技术开发文档

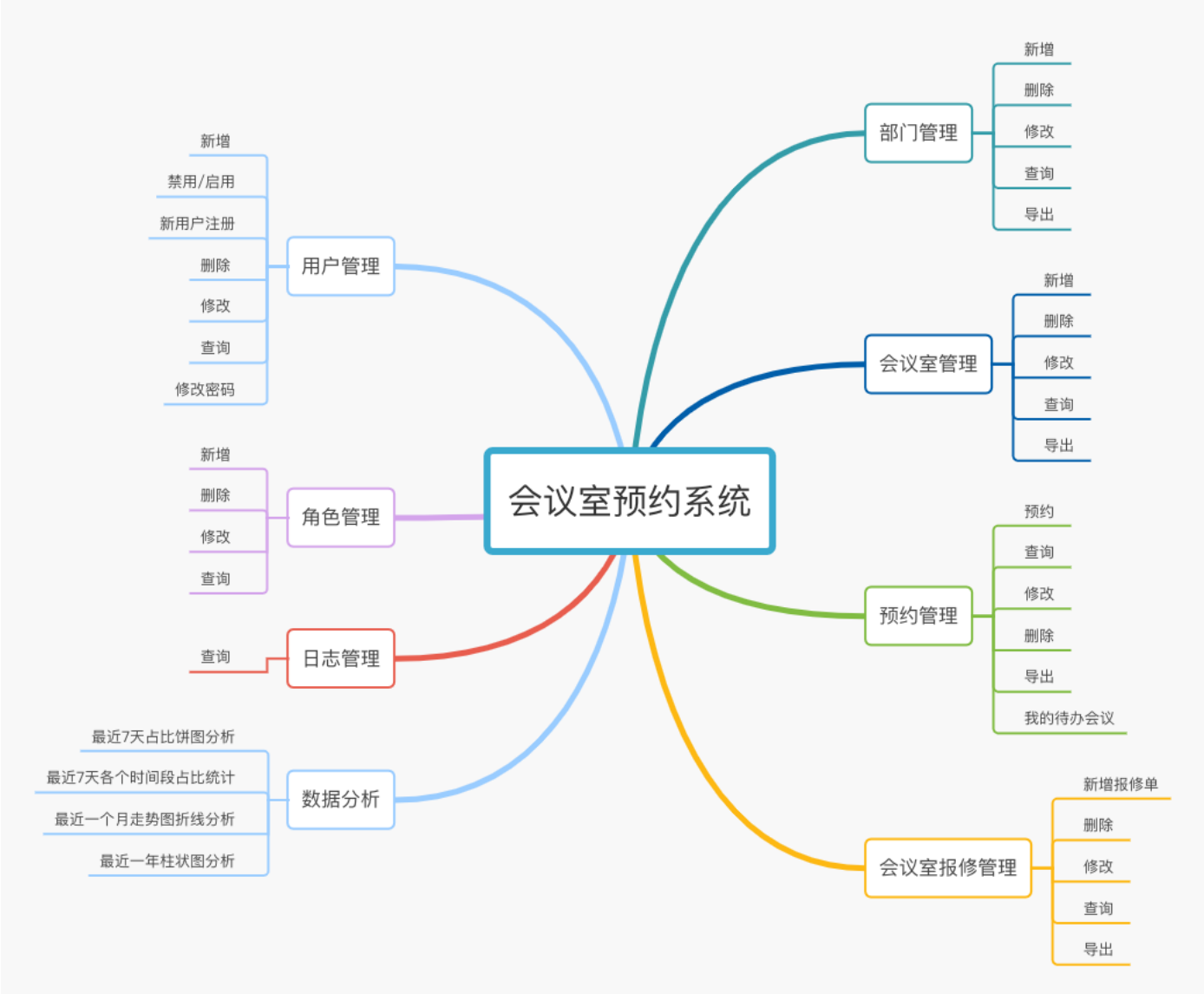
前言

为了各位同学可以更好的理解和学习整个项目，学长花了大量的时间写下这篇技术开发文档，希望大家认真去看。学长会通过大量画图去讲解，这样更便于大家理解。这里跟大家推荐一款画图软件，不需要安装，可以直接在线画图，我个人非常喜欢，这款软件名称叫做**processon**，学长画的图也都是基于这款软件画的。

