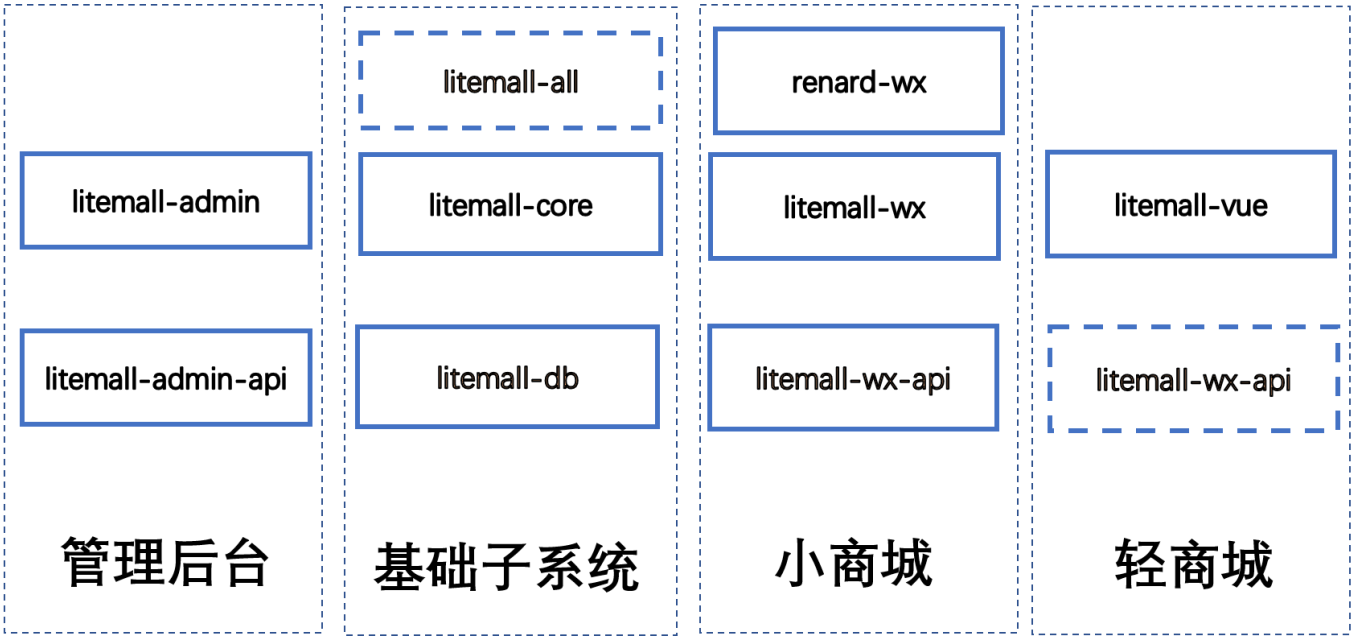
1. 系统架构

1.1 简介

litemall是一个简单的商场系统，基于现有的开源项目，重新实现一个完整的前后端项目，包含小程序客户端、移动客户端和网页管理端。



项目的架构是四个系统和九个模块：

- 基础系统子系统(platform)
  由数据库、litemall-core模块、litemall-db模块和litemall-all模块组成;
- 小商场子系统(wxmall，即weixin mall)
  由litemall-wx-api模块、litemall-wx模块和renard-wx模块组成;
- 轻商城子系统(mobmall，即mobile mall)
  由litemall-wx-api模块和litemall-vue模块组成。 注意，目前这里移动商城子系统的后端和小商场子系统是一样的。
- 管理后台子系统(admin)
  由litemall-admin-api模块和litemall-admin模块组成。

而九个模块的开发设计到三种技术栈：

- Spring Boot技术栈

采用IDEA开发工具，开发litemall-core、litemall-db、litemall-admin-api、 litemall-wx-api和litemall-all共五个模块；

- miniprogram（微信小程序）技术栈
  采用微信小程序开发工具，开发litemall-wx模块和renard-wx模块；
- Vue技术栈
  采用VSC开发工具，开发litemall-admin模块和litemall-vue模块。

1.1.1 项目特点

项目存在以下特点：

- 数据库方面，只是简单的表，表和表之间的依赖关系没有采用外键设计，而是依赖Java代码在service层面或者业务层面保证。这样做的好处是数据库频繁改动很方便，不会因为外键而导致数据库难以修改;
- 涉及三种技术栈，但是每种技术栈仅涉及最基础的技术；
  - 后端技术栈，仅涉及 Spring，Spring Boot, Spring MVC和Mybatis技术，其他后端技术暂时不采用;
  - 小程序技术栈，仅涉及miniprogram官方文档；
  - 前端技术栈，仅涉及vue, vuex, vue-route和element技术；
- 安全方面，仅采用最基本的代码，提供简单基本的安全服务;
- 性能方面，没有涉及内存数据库缓存功能，而是完全依赖MySQL;
- 对象存储服务方面，支持本地存储和第三方云存储方案。
- 消息通知方面，支持邮件通知、第三方云短信通知和微信模板通知；
- 部署方便，支持多服务部署和一键部署脚本；
- 文档全面，虽然还在开发中，但是规划中文档和代码注释一定会完成，帮助开发者理解项目。

总之，目前的系统只是为了学习技术和业务而开发的一个简单商场原型系统。虽然缺失很多企业级功能，但是是完整和合理的原型系统。

注意：

> 以上特点并不一定是优点。

1.2 系统功能

从业务功能上，目前由六个业务模块组成：

- 会员业务模块
- 商场业务模块
- 商品业务模块
- 推广业务模块
- 系统业务模块
- 配置业务模块

1.2.1 小商城功能

- 首页
- 专题列表、专题详情
- 分类列表、分类详情
- 品牌列表、品牌详情
- 新品首发、人气推荐
- 团购
- 搜索
- 商品详情
- 商品评价列表、商品评价
- 购物车
- 下单
- 个人
- 订单列表、订单详情
- 地址列表、地址添加、地址删除
- 收藏、足迹、关于

1.2.2 轻商城功能

目前还在开发中，不稳定

以下是准备完成的功能：

- 首页
- 专题列表、专题详情
- 分类列表、分类详情
- 品牌列表、品牌详情
- 新品首发、人气推荐
- 团购
- 搜索
- 商品详情

- 商品评价列表、商品评价
- 购物车
- 下单
- 个人
- 订单列表、订单详情
- 地址列表、地址添加、地址删除
- 收藏、足迹、关于

1.2.3 管理平台功能

- 会员管理
  - 会员管理
  - 收货地址
  - 会员收藏
  - 会员足迹
  - 搜索历史
  - 意见反馈
- 商城管理
  - 行政区域
  - 品牌制造商
  - 订单管理
  - 商品类目
  - 通用问题
  - 关键词
  - 渠道管理（待定）
- 商品管理
  - 商品列表
  - 商品上架
  - 商品编辑
  - 用户评论
- 推广管理
  - 广告管理
  - 专题管理
  - 团购规则
  - 团购活动
- 系统管理
  - 管理员
  - 对象存储
  - 权限管理

    - 定时任务（待定）
    - 操作日志
- 配置管理
    - 商场配置
    - 小程序配置
    - 运费配置
    - 订单配置
- 统计报表
    - 用户统计
    - 订单统计
    - 商品统计

1.3 项目技术

1.3.1 技术参考

1.3.1.1 Spring Boot技术

Spring Boot技术栈参考以下文档或者项目：

1. MySQL
   了解创建数据库和表、添加、查询、更新和删除即可。
2. Spring Boot 2.x
    - https://docs.spring.io/spring-boot/docs/2.1.5.RELEASE/reference/htmlsingle/#getting-started-introducing-spring-boot
    - https://docs.spring.io/spring-boot/docs/2.1.5.RELEASE/reference/htmlsingle/#using-boot-maven
      这里需要了解RestController, Service等注解，以及如何使用自动化配置。 Spring Boot支持很多功能，开发者使用时查阅。
3. Mybatis
    - http://www.mybatis.org/mybatis-3/
    - http://www.mybatis.org/mybatis-3/java-api.html
    - http://www.mybatis.org/mybatis-3/sqlmap-xml.html
      这里可以简单了解，而使用Mybatis Generator来生成Java代码使用即可。
4. Mybatis Generator
    - http://www.mybatis.org/generator/running/runningWithMaven.html
    - http://www.mybatis.org/generator/generatedobjects/results.html

