1. 开发规划

1.1人员分工

角色	分工	人员
项目经理	项目主体设计、文档撰写、整合gateway、openfeign和hystrix、整合ouath2和keycloak	王昱
开发人员	测试模块、books微服务、文档撰写	胡锦浩
开发人员	文档撰写、orders微服务编写、kafka改写代码	姚聪

1.2开发环境及工具

开发语言：java

开发软件：idea，webstorm

开发框架：Spring Boot，Vue

2.总体设计

2.1项目介绍

改写第三次作业，主题功能一致，改写订单购买功能为kafka实现

前端接口：localhost:9000

后端接口：

BooksService: localhost:8400

OrdersService: localhost:8402

GatewayService: localhost:80

RegistryService: localhost:8761

后端接口文档（swagger）：

BooksService:localhost:8400/doc.html，另附”接口文档.md”文件

2.2 项目结构

```
├.idea
├authorization_service ------------------------ // 自定义认证服务器（使用keycloak后弃用）
│  ├.mvn
│  │  └wrapper
│  ├src
│  │  ├main
│  │  │  ├java
│  │  │  │  └bjtu
│  │  │  │      └edu
│  │  │  │          └authorization_service
│  │  │  │              ├config
│  │  │  │              └controller
│  │  │  └resources
│  │  └test
│  │      └java
│  │          └bjtu
│  │              └edu
│  │                  └authorization_service
│  └target
│      ├classes
│      │  └bjtu
│      │      └edu
│      │          └authorization_service
│      │              ├config
│      │              └controller
│      ├generated-sources
│      │  └annotations
│      ├generated-test-sources
│      │  └test-annotations
│      └test-classes
│          └bjtu
│              └edu
│                  └authorization_service
├books_service----------------------------- // 书籍微服务
│  ├.mvn
│  │  └wrapper
│  ├src
│  │  ├main
│  │  │  ├java
│  │  │  │  └edu
│  │  │  │      └bjtu
│  │  │  │          └books_service
```

```
│   │   │   │               ├─config
│   │   │   │               ├─controller
│   │   │   │               ├─entity
│   │   │   │               ├─repository
│   │   │   │               └─service
│   │   │   └─resources
│   │   └─test
│   │       └─java
│   │           └─edu
│   │               └─bjtu
│   │                   └─books_service
│   └─target
│       ├─classes
│       │  └─edu
│       │      └─bjtu
│       │          └─books_service
│       │              ├─config
│       │              ├─controller
│       │              ├─entity
│       │              ├─repository
│       │              └─service
│       ├─generated-sources
│       │  └─annotations
│       ├─generated-test-sources
│       │  └─test-annotations
│       └─test-classes
│           └─edu
│               └─bjtu
│                   └─books_service
├─gateway_service------------------------ // 网关微服务
│   ├─.mvn
│   │  └─wrapper
│   ├─src
│   │  ├─main
│   │  │  ├─java
│   │  │  │  └─bjtu
│   │  │  │      └─edu
│   │  │  │          └─gateway_service
│   │  │  │              └─controller
│   │  │  └─resources
│   │  └─test
│   │      └─java
│   │          └─bjtu
│   │              └─edu
│   │                  └─gateway_service
│   └─target
│       ├─classes
│       │  └─bjtu
│       │      └─edu
│       │          └─gateway_service
│       │              └─controller
│       ├─generated-sources
│       │  └─annotations
│       ├─generated-test-sources
│       │  └─test-annotations
│       └─test-classes
│           └─bjtu
│               └─edu
│                   └─gateway_service
├─orders_service------------------------ // 订单微服务，调用books微服务
│   ├─.mvn
```

```
│   │   └─wrapper
│   ├─src
│   │   ├─main
│   │   │   ├─java
│   │   │   │   └─bjtu
│   │   │   │       └─edu
│   │   │   │           └─orders_service
│   │   │   │               ├─controller
│   │   │   │               │   └─dto
│   │   │   │               ├─entity
│   │   │   │               ├─mapper
│   │   │   │               └─service
│   │   │   │                   └─impl
│   │   │   └─resources
│   │   │       ├─mapper
│   │   │       ├─static
│   │   │       └─templates
│   │   └─test
│   │       └─java
│   │           └─bjtu
│   │               └─edu
│   │                   └─orders_service
│   └─target
│       ├─classes
│       │   ├─bjtu
│       │   │   └─edu
│       │   │       └─orders_service
│       │   │           ├─controller
│       │   │           ├─entity
│       │   │           ├─mapper
│       │   │           └─service
│       │   │               └─impl
│       │   └─mapper
│       ├─generated-sources
│       │   └─annotations
│       ├─generated-test-sources
│       │   └─test-annotations
│       └─test-classes
│           └─bjtu
│               └─edu
│                   └─orders_service
├─registry_service------------------------ // 服务注册中心
│   ├─.mvn
│   │   └─wrapper
│   ├─src
│   │   ├─main
│   │   │   ├─java
│   │   │   │   └─edu
│   │   │   │       └─bjtu
│   │   │   │           └─registryservice
│   │   │   └─resources
│   │   └─test
│   │       └─java
│   │           └─edu
│   │               └─bjtu
│   │                   └─registryservice
│   └─target
│       ├─classes
│       │   └─edu
│       │       └─bjtu
│       │           └─registryservice
```

```
│       ├─generated-sources
│       │  └─annotations
│       ├─generated-test-sources
│       │  └─test-annotations
│       └─test-classes
│           └─edu
│               └─bjtu
│                   └─registryservice
│
└─users_service------------------------ // 自定义资源服务器（使用keycloak后弃用）
    ├─.mvn
    │  └─wrapper
    ├─src
    │  ├─main
    │  │  ├─java
    │  │  │  └─bjtu
    │  │  │      └─edu
    │  │  │          └─users_service
    │  │  │              ├─config
    │  │  │              └─controller
    │  │  └─resources
    │  └─test
    │      └─java
    │          └─bjtu
    │              └─edu
    │                  └─users_service
    └─target
        ├─classes
        │  └─bjtu
        │      └─edu
        │          └─users_service
        │              ├─config
        │              └─controller
        ├─generated-sources
        │  └─annotations
        ├─generated-test-sources
        │  └─test-annotations
        └─test-classes
            └─bjtu
                └─edu
                    └─users_service
```