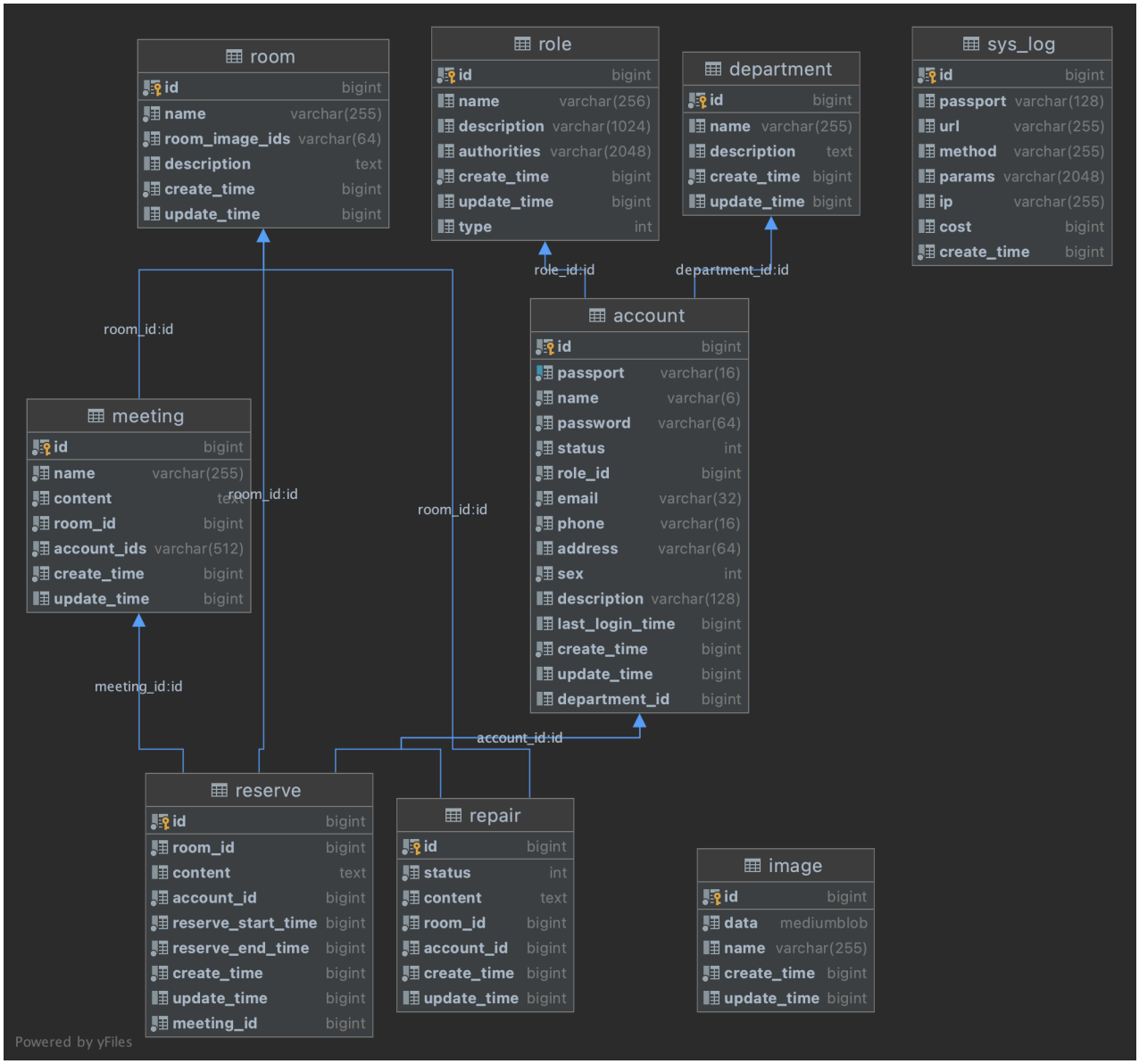
点击学长分享的链接去注册，可以领取7天会员： https://www.processon.com/i/5c03ae0fe4b0f012f23affd5