- http://www.mybatis.org/generator/generatedobjects/exampleClassUsage.html

5. Mybatis PageHelper
   - https://github.com/pagehelper/Mybatis-PageHelper/blob/master/wikis/en/HowToUse.md

1.3.1.2 小程序技术

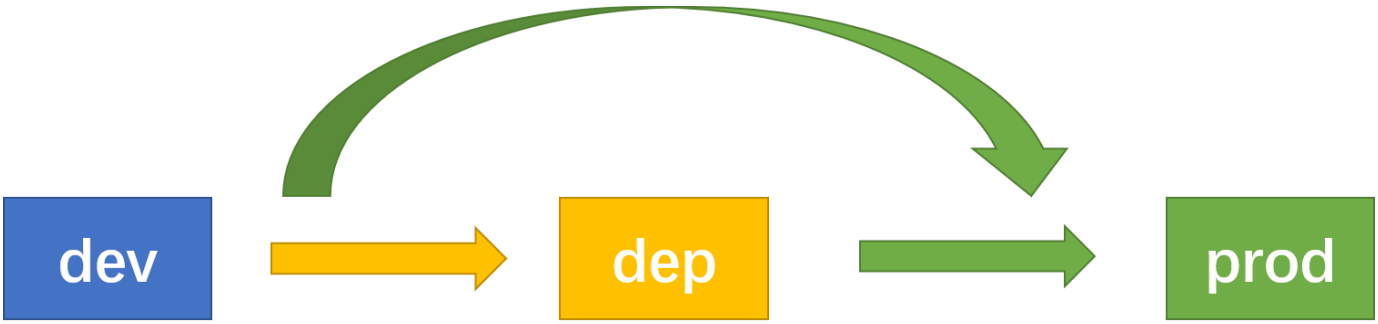
1. 小程序
   - https://developers.weixin.qq.com/miniprogram/dev/index.html
   - https://developers.weixin.qq.com/miniprogram/dev/component/
   - https://developers.weixin.qq.com/miniprogram/dev/api/
   - https://developers.weixin.qq.com/community/develop
     建议小程序方面遇到问题，可以到官方社区查找。
2. 微信支付
   - https://pay.weixin.qq.com/wiki/doc/api/wxa/wxa_api.php?chapter=7_3&index=1

1.3.1.3 Vue技术

1. Vue
   - https://cn.vuejs.org/index.html
2. Vant
   - https://youzan.github.io/vant/#/zh-CN/intro
3. Element
   - https://element.eleme.cn/#/zh-CN/component/installation
4. vue-element-admin
   - https://github.com/PanJiaChen/vue-element-admin
   - https://panjiachen.github.io/vue-element-admin-site/zh/

1.3.2 项目阶段

接下来，从项目的开发、部署（测试）和上线三个阶段介绍litemall。



首先需要明确的是三个不同阶段：

- dev

即develop或者development, 这里指开发阶段，通常代码是直接在本地编译、运行和测试。 此外，这里服务访问地址通常是localhost。这里的“用户”主要是指开发者本身。

- dep

即deploy或者deployment，这里指部署（测试阶段），通常代码已经编译打包运行在远程服务器中， 可以对外服务。此外，这里服务访问地址通常是IP地址。如果IP是公网IP，那么 部署以后就可以对外服务；如果是内网地址，那么只能内网访问。这里的“用户”主要是 指开发者本身、测试者；当然，如果是局域网或者不介意IP访问的，那么这里的“用户” 也可能是最终使用者用户。

- prod

即product或者production，这里指上线阶段，通常也是代码编译打包运行在远处服务器中可以对外服务。 此外，这里服务访问地址通常是域名地址，同时端口是80web端口。上线以后直接面向的是最终用户。 虽然服务的代码本身和dep是完全一样的，但是考虑到场景的不同，上线阶段可能在运行环境方面需要做 调整，例如采用反向代理屏蔽内部实际项目结构。此外，最大的不同应该是上线环境下要使用域名和80端口， 而部署阶段则更为自由。

其次，需要明确的是，这里划分三种阶段不是简单的文档说明，还直接影响项目本身的行为 和代码编译结果，因此开发者需要清晰的了解；

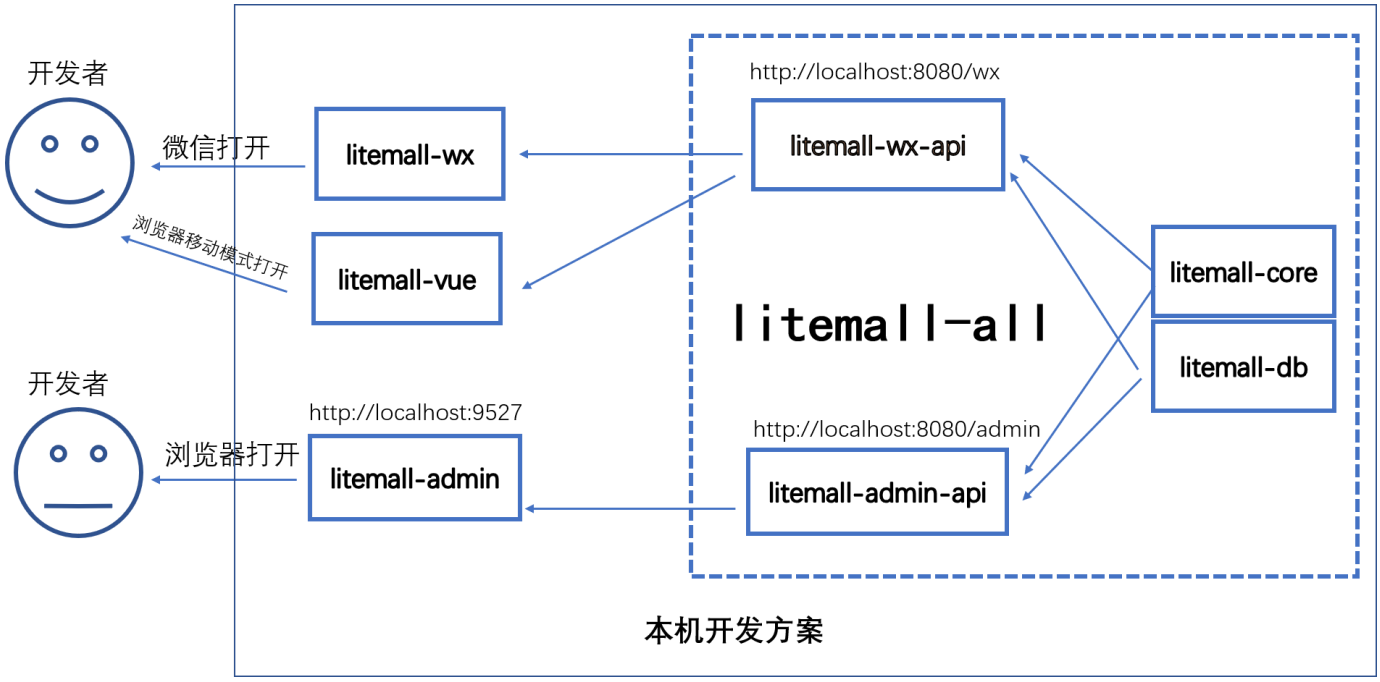
最后，其实dep和prod不存在先后关系。例如，如果开发者已经存在域名和生产环境，可以直接跳过dep阶段，而直接部署在线上环境中。因此有些时候，这里部署和上线是一个阶段。

当然，这里仍然建议先dep后prod，是因为对于第一次开发而言，先dep阶段可以避免对域名、https证书等非业务相关工作的干扰。

此外，有些业务功能（例如微信支付）必须是域名访问，那么开发和部署阶段可以先采用模拟或跳过的形式， 先不开发和测试这样业务功能，等其他功能开发完毕和部署测试成功以后，再来开发这些线上环境才能 运行的功能，此时会有一个好的基础。

接下来，分别从开发阶段、部署阶段和上线阶段三种阶段，分别介绍不同的方案实践要点。

1.4 开发方案



如图所示，当前开发阶段的方案：

- MySQL数据访问地址 `jdbc:mysql://localhost:3306/litemall`
- litemall-wx-api后端服务地址 `http://localhost:8080/wx` ，数据则来自MySQL
- litemall-admin-api后端服务地址 `http://localhost:8080/admin` ,数据则来自MySQL
- litemall-admin前端访问地址 `http://localhost:9527` , 数据来自litemall-admin-api
- litemall-wx没有前端访问地址，而是直接在微信小程序工具上编译测试开发，最终会部署到微信官方平台（即不需要自己部署web服务器），而数据则来自litemall-wx-api

1.4.1 数据库

数据库环境设置过程如下：

1. 安装MySQL;
2. 创建数据库、用户权限、数据库表和测试数据; 数据库文件存放在litemall-db/sql文件夹中，请开发者在MySQL中 按照顺序运行以下脚本:
   - litemall_schema.sql，用于创建数据库、用户和权限;
   - litemall_table.sql，用于创建表;
   - litemall_data.sql，用于导入测试数据。