2.3数据库设计:

Book表

对象 books @mytest (本地root) - 表

开始事务 文本 筛选 排序 导入 导出

ID	bookname	author	intro	price
1	红楼梦	曹雪芹	小说以贾、	50.56
2	三少爷的剑	古龙	《三少爷(	30.70
7	西游记	吴承恩	猴子取经	40.50
10	test	testAuthor	用于更新	12.50
11	book1	Author1	用于插入	50.00

Users表

开始事务 文本 筛选 排序

id	password	username
1	123456	admin
2	123456	test

Ordes表

开始事务 文本 筛选 排序 导入 导出

id	bid	date	uid	state
2	1	2021-06-09 10:27:49	1	0
6	1	2021-06-09 20:38:24	2	1
7	1	2021-06-09 21:51:17	2	1
8	1	2021-06-09 21:51:17	2	0
9	1	2021-06-09 21:51:18	2	0

3.功能实现

设计方案：通过spring cloud编写微服务，其中booksService、ordersService等业务功能的微服务只关注业务功能的实现，不负责安全和权限方面的编写。使用gateway做网关，统一入口的同时，在gateway上集成安全、授权等服务。

关于authorization_service和users_service：一开始我编写了authorization_service作为认证服务器，users_service作为资源服务器，提供用户的资源，之后学习使用了keycloak，便把这两个替换了。

3.1 微服务

1. registryService

使用eureka作为注册中心，提供服务注册和发现

2. booksService

实现书籍服务的增删改查，注册至注册中心

3. ordersService

通过openfeign调用booksService，实现订单的增删改查，注册至注册中心

4. gatewayService

统一入口，集成hytrix实现简单的熔断，集成ouath2实现安全

5.event_processor

对kafka消息队列中的消息进行处理，并将新的消息放入kafka消息队列

####

3.2 集成功能及实现

1. eureka注册中心实现

registryService 的pom.xml引入eureka-server依赖，配置8761端口，并在主程序入口通过注解@EnableEurekaServer开启eureka服务

2. 服务消费者和提供者注册至eureka

```
spring:
  application:
    name: BOOKS-SERVICE                      #设定微服务名
eureka:
  client:
    service-url:
      defaultZone: http://localhost:8761/eureka/      #服务注册中信地址
  instance:
    hostname: localhost
    prefer-ip-address: true          #将IP注册到服务注册中心
```

