最后一项0  var huanhangNum = (splitArray[index]-1) > 0? (splitArray[index]-1) > 0: 0;  totalHeight += Math.ceil(item.length/Math.floor((windowW-80/pixelRatio)/fontSize))\*lineHeight + huanhangNum\*lineHeight;  });  return Math.ceil(totalHeight/windowH)+1;  } |

B. 性能问题

虽然分栏能初步模拟分页效果，但是还是存在不少问题，第一: 分栏形成的页面是连续排列的，可以支持滑动操作，但是并不能支持仿真的翻页效果。第二：如果分出的栏目过多，性能就会比较差（大概在20~30个栏目会有明显的性能下降）。好在利用的分栏的也不过是一章节的内容，理论上不会有太大的性能问题。

C. 如何检测用户手势

阅读器支持的用户手势有三种，一种是点击阅读器右边往后翻，点击左边往前翻，第二种是触屏右滑往后翻，左滑往前翻，第三种是点击屏幕中央调起阅读器控制栏。首先我们在阅读控件上绑定touchstart、touchmove、touchend三个事件，三者在一次触屏事件中被顺序触发，这三个事件都有一个touches的数组，分别包含了触屏开始位置的x坐标和y坐标，触屏滑动时当前位置的x坐标和y坐标，以及触屏结束位置的x坐标和y坐标。

对于第一种翻页操作，如果touchmove在一次触屏动作中没有被触发，则认为用户是点击了屏幕，然后比较点击位置距离屏幕左边以及屏幕右边的距离来判断阅读器是向前翻页还是向后翻页。

对于第二种翻页操作，需要在touchmove被触发的前提下，对比触屏开始位置的x坐标和触屏滑动过程中当前位置x的坐标，如果触屏滑动过程中当前位置x坐标变大，则认为用户向右滑动了，阅读器需要往前翻页，反之如果触屏结束时x坐标变小了，则认为用户向左滑动了，阅读器需要往后翻页。

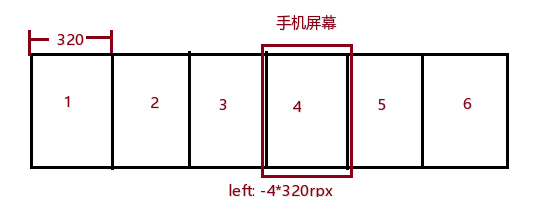
|  |
| --- |
| var currentX = event.touches[0].pageX;  var currentY = event.touches[0].pageY;  // 判断用没有滑动而是点击屏幕的动作  hasRunTouchMove = true;  var direction = 0;  if ((currentX - self.data.touches.lastX) < 0){  direction = 0; // 左滑  }else if(((currentX - self.data.touches.lastX) > 0)){  direction = 1; // 右滑  } |

对于第三种，检测用户是否点击了屏幕中央，这是用户调起阅读器控制栏的操作。同检测滑动方向的原理大致相同，不同点是检测滑动方向使用的touchstart的触点位置和touchmove的当前触点位置坐标做的对比，而检测用户是否点击屏幕中央是touchstart的触点位置和touchend的触点结束位置坐标的对比，如果x坐标差值以及y坐标差值均在一屏幕中心点的容差范围内（H5阅读器中设置的是[-50rpx, 50rpx]），则认为用户点击了屏幕中央，根据当前控制栏是否已经显示来隐藏或显示控制栏。

|  |
| --- |
| // 判断用户的点击事件，决定是否显示控制栏  if(hasRunTouchMove == false){  var y = self.data.touches.lastY;  var x = self.data.touches.lastX;  var h = self.data.windows.windows\_height/2;  var w = self.data.windows.windows\_width/2;  if(x && y && y >= (h-50) && y <= (h+50) && x >= (w-60) && x <= (w+60)){  self.setData({control: {all: self.data.control.all == '0'? '1': '0', control\_tab: 1, control\_detail: 0, target: ''}, isShowFontSelector: 0});  return;  }  } |

### 4.3.3 左右滑动翻页动画的实现

前面提到分页其实就是使用分栏属性将一大段文本分成若干栏，每一栏的宽度都是一屏幕的宽度，那如何实现左滑时内容从右向左缓慢移入的动画呢？首先我们将承载章节的内容的HTML元素的定位设置为相对定位relative, 这样我们就能通过设置元素的left属性来实现内容向左或者向右滑动。



