A1-18

2019-2020 学年冬季学期

《计算思维实训（2）》(0830A031)

课程报告

成绩 (百分制)	

学　　号	17122842	学　　院	计算机工程与科学学院
姓　　名	魏亮亮	手工签名	
报告题目	SSM、Vue 框架学习及个人博客的搭建		
实训报告成绩 50%	实训过程描述：清晰、全面。（5%）		
	报告主要部分：1、计算思维实例叙述清楚、有意义；2、分析到位；3、思路独特或有创意；4、关键实现技术描述清楚详细；5、内容丰富；6、不直接粘贴所有代码；7、代码有注释或说明。（25%）		
	实训收获体会：感受真实、深刻，建议有价值。（10%）		
	书写格式：书写规范、用词正确、无明显的错别字，图表、代码清晰规范，格式协调、效果好，参考文献书写、引用规范合理。（10%）		
工作实绩 50%	1、平时表现、进步情况（25%），2、工作实绩（25%）		

教师对该生工作实绩简要评述：

教师签名：

日期：　　年　　月　　日

本学期实训过程概述

1 本学期实训总体概况与实训过程

1.1. 总体概况

经过与指导老师的讨论，我联系上了曹老师的研究生，打算在他的带领下参与一些项目。我原本的打算是能够在研究生的带领下，边学边实战，参与到研究生的项目中。不过经过我与学长的讨论，发现我对他的项目所用到的技术框架（React + Spring Boot + Elastic Search）竟一无所知。商量之后，他给我展示了他自己以前搭的小论坛 Demo，让我自己去学习相关的框架，然后自行搭建一个小项目。由于学长对具体的项目并没有要求，于是我便自行选择搭建的项目类型。

经过一些摸索，我发现做一个个人博客是比较适合我的。一方面是可以促进自己对流行现代 Web 的 SSM + Vue 框架的了解及使用，另一方面是如果这网站成功搭建起来，也方便自己以后进行功能上的补充工作。

于是接下来我就开始选择具体的技术框架进行学习。考虑到当时在网站开发方面，我只掌握了 Java、Golang、MySQL 和前端三件套的基本使用，所以我选择了开发起来较为简便的 Spring Boot 后端框架和目前国内较为流行的 Vue.js 前端框架作为项目的基础技术选择。

在对上述技术框架经过了较长时间的学习和一定程度的了解后，我开始对网络上已有的程序员个人博客的具体形式和使用的各种中间框架进行了解，方便明白项目的需求和实现过程。 在此期间，我知道了搭建一个网站不仅仅只需要学习 SSM + Vue.js，而且出于安全、性能等方面的考虑，还必须要学习鉴权层、消息队列层、Redis 缓存数据库、消息队列等各种中间件，又可选数据检索（如 Elastic Search）、前后端文档（如 Swagger）、数据库连接池、登陆加密等中间件。刚才罗列的只是后端的中间件要求，使用 Vue.js 时，又有 View UI 这样的组件库、EChart 这样的图标库。所以，需要学习的东西确实很多。

在对必需的中间件和各种前端进行学习后，我终于开始了项目的搭建。其间遇到很多困难，比如对于框架的不熟悉和下一步如何进行，几度暂停编写，重新翻看技术博客和官方教程，最后整个项目还是能够 Run 起来的。这也算是完成了整体的项目要求。

1.2. 后端 SSM 学习过程

由于 Spring Boot 框架是 SSM（Spring + Spring MVC + MyBatis / MyBatis Plus）的高度整合，所以如果要学习 Spring Boot 框架，必须先了解 Spring，进一步再了解 Spring MVC，最后再了解 Spring Boot。

- Spring 的 IOC 和 DI

Spring 框架是一个 Java 平台，它为开发 Java 应用程序提供全面的基础架构支

持。Spring 负责基础架构，因此程序员可以专注于应用程序的逻辑开发，而不用为繁琐的基础配件花费大量的时间。

Spring 有两大特性：IOC（Inversion Of Control）控制反转和 DI（Dependency Injection）依赖注入。IOC 控制反转即控制权的转移，将我们创建对象的方式反转了,以前对象的创建是由我们开发人员自己维护,包括依赖关系也是自己注入。使用了 Spring 之后，对象的创建以及依赖关系可以由 Spring 完成创建以及注入，DI 反转控制就是反转了对象的创建方式，从我们自己创建反转给了程序创建(Spring)。依赖注入思想可以这么解释：Spring 这个容器中，替你管理着一系列的类，前提是你需要将这些类交给 Spring 容器进行管理，然后在你需要的时候，不是自己去定义，而是直接向 Spring 容器索取，当 Spring 容器知道你的需求之后，就会去它所管理的组件中进行查找，然后直接给你所需要的组件。实现 IOC 思想则需要 DI 做支持。

依赖注入时，有 3 种注入方式和 2 种注入类型。IOC 和 DI 的显著优势是降低组件之间的耦合度，实现软件各层之间的解耦。

实现注入，则可以使用 xml 和注解两种形式。Spring 作为整个现代 JavaWeb 体系的底层，拥有两种形式，导致其它框架都有这两种注入形式。

- Spring AOP

实现 IOC 和 DI 后，软件自然会拥有 AOP（面向切面编程）的优势。AOP（Aspect Oriented Programming），即面向切面编程，可以说是 OOP（Object Oriented Programming，面向对象编程）的补充和完善。

OOP 引入封装、继承、多态等概念来建立一种对象层次结构，用于模拟公共行为的一个集合。不过 OOP 允许开发者定义纵向的关系，但并不适合定义横向的关系，例如日志功能。日志代码往往横向地散布在所有对象层次中，而与它对应的对象的核心功能毫无关系对于其他类型的代码，如安全性、异常处理和透明的持续性也都是如此，这种散布在各处的无关的代码被称为横切（cross cutting），在 OOP 设计中，它导致了大量代码的重复，而不利于各个模块的重用。

AOP 技术恰恰相反，它利用一种称为"横切"的技术，剖解开封装的对象内部，并将那些影响了多个类的公共行为封装到一个可重用模块，并将其命名为"Aspect"，即切面。所谓"切面"，简单说就是那些与业务无关，却为业务模块所共同调用的逻辑或责任封装起来，便于减少系统的重复代码，降低模块之间的耦合度，并有利于未来的可操作性和可维护性。

AOP 有许多核心的概念：横切关注点、切面（aspect）、连接点（joinpoint）等等。Spring 则对 AOP 有着良好的支持，程序员采用 AOP 编程十分方便：定义普通业务组件；定义切入点，一个切入点可能横切多个业务组件；定义增强处理，增强处理就是在 AOP 框架为普通业务组件织入的处理动作。

学习完 Spring 的 IOC、DI 和 AOP，还有 Spring 的事务、Spring 和 JDBC 的

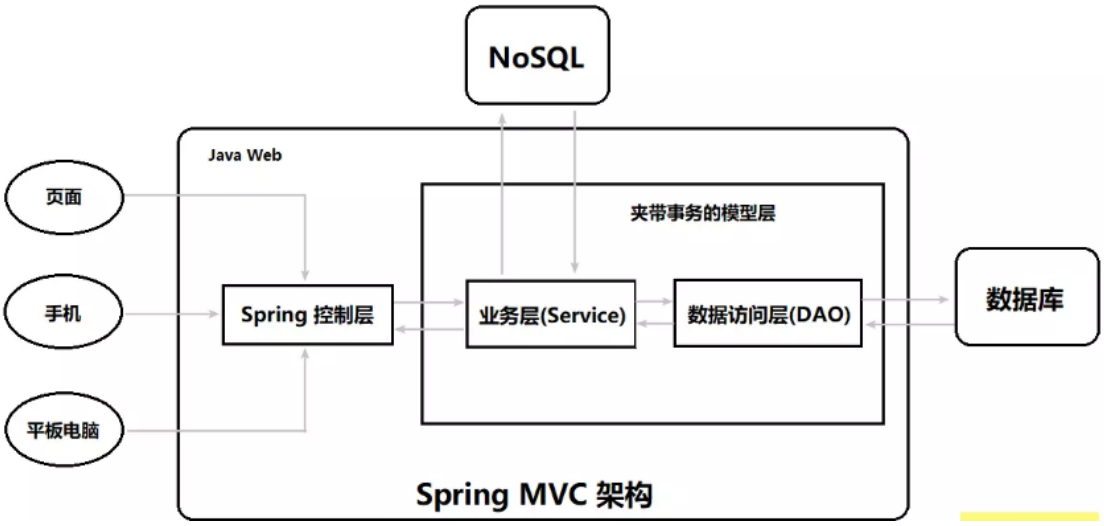
连接等。这些就算是 Spring 的基础内容了，这里讲的比较多，也是因为这些概念极为重要，贯穿了之后后端框架学习和编写的始终。

- Spring MVC

Spring MVC 是 Spring 提供的一个强大而灵活的 web 框架。借助于注解，Spring MVC 提供了几乎是 POJO 的开发模式，使得控制器的开发和测试更加简单。这些控制器一般不直接处理请求，而是将其委托给 Spring 上下文（Spring 上下文也是一个比较重要的概念）中的其他 bean，通过 Spring 的依赖注入功能，这些 bean 被注入到控制器中。

Spring MVC 主要由 DispatcherServlet、处理器映射、处理器(控制器)、视图解析器、视图组成。他的两个核心是处理器映射（选择使用哪个控制器来处理请求）和视图解析器（选择结果应该如何渲染）。通过这两点，Spring MVC 保证了如何选择控制处理请求和如何选择视图展现输出之间的**松耦合**。这样也极其符合 MVC 架构的设计理念。