注意：

> 建议采用命令行或者MySQL Workbench。如果采用Navicat可能导入失败。

如果开发者运行litemall_schema.sql失败，可以打开该文件：

```
drop database if exists litemall;
drop user if exists 'litemall'@'localhost';
create database litemall default character set utf8mb4 collate utf8mb4_unic
use litemall;
create user 'litemall'@'localhost' identified by 'litemall123456';
grant all privileges on litemall.* to 'litemall'@'localhost';
flush privilege
```

可以看到几个命令，用于创建数据库、用户和访问权限，因此开发者可以利用 命令或者工具完成这里的功能即可。

1.4.2 Spring Boot开发环境

1. 安装JDK8
2. 安装Maven
3. 安装Git（可选）
4. 安装IDEA Community，建议安装Maven插件和Git插件。
   这里IDEA社区版即可，不要求IDEA商业版。
   Eclipse没有试过，但应该也是可行的。
5. IDEA导入本项目
6. 采用Maven命令安装依赖库
   例如：

   ```
   cd litemall
   mvn install
   ```

   或者采用IDEA的Maven插件安装本项目依赖库，点击 `install`

7. 采用Maven命令编译本项目
   例如：

   ```
   cd litemall
   mvn compile
   ```

   此时可以看到，litemall-wx-api等模块多了target文件夹，里面是编译出的文件。

或者采用IDEA的Maven插件编译本项目，点击`compile`
如果采用IDEA也可以跳过当前步骤，直接步骤8（因为运行时会自动编译再运行）。

8. 采用Maven命令运行本项目的litemall-all
   例如：

```
cd litemall/litemall-all
mvn spring-boot:run
```

如果采用IDEA，则litemall-all模块的Application类 右键`Run Application.main()`方式运行该模块，

打开浏览器，输入

```
http://localhost:8080/wx/index/index
http://localhost:8080/admin/index/index
```

如果出现JSON数据，则litemall-all模块运行正常。

注意：

> 1. 上述步骤中，既介绍了Maven命令方式，也介绍了IDEA方式，
>   但是建议开发者开发阶段采用IDEA。
> 2. 上述步骤只是一种实践方式，开发者可不拘泥于这些步骤，多实践。
>   当然，如果开发者不采用这里步骤而出现问题，请自行解决。
> 3. 开发者使用IDEA导入项目或者运行项目时可能会出现**软件卡顿**的现象，这通常是litemall-admin或者litemall-vue的
>   node_modules文件夹内自动下载了大量的依赖库，当IDEA尝试索引该文件夹内的大量文件时
>   则出现IDEA卡顿的现象，具体解决方式可以参见FAQ

1.4.3 微信小程序开发环境

1. 安装微信小程序开发工具；
2. 导入本项目的litemall-wx模块(或者renard-wx模块)文件夹；
3. 编译前，请确定litemall-all模块已经运行，而litemall-wx模块的config文件夹中的api.js已经设置正确的后端数据服务地址；

4. 点击 `编译` ，如果出现数据和图片，则运行正常

注意：

> 开发者编译以后，可以看到图片和数据，但是采用微信登录或者微信支付是肯定会失败的。 原因是这里的配置信息没有正确设置，例如appid，具体详细配置见1.4.5节。

1.4.4 Vue开发环境

1. 安装nodejs
2. 安装依赖库

```
cd litemall/litemall-admin
npm install -g cnpm --registry=https://registry.npm.taobao.org
cnpm install
```

3. 编译并运行

```
cnpm run dev
```

然后，打开浏览器，输入 `http://localhost:9527` 。 如果出现管理后台登录页面，则表明管理后台的前端运行正常；

4. 请确定litemall-all模块已经运行，然后点击 `登录` ，如果能够成功登录，则表明管理后台的前端和后端对接成功，运行正常。

本项目采用VSC（Visual Studio Code）开发litemall-admin模块，开发者也可以采用其他熟悉的IDE。

1.4.5 项目配置

当安装好Spring Boot开发环境、Vue开发环境和小程序开发环境以后，启动相应的模块，已经可以看到一些数据或者效果。 但是还有一些特性或者功能没有启动，这是因为需要开发者进行配置才能正确启用。

项目配置结构

1. 管理后台前端，即litemall-admin模块，配置文件在litemall-admin中，存在三个配置文件 `env.development`，`env.deployment` 和 `.env.production`。这里面配置信息都是一样，最主要的配置是 `VUE_APP_BASE_API`，即管理后台的服务根地址。
   - 开发阶段，开发者运行命令 `cnpm run dev`，这里就会采用 `env.development` 配置文件；
   - 部署阶段，当开发者运行命令 `cnpm run build:dep`，这里就会采用 `env.deployment` 配置文件；
   - 上线阶段，当开发者运行命令 `cnpm run build:prod`，这里就会采用 `.env.production` 配置文件。
2. 小商场前端，即litemall-wx模块，配置文件是 `litemall-wx/project.config.json` 和 `litemall-wx/api.js`。 这里面最主要的配置信息是 `project.config.json` 中的 `appid`，开发者需要设置自己申请的appid； 以及 `apis.js` 中的 `WxApiRoot`，即小商场服务根地址。

```
// 本机开发时使用
 var WxApiRoot = 'http://localhost:8080/wx/';
// 局域网测试使用
// var WxApiRoot = 'http://192.168.0.101:8080/wx/';
// 云平台部署时使用
// var WxApiRoot = 'http://122.51.199.160:8080/wx/';
// 云平台上线时使用
// var WxApiRoot = 'https://www.menethil.com.cn/wx/';
```

3. 管理后台后端和小商城后端，即多个Spring Boot模块，配置文件是每个模块的 `litemall-xx/src/main/java/resources` 的 `application.yml` 和 `application-xx.yml` 配置文件。这里会发现每个模块都会有两个配置文件，但是实际上当前模块的配置信息 都是在 `application-xx.yml` 文件中，而 `application.yml` 文件仅仅用于引入其他模块的配置文件。
   例如litemall-all模块的 `application.yml` 的内容是

```
spring:
    profiles:
        active: db, core, admin, wx
    message:
        encoding: UTF-8
```

因此启动litemall-all模块时，程序首先加载litemall-all的 `application.yml`，然后通过 `spring.profiles.active` 信息 再次依次加载 `application-db.yml`，

`application-core.yml`，`application-admin.yml` 和 `application.yml-wx` 四个配置文件。

这里后端服务模块的配置如下所示。

1.4.5.1 日志配置

如果开发者启动litemall-all模块，则需要配置该模块的 `logback-spring.xml` 文件

```
    <logger name="org.mybatis" level="ERROR" />
    <logger name="org.springframework" level="ERROR" />
    <logger name="org.linlinjava.litemall.core" level="DEBUG" />
    <logger name="org.linlinjava.litemall.db" level="DEBUG" />
    <logger name="org.linlinjava.litemall.admin" level="DEBUG" />
    <logger name="org.linlinjava.litemall.wx" level="DEBUG" />
    <logger name="org.linlinjava.litemall" level="DEBUG" />
```

具体如何配置，请自行学习Spring Boot的日志配置和logback日志配置。

`org.linlinjava.litemall.core` 定义litemall-core模块的日志级别 `org.linlinjava.litemall.db` 定义litemall-db模块的日志级别 `org.linlinjava.litemall.wx` 定义litemall-wx-api模块的日志级别 `org.linlinjava.litemall.admin` 定义litemall-admin-api模块的日志级别 `org.linlinjava.litemall` 而定义litemall所有后端模块的日志级别

当然，如果开发者这里启动litemall后端模块级别是DEBUG时，可能会发现并没有很多日志， 这是因为代码内部没有写很多日志，开发者可以根据需要添加。

注意：

> 如果开发者独立启动litemall-wx-api模块，那么则需要配置litemall-wx-api模块的 日志配置方式。

1.4.5.2 数据库连接配置

在litemall-db模块的 `application-db.yml` 文件中配置数据库连接和druid：

```
spring:
  datasource:
    druid:
      url:  jdbc:mysql://localhost:3306/litemall?useUnicode=true&characterEn
```

```
        driver-class-name:  com.mysql.jdbc.Driver
        username:  litemall
        password:  litemall123456
        initial-size:  10
        max-active:  50
        min-idle:  10
        max-wait:  60000
        pool-prepared-statements:  true
        max-pool-prepared-statement-per-connection-size:  20
        validation-query:  SELECT 1 FROM DUAL
        test-on-borrow:  false
        test-on-return:  false
        test-while-idle:  true
        time-between-eviction-runs-millis:  60000
        filters:  stat,wall
```

1.4.5.3 微信登录配置

微信登录需要配置两个地方， 首先是小商场前端litemall-wx模块（或renard-wx模块）中`project.config.json`文件的appid 其次是小商场后端litemall-core模块的`application-core.yml`文件：

```
litemall:
  wx:
    app-id: wxa5b486c6b918ecfb
    app-secret: e04004829d4c383b4db7769d88dfbca1
```