滑动动画有两部分，一是触屏过程中，手指不离开屏幕左右滑动，此时并不会触发翻页，而是需要随着手指滑动页面也需要移动相同的距离。实现这种效果只需要在touchmove事件中对比滑动过程中手指触点距离touchstart其实位置的距离，将left值增加或者减少对应的值就可以了。一旦手指离开了屏幕，即触发了touchend事件，就开始执行页面翻页。

二是翻页效果，这里只是实现了一个简单的从左到右缓慢滑动到指定的位置的动画，其原理也十分简单，就是使用js计时器让left的值从起始值缓慢变化到目标值，具体代码实现如下：

|  |
| --- |
| targetLeftValue = (-1)\*self.data.windows.windows\_width\*currentIndex;  pingjunValue = Math.abs(targetLeftValue - self.data.leftValue)/4;  //500ms其实函数只执行了4次，第一次会等待100ms才会开始函数  isMoving = 1; //开始计时的时候将标志置1  //使用计时器实现动画效果  moveTime = setInterval(function(){  ++ leftTimmerCount;  var currentLeftValue = self.data.leftValue;  //如果达到了目标值，立即停止计时器  //调试发现有些时候这个if的跳转会莫名的不成立，所以做个限制，函数被执行了4次之后，无论条件是否成立，将leftValue设置为目标值，并结束计时器  if(leftTimmerCount == 4){  clearInterval(moveTime);  isMoving = 0;  leftTimmerCount = 0;  self.setData({leftValue: targetLeftValue});  return;  }  if(currentLeftValue == targetLeftValue){  clearInterval(moveTime);  isMoving = 0;  leftTimmerCount = 0;  // console.log('向 左 滑动的计时器结束了，isMoving为0');  return;  }  self.setData({leftValue: currentLeftValue-pingjunValue});  },75); |

### 4.3.4 阅读器风格切换、字体设置、查看目录

H5阅读器定义了四种风格，日间风格、夜间风格、护眼模式、咖啡模式。可以通过点击屏幕中央选择自己喜欢的阅读风格。把需要适配风格的属性值存储在data的一个属性中，用户点击选择风格的时候将正确的风格属性值添加到对应wxml元素上。另外图标颜色的切换使用雪碧图，首先将各种风格的图标都写成雪碧图的类名，然后在用户选择风格的时候将图标元素的类名改成对应风格的类名就好了。下面就是部分风格的属性值：

|  |
| --- |
| colorStyle: {content\_bg: '#f5f9fc', styleNum:1, slider\_bg: '#fd9941', slider\_none\_bg: '#dbdbdb', control\_bg: '#ffffff', control\_fontColor: '#fd9941'};  colorStyle: {content\_bg: '#f5f0da', styleNum:2, slider\_bg: '#a6832f', slider\_none\_bg: '#dbd6c3', control\_bg: '#f8f3e0', control\_fontColor: '#a6832f'};  colorStyle: {content\_bg: '#c0edc6', styleNum:3, slider\_bg: '#359112', slider\_none\_bg: '#a7ccab', control\_bg: '#ccf1d0', control\_fontColor: '#359112'};  colorStyle: {content\_bg: '#1a1e21', styleNum:4, slider\_bg: '#bb7333', slider\_none\_bg: '#212528', control\_bg: '#101417', control\_fontColor: '#bb7333'}; |

字体的切换原理和切换风格一致，下面讲下查看目录的功能。因为考虑到一本书籍的章节可能十分多，所以在查看章节需要做分页，即每次请求章节只有当前章节的前后20章，当用户翻到当前章的后面第15章的时候加载后面的40章以保证翻页的流畅。前端实现分页其实是后端接口做到的，后端先从数据库中按照章节顺序取出某一本书的所有章节，然后根据前端传过来的当前章节，返回这一章前后20章。