为解决持久层中一直未处理好的数据库事务的编程，又为了迎合 NoSQL 的强势崛起，Spring MVC 给出了方案：

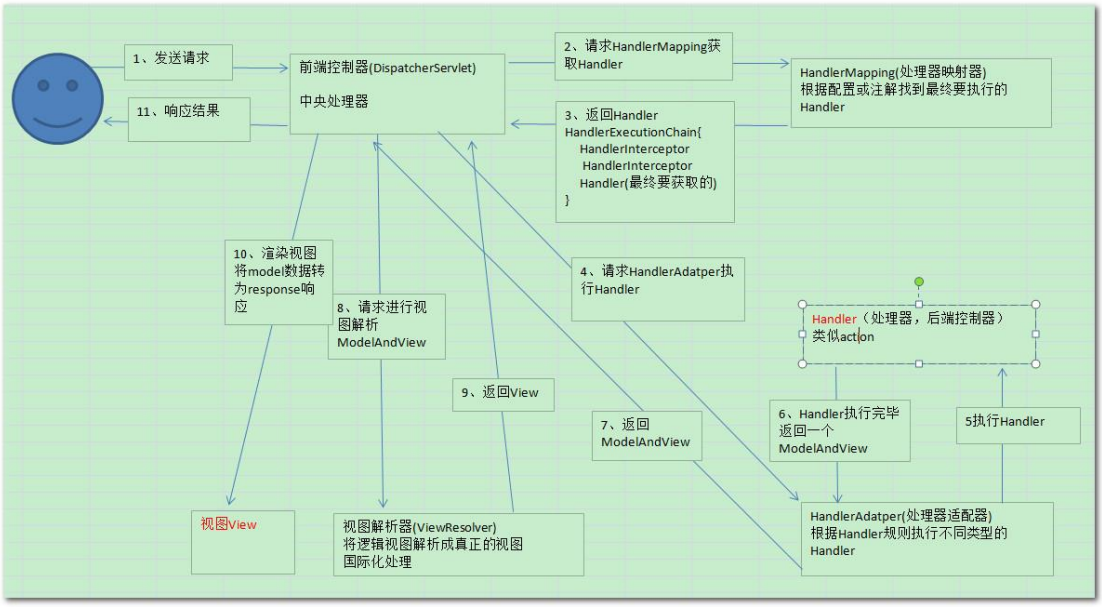


传统的模型层被拆分为了**业务层**(Service)和**数据访问层**（DAO, Data Access Object）。在 Service 下可以通过 Spring 的声明式事务操作数据访问层，而在业务层上还允许我们访问 NoSQL 这样就能够满足异军突起的 NoSQL 的使用了，它可以大大提高互联网系统的性能。

- Spring MVC 工作原理

下文中的 Spring Boot 其实对 Spring MVC 进行了默认集成。为了学习 Spring MVC 这一优秀的应用 MVC 构架的框架，了解 Spring MVC 的工作原理是十分必要的。

以下为 Spring MVC 工作流程图：



流程说明：

1) 用户发送请求至前端控制器 DispatcherServlet。
2) DispatcherServlet 收到请求调用 HandlerMapping 处理器映射器。
3) 处理器映射器找到具体的处理器(可以根据 xml 配置、注解进行查找)，生成处理器对象及处理器拦截器(如果有则生成)一并返回给 DispatcherServlet。
4) DispatcherServlet 调用 HandlerAdapter 处理器适配器。
5) HandlerAdapter 经过适配调用具体的处理器(Controller，也叫后端控器)。
6) Controller 执行完成返回 ModelAndView。
7) HandlerAdapter 将 controller 执行结果 ModelAndView 返回给 DispatcherServlet。
8) DispatcherServlet 将 ModelAndView 传给 ViewReslover 视图解析器。
9) ViewReslover 解析后返回具体 View。
10) DispatcherServlet 根据 View 进行渲染视图（即将模型数据填充至视图中）。
11) DispatcherServlet 响应用户。

根据以上的流程图，我们可以知道 Spring MVC 拥有如下组件：

- 前端控制器 DispatcherServlet（不需要工程师开发）,由框架提供

作用：接收请求，响应结果，相当于转发器，中央处理器。有了 DispatcherServlet 减少了其它组件之间的耦合度。

用户请求到达前端控制器，它就相当于 MVC 模式中的 C，DispatcherServlet 是整个流程控制的中心，由它调用其它组件处理用户的请求，DispatcherServlet 的存在降低了组件之间的耦合性。

- 处理器映射器 HandlerMapping(不需要工程师开发),由框架提供

作用：根据请求的 URL 查找 Handler

HandlerMapping 负责根据用户请求找到 Handler 即处理器，Spring MVC 提供了不同的映射器实现不同的映射方式，例如：配置文件方式，实现接口方式，注解方式等。

- 处理器适配器 HandlerAdapter

作用：按照特定规则（HandlerAdapter 要求的规则）去执行 Handler

通过 HandlerAdapter 对处理器进行执行，这是适配器模式的应用，通过扩展适配器可以对更多类型的处理器进行执行。

- 处理器 Handler(**需要工程师开发**)

注意：编写 Handler 时按照 HandlerAdapter 的要求去做，这样适配器才可以去正确执行 Handler

Handler 是继 DispatcherServlet 前端控制器的后端控制器，在 DispatcherServlet 的控制下 Handler 对具体的用户请求进行处理。

由于 Handler 涉及到具体的用户业务请求，所以一般情况需要工程师根据业务需求开发 Handler。

- 视图解析器 ViewResolver(不需要工程师开发),由框架提供

作用：进行视图解析，根据逻辑视图名解析成真正的视图（view）

View Resolver 负责将处理结果生成 View 视图，View Resolver 首先根据逻辑视图名解析成物理视图名即具体的页面地址，再生成 View 视图对象，最后对 View 进行渲染将处理结果通过页面展示给用户。

一般情况下需要通过页面标签或页面模版技术将模型数据通过页面展示给用户，需要由工程师根据业务需求开发具体的页面。

- 视图 View(**需要工程师开发**)

View 是一个接口，实现类支持不同的 View 类型

上述步骤虽然复杂，但是一般来讲我们只需要自定义 Handler 和 View 就足够了。了解这个步骤十分重要，这样我们才能开始学习 Spring Boot。

- Spring Boot

初期的 Spring 通过代码加配置的形式为项目提供了良好的灵活性和扩展性，但随着 Spring 越来越庞大，其配置文件也越来越繁琐，太多复杂的 xml 文件也一直是 Spring 被人诟病的地方，特别是近些年其他简洁的 WEB 方案层出不穷，如基于 Python 或 Node.js，几行代码就能实现一个 WEB 服务器，对比起来，大家渐渐觉得 Spring 那一套太过繁琐，

Spring Boot 是在现有 Spring 框架的基础上发布的框架，它使用全新的开发模型，通过默认配置了很多框架的使用方式，实现自动配置，降低项目搭建的复杂度，使 Java 开发非常容易。

Spring Boot 其实有很多可以讲解的地方，但限于篇幅内容，难以展开详细讲述。在中间件部分可以瞥见其强大的功能。

Spring Boot 可以根据开发者的需求，很方便地与各种中间件进行集成。只需要在 Spring Boot 项目下的 pom.xml 文件中写入对应版本的中间件的 jar 包的依赖，让 IDE 的依赖管理程序 Maven 或 Gradle 进行管理。当然，这涉及到 Spring Boot 本身的版本（1.5or2.x）、各种中间件的版本还有两者的接口 jar 包的版本适配问题。有的时候，版本适配问题会造成很大的困扰。

- Spring Boot 自动配置原理

Spring Boot 能自动配置各种类的 jar 包，并且运行开发者实现自己配置的 jar 包。了解 Spring Boot 的自动配置和启动原理也是十分必要的。

我们开发任何一个 Spring Boot 项目，都会用到如下的启动类：

```
@SpringBootApplication
public class QuickApplication {

    public static void main(String[] args) {
        SpringApplication.run(QuickApplication.class);
    }
}
```

可以看到其上有@SpringBootApplication 的 Annotation 定义和 SpringApplication.run 的静态类定义。

我们点开@SpringBootApplication 的源码：

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@SpringBootConfiguration
@EnableAutoConfiguration
@ComponentScan(excludeFilters = {
                @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
                @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFi
public @interface SpringBootApplication {

        /**
         * Exclude specific auto-configuration classes such that they will never be ap
         * @return the classes to exclude
         */
        @AliasFor(annotation = EnableAutoConfiguration.class)
        Class<?>[] exclude() default {};

        ... ... ...

}
```

虽然定义使用了多个 Annotation 进行了原信息标注，但实际上重要的只有三

个 Annotation：@SpringBootConfiguration、@EnableAutoConfiguration 以及@ComponentScan。

所以@SpringBootApplication 其实就等于@ SpringBootConfiguration + @EnableAutoConfiguration + @ComponentScan。

对三者解析：

- @ SpringBootConfiguration：等同于 @Configuration，既标注该类是 Spring 的一个配置类，如下 @SpringBootConfiguration 源码可知：

```
@Target({ElementType.TYPE})
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Configuration
public @interface SpringBootConfiguration {
}
```

- @ComponentScan: 它对应 XML 配置中的元素，@ComponentScan 的功能其实就是自动扫描并加载符合条件的组件（比如@Component 和@Repository 等）或者 bean 定义，最终将这些 bean 定义加载到 IOC 容器中。
- @EnableAutoConfiguration: Spring Boot 自动配置功能开启，按住 Ctrl 点击查看注解源码：

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import(AutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {

        ...
}
```

最关键的要属@Import(EnableAutoConfigurationImportSelector.class)，借助 EnableAutoConfigurationImportSelector，@EnableAutoConfiguration 可以帮助 Spring Boot 应用将所有符合条件的@Configuration 配置都加载到当前 Spring Boot 创建并使用的 IOC 容器。