这里的`app-id`和`app-secret`需要开发者在微信公众平台注册获取。

注意

> 这里开发者可能会疑惑：小商场后端应该配置在litemall-wx-api模块的`application-wx.yml`文件更合适。 这里放置在`application-core.yml`文件中是因为litemall-core模块也依赖小程序appid配置信息。

1.4.5.4 微信支付配置

在litemall-core模块的`application-core.yml`文件中配置微信支付：

```
litemall:
```

```
  wx:
    mch-id: 111111
    mch-key: xxxxxx
    notify-url: https://www.example.com/wx/order/pay-notify
```

这里的 `mch-id` 和 `mch-key` 需要开发者在微信商户平台注册获取。

而这里的 `notify-url` 则应该是项目上线以后微信支付回调地址，当微信支付成功或者失败， 微信商户平台将向回调地址发生成功或者失败的数据，因此需要确保该地址是 litemall-wx-api模块的WxOrderController类的payNotify方法所服务的API地址。

开发阶段可以采用一些技术实现临时外网地址映射本地，开发者可以百度关键字“微信 内网穿透”自行学习。

1.4.5.5 邮件通知配置

在litemall-core模块的 `application-core.yml` 文件中配置邮件通知服务：

```
litemall:
  notify:
    mail:
      # 邮件通知配置,邮箱一般用于接收业务通知例如收到新的订单，sendto 定义邮件接收者，通
      enable: false
      host: smtp.exmail.qq.com
      username: ex@ex.com.cn
      password: XXXXXXXXXXXXX
      sendfrom: ex@ex.com.cn
      sendto: ex@qq.com
```

配置方式： 1. 邮件服务器开启smtp服务 2. 开发者在配置文件中设置 `enable` 的值 `true` ，然后其他信息设置相应的值。 这里只测试过QQ邮箱，开发者需要自行测试其他邮箱。

应用场景： 目前邮件通知场景也很简单，就是用户下单以后系统会自动向 `sendto` 用户发送一封邮件，告知用户下单的订单信息。 以后可能需要继续优化扩展。当然，如果不需要邮件通知订单信息，可以默认关闭即可。

验证配置成功： 当配置好信息以后，开发者可以运行litemall-core模块的 `MailTest` 测试类， 独立发送邮件，然后登录邮箱查看邮件是否成功接收。

1.4.5.6 短信通知配置

在litemall-core模块的 `application-core.yml` 文件中配置短信通知服务：

```
litemall:
  notify:
    # 短消息模版通知配置
    # 短信息用于通知客户，例如发货短信通知，注意配置格式：template-name，template-ter
    sms:
      enable: false
      # 如果是腾讯云短信，则设置active的值tencent
      # 如果是阿里云短信，则设置active的值aliyun
      active: tencent
      sign: litemall
      template:
        - name: paySucceed
          templateId: 156349
        - name: captcha
          templateId: 156433
        - name: ship
          templateId: 158002
        - name: refund
          templateId: 159447
      tencent:
        appid: 111111111
        appkey: xxxxxxxxxxxxx
      aliyun:
        regionId: xxx
        accessKeyId: xxx
        accessKeySecret: xxx
```

配置方式： 1. 腾讯云短信平台或者阿里云短信平台申请，然后设置四个场景的短信模板； 2. 开发者在配置文件设置 `enable` 的值 `true` ，设置 `active` 的值 `tencent` 或 `aliyun` 3. 然后配置其他信息，例如腾讯云短信平台申请的appid等值。 这里只测试过腾讯云短信平台和阿里云短信平台，开发者需要自行测试其他短信云平台。

应用场景： 目前短信通知场景只支持支付成功、验证码、订单发送、退款成功四种情况。 以后可能需要继续优化扩展。

验证配置成功： 当配置好信息以后，开发者可以litemall-core模块的 `SmsTest` 测试类中设置手机号和 模板所需要的参数值，独立启动 `SmsTest` 测试类发送短信，然后查看手机是否成功接收短信。

短信模板参数命名： 这里存在一个问题，即腾讯云短信的官方平台中申请短信模板格式的模板参数是数组， 例如“你好，验证码是{0}，时间是{1}"; 而阿里云短信的官方平台中申请短信模板的模板参数是JSON, 例如“你好，验证码是{param1}，时间是{param2}"。 为了保持当前代码的通用性，本项目采用数组传递参数，而对阿里云申请模板的参数做了一定的假设： 1. 腾讯云模块参数，申请模板时按照官方设置即可，例如“你好，验证码是{0}，时间是{1}"; 2. 阿里云模板参数，本项目假定开发者在官方申请的参数格式应该采用"{ code: xxx, code1: xxx, code2: xxx }"， 例如“你好，验证码是{code}，时间是{code1}"。开发者可以查看 `AliyunSmsSender` 类的 `sendWithTemplate` 方法的 源代码即可理解。如果觉得不合理，可以自行调整相关代码。

1.4.5.8 物流配置

物流配置是查询商品物流信息，这里主要是基于第三方快递鸟服务。

在litemall-core模块的 `application-core.yml` 文件中配置快递鸟物流服务：

```
litemall:
  notify:
  # 快鸟物流查询配置
  express:
    enable: false
    appId: "XXXXXXXXX"
    appKey: "XXXXXXXXXXXXXXXXXXXXXXXXX"
    vendors:
    - code: "ZTO"
      name: "中通快递"
    - code: "YTO"
      name: "圆通速递"
    - code: "YD"
      name: "韵达速递"
    - code: "YZPY"
      name: "邮政快递包裹"
    - code: "EMS"
      name: "EMS"
    - code: "DBL"
      name: "德邦快递"
    - code: "FAST"
      name: "快捷快递"
    - code: "ZJS"
      name: "宅急送"
    - code: "TNT"
      name: "TNT快递"
    - code: "UPS"
      name: "UPS"
    - code: "DHL"
      name: "DHL"
    - code: "FEDEX"
      name: "FEDEX联邦(国内件)"
    - code: "FEDEX_GJ"
```

```
34        name: "FEDEX联邦(国际件)"
```

配置方式： 1. 快递鸟平台申请； 2. 开发者在配置文件设置 `enable` 的值 `true` ，然后其他信息设置 快递鸟平台中的appId和appKey。

应用场景： 小商场查询订单详情时，如果商品已发货，小商城后端会返回详细物流信息。

验证配置成功： 当配置好信息以后，开发者可以litemall-core模块的 `ExpressTest` 测试类中设置快递公司编码和 真实测试快递单号，独立启动 `ExpressTest` 测试类查询物流信息。

注意：

> 一部分快递公司（例如顺丰速运、申通快递等）的轨迹查询在开发环境下不支持， 具体支持情况或者使用限制请阅读官方资料

1.4.5.9 对象存储配置

对象存储，即存储和下载文件。

在litemall-core模块的 `application-core.yml` 文件中配置对象存储服务：

- 本地对象存储配置
  如果开发者采用当前服务器保存上传的文件，则需要配置：

```
litemall:
storage:
  # 当前工作的对象存储模式，分别是local、aliyun、tencent、qiniu
  active: local
  # 本地对象存储配置信息
  local:
    storagePath: storage
    # 这个地方应该是wx模块的WxStorageController的fetch方法对应的地址
    address: http://localhost:8080/wx/storage/fetch/
```

配置方式： 配置文件设置 `active` 的值 `local` ，表示当前对象存储模式是本地对象存储； 而 `storagePath` 是上传文件保存的路径； `address` 则是访问文件的对外路径。

- 阿里云对象存储配置

```
litemall:
  storage:
    # 当前工作的对象存储模式，分别是local、aliyun、tencent、qiniu
    active: aliyun
    aliyun:
      endpoint: oss-cn-shenzhen.aliyuncs.com
      accessKeyId: 111111
      accessKeySecret: xxxxxx
      bucketName: litemall
```

配置方式： 1. 阿里云官网注册 2. 配置文件设置 `active` 的值 `aliyun` ，表示当前对象存储模式是阿里云对象存储；

- 腾讯云对象存储配置

```
litemall:
  storage:
    # 当前工作的对象存储模式，分别是local、aliyun、tencent、qiniu
    active: tencent
    # 腾讯对象存储配置信息
    # 请参考 https://cloud.tencent.com/document/product/436/6249
    tencent:
      secretId: 111111
      secretKey: xxxxxx
      region: xxxxxx
      bucketName: litemall
```