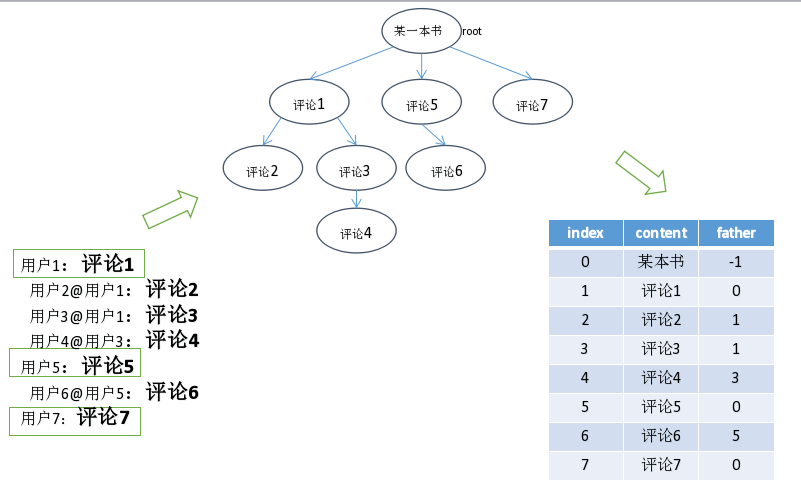
## 4.4 书籍详情模块

### 4.3.1 书籍详情页界面设计

### 4.3.2 书评和点赞功能实现

先说点赞功能，比较简单，只需要在每条评论的数据结构中上一个就点赞数量的字段就好了，每次用户点赞的时候，将字段加一。前端在用户点赞之后将点赞的图标切换至已点赞的状态。

书评的实现是基于树实现的。先来讨论下如何存储一本书的所有评论，如下图所示，假设



## 4.4 书城模块

### 4.4.1 书籍详情页界面设计

### 4.4.2 推荐专题和排行榜

微书在书城放置了书籍推荐的专题，我们会定期放置和话题相关的优秀书籍于其中。如果用户正好对此话题感兴趣，推荐专题可以帮助用户找到大量感兴趣的书籍。其实现原理即创建了单独的表，表中除了存放专题名字、发布时间，封面图片之外，有一个数组来存放小说基本信息表的书籍id，代表出现在这本专题的所有书籍。在专题页面中，用户可以点击每一本书查看书籍详情，也可以收藏这个专题，被收藏的专题将会出现用户个人中心，方便用户即使在专题不在架的情况下也能随时翻到以前自己收藏的专题。而要实现用户收藏专题，同样也需要在myAppUser中新增一个myTopic字段来记录用户专题数据

排行榜是指的起点小说排行榜和纵横小说排行榜，它们每个都包含总榜、玄幻、历史、言情、武侠、科技6类小说，而每个列别下的排行榜都包含这个分类下的排行前十的小说。排行版的实现前端上面没太多的难点，关于排行榜的数据来源，会在接下来的爬虫中详细说明。

## 4.5 个人中心模块

### 4.5.1 个人中心界面设计

### 4.5.2 个人信息

为了减少用户在注册时候的输入，除去密码和用户名以及邮箱三个必选项之外，其他都可以不填。用户注册完成之后，可以在个人中心点击自己的头像去完善个人信息。个人信息页面需要后端提供两个接口，一是获取用已有的个人信息getUserInfo，二是更新用户信息的接口updateUserInfo。用户查看信息和更新信息并没有分页面实现，二是在用户第一次进入页面即认为是查看个人信息，点击最下方的修改按钮，切换至修改个人信息状态，当用户信息点击提交，重新返回查看个人信息状态。

### 4.5.3 个人设置

个人设置主要为用户提供阅读器风格设置，以及书籍更新推送、清除缓存这些常用设置。所有这些设置在被更改之后都会被更新到用户的setting的对象中。在进入用户微书，获取这些设置，并应用到阅读中。

### 4.5.4 我的消息

个人消息是书评中其他用户回复当前用户的消息，微书没有提供直接的聊天功能，但是我们会整合用户收到以及发出去的评论以及回复，并在用户进入微书的时候给予用户消息提示。评论消息都是存放在小说基本信息表的comments字段中，在评论比较多的情况下去查找所有的评论找出用户有关的评论是一种不明智的做法，可以在用户发表书评的时候，提交接受评论的用户的id以及评论内容到这个用户的记录个人消息的字段中，这样即保证了消息的及时性，也避免了大量的查找。