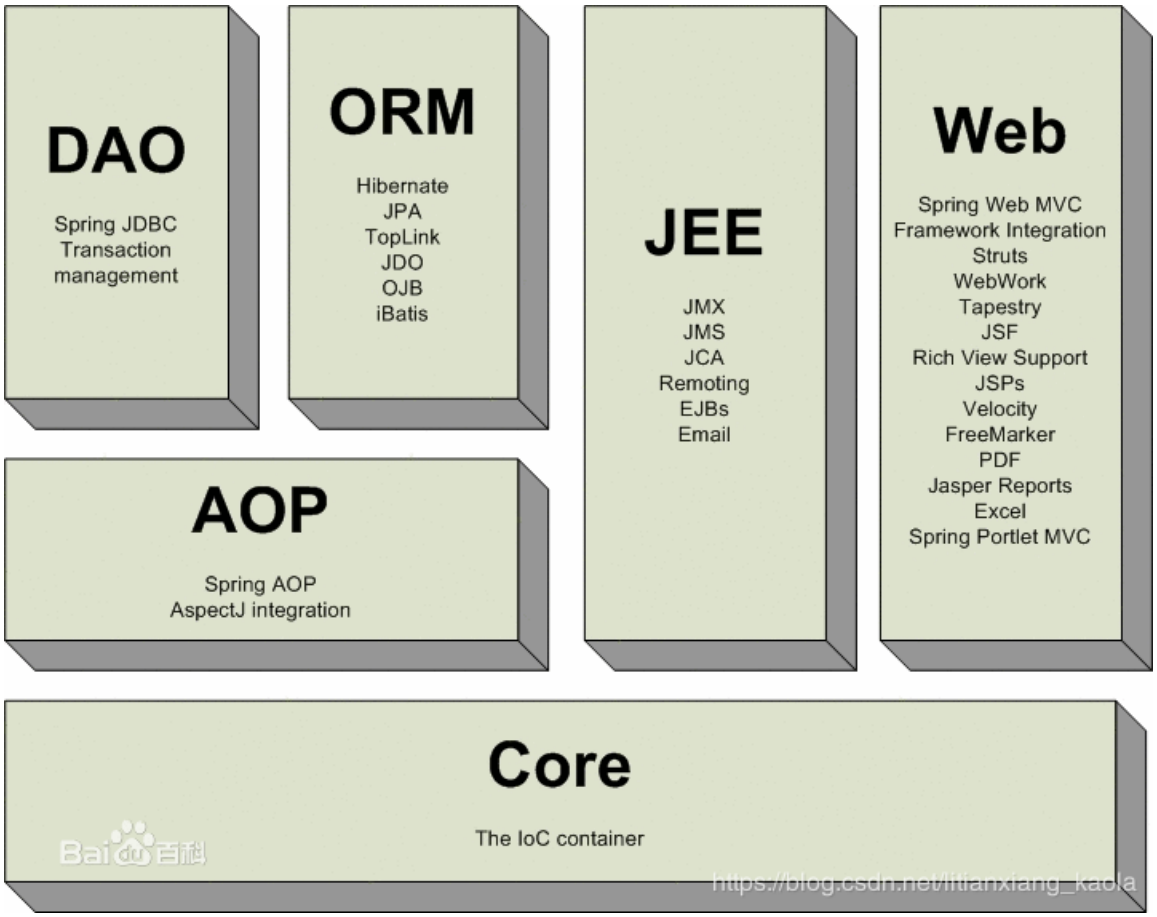
- 小结

简单介绍了 Spring、Spring MVC 和 Spring Boot，那三者有什么关系呢？

总的来说，Spring 就像一个大家族，有众多衍生产品例如 Spring Boot，Spring Security，Spring JPA 等等。但他们的基础都是 Spring 的 IOC 和 AOP，IOC 提

供了依赖注入的容器，而 AOP 解决了面向切面的编程，然后在此两者的基础上实现了其他衍生产品的高级功能；Spring MVC 是基于 Servlet 的一个 MVC 框架，主要解决 Web 开发的问题；因为 Spring 的配置非常复杂，各种 xml，properties 处理起来比较繁琐。于是为了简化开发者的使用，Spring 社区创造性地推出了 Spring Boot，它遵循约定优于配置，极大降低了 Spring 使用门槛，但又不失 Spring 原本灵活强大的功能。

下面用一张图来描述三者的关系：



Spring MVC 和 Spring Boot 都属于 Spring，Spring MVC 是基于 Spring 的一个 MVC 框架，而 Spring Boot 是基于 Spring 的一套快速开发整合包。

- MyBatis/MyBatis Plus

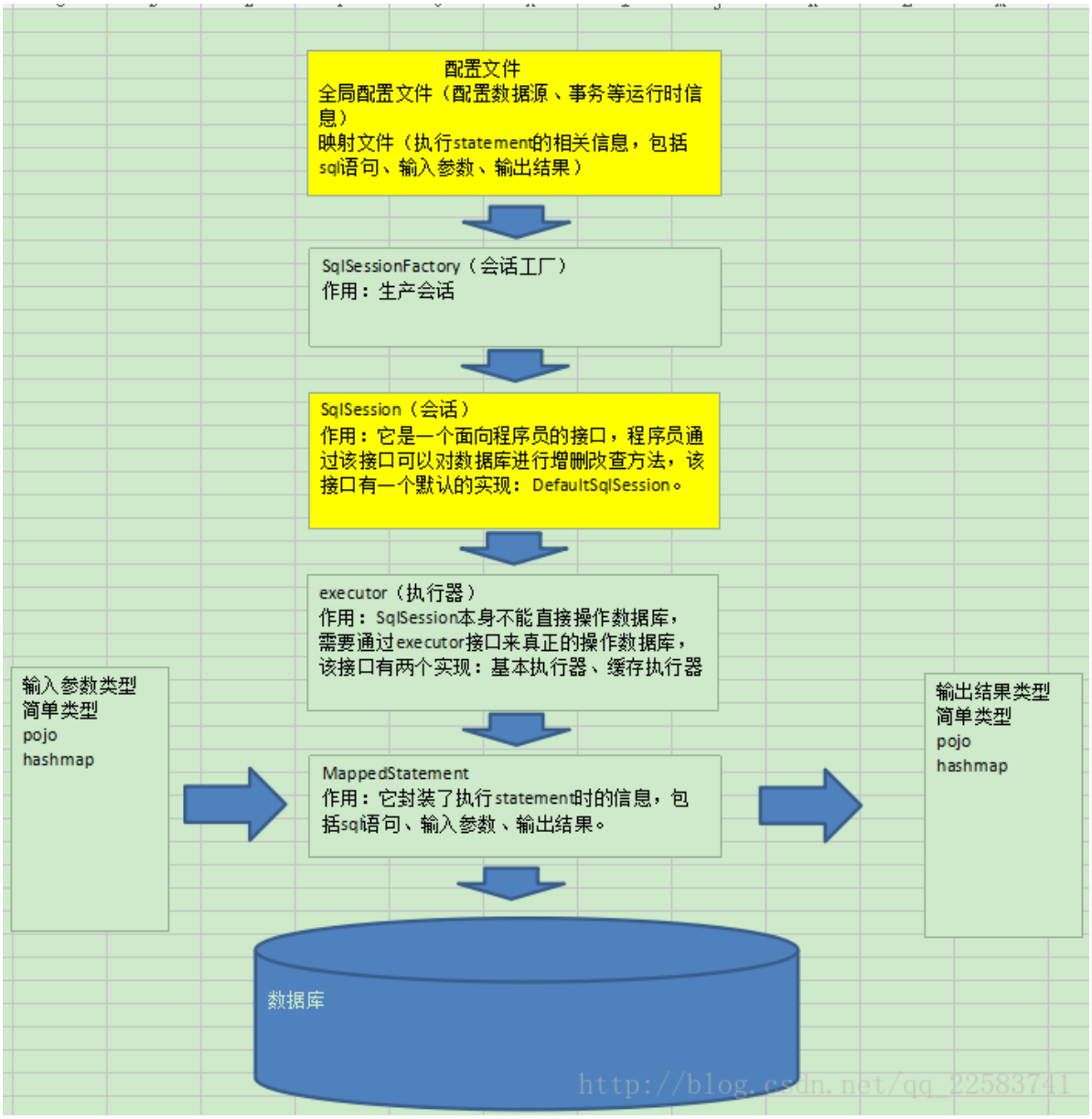
ORM（Object Relational Mapping）对象关系映射，一般指持久化数据和实体对象的映射。在没有 ORM 框架时，驱动返回类型和对象不能良好映射，而且 SQL 语句的学习成本高及易错率也高（多种数据库语句难以全部掌握）。使用 ORM 则可以改善这些问题。

现有的流行 ORM 框架主要有两个体系：.Net 和 Java 体系。MyBatis 则是 Java 体系 ORM 框架的一种，直接基于 JDBC 做简单的映射包装，所以从性能的角度来讲，JDBC 优于 MyBatis，MyBatis 优于 Hibernate。MyBatis Plus 则是在 MyBatis

上进行功能强化并简化开发、提高效率的一个开源框架。

同时，MyBatis 有两种开发形式：xml 或完全基于注解。两种形式都可以与 Spring Boot 进行深度集成。

MyBatis 框架原理:



MyBatis 核心对象：

1) SqlSessionFactory 与 SqlSessionFactoryBuilder

SqlSessionFactory 是 MyBatis 的核心对象，它是创建 SqlSession 的工厂类。

SqlSessionFactory 对象的实例可以通过 SqlSessionFactoryBuilder 对象类获得，而 SqlSessionFactoryBuilder 则可以从 XML 配置文件或一个预先定制的 Configuration 的实例构建出 SqlSessionFactory 的实例。

每一个 MyBatis 的应用程序都是以一个 SqlSessionFactory 实例为核心。

同时，SqlSessionFactory 是线程安全的，SqlSessionFactory 一旦被创建，应该在应用执行期间都存在。在应用运行期间不要重复创建多次，建议使用单例模式。

2) SqlSession

SqlSession 是 MyBatis 的关键对象，类似于 JDBC 中的 Connection，它是应用程序与持久层之间执行交互操作的一个单线程对象。

SqlSession 的底层封装了 JDBC 连接，可以用 SqlSession 实例来直接执行被映射的 SQL 语句。

SqlSession 是线程不安全的，每个线程都应该有它自己的 SqlSession 实例，SqlSession 的实例不能被共享。不能将 SqlSession 实例的引用当做一个类的静态属性。

3) Mapper:

Mapper 是从 SqlSession 中获取的(实际上获取到的是一个代理对象)，他的作用是发送 SQL 来操作数据库中的数据，它的生命周期应该是在 SqlSession 事务方法之内。

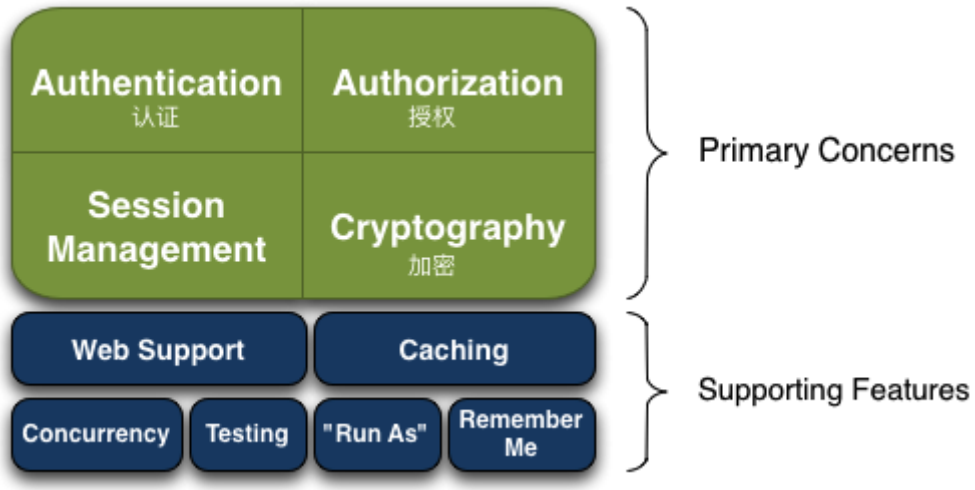
1.3. 后端相关中间件学习小结

为了实现相关的功能，该项目所用到的中间件将一一解析。

- **Shiro，用于后端鉴权 authentication**

Shiro 是一个强大的简单易用的 Java 安全框架，主要用来更便捷的认证，授权，加密，会话管理。Shiro 首要的和最重要的目标就是容易使用（相对于 Spring Security 这样的重量级框架）并且容易理解。