配置方式： 1. 腾讯云官网注册 2. 配置文件设置 `active` 的值 `tencent` ，表示当前对象存储模式是腾讯云对象存储；

- 七牛云对象存储配置

```
litemall:
  storage:
    # 当前工作的对象存储模式，分别是local、aliyun、tencent、qiniu
    active: qiniu
    # 七牛云对象存储配置信息
    qiniu:
      endpoint: http://pd5cb6ulu.bkt.clouddn.com
      accessKey: 111111
      secretKey: xxxxxx
      bucketName: litemall
```

配置方式： 1. 七牛云官网注册 2. 配置文件设置 `active` 的值 `qiniu` ，表示当前对象存储模式是七牛云对象存储；

1.4.5.10 其他配置

除上述配置信息，本项目还存在其他配置。 目前仅采用默认值即可，开发者可以自己实践或者扩展新的配置信息。

1.5 部署方案

在1.4节中介绍的是开发阶段时一些关键性开发流程。本节将介绍代码开发成功以后开始部署项目时一些关键性流程。

首先，需要明确的是开发时项目使用的服务地址是本地地址，即localhost；而部署时则应该根据具体情况设置合理的服务器地址和端口。

其次，需要明确的是各模块之间的关系：

- litemall-wx-api模块会包含litemall-core模块和litemall-db模块，部署在服务器中
- litemall-admin-api模块会包含litemall-core模块和litemall-db模块，部署在服务器中
- litemall-all模块则会包装litemall-wx-api模块和litemall-admin-api模块；
- litemall-wx模块部署在微信开发者工具中，此外数据API地址指向litemall-wx-api所在服务qi地址
- litemall-admin编译出的静态文件放在web服务器或者tomcat服务器，此外服务器地址设置指向3中litemall-admin-api所在地址

最后，**如果项目部署云服务器，则根据开发者的部署环境在以下文件中或代码中修改相应的配置。**

1. MySQL数据库设置合适的用户名和密码信息；
2. 后端服务模块设置合适的配置信息；
3. 小商场前端litemall-wx模块 `config/api.js` 的 `WxApiRoot` 设置小商场后端服务的服务地址；
4. 管理后台前端litemall-admin模块 `.env.deployment` 中的 `VUE_APP_BASE_API` 设置管理后台后端服务的服务地址。

实际上，最终的部署方案是灵活的：

- 可以是同一云服务器中安装一个Spring Boot服务，同时提供litemall-admin、litemall-admin-api和litemall-wx-api三种服务
- 可以单一云服务器中仅安装一个tomcat/nginx服务器部署litemall-admin静态页面分发服务，然后部署两个Spring Boot的后端服务；
- 也可以把litemall-admin静态页面托管第三方cdn，然后开发者部署两个后端服务
- 当然，甚至多个服务器，采用集群式并发提供服务。

注意

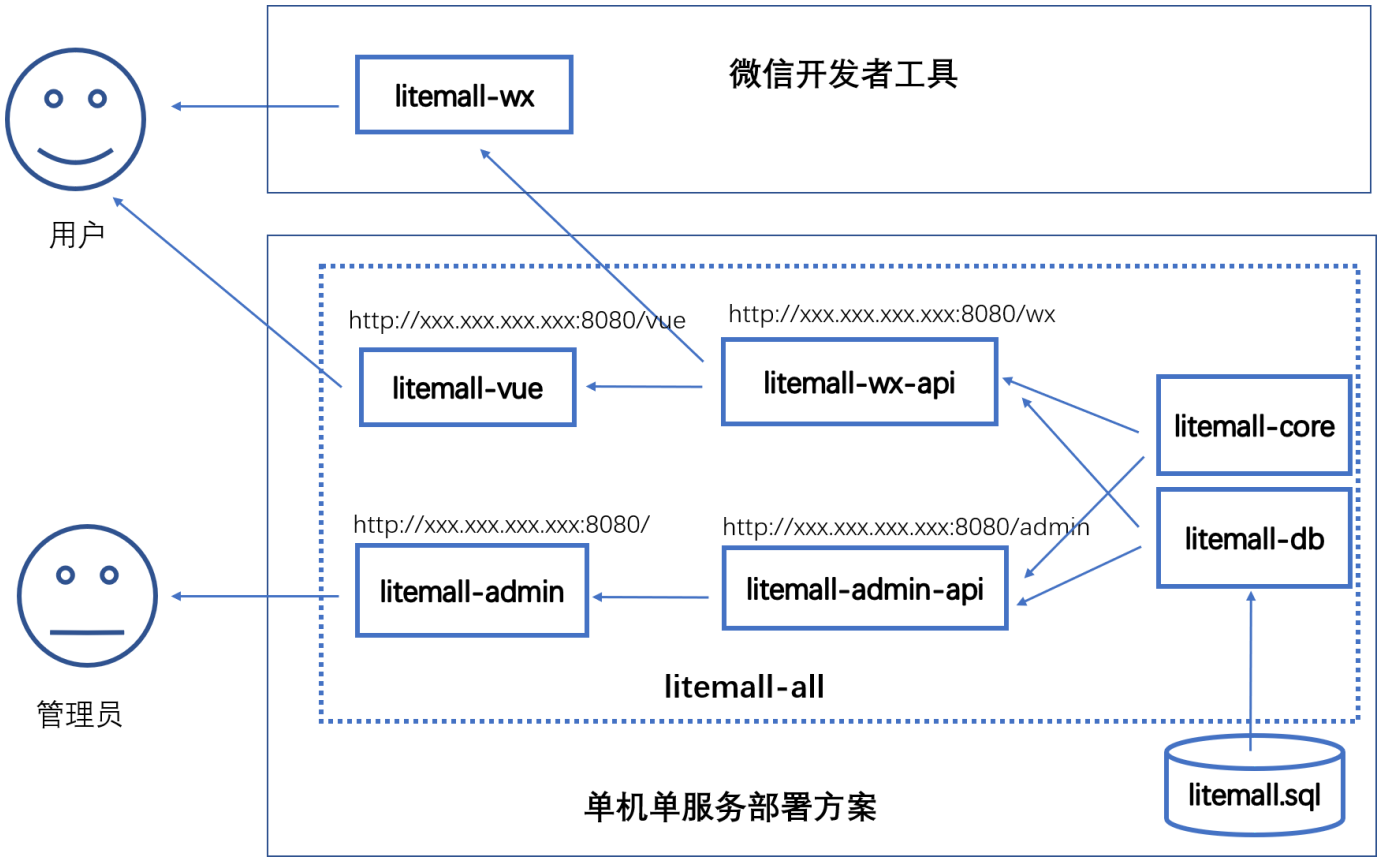
> 1. `本机` 指的是是当前的开发机
> 2. `云服务器` 指的是开发者购买并部署的远程服务器

以下简单列举几种方案。

1.5.1 单机单服务部署方案

本节介绍基于腾讯云的单机单服务部署方案，面向的是服务器数据和应用部署在云服务器单机中用于演示的场景。 其他云应该也是可行的。

主要流程是：创建云服务器，安装ubuntu操作系统，按照JDK和MySQL应用运行环境，部署单一Spring Boot服务。



1.5.1.1 云服务器

1. 创建云服务器
   请参考腾讯云、阿里云或者其他云平台的官方文档进行相关操作。 建议最低配置是**1核2G**。
2. 安装操作系统
   本项目采用ubuntu 16.04.1，但是并不限制其他操作系统。
3. 创建安全组

   目前允许的端口：8080，80，443，22，3306
   注意： 这里其实只需要8080端口，允许其他端口只是方便开发阶段的测试和调试。 特别是3306端口，作为MySQL的远程访问端口，请在上线阶段关闭。
4. 设置SSH密钥（可选）
   建议开发者设置SSH密钥，可以免密码登录云服务器，以及用于脚本自动上传应用。
5. 使用PuTTY远程登录云服务器
   如果开发者设置SSH密钥，可以采用免密码登录；否则采用账号和密码登录。

1.5.1.2 OpenJDK8

这里可以安装openjdk-8-jre

```
sudo apt-get update
sudo apt-get install openjdk-8-jre
```

如果希望采用jdk，而不是jre，则可以运行

```
sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

注意

> 如果用户想采用Oracle JDK8或者其他JDK环境，请查阅相关资料安装。

1.5.1.3 MySQL