1. 爬虫

微书的所有书籍信息均是从小说发布网站爬取的，这里有必要介绍爬虫实现原理。我们将微书的爬虫分为以下几个模块：爬虫配置、主程序、数据存储、工具类，下面是它们的详细介绍：

1. 爬虫配置

爬虫配置是根目录的config.js，该js定义数据库的链接信息，上传图片的七牛云服务器配置，以及小说发布网站地址和承载关键信息的html元素类名，这主要是为了应对小说源站点的更新和改版。实际的爬虫代码中从配置中读取这些信息，一旦源站改版了，也只需要改版这些关键信息，爬虫就可以就继续执行了。

**var** websiteConfig = [  
 {  
 **factionName**: **'大主宰'**,  
 **des**: **'xxxx'**,  
 **headerImage**: **'http://qiniu.andylistudio.com/myblog/images/dazhuzai.jpg'**,  
 **author**: **'烟雨江南'**,  
 **allResources**: [  
 {  
 **sourceName**: **'爱下电子书'**,  
 **url**: **'http://read.ixdzs.com/66/66485/'**,  
 **coreUrl**: **'http://read.ixdzs.com/66/66485/'**,  
 **des**: **'爱下电子书-大主宰，再给出的页面中罗列了所有的小说章节，在爬取的时候得分不同的情况'**,  
 **firstSign**: **'.catalog .chapter > a'**,  
 **inWhatAttr**: **'href'**,*//链接存储的属性* **secondSign**: **'.content'** }  
 ]  
 }  
];

1. 主程序

主程序是爬虫核心模块，它有分为定时控制模块、搜索模块、基本信息爬取模块、爬取章节内容模块、日志记录模块

定时控制模块：使用node-schedule包来控制爬虫执行时间，目前是每天晚上6点定时更新。下面是node-schedule的时间定义格式。

# ┬ ┬ ┬ ┬ ┬ ┬ │ │ │ │ │ | │ │ │ │ │ └ day of week (0 - 7) (0 or 7 is Sun) │ │ │ │ └───── month (1 - 12) │ │ │ └────────── day of month (1 - 31) │ │ └─────────────── hour (0 - 23) │ └──────────────────── minute (0 - 59) └───────────────────────── second (0 - 59, OPTIONAL)

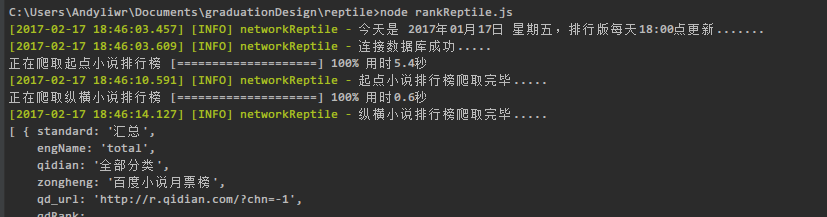
搜索模块：当爬虫去爬取书籍的时候，唯一需要传递的参数就是书籍名字。也就是只要指定了书籍名字，爬虫会自动去寻找书籍地址，并且爬取它的内容，并存储到数据库中。搜索模块就是用来寻找小说地址的。实现原理就是使用superagent去模拟搜索动作找到最合适的搜索结果，得到访问这本书的地址。搜索结果可能有两种一种是唯一搜索结果，也就是通过源站只找到一本相同名字的书籍，这种会直接返回此书籍的访问地址。另一种是拥有多个搜索结果的情况，对于这种，我们新增了一个子爬虫去爬取每一个符合结果的书籍，对比每一本书的热度值选取最高热度值的书籍作为搜索结果。  
**var** compareHotEp = **new eventproxy**();