3. gateway集成

gatewayService 的pom.xml引入gateway依赖，配置80端口，集成hytrix编写hytrixController提供一个服务降级接口。

```
@RestController
public class hystrixController {
    @RequestMapping(value = "/downgrade",produces = "text/html;charset=UTF-8")
    public String downgrade(){
        return "<html><body><div style='width:800px;margin:auto;text-
align:center;font-size:24px'>服务器忙，请稍后重试</div></body></html>";
    }
}
```

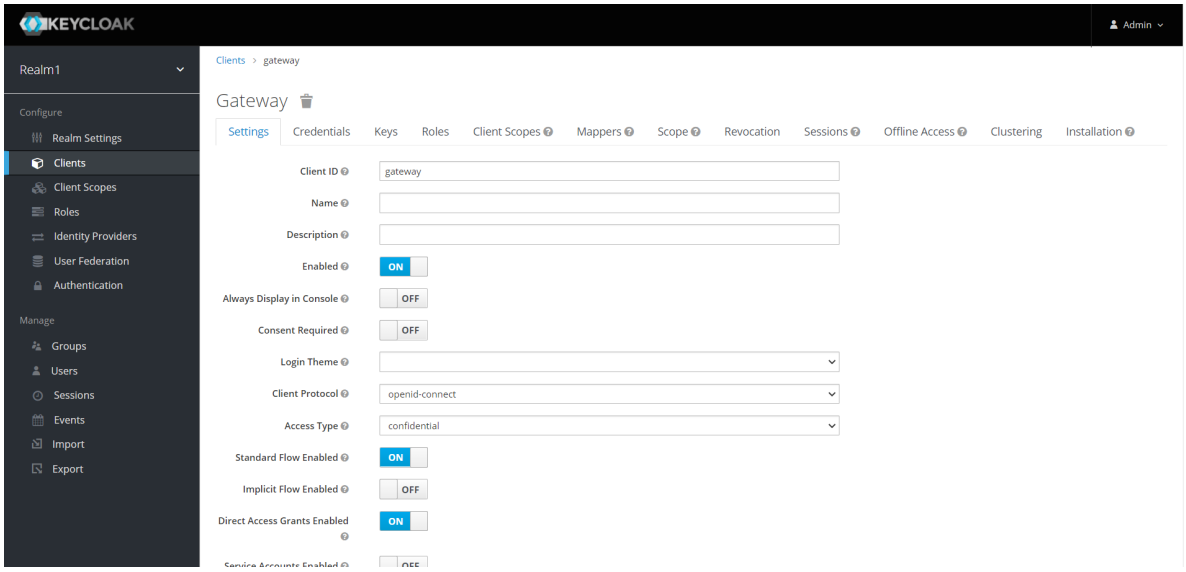
yml配置

```
spring:
  application:
    name: API-GATEWAY
  cloud:
    gateway:
      metrics:
        enabled: true
      discovery:
        locator:
          enabled: true
          lower-case-service-id: true
      routes:                          #为每个微服务提供对应的路由
        - id: BOOKS-SERVICE
          uri: lb://BOOKS-SERVICE
          predicates:
            - Path=/api/books/**
          filters:
            - StripPrefix=1
            - name: Hystrix            #为books微服务提供熔断降级
              args:
                name: fallbackcmd
                fallbackUri: forward:/downgrade
        - id: ORDERS-SERVICE
          uri: lb://ORDERS-SERVICE
          predicates:
            - Path=/api/orders/**
          filters:
            - StripPrefix=1
            - name: Hystrix
              args:
                name: fallbackcmd
                fallbackUri: forward:/downgrade
```