```
sudo apt-get update
sudo apt-get install mysql-server
```

```
sudo apt-get install mysql-client
```

如果配置MySQL，可以运行命令

```
sudo mysql_secure_installation
```

1.5.1.4 项目打包

1. 在服务器或者开发机打包项目到deploy；

```
cd litemall
cat ./litemall-db/sql/litemall_schema.sql > ./deploy/db/litemall.sql
cat ./litemall-db/sql/litemall_table.sql >> ./deploy/db/litemall.sql
cat ./litemall-db/sql/litemall_data.sql >> ./deploy/db/litemall.sql

cd ./litemall-admin
cnpm install
cnpm run build:dep

cd ..
mvn clean package
cp -f ./litemall-all/target/litemall-all-*-exec.jar ./deploy/litemall/l
```

   这里脚本的作用是：
   1. 把数据库文件拷贝到deploy/db文件夹；
   2. 编译litemall-admin项目；
   3. 编译litemall-all模块，同时把litemall-admin编译得到的静态文件拷贝到litemall-all模块的static目录。
2. 修改litemall文件夹下面的*.yml外部配置文件，当litemall-all模块启动时会 加载外部配置文件，而覆盖默认jar包内部的配置文件。 例如，配置文件中一些地方需要设置成远程服务器的IP地址

此时deploy部署包结构如下：

- bin 存放远程服务器运行的脚本，包括deploy.sh脚本和reset.sh脚本
- db 存放litemall数据库文件
- litemall 存放远程服务器运行的代码，包括litemall-all二进制可执行包和litemall外部配置文件

- util 存放开发服务器运行的脚本，包括package.sh脚本和lazy.sh脚本。 由于是本地开发服务器运行，因此开发者可以不用上传到远程服务器。

1.5.1.5 项目部署

1. 远程服务器环境（MySQL和JDK1.8）已经安装好，请确保云服务器的安全组已经允许相应的端口。
2. 导入db/litemall.sql

```
cd /home/ubuntu/deploy/db
mysql -h localhost -u $ROOT -p$PASSWORD < litemall.sql
```

3. 启动服务

```
sudo service litemall stop
sudo ln -f -s /home/ubuntu/deploy/litemall/litemall.jar /etc/init.d/lit
sudo service litemall start
```

4. 测试是否部署成功(xxx.xxx.xxx.xxx是云服务器IP）：

```
http://xxx.xxx.xxx.xxx:8080/wx/index/index
http://xxx.xxx.xxx.xxx:8080/admin/index/index
http://xxx.xxx.xxx.xxx:8080/#/login
```

注意：

> 开发者访问以上三个地址都能成功，但是管理后台点击登录时会报错网络连接不成功。 这里很可能是开发者litemall-admin模块的 `config/dep.env.js` 或者 `condig/prod.env.js` 没有设置正确的管理后台后端地址，例如这里的 `http://xxx.xxx.xxx.xxx:8080/admin`

1.5.1.6 项目辅助脚本

在前面的项目打包和项目部署中都是采用手动命令来部署。 这里可以写一些脚本简化：

- util/packet.sh

在开发服务器运行可以自动项目打包

- util/lazy.sh

在开发服务器运行可以自动项目打包、项目上传远程服务器、自动登录系统执行项目部署脚本。

注意：

> 1. 开发者需要在util/lazy.sh中设置相应的远程服务器登录账号和密钥文件路径。
> 2. 开发者需要在bin/reset.sh设置远程服务器的MySQL的root登录账户。

- bin/deploy.sh

在远程服务器运行可以自动部署服务

- bin/reset.sh

在远程服务器运行可以自动项目导入数据、删除本地上传图片、再执行bin/deploy.sh部署服务。

注意：

> 开发者需要在bin/reset.sh设置远程服务器的MySQL的root登录账户。

总结，当开发者设置好配置信息以后，可以在本地运行lazy.sh脚本自动一键部署:

```
cd litemall
./deploy/util/lazy.sh
```

不过由于需要设置的信息会包含敏感安全信息，强烈建议开发者参考这里的deploy文件夹， 然后实现自己的deploy文件夹，妥善处置外部配置文件和脚本中的敏感安全信息!!!

1.5.2 单机多服务部署方案

1.5.3 集群式云部署方案

由于本项目是面向微小型企业的小商城系统，因此预期的分布式部署方案是

1. 专门的云数据库部署数据

2. 专门的云存储方案
3. 专门的CDN分发管理后台的静态文件
4. 一台云服务器部署管理后台的后端服务
5. 一台或多台云服务器部署小商场的后端服务

虽然由于环境原因没有正式测试过，但是这种简单的集群式场景应该是可行的。 在1.5.2节中所演示的三个服务是独立的，因此延伸到这里分布式是非常容易的。

但是，如果需要实现互联网式分布式云部署，目前的项目架构和方案不支持。 至少每个功能模块应该是独立服务系统。此外，需要引入单点登录系统、集群、缓存 和消息队列等多种技术。因此如果开发者需要这种形式的分布式方案，请参考其他项目。

1.6 上线方案

在1.5节部署方案中，我们介绍了多种部署的方案，但是实际上这些方案都不能立即用于正式环境：

1. 正式环境需要域名和HTTPS证书
2. 小商场的小程序端对服务器域名存在接入要求。

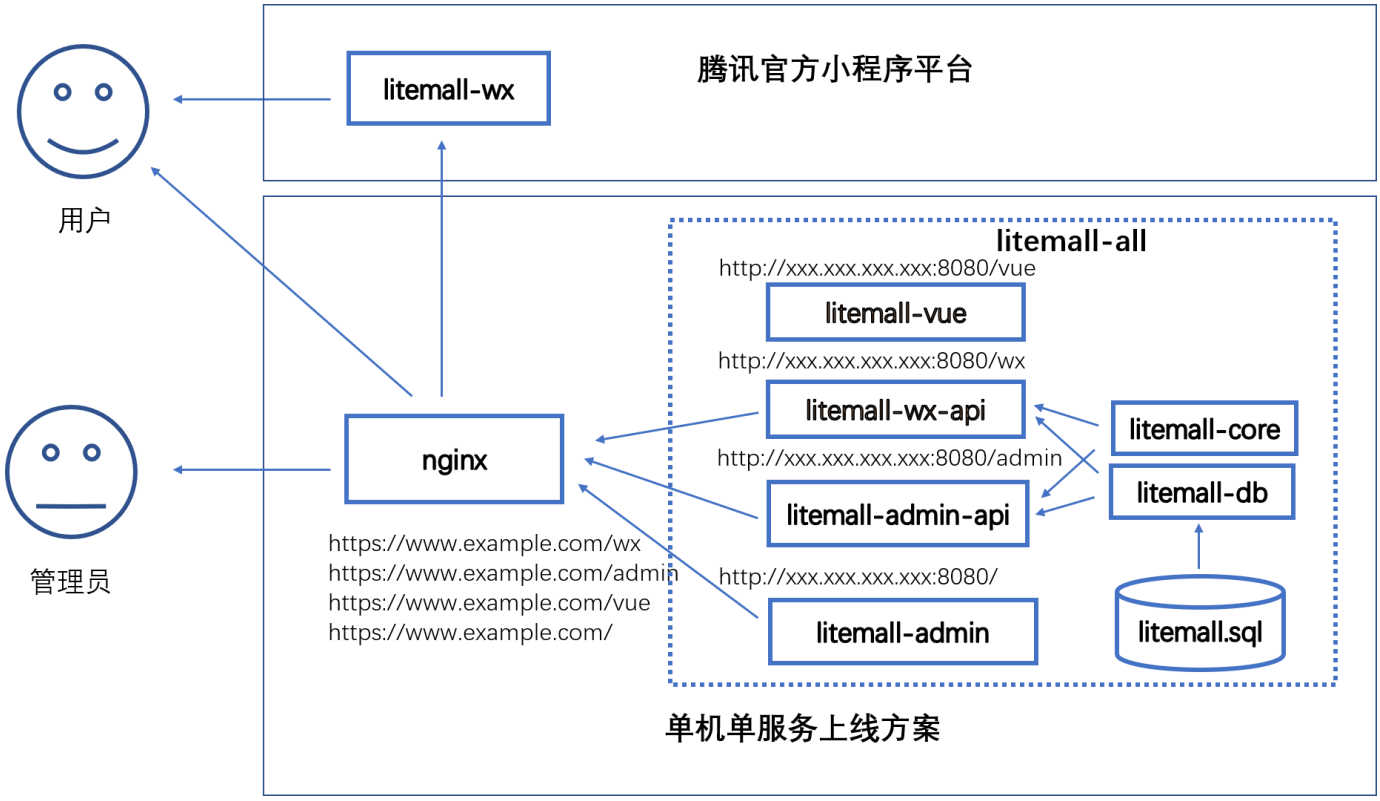
本节采用`www.example.com`域名作为示例。

注意

> `www.example.com`仅作为实例，不是真实环境下的域名。

这里列举一种基于1.5.1的单机单服务上线方案，即一个all后端服务，同时提供三种数据：

- 提供管理后台的前端文件；
- 提供管理后台前端所需要的数据；
- 提供小商城前端所需要的数据。



开发者可以基于自身业务采用其他上线方案。

1.6.1 域名

1. 注册域名，通常商业性的网站采用 `.com`
2. 解析域名到服务器公网IP，采用 `ping` 命令查看是否解析成功
3. 备案

1.6.2 nginx

https://www.digitalocean.com/community/tutorials/how-to-install-nginx-on-ubuntu-16-04

1.6.2.1 nginx安装

采用命令