compareHotEp.after(**'hasFinishedHot',** factionDisplayUrlArr.**length**, **function** (allHots) {  
 *//排序，找到最大值* allHots.**sort**(**function** (hot1, hot2) {  
 **return** hot2.**hot** - hot1.**hot**;  
 });  
 factionDisplayUrl = factionDisplayUrlArr[allHots[0].**idx**].**attribs**.**href**;  
 **var** factionUrl = factionDisplayUrl.replace(/www/, **'read'**);  
 factionUrl = factionUrl.replace(/\/d/, **''**);  
 logger.**info**(**'|'** + factionName + **'| 在 |爱下电子书| 的地址是：'** + factionUrl);  
 *getFactionList*(factionName, factionUrl, **'regular'**);  
});  
*//遍历每个结果项去获取热度数据*factionDisplayUrlArr.each(**function** (index, element) {  
 **var** thisUrl = $(element).attr(**'href'**);  
 **superagent**.get(thisUrl)  
 .end(**function** (err, res) {  
 **if** (err) {  
 logger.warn(**'获取 第'** + index + **'结果项 的热度数据失败，地址：'** + thisUrl);  
 compareHotEp.emit(**'hasFinishedHot'**, {**idx**: index, **hot**: 0});  
 **return**;  
 }  
 **var** $ = **cheerio**.load(res.**text**);  
 **var** hotValue = $(**'.d\_info .d\_ac li:nth-child(5)'**).**text**().replace(/热度：/, **''**);  
 compareHotEp.emit(**'hasFinishedHot'**, {**idx**: index, **hot**: hotValue});  
 });  
 });  
 **return**;

}

1. 基本信息爬取模块：用来获取这本书籍的书籍名、封面图片、作者、书籍描述、以及访问它的章节列表的地址。原理和搜索差不多，使用superagent去模拟访问搜索获得的书籍地址，使用cheerio去解析获取到的html，然后从Html元素中获取到正确的数据。
2. 章节列表爬取模块：章节列表和基本信息模块上的区别在于需要使用正则去过滤出每一章的章节数以及章节名，另一个区别是需要使用eventproxy并发的爬取每一章节的内容，但是并非每一个并发线程都会正确的返回，这个要做好判断，对于爬取的失败的章节执行重新爬取。

**var** sectionTitleReg = /(^\d+(\.)\*( )\*(第[零一二三四五六七八九十百千万0-9]+章))|(^第[零一二三四五六七八九十百千万0-9]+章)|(^[零一二三四五六七八九十百千万])|(^\d+(\.)\*[^零一二三四五六七八九十百千万0-9])/igm;  
**var** matchResult = aText.match(sectionTitleReg);  
*//当且仅当titile通过正则检测，并且章节数大于最新章节数，才会被加到待访问队列中***if** (matchResult != **null** && matchResult.**length** > 0) {  
 allSections.push({  
 **sectionNum**: chinese\_parseInt(myAppTools.*removeNaN*(matchResult[0])),  
 **sectionTitle**: aText.replace(sectionTitleReg, **''**).trim(),  
 **contentUrl**: url + $element.attr(**'href'**)  
 })  
}

5. 日志模块：这个模块主要是将爬虫执行输出的日志记录并写入到本地的log文件中，方便查找爬虫执行出现错误。我们使用logger这个工具，下面是looger执行的截图：

1. 数据库操作模块

此模块主要用于定义表结构，初始化数据库，以及存储爬虫的爬取结果。使用mongoose这个数据库操作包，对数据执行查询、更新以及删除。在这个模块对数据操作做了大量的封装，下面这些封装函数：

exports.*configLog* = *configLog*; *// 配置日志*exports.*initDB* = *initDB*; *// 初始化数据库*exports.*saveFaction* = *saveFaction*; *// 存储爬取到的章节内容*exports.*updateSectionList* = *updateSectionList*; *// 更新章节列表*exports.*getNewestSectionNum* = *getNewestSectionNum*; *// 获取最新章节*exports.*updateRank* = *updateRank*; *// 更新排行榜*exports.*emptyFaction* = *emptyFaction*; *// 清空某本小说*exports.*getSlipSection* = *getSlipSection*; *// 获取不连续的章节*

1. 工具类

这个js分装了一些常用的数据操作函数，包括了去除重复项、克隆对象、去除非数字、连接json对象、比较两个json是否相同等等

# 4 微书测试

## 4.1 测试项目

## 4.2 测试用例

# 参考文档

## 2.1 业务层面

微书的产品的定位即是为了满足用户无需下载，免费阅读的需求。适应的用户群体主要是城市上班族，在闲暇时光能借助微书不付费地读到自己喜欢的一些书籍，微信小程序带来的便捷体验和及时消息提醒，能够让用户在第一时间了解书籍的更新状态。