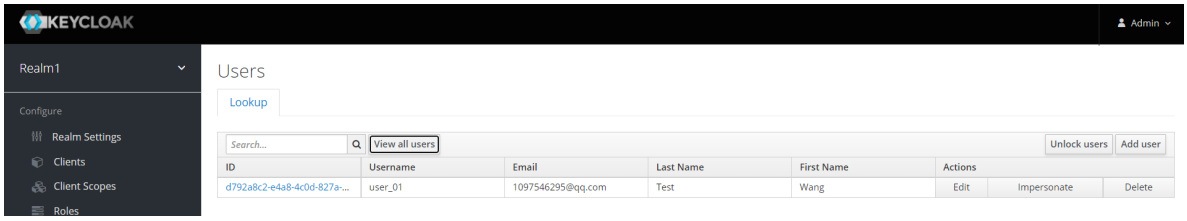
4. gateway集成oauth2，keycloak

1.配置keycloak

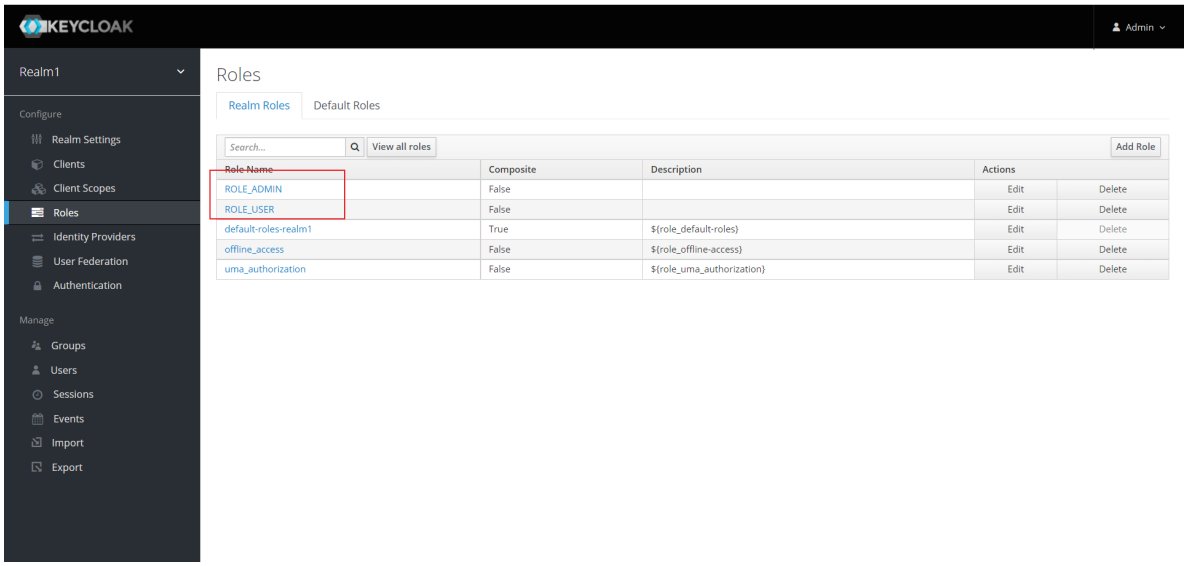
创建Realm1域，并创建client：gateway，用于保护gateway应用服务。



创建user_01用户



创建角色并分配给用户



2.引入oauth2依赖

3.配置

这里我直接把keycloak作为认证服务器和资源服务器，既提供用户认证，也提供用户资源信息。

```
spring:
  security:
    oauth2:
      client:
        provider:
          keycloak:
            issuer-uri: http://localhost:8080/auth/realms/Realm1
        registration:
          gateway:
```

```
            authorization-grant-type: authorization_code
            client-id: gateway
            client-secret: f1bf159f-1f5e-4969-9062-db4da0d17420
            provider: keycloak
            redirect-uri: http://localhost:80/login/oauth2/code/spring-gateway
            client-authentication-method: post
            scope: openid,profile,email
      resourceserver:
        jwt:
          jwk-set-uri: http://localhost:8080/auth/realms/Realm1/protocol/openid-
connect/certs
```

4.编写securityConfig

5.kafka集成

event_processor的相关设置：

yml配置：针对order、提交订单、付费注册了三个群组，并设置了kafka种producer和consumer的相关配置。

```
spring:
  application:
    name: EVENT-PROCESSOR

  kafka:
    producer:
      bootstrap-servers: 9092
      key-serializer: org.apache.kafka.common.serialization.StringSerializer
      value-serializer: org.springframework.kafka.support.serializer..JsonSerializer

    consumer:
      auto-offset-reset: earliest
      bootstrap-servers: 9092
      group-id: group-id
      key-deserializer: org.apache.kafka.common.serialization.StringDeserializer
      properties:
        spring:
          json:
            trusted:
              packages: '*'
      value-deserializer:
org.springframework.kafka.support.serializer.JsonDeserializer

    topic:
      order: orderTopic
      delivery: deliveryTopic
      pay: payTopic

server:
  port: 8086
```