```
sudo apt-get update
sudo apt-get install nginx
```

有的文档会指出需要防火墙设置，但是腾讯云服务器防火墙默认没有开启。 开发者这里自己可以开启设置，或者直接不开启。

打开浏览器，输入以下地址：

```
http://www.example.com
```

此时，如果看到nginx的欢迎页面，则安装成功。

安装以后：

- `/var/www/html`：默认静态web文件目录
- `/etc/nginx`：
- `/etc/nginx/nginx.conf`：
- `/etc/nginx/sites-avaiable`：
- `/etc/nginx/sites-enabled`：
- `/etc/nginx/snippets`：
- `/var/log/nginx/access.log`：
- `/var/log/nginx/error.log`：

1.6.2.2 https

1. 申请证书
   可以参考腾讯云 域名型证书申请流程
2. 下载证书
   这里使用nginx文件夹下面的密钥文件，例如 `1_www.example.com_bundle.crt` 和 `2_www.example.com.key`
3. 部署证书到nginx
   可以参考腾讯云 证书安装指引 把两个密钥文件保存的 `/etc/nginx` 文件夹，然后修改 `/etc/nginx/nginx.conf` 文件：

```
    server {
      listen 443;
      server_name www.example.com;
      ssl on;
      ssl_certificate /etc/nginx/1_www.example.com_bundle.crt;
      ssl_certificate_key /etc/nginx/2_www.example.com.key;
      ssl_session_timeout 5m;
      ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
      ssl_ciphers ECDHE-RSA-AES128-GCM-SHA256:HIGH:!aNULL:!MD5:!RC4:!DHE;
      ssl_prefer_server_ciphers on;
    }
```

4. 重启nginx

打开浏览器，输入以下地址：

```
https://www.example.com
```

此时，可以看到https协议的nginx欢迎页面。

1.6.2.3 反向代理Spring Boot后端

修改 `/etc/nginx/nginx.conf` 文件，配置nginx静态web文件目录

```
server {
    location / {
        proxy_pass  http://localhost:8080;
        proxy_set_header    Host    $host;
        proxy_set_header    X-Real-IP   $remote_addr;
        proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
    }
}
```

打开浏览器，输入以下地址：

```
https://www.example.com/wx/index/index
https://www.example.com/admin/index/index
https://www.example.com/admin/index/index
```

此时，看到后端数据说明反向代理配置成功。

1.6.2.4 全站加密

服务器自动把http的请求重定向到https

```
server {
    listen 80;
    server_name www.example.com;
    rewrite https://$server_name$request_uri? permanent;
}
```

打开浏览器，输入以下地址：

```
http://www.example.com
```

总结，经过以上不同方面的配置，nginx这里最终的配置是如下： 1. 证书 `1_www.example.com_bundle.crt` 和 `2_www.example.com.key` 放置在 `/etc/nginx/` 文件夹内。 2. 把 `/etc/nginx/nginx.conf` 文件进行修改，具体可以参考本项目的nginx.conf 3. 重启nginx

注意：

> 更多配置方法和功能，请开发者自行学习。

1.6.3 小商场上线

在1.6.2.3节"反向代理Spring Boot后端"成功以后，其实小商场的后端已经上线成功。 这里介绍小商场的前端上线过程：

上线之前需要修改代码或者配置文件： 1. litemall-wx-api模块的WxOrderController类的payNotify方法的链接换成合适的地址。

注意：

> 换成什么地址都可以，但是这里不应该暴露出来。也就是说这个地址是微信商户平台 和这里的小商场后端服务之间的交互API，对外公开会存在安全隐患。

1. litemall-core模块需要配置application-core.yml

```
litemall:
    wx:
        app-id: wxa5b486c6b918ecfb
        app-secret: e04004829d4c383b4db7769d88dfbca1
```

```
5           mch-id: 111111
6           mch-key: xxxxxx
7           notify-url: http://www.example.com/wx/order/pay-notify
```

这里的 `litemall.wx.notify-url` 就是前面开发者自定义的地址。

2. litemall-wx模块的 `project.config.json` 文件调整相应的值， 特别是 `appid` 要设置成开发者申请的appid。

1.6.4 管理后台上线

在1.6.2.3节"反向代理Spring Boot后端"成功以后，其实管理后台已经上线成功， 包括管理后台的前端和后端，会同时对外提供管理后台的前端文件和后端数据。 当然，这里开发者需要自己的线上环境在以下文件中或代码中修改相应的配置。

1. MySQL数据库设置合适的用户名和密码信息；
2. 管理后台后端服务模块设置合适的配置信息，建议开发者参考deploy/litemall的外部配置文件，
   这样可以避免开发者对模块内部的开发配置文件造成修改；
3. 管理后台前端litemall-admin模块 `.env.production` 中的 `VUE_APP_BASE_API` 设置管理后台后端服务的服务地址。

1.6.5 项目评估

本项目只是参考项目，项目代码质量和功能不可能符合开发者的最终需求， 因此开发者**请务必仔细评估项目代码。**

特别提醒，上线前管理员用户名和密码请更改，不能采用这里的默认值。

1.6.6 项目优化

以下是部署方案中出现而在上线方案中可以优化的一些步骤。

1.6.6.1 管理后台前端文件启动优化

litemall-admin编译得到的前端文件在第一次加载时相当耗时，这里需要一些措施来优化启动速度

- 静态文件托管CDN
  在上节中，采用Spring Boot来分发管理后台的静态文件。 这里可以进一步地，把静态文件托管到CDN，当然这里是需要收费。

- gzip压缩
- 动态加载

1.6.6.2 后端服务内部访问

原来后端服务（包括小商城的后端服务和管理后台的后端服务）可以通过域名或者IP直接对外服务，而这里采用nginx反向代理后可以 通过80端口访问后端服务。因此，会存在这样一种结果：

- 用户可以https协议的80端口访问后端服务（nginx反向代理）
- 用户也可以通过http协议的8080访问后端服务（spring boot）
  由于http不是安全的，这里可能存在安全隐患

而如果取消后端服务的对外访问，这样可以保证用户只能采用安全的https协议访问后端服务。 同时，对外也能屏蔽内部具体技术架构细节。

1.6.6.4 nginx优化

建议开发者根据自己业务或架构情况优化。

1.6.7 项目安全

项目一旦正式上线，即对外正式服务。但是服务同时，可能会存在安全隐患甚至黑客攻击。

本节仅列举一些注意事项，欢迎开发者补充和完善。

1.6.7.1 账户安全

这里的账号安全，既包括商城端用户账户，也包括管理后台端管理员账户。

目前账号安全还缺乏一点的保护措施，例如

- 用户密码失败超过阈值，则显示验证码；
- 用户密码失败超过阈值，则取消登录；
- 用户密码失败超过阈值，则需要手机验证码；

1.6.7.2 关键业务记录

有关订单或者金钱相关的操作，建议开发者尽可能记录在数据库中，以便以后回溯。

1.6.7.3 API统一调整

本项目公布了参考API接口，如果出现BUG可能会被黑客作为入口。 建议开发者上线之前可以统一调整接口，以减少安全隐患。

1.6.7.4 对账

本项目管理后台没有对账功能，建议开发者可以开发对账比对商场的状态是否正常。

1.6.7.5 取消或者限制退款

本项目不支持自动退款功能，而是在管理后台通过管理员点击退款按钮来人工退款。 但是仍然可能存在隐患，例如黑客通过漏洞进入管理后台从而进行不合理的退款操作。

因此建议开发者可以取消管理后台的退款按钮，而仅仅保持退款信息，管理员可以登录 微信官方支付平台进行退款操作。

或者建议开发者基于一定的业务逻辑或场景限制管理后台的退款功能。例如，设置当天 退款限额从而保证不会产生无限退款操作。

1.6.7.6 资源限制访问

一些API操作涉及到后端服务器资源，因此需要做一定的限制，防止有限资源被恶意消耗。

有限资源可能包括：

- 验证码
- 图片上传

一些限制措施可能包括：

- 限制单个IP的访问频率
- 限制用户上传图片数量

1.6.7.n 跟踪本项目进展

一旦有开发者反馈BUG，本项目会优先解决并及时上传补丁。 因此建议开发者跟踪本项目进展，留意每次BUG修复的commit。

同时也希望开发者发现任何BUG都及时反馈。

目前还不存在LTS版本，未来业务稳定后可能会发布。

1.7 项目管理

这里简述一些当前项目开发的要点。

1.7.1 项目.gitignore

当前项目的.gitignore不是单一文件，而是多个模块都存在：

- deploy/.gitignore
- litemall-admin/.gitignore
- litemall-admin-api/.gitignore
- litemall-core/.gitignore
- litemall-db/.gitignore
- litemall-wx-api/.gitignore
- litemall-all/.gitignore
- .gitignore

开发者可以采用单一.gitignore文件。

1.7.2 项目自动部署

1.7.2.1 deploy部署

当前项目存在deploy部署文件夹，这个是上述1.5.1节部署腾讯云服务器所采取的一些脚本。

流程如下： 1. util脚本是当前开发服务器运行，用来打包项目和上传腾讯云服务器； 2. 打包项目时，会编译打包项目相关模块到litemall和db文件夹中； 3. bin脚本是云服务器运行，用来安装数据库、导入数据、启动项目服务。

这里deploy部署方式比较简单不灵活，开发者可以参考开发自己的项目脚本。

1.7.2.2 .gitlab-ci.yml部署

目前不支持

1.7.2.3 docker部署

目前不支持

1.7.3 项目代码风格

由于本项目涉及三种技术栈，因此针对这三种技术栈也存在三种代码风格。

如果开发者想要贡献代码，建议尽可能保证代码符合这里的规范。

1.7.3.1 Spring Boot技术栈代码风格

这里的代码风格采用IDEA默认代码风格。

修改代码后，利用 `Code` 菜单的 `Reformat Code` 即可格式化代码。

1.7.3.2 小程序技术栈代码风格

这里的代码风格采用微信开发者工具默认代码风格。

修改代码以后，利用 `编辑` 菜单的 `格式化代码` 即可格式化代码。

1.7.3.3 Vue技术栈代码风格

这里的代码风格采用ESLint配置代码风格，具体参考vue-element-admin下项目的 ESLint文档, 特别是 `vscode 配置 ESLint` 内容。

注意：

> Visual Studio Code编辑器中右键存在 `格式化代码` 的选项，但是请不要使用这种方式， 因为VSC自带的格式化代码风格和ESLint代码风格可能不完全一致。

1.7.4 Spring Boot多模块多阶段配置

目前后端服务采用Spring Boot多模块方案，结构清晰、易于测试。

但是存在一个问题，即多模块配置依赖。 例如，litemall-db模块存在数据库配置信息，那么其他模块如何引入 litemall-db模块的配置信息呢?

最简单的方式，就是其他模块把litemall-db模块的配置信息拷贝到自己的 application配置文件中，但是问题就是数据库信息一旦改变则其他模块又要 再次手动修改，非常不方便。

目前本项目采用一种基于 `spring.profiles.active` 的方式，细节如下： 1. litemall-db模块存在application.yml和application-db.yml两个配置文件， 在application-db.yml配置文件中存放数据库

配置信息； 2. litemall-core模块也存在application.yml和application-core.yml两个配置文件, 在application-core.yml配置文件中存放core模块的一些配置信息，而在application.yml 中存在这样一个配置：