另外，大家写论文时不要直接复制学长的文档，这样重复率容易太高。最好是在理解学长内容的基础上，用自己的文字来表述，还有画图也是一样，你可以参考，但是不建议直接截图使用，这个大家**切记、切记、切记！！！**

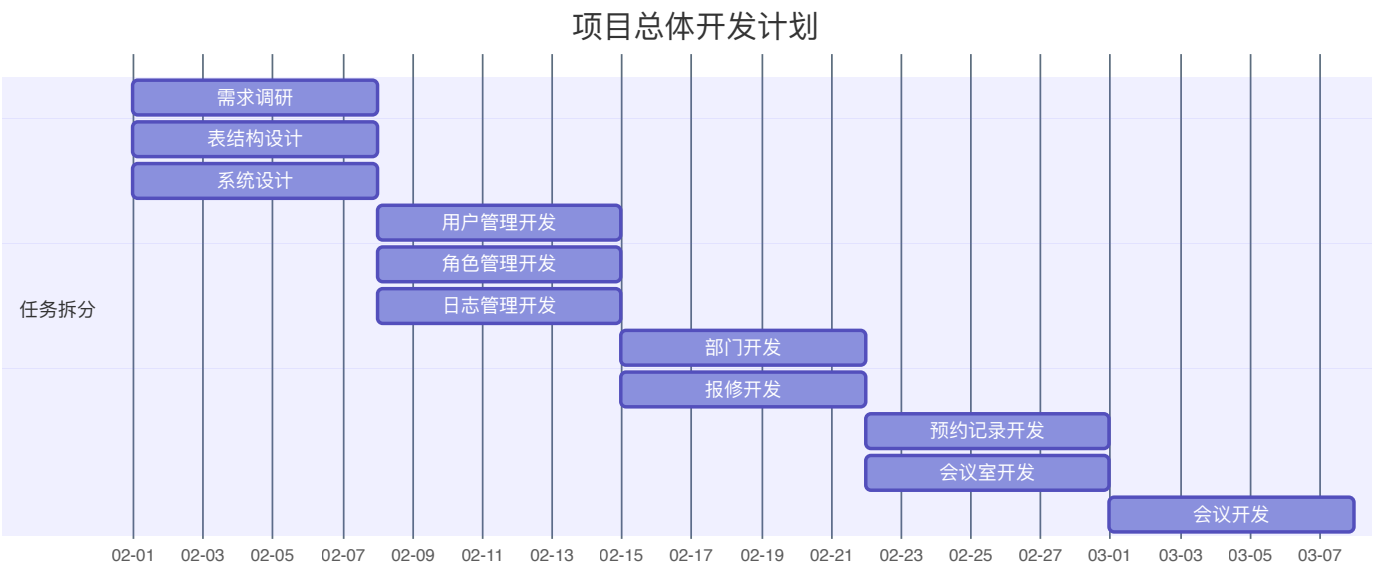
总体功能设计



数据库ER图



项目开发计划



硬件&软件要求

1. **开发工具**：基于**Intellij IDEA**开发 (简称Idea)，**idea**可以用来开发前后端代码；
2. **数据库**：使用**MySQL8**关系型数据库（不支持MySQL5），数据库操作工具推荐使用**Navicat**；
3. **编程语言**：后端基于**Java**开发，**jdk1.8**
4. **缓存**：使用**Redis**内存数据库
5. **CPU要求**：至少**1核1G**内存的机器
6. **磁盘要求**：至少需要**200M**磁盘可用空间

技术点&框架

Java

本项目后端开发语言用得是Java，jdk版本是1.8。这里学长简单介绍下Java，主要是帮助一些小白同学更好的理解这个技术，如果你是老手了，那直接跳过这部分。

什么是Java?

Java 是由 Sun Microsystems 