Consumer配置：设置Consumer的相关配置

```
@Configuration
@EnableKafka
public class KafkaConsumerConfig {

    @Value("${spring.kafka.consumer.bootstrap-servers}")
```

```
    private String kafkaServer;

    @Value("${spring.kafka.consumer.group-id}")
    private String kafkaGroupId;

    @Bean
    public ConsumerFactory<String, Order> consumerConfig() {
        // TODO Auto-generated method stub
        Map<String, Object> config = new HashMap<>();
        config.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, kafkaServer);
        config.put(ConsumerConfig.GROUP_ID_CONFIG, kafkaGroupId);
        config.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG,
StringDeserializer.class);
        config.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
JsonDeserializer.class);
        //config.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG,
JsonDeserializer.class);
        //config.put(JsonDeserializer.TRUSTED_PACKAGES, "*");
        return new DefaultKafkaConsumerFactory<>(config, new StringDeserializer(), new
JsonDeserializer<>(Order.class));
    }

    @Bean
    public KafkaListenerContainerFactory<ConcurrentMessageListenerContainer<String,
Order>> kafkaListenerContainerFactory() {
        ConcurrentKafkaListenerContainerFactory<String, Order> listener = new
ConcurrentKafkaListenerContainerFactory<>();
        listener.setConsumerFactory(consumerConfig());
        return listener;
    }
}
```

Producer配置：设置Producer的相关配置：

```
@Configuration
public class KafkaProducerConfig {

    @Value("${spring.kafka.producer.bootstrap-servers}")
    private String kafkaServer;

    @Bean
    public ProducerFactory<String, Order> producerFactory() {
        Map<String, Object> config = new HashMap<>();
        config.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,kafkaServer);
        config.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
StringSerializer.class);
        config.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
JsonSerializer.class);

        return new DefaultKafkaProducerFactory<>(config);
    }

    @Bean
    public KafkaTemplate<String,Order> kafkaTemplate(){
        return new KafkaTemplate<>(producerFactory());
    }
```

DeliveryService：添加kafka监听器，收听订单生成的消息

```
@KafkaListener(topics = "${spring.kafka.topics.delivery}", groupId =
"${spring.kafka.consumer.group-id}")
public void process_delivery_events(Order order){
    order.setstate(0);
    order.setdate(new SimpleDateFormat("yyyy.MM.dd.HH.mm.ss").format(new Date()));

    this.kafkaTemplate.send(orderTopic,order);
}
```

PayService：添加kafka监听器，收听对订单进行付费的消息

```
@KafkaListener(topics = "${spring.kafka.topics.pay}", groupId =
"${spring.kafka.consumer.group-id}")
public void process_payment_events(Order order){
    order.setstate(1);
    order.setdate(new SimpleDateFormat("yyyy.MM.dd.HH.mm.ss").format(new Date()));

    this.kafkaTemplate.send(orderTopic, order);
}
```

OrderServiceImpl中的相关配置：

在该微服务中也要对Consumer和Producer进行和event_processor中两者一样的配置。

生成订单时，将对应的order放入kafka消息队列中：

```
@Override
public int addOne(Order order) {
    this.kafkaTemplate.send(deliverName,order);
    return orderMapper.addOne(order);
}
```

对订单付费时，将对应的order放入kafka消息队列中：

```
@Override
public int buyConfirm(Order order) {
    this.kafkaTemplate.send(payName,order);
    return orderMapper.updateOne(order);
}
```