```
    spring:
        profiles:
            active: core, db
```

```
因此，如果单独启动litemall-core模块，则会先读取application.yml配置文件，然后基于
系统会根据`spring.profiles.active`读取application-db.yml和application-core.yml
因此就会自动读取litemall-db模块的配置文件。
```

1. 以此类推，在litemall-all模块中存在application.yml配置文件，其中内容是

```
spring:
    profiles:
        active:  db, core, admin, wx
```

因此，系统启动litemall-all模块以后，则会先读取application.yml配置文件，然后基于`spring.profiles.active`进一步读取application-db.yml、application-core.yml、application-admin.yml和application-wx.yml四个模块的配置文件。

但是，虽然以上方案解决了多模块配置依赖问题，但是又会导致另外一个问题，如何支持不同profile， 也就是开发阶段、测试阶段和上线阶段配置不同。

这里介绍本项目的思路，就是基于Spring Boot的配置加载顺序，采用外部配置文件覆盖jar包内部配置文件。 1. 开发阶段，系统的配置信息在模块的resources目录配置文件中； 2. 测试或者部署阶段，系统打包成一个litemall.jar二进制jar包，jar包内部配置文件是之前 开发阶段的配置文件，此时在litemall.jar的同级目录创建相同的配置文件，在这些配置文件则 保存了测试或者部署阶段的配置信息。启动litemall.jar时，系统会读取当前目录的配置文件，而 不再读取jar包内部的配置文件。 3. 上线阶段，同样地，在litemall.jar包同级目录创建上线配置文件。

此外，这里还可以采用另外一种思路，如下图：

其实原理也很简单，就是配置文件采用application-{module}-{profile}.yml来支持不同模块不同阶段的配置需求。

1.7.5 前后端校验

本项目是前后端分离项目，当用户或者管理员在系统中输入数据时， 数据需要进行两层校验。

- 第一层是前端校验，是对参数格式校验。
- 第二层是后端校验，不仅对参数校验，还会根据业务场景进行校验。

注意

> 目前项目校验思路是这样，但是实际代码的校验还不完善， 例如前端校验代码不完善，导致用户体验较差。

1.7.6 后端响应错误码

后端服务的响应结果是：

```
{
    errno：  错误码，
    errmsg：错误消息，
    data：   响应数据
}
```

当errno是0时，则data保存业务数据； 当error是非0时，则业务失败，errmsg保存具体错误信息。

目前，errno存在四种形式：

- 4xx，前端错误，说明前端开发者需要重新了解后端接口使用规范：
  - 401，参数错误，即前端没有传递后端需要的参数；
  - 402，参数值错误，即前端传递的参数值不符合后端接收范围。
- 5xx，后端系统错误，除501外，说明后端开发者应该继续优化代码，尽量避免返回后端系统错误码：
  - 501，验证失败，即后端要求用户登录；
  - 502，系统内部错误，即没有合适命名的后端内部错误；
  - 503，业务不支持，即后端虽然定义了接口，但是还没有实现功能；
  - 504，更新数据失效，即后端采用了乐观锁更新，而并发更新时存在数据更新失效；

    - 505，更新数据失败，即后端数据库更新失败（正常情况应该更新成功）。
- 6xx，管理后台后端业务错误码，具体见litemall-admin-api模块的 `AdminResponseCode` 类。
- 7xx，小商城后端业务错误码，具体见litemall-wx-api模块的 `WxResponseCode` 类。

需要指出的是，小商场后端可能返回4xx、5xx和6xx错误码；管理后台后端则可能返回4xx、5xx和7xx错误码。 这样设计原因是方便小商场前端和管理后台前端区别对待。

小商城前端处理后端响应错误码，存在三种处理方式：

- 如果是4xx，说明前端开发者请求后端API时使用方式存在问题。
  例如，后端需要参数“name”，但是前端却没有传值，这个时候后端返回”用户名不对“
  没有任何意义，因为这里前端使用错误。相反，简单地返回“参数不对”反而会及早提醒
  前端开发者使用出现了问题。
- 如果是5xx，（除501外）说明后端系统出现错误，后端开发者应该修复或者优化，此外
  前端可以在请求响应处统一处理5xx错误，而不是把错误信息返回到具体页面。
  例如，后端返回“更新数据失败”，说明数据库更新时出现异常，因此前端请求响应处
  统一简单报错“系统出错，联系管理员”，这样管理员可以及时联系后端开发者。而后端开发者
  则需要评估具体错误码和错误信息，例如这里的“更新数据失败”很可能是数据表调整字段
  导致Java代码的模型对象和数据库表不一致，此时后端开发者就可以及时修复。
  此外，对于501验证失败，则前端请求响应处可以统一处理跳转登录页面。
- 如果是6xx，则说明是具体业务错误，此时前端需要在业务具体页面显示错误信息即可，同时
  这里也要求后端开发者书写良好友好的业务错误信息，因为会向最终用户显示。

和小商场前端类似，管理后台前端处理后端响应错误码也存在三种类似的处理方式。

注意：

> 这里的4xx和5xx错误码，和HTTP中的4xx和5xx状态码不是一个概念。

1.7.7 TODO

本项目存在一些TODO，**强烈建议**开发者上线前仔细审阅是否存在问题和做相应调整。 开发者可以使用IDE找到这些TODO。

下面列出一些重要的TODO：

1.7.7.1 微信退款TODO

管理后台管理员点击退款按钮时，管理后台会通过微信退款API请求微信商户平台退款。 但是从安全角度考虑，**强烈建议**开发者删除微信退款代码，而登录微信商户平台手动退款。 或者开发者添

加安全相关代码，例如实现短信验证码。

见 `AdminOrderController` 类

再次提醒，本项目不承担任何使用后果。

1.7.7.2 未完善TODO

有些业务只是实现基本功能，因此这里TODO提醒开发者自行思考。

1.7.7.3 重构TODO

有些业务需求不是很清晰，导致实现时可能存在不合理地方，这里TODO提醒 开发者审阅代码逻辑。