下面这幅图可以了解 Shiro 的特性：



Shiro 四个主要的功能

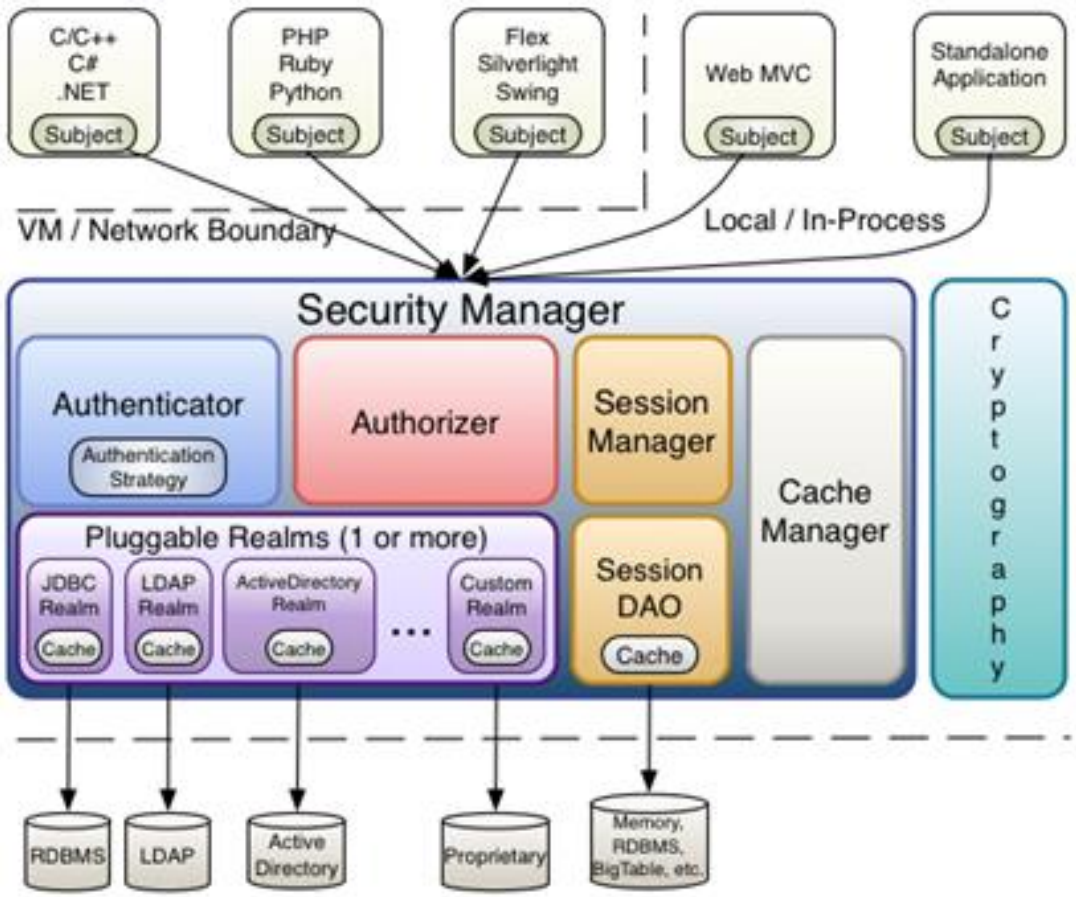
(1) Authentication：身份认证/登录，验证用户是不是拥有相应的身份；

(2) Authorization：授权，即权限验证，判断某个已经认证过的用户是否拥有某些权限访问某些资源，一般授权会有角色授权和权限授权；

(3) SessionManager：会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有信息都在会话中；会话可以是普通 JavaSE 环境的，也可以是如 Web 环境的，web 环境中作用是和 HttpSession 是一样的；

(4) Cryptography：加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储

Shiro 架构:



从图中我们可以看到不管是任何请求都会经过 SecurityManager 拦截并进行相应的处理，Shiro 几乎所有的功能都是由 SecurityManager 来管理。

其中:

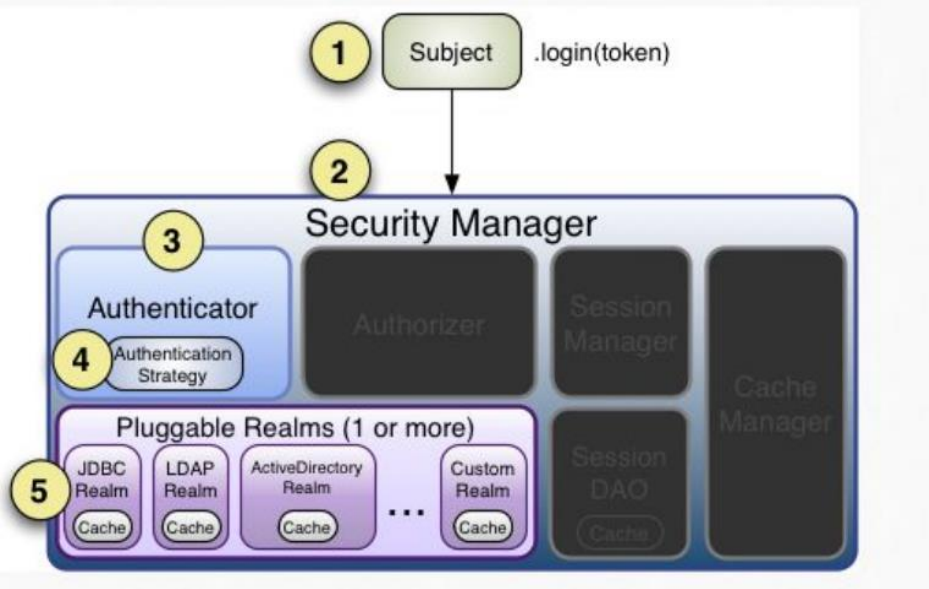
- Subject：主体，请求过来的"用户"
- SecurityManager：是 Shiro 的心脏；所有具体的交互都通过 SecurityManager 进行拦截并控制；它管理着所有 Subject、且负责进行认证和授权、及会话、缓存的管理

- Authenticator：认证器，负责主体认证的，即确定用户是否登录成功，我们可以使用 Shiro 提供的方法来认证，还可以自定义去实现，自己判断什么时候算是用户登录成功
- Authorizer：授权器，即权限授权，给 Subject 分配权限，以此很好的控制用户可访问的资源
- Realm：一般我们都需要去实现自己的 Realm，可以有 1 个或多个 Realm，即当我们进行登录认证时所获取的安全数据来源(帐号/密码)
- SessionManager：为了可以在不同的环境下使用 session 功能，Shiro 实现了自己的 SessionManager ，可以用在非 web 环境下和分布式环境下使用
- SessionDAO：对 session 的 CURD 操作
- CacheManager：缓存控制器，来管理如用户、角色、权限等的缓存的
- Cryptography：密码模块，Shiro 提高了一些常见的加密组件用于如密码加密/解密的。

了解了以上 Shiro 的核心类，则可以了解 Shiro 的主要功能的原理。

(1) 认证：Subject 认证主体包含两个信息:Principals:身份,即用户名；Credentials:凭证,即密码。

认证流程：



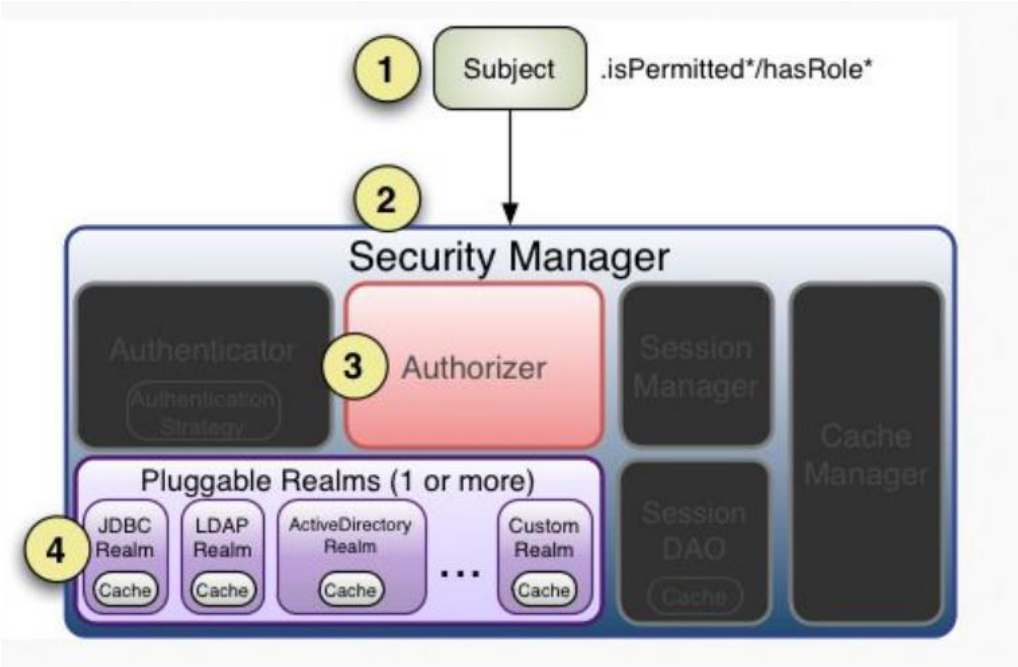
- 用户发送请求进行 Subject 认证(调用 subject.login(token))
- SecurityManager 会去 Authenticator(认证器)中查找相应的 Realms(可能不止一个)源

- Realms 可以根据不同类型的 Realm 中去查找用户信息,并进行判断是否认证成功

(2) 权限授权就是访问控制,在应用中控制谁能访问哪些资源

核心类

- 权限：即操作某个资源的权限，这些资源可以是某个链接，也可以是某个图片，也可以是对某个模块的数据的 CURL
- 角色：即权限的集合，一个角色可以有多个权限
- 用户：代表访问的用户，即 Subject