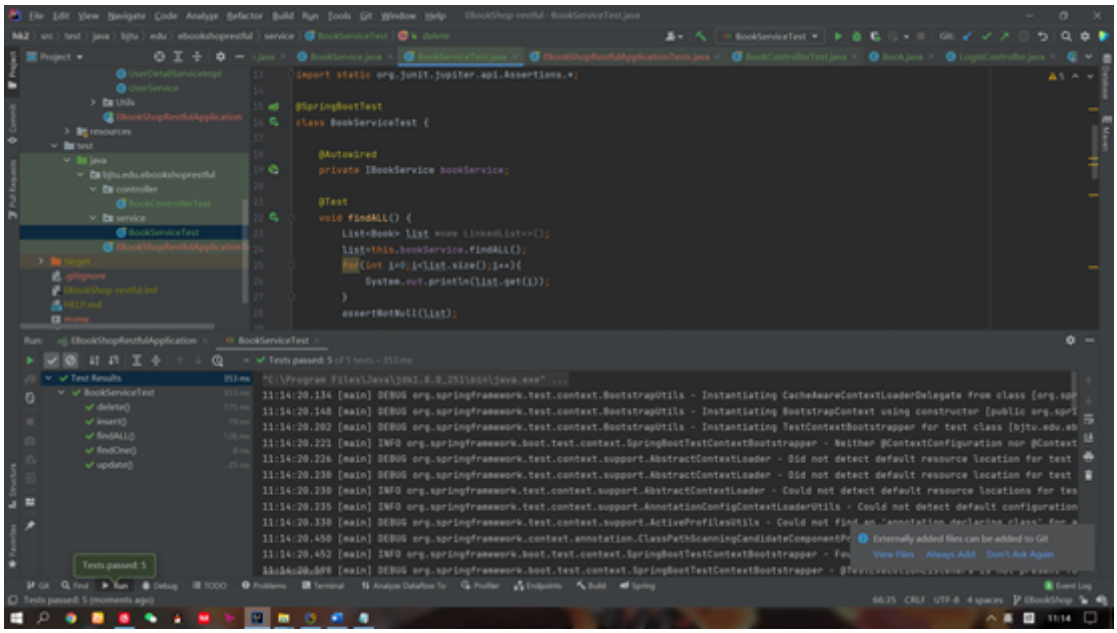
添加kafka监听器，接收相关操作处理后的返回消息：

```
@KafkaListener(topics = "${spring.kafka.topics.order}", groupId =
"${spring.kafka.consumer.group-id}")
public void getresult(Order order){
    System.out.println(order.getid()+"操作完成");
}
```

4 测试

4.1 单元测试

本项目针对Service层进行单元测试，实现了BookServiceTest类，分别初始化实例并实现了对数据库增删改查功能的测试，共进行了5个测试，通过5个，未通过0个，测试通过



4.2 集成测试

本项目针对Controller层进行集成测试，分别实现了针对BooksController,OrderController的测试类，对项目的增删改查功能进行测试，通过设置不同的边界值，共进行了9个测试，通过6个，未通过3个，测试通过