## 2.2 产品层面

微书主要由我的书架、书城、个人中心、H5阅读器，以及书籍详情页、登录注册这些模块构成。在设计上使用小程序推荐的底部tab切换，整体的色调选择了橙色作为主色调，页面切换使用了左右滑动动画。产品功能结构图如图1：

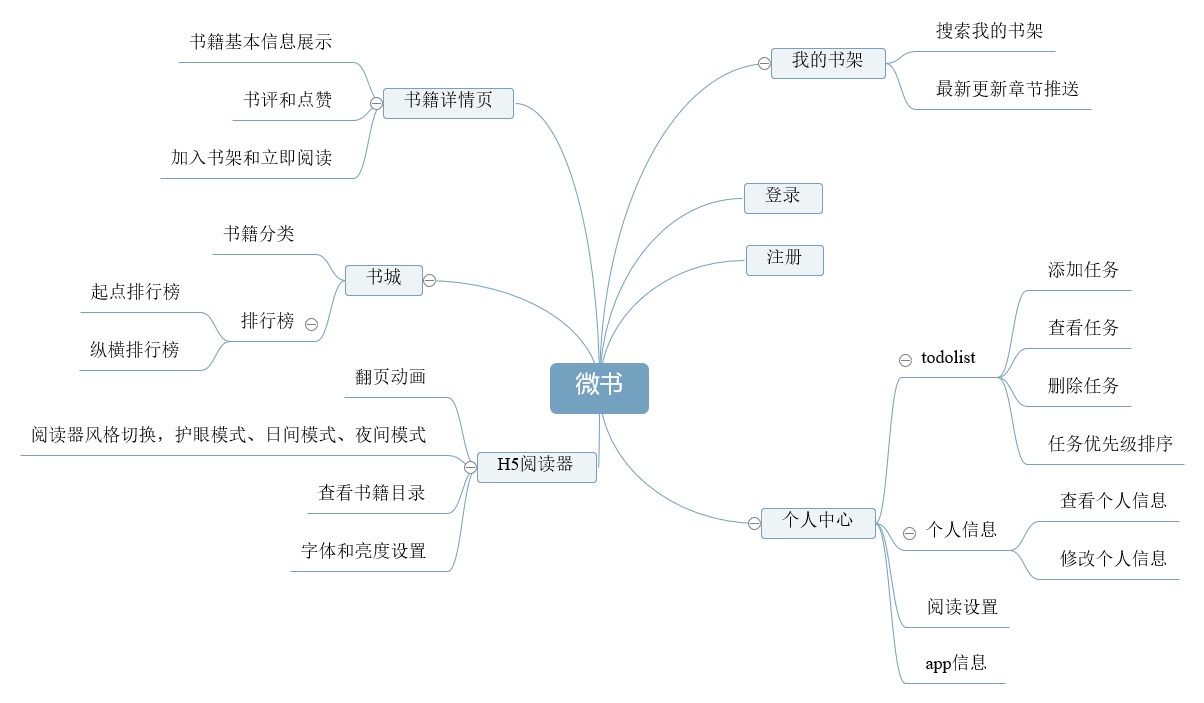
[[1]](#endnote-0)

图1. 产品功能结构图

## 2.3 技术层面

后端方面，微书选用了国外十分具有生产力的loopback框架，loopback一款nodejs的全栈框架，因为已经有小程序作为前端技术，所以该框架主要使用了其API接口管理以及它和数据库mongo的交互。数据库方面，由于存储主要都是书籍章节内容这些文本数据，使用mongo存储这些数据在数据库查询方面十分便捷，且mongo作为一个非结构化数据库，在存储书籍这些比较零散数据的时候具有很大的优势。前端方面，使用了最近比较热门的小程序作为实现前端的主要技术，小程序基于MVC的架构以及微信已给出的一些api使得它具有构建大型应用的技术前提，同时自身的便捷也使得基于微信小程序的产品具有更多为人所知所用的机会。

1. [↑](#endnote-ref-0)