流程：

- 当用户访问应用中的某个资源时，会被 SecurityManager 拦截.
- SecurityManager 会去调用 Authorizer(授权器)
- Authorizer 会根据 Subject 的身份去相应的 Realm 中去查找该 Subject 是否有权限去访问该资源

- **Elastic Search，用于实现文章搜索功能**

Elasticsearch，基于 Lucene，隐藏复杂性，提供简单易用的 Restful API 接口、Java API 接口（还有其他语言的 API 接口）。Elasticsearch 是一个实时分布式搜索和分析引擎。它用于全文搜索、结构化搜索、分析。

- 全文检索：将非结构化数据中的一部分信息提取出来，重新组织，使其变得有一定结构，然后对此有一定结构的数据进行搜索，从而达到搜索相对较快的目的。
- 结构化检索：我想搜索商品分类为电子商品都有哪些，select * from products where category_id=‘电子’
- 数据分析：电商网站，最近一周电子类商品销量排名前 10 的商家有哪些；新闻网站，最近 1 个月访问量排名前 3 的新闻版块是哪些等等。

核心概念：Index（索引-数据库）、Type（类型-表）、Document（文档-行）、Field（字段-列）、mapping（映射-约束）。

Elastic Search 的官方文档有很方便的查找方法，因此具体操作都是直接查阅官方文档。

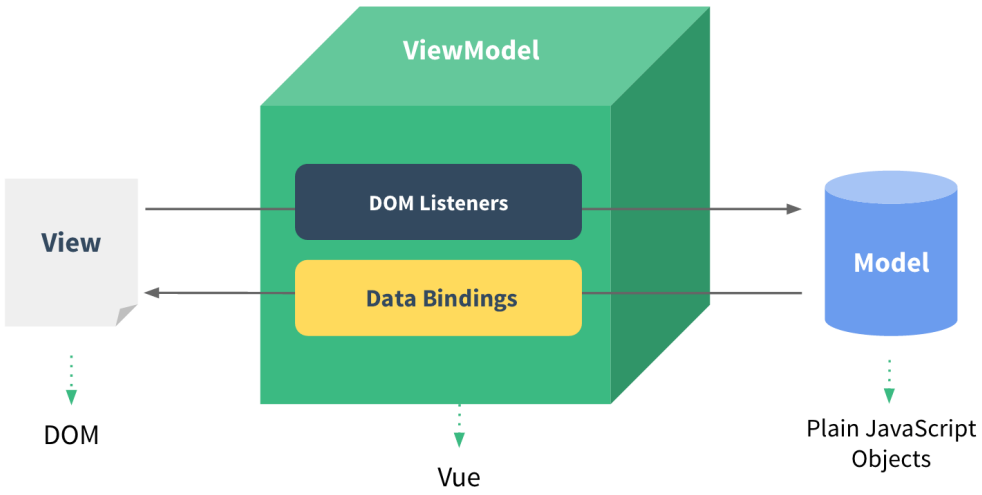
此外，程序后端中还用到了 AMQP（消息队列层）、Swagger（前后端接口文档）、Druid（数据连接池）、Kaptcha（可高度配置的实用验证码生成工具，用于生成验证码）、Jasypt（对关键数据进行加密）、Redis（用于后端数据缓存）等框架。因为相对前面几个库来说讲解这些库的意义不是特别大，篇幅限制所以略去不写。

1.4. 前端 Vue.js 和相关框架学习过程

- **Vue.js 全家桶**

Vue.js 是当下很火的一个 JavaScript MVVM 库，它是以数据驱动和组件化的思想构建的。相比于 Angular.js，Vue.js 提供了更加简洁、更易于理解的 API，使得我们能够快速地上手并使用 Vue.js。

下图不仅概括了 MVVM 模式（Model-View-ViewModel），还描述了在 Vue.js 中 ViewModel 是如何和 View 以及 Model 进行交互的。



当创建了 ViewModel 后，双向绑定是如何达成的呢？

首先，我们将上图中的 DOM Listeners 和 Data Bindings 看作两个工具，它们是实现双向绑定的关键。

从 View 侧看，ViewModel 中的 DOM Listeners 工具会帮我们监测页面上 DOM 元素的变化，如果有变化，则更改 Model 中的数据；从 Model 侧看，当我们更新 Model 中的数据时，Data Bindings 工具会帮我们更新页面中的 DOM 元素。

同时，Vue 中使用了高度灵活的组件、路由，方便开发者可以将一个个小的

部分抽离出来，实现页面的跳转。将这些代码抽离出来，放到另一个 Vue 文件中，可以进一步提高系统内部的松耦合性。Vue 的路由器一般是 Vue.js 官方提供的 Vue-Router 组件。

Vue.js 的 Ajax 部件一般使用的是 axios 框架。Axios 为 Vue.js 提供了一个松耦合度、异步的请求处理部件，方便系统提供更便捷的服务。

由于 Vue.js 项目是由各个 Vue 部件组合而成，然后系统中有一些数据是全局的，所以需要对这些全局的变量（“状态”）进行管理。Vue.js 的相关模块是 Vuex。

Vue.js 除了需要安装这些组件之外，还需要安装很多其他的库。为了方便对这些库进行管理，Vue 拥有 Node.js 为基础的 Vue-cli 库管理工具。

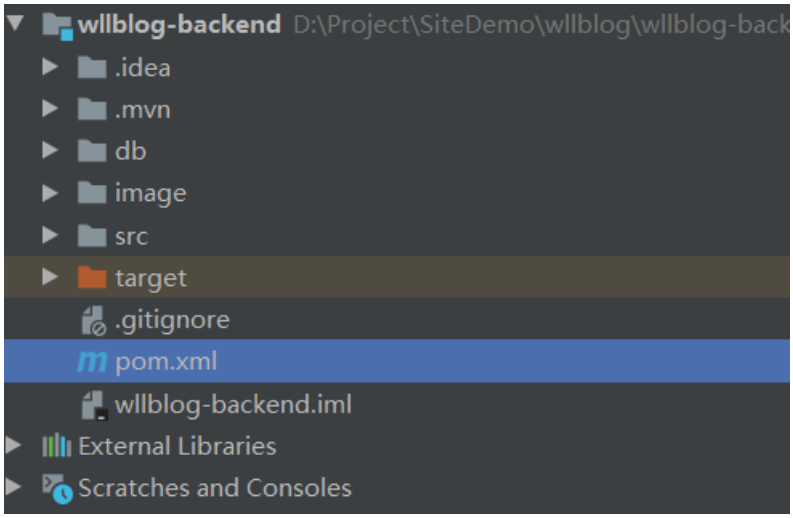
- **其他组件库**

为了进一步方便程序员开发，其他公司出现了类似 View UI 和 Element UI 这样的组件库，提供例如按钮、布局、列表、卡片等的组件，且可以进行自定义的改善。本项目展示前端使用的是 View UI，管理前端使用的是 Element UI。

前端程序员编写 CSS 样式的时候总感觉不是特别灵活，可以使用例如 Stylus 的兼容 CSS 的样式语言。Stylus 使用类“Python”的语言，对 CSS 进行灵活的开发。

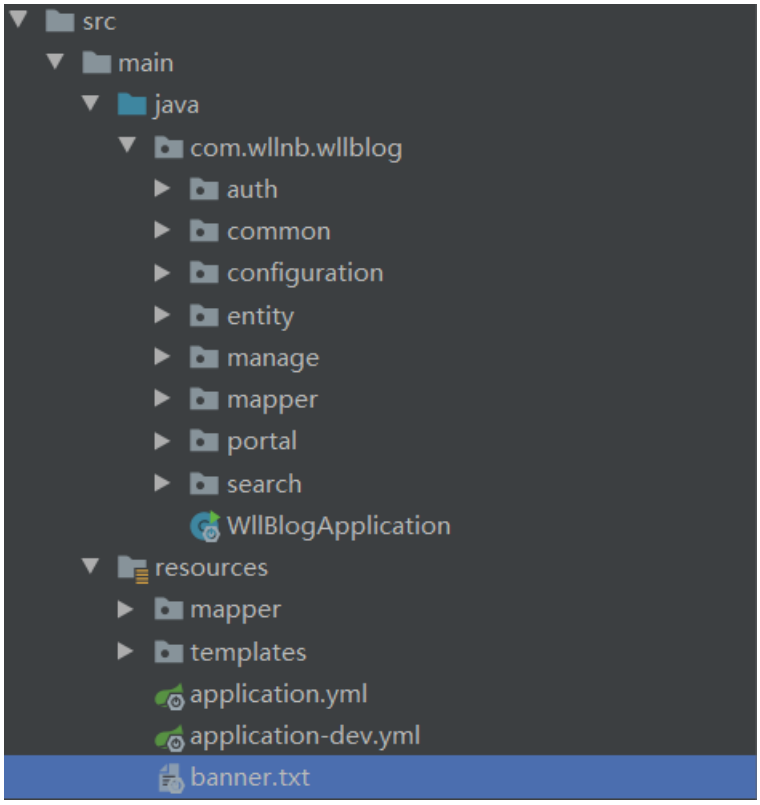
1.5. 项目后端代码讲解

- **整体代码截图：**

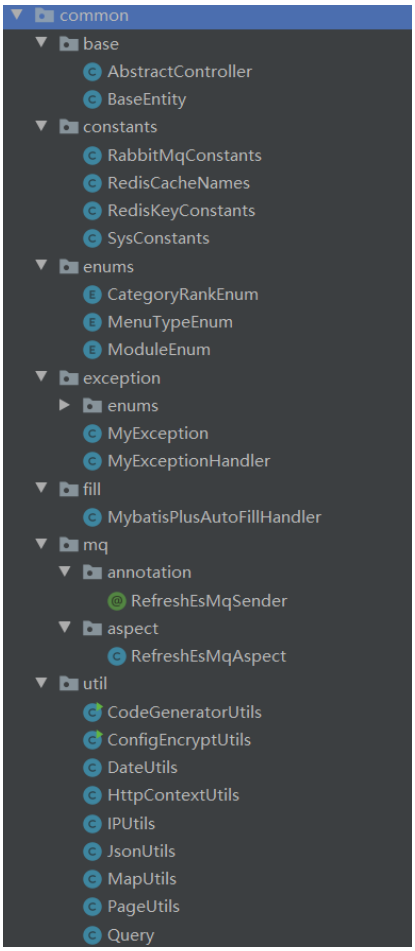


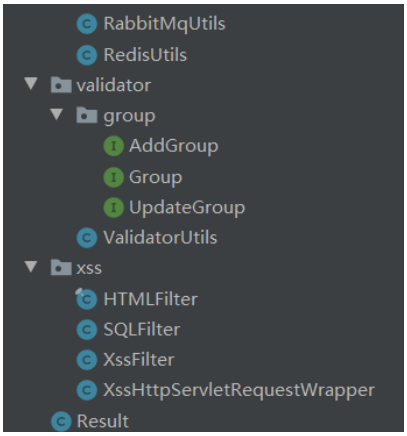
其中 db 文件夹下存放了项目数据库的相关 sql 文件。image 文件夹下有图片。src 文件夹下存放源代码。