5 前端

5.1实现步骤

1.使用上次的HTML模板

2.通过vue-cli创建前端工程项目

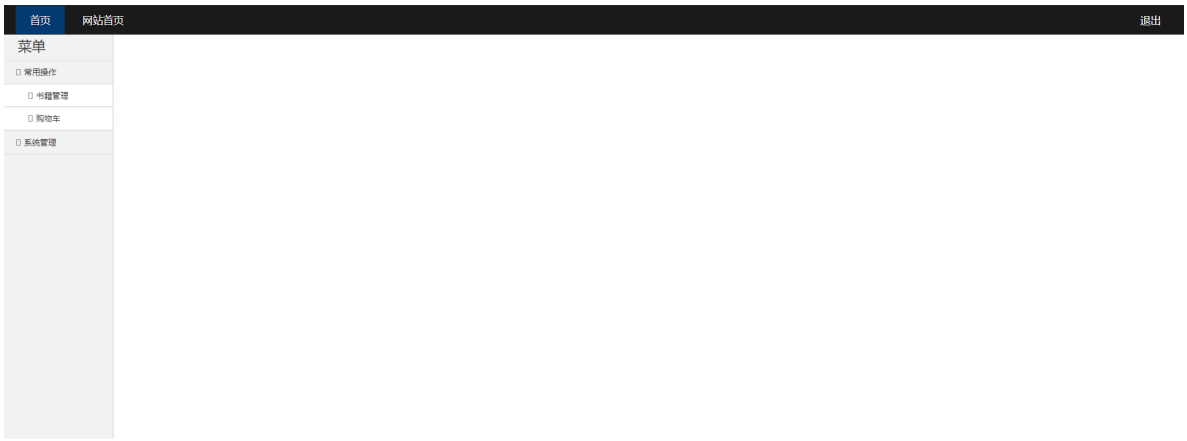
3.改造模板为vue形式，同时提出共同部分和特殊部分形成组件

4.通过vue-router实现前端导航

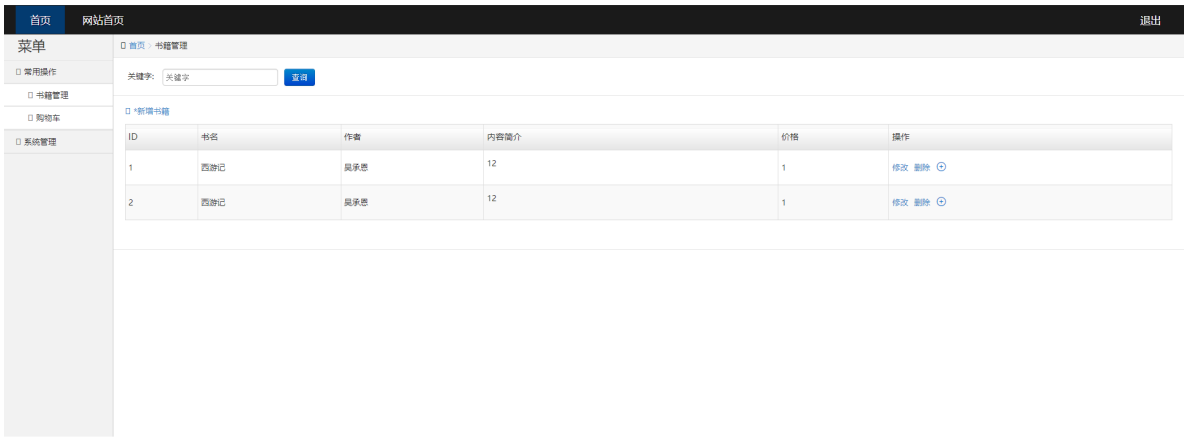
5.通过axios实现异步发送http请求

5.2界面

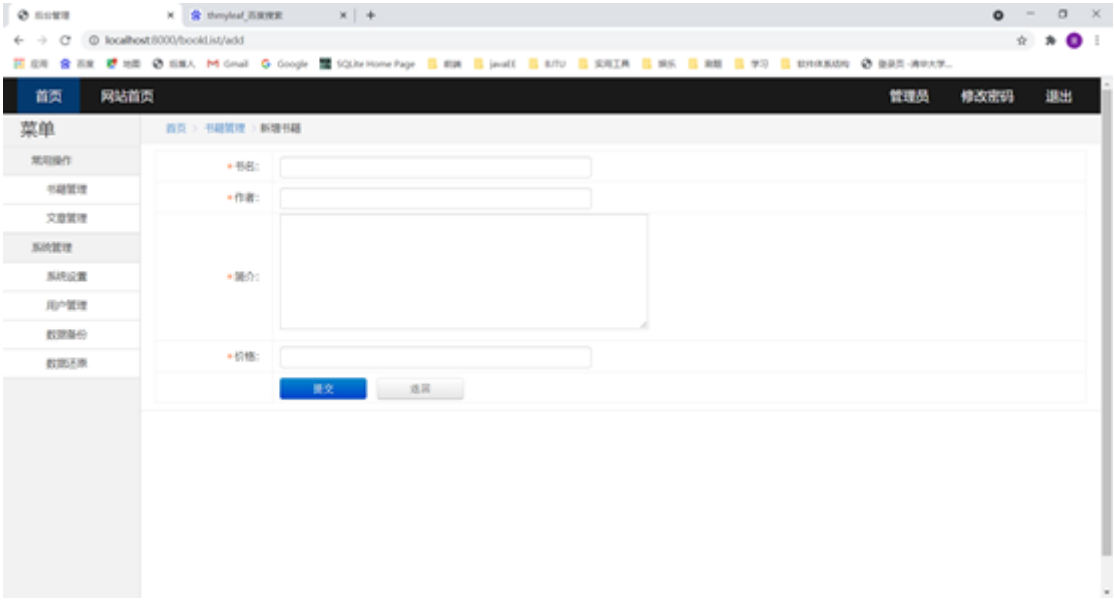
1 主界面(index.vue)



2书籍列表界面及加入订单(bookList.vue)



3增加、修改书籍界面(insert.vue,modify.vue)


首页
网站首页
管理员
修改密码
退出
菜单
常用操作
书籍管理
文章管理
系统管理
系统设置
用户管理
数据备份
数据还原
首页 / 书籍管理 / 新增书籍
书名:
作者:
简介:
价格:
提交
返回

4订单界面


首页
网站首页
退出
菜单
常用操作
书籍管理
购物车
系统管理
购物车
ID
书名
作者
简介
价格
操作
1
西游记
吴承恩
12
1
2
西游记
吴承恩
12
1
付款