src 文件夹结构截图：

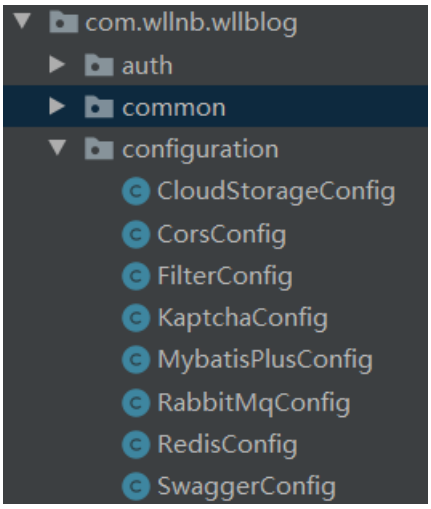


其中 src/main/java/com.wllnb.wllblog/存放源代码，src/main/resources 存放相关的 mapper 文件。

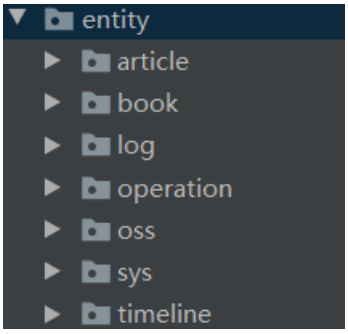


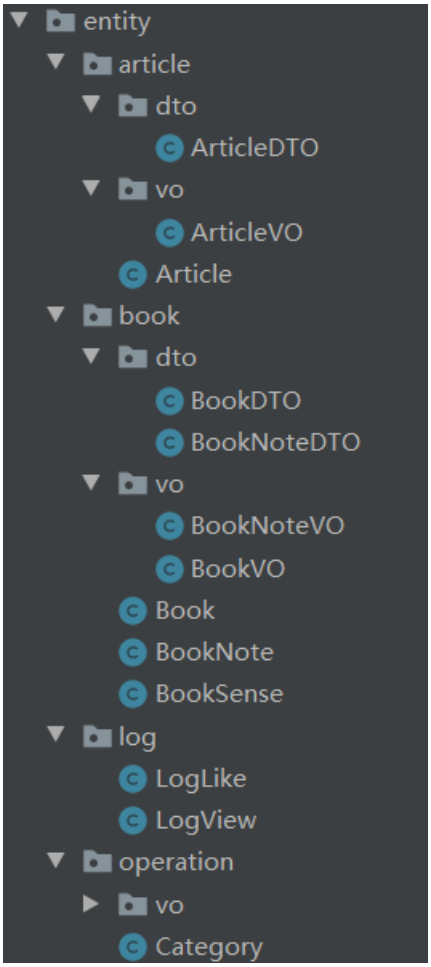


其中 src/main/java/com.wllnb.wllblog/common 存放基础的代码。Result 对象为网络请求的结果。base 文件夹存放基础的 Controller 和 BaseEntity，后续所有的 Controller 和 Entity 都分别由这两个对象继承而来。constants 存放系统常量和各个中间件的常量。enums 存放一些枚举项，如菜单分级枚举、分类级别枚举、模块枚举。exception 存放 exception 处理(Handler)、错误枚举（404 等等）和自定义 exception 对象。fill 存放 MyBatisPlus 自动填充的字段对象。mq 存放自定义的 Aspect 和 annotation，用于 AMQP 的消息发送和接收。util 中则存放各种集合包。Validator 存放验证对象，在各个对象需要校验时使用。xss 则存放前端传送来的数据的 Filter 对象，防止 SQL 注入、HTML 注入。

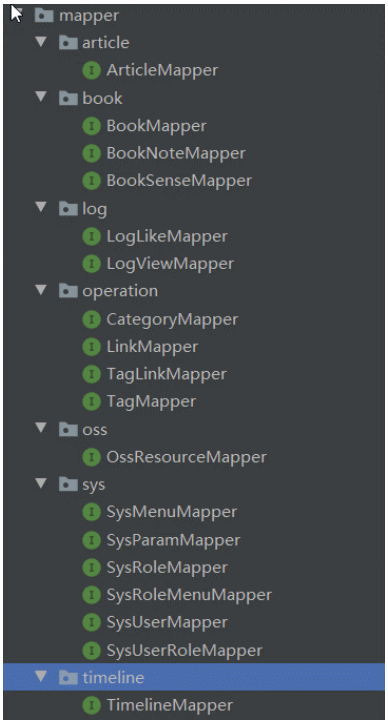


src/main/java/com.wllnb.wllblog/configuration 存放配置代码。



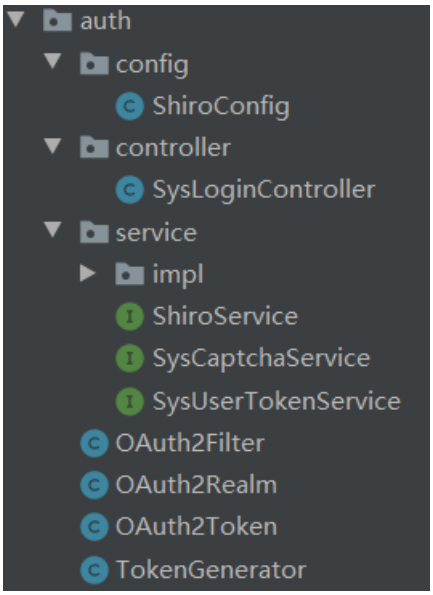


src/main/java/com.wllnb.wllblog/entity 存放项目的实体对象。每个对象其中又分 DTO、VO 层次编写对象。

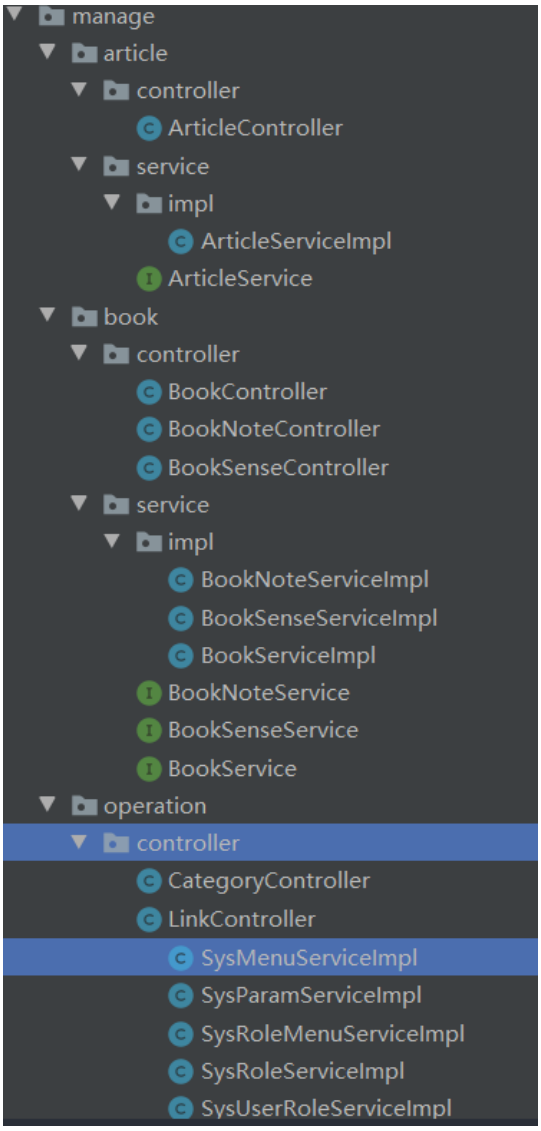


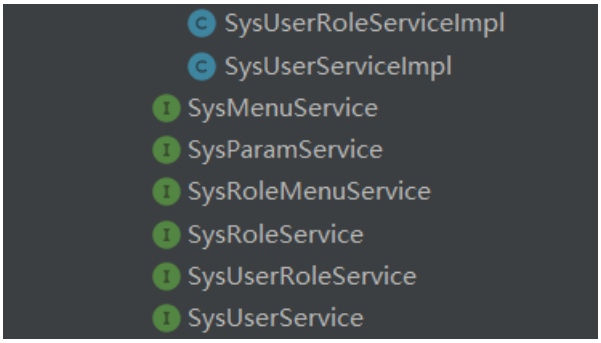
src/main/java/com.wllnb.wllblog/mapper 存放项目与数据库进行 ORM 的

各个实体的接口。

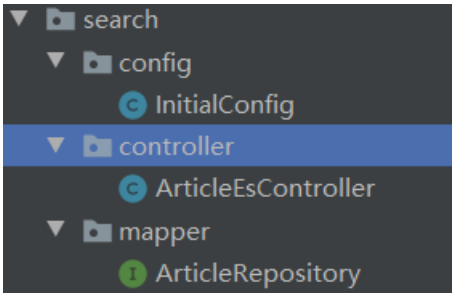


src/main/java/com.wllnb.wllblog/auth 存放验证服务代码。

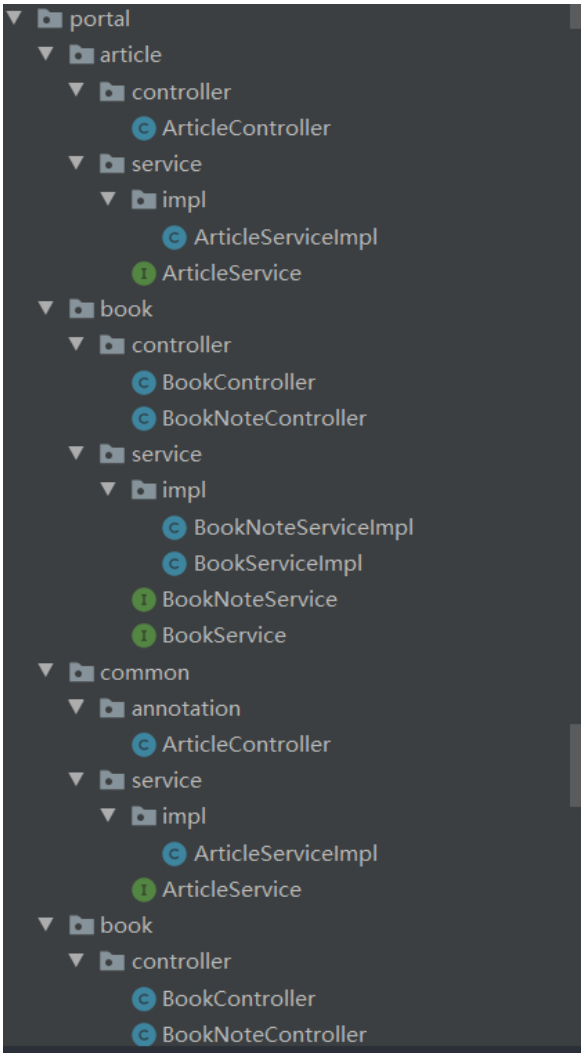


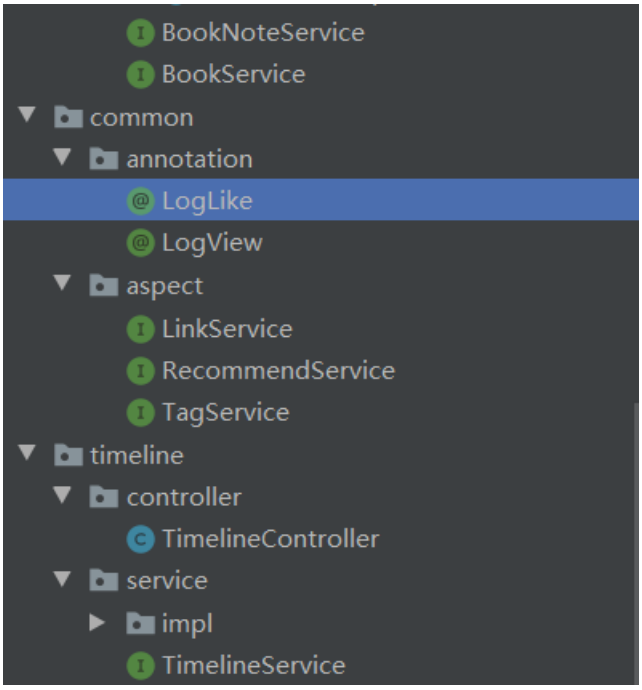


src/main/java/com.wllnb.wllblog/manage 存放后台管理部分的逻辑代码。



src/main/java/com.wllnb.wllblog/search 对文章进行搜索。





src/main/java/com.wllnb.wllblog/portal 存放对移动设备的支持的代码。

- **依赖项（pom.xml 文件）**

  项目使用到的依赖项主要为：lombok（getter 和 setter）、swagger、spring-aop（实现 spring 的 AOP 功能）、kaptcha（验证码）、qiniu（七牛云图床的依赖）、jasypt（实现对七牛云图床的密钥进行再加密，防止代码中出现明文密钥）、mysql-connector（Spring Boot 连接 MySQL）、mybatis 和 mybatis-plus、druid（数据库连接层）、redis、elasticsearch、amqp。pom.xml 配置了对上述依赖项的管理（添加和版本管理）。

- **依赖项配置（./src/main/resources/application.yml 和 application-dev.yml）**

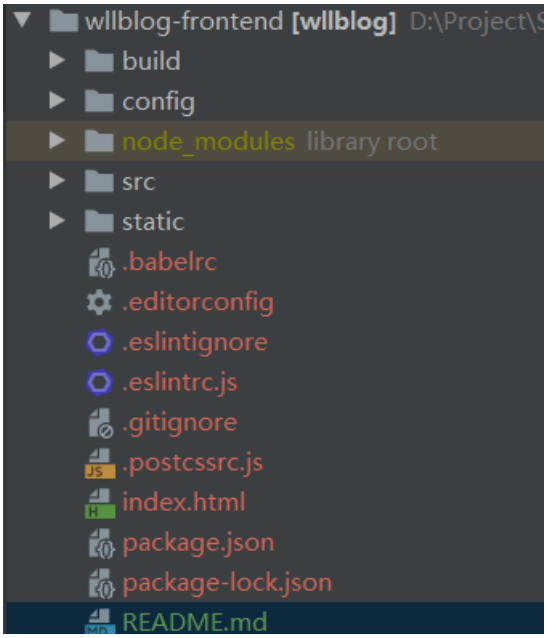
  配置上述依赖项。唯一需要注意的是要运行 Elastic Search 的话，需要在本地创建好如下的 cluster 节点：

```
data:
  elasticsearch:
    #      rest:
    #        uris: http://localhost:9300
    #      repositories:
    #        enabled: true
    cluster-name: docker-cluster
    cluster-nodes: 127.0.0.1:9300
```

还有一方面，由于本人已经租好了国内服务器。由于限制，项目具体上线需要备案域名，而我卡在域名备案上，整体项目也不能上线。七牛云图床测试的域名超过了 30 天，测试域名也已经被收回，导致本来存储在图床 bucket 上的图片无法正常加载。

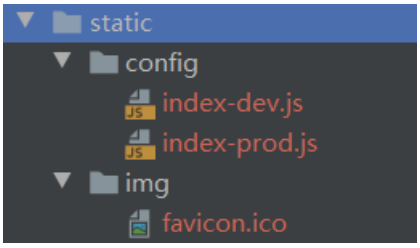
1.6. 后台管理前端

- **项目文件结构展示：**

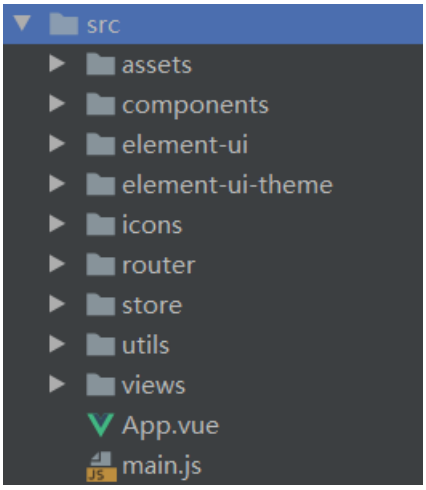


由于前端项目使用的是 Vue-cli 2.x，所以项目文件目录的结构看上去比较混乱。如果使用 Vue-cli 3.x 版本，整体看上去会比较清晰一点。

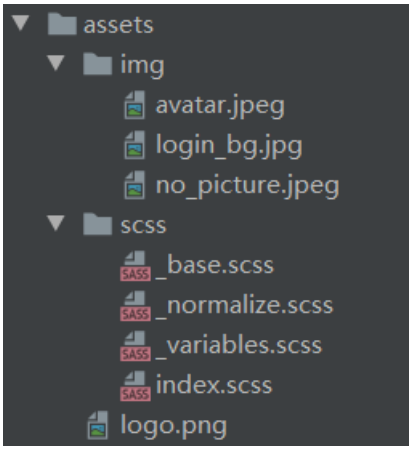
文件夹下的 build、config、node_modules 文件夹是自动生成的，不需要进行管理，我们关心的是 src 和 static 文件夹和整体项目下的 index.html 文件和 package.json 文件。



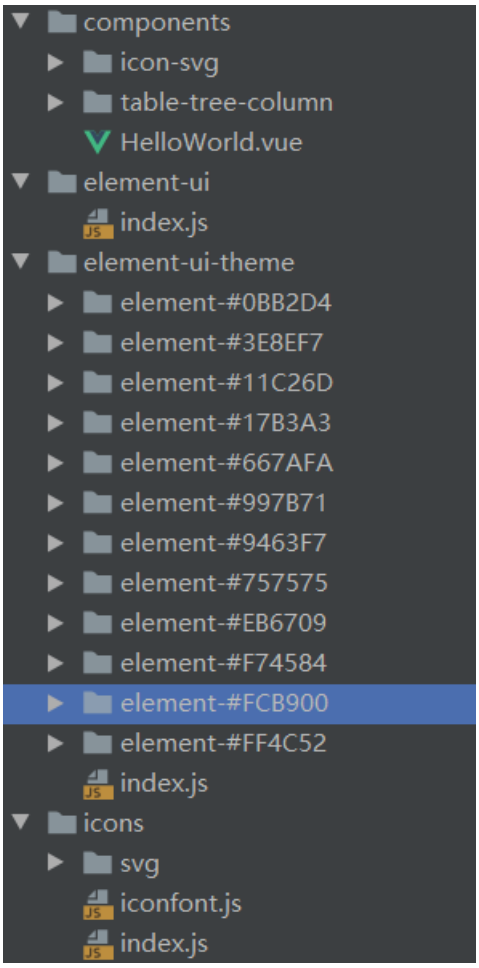
static 文件夹下有不同生产环境的配置文件，可以设置相关的请求 URL。



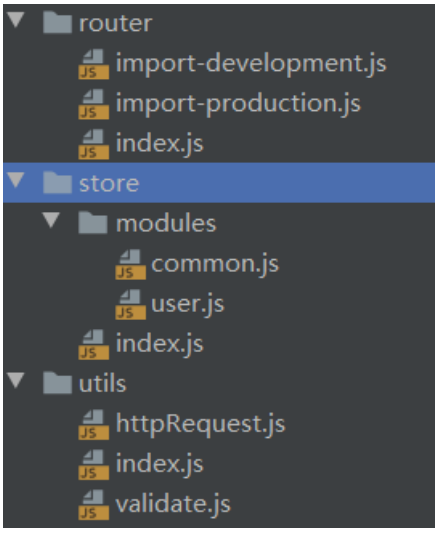
src 文件夹目录结构如上。main.js 进行项目的整体配置，App.vue 进行初始界面的配置。



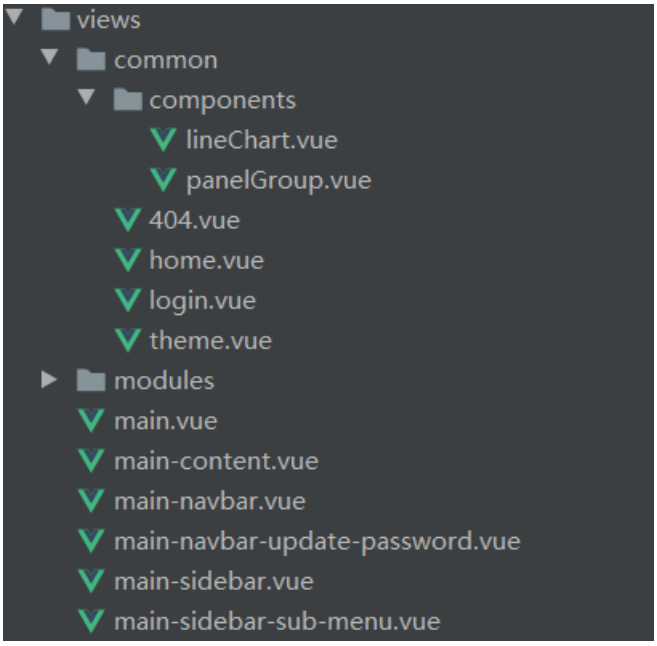
src/assets 文件夹下存放静态的图片和全局的 SCSS 配置文件。



src/components、src/element-ui、src/icons 三个文件夹存放的是配色方案、icons 等组件。



src/router 存放的是项目的路由信息。src/store 存放的是系统的全局状态。src/utils 存放的是封装过后的请求处理、一些 utils 函数还有验证用户输入的函数。



src/views 存放的是项目的视图信息。

由于前端项目的目录结构和前端后台管理的目录结构类似，所以此处不再列出。

1.7 前端功能展示

- **博客主界面**


WLL's Blog Self-Discipline and Social Commitment.
文章 阅读 时光轴 关于
文章 | Articles View More
最新 点赞最多 推荐
关于本站和博主 置顶
本站相关 关于
关于本站和博主 查看更多
At time / 2020-2-2 52 阅读 1 喜欢
阅读 | Books View More
深入理解Java虚拟机（第2版） 正在阅读
Java JVM
周志明，资深Java技术专家，对JavaEE企业级应用开发、OSGi、Java虚拟机和工作流等都有深入的研究，并在大量的实践中积累了丰富的经... 查看更多
At time / 2019-3-4 20 阅读 0 喜欢
笔记 | Notes View More
最新 点赞最多 推荐
WLLNB
Write the code and keep it.
技能值
Java
Vue
Go
Cloud
推荐阅读
最热阅读
关于本站和博主
本站相关 关于
2020-2-2 54 1
关于本站和博主
友情链接
标签墙
Java [4] JVM [2]
本站相关 [1] 关于 [1]
面试 [1] ElasticSearch [1]
XXXXX --------- 版权所有©2019 - 2020 --------- 以商业目的使用本网站内容需获许可，非商业目的使用授权遵循CC BY-NC 4.0

界面展示：

点击 nav-bar“文章”，可查看文章列表界面：

目前只有一篇文章，后续再继续插入数据。

点入文章，可查看文章信息：

点入书籍，可查看书籍信息：

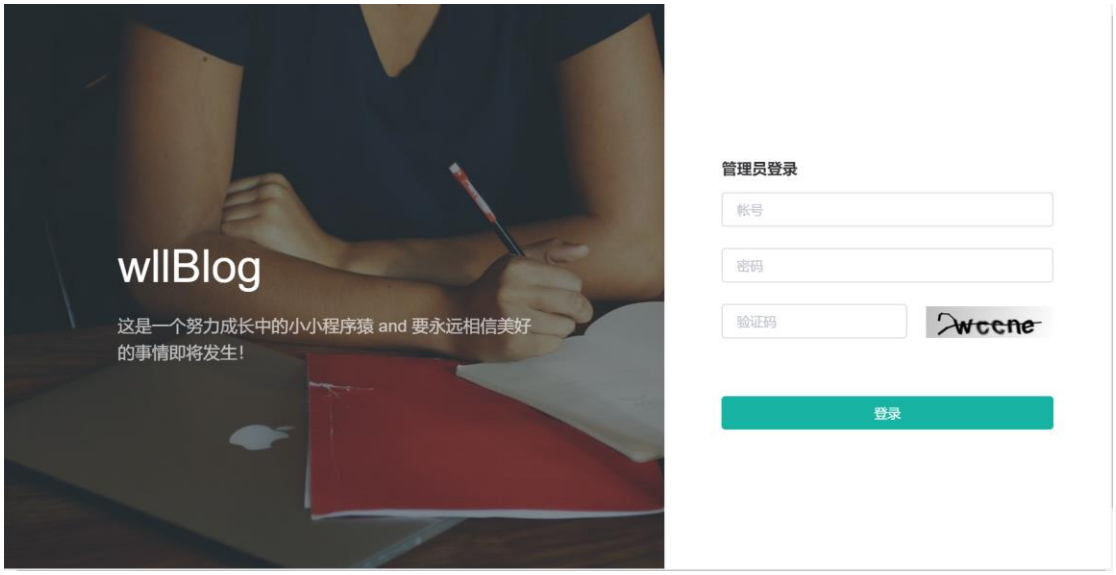
书籍下方设置有评论区，可以通过登录 GitHub 进行评论。

点击 nav-bar“阅读”，可查看多级阅读界面：

点击 nav-bar“时光轴”，可查看时光轴

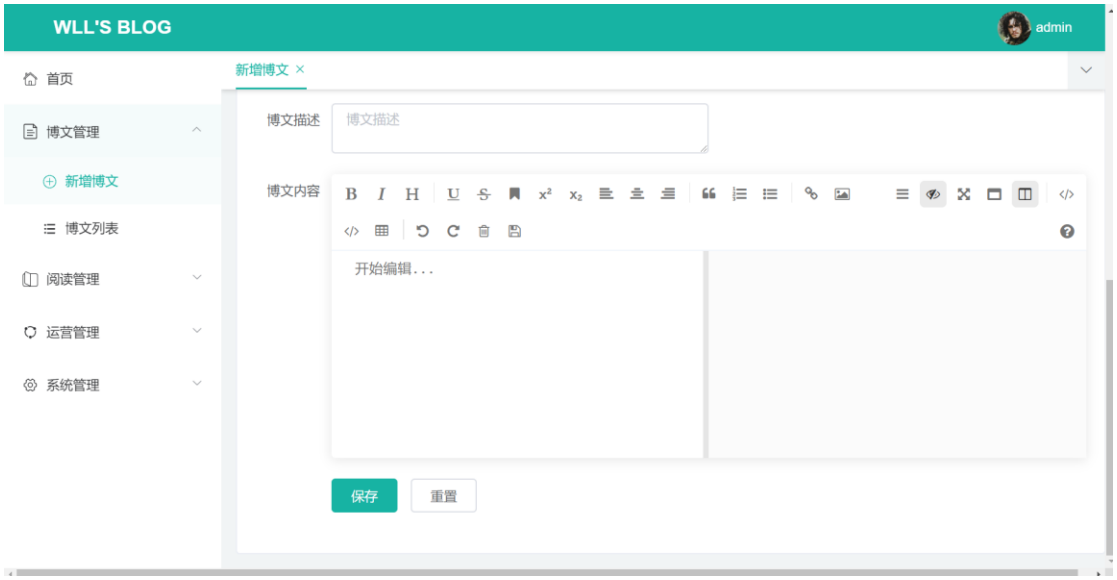
- **后台管理前端主界面：**

  Login 界面

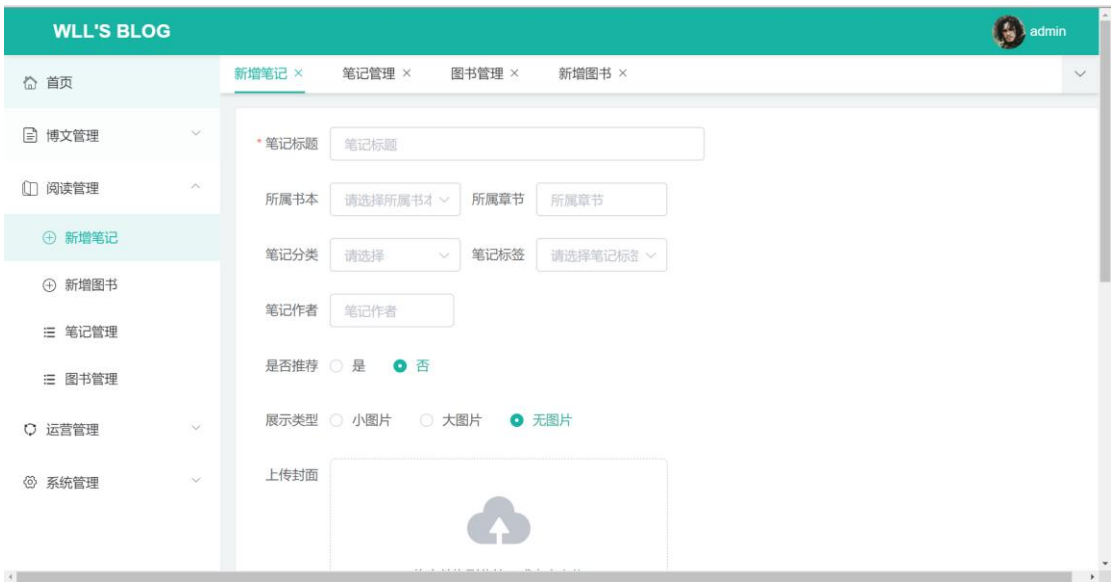


输入管理员账号/密码，验证码后，可进入管理主页面：

点击左侧博文管理菜单下的新增博文、博文列表可以添加、查看博文：



点击左侧阅读管理菜单下的新增笔记、新增图书、笔记管理、图书管理可以添加、查看笔记和图书（界面和上述相似，截图不再给出）：



点击左侧运营管理菜单下的分类管理、标签管理、友联管理、推荐管理可以管理相关类型对象（界面和上述相似，截图不再给出）：

点击左侧系统管理菜单下的管理员列表、角色管理、菜单管理、系统参数、SQL 监控可以管理相关类型对象（界面和上述相似，截图不再给出）：

项目的基本功能已经展示完毕。

2 对计算思维实训的认识与体会

个人感觉，计算思维是使用计算机解决问题的一种思维。无论是以工程角度还是以科学角度，都以计算思维利用计算机为工具解决问题。当然，问题有宏观微观之分。如果系统比较庞大，则需要对系统的功能进行分解再逐步求解。

虽然这门课只有两个学分，但通过自学，可以让它给个人带来的价值变得很大。从刚开始连怎么入门相关框架都不知道，到现在已经能独立使用这个框架进行小项目开发。当然，越学习越感觉自己能力的不足，越学习越感觉自己需要学习。

这也是我第一次使用现在比较流行的 Web 框架，知道需要学习的还很多。虽然这个网站还有很多功能上的不足，但写这个网站的过程显得尤为重要。这个网站也花费了我大量的时间，甚至不是因为疫情给我带来了一两周的时间，我无法完成这个项目。也希望自己能更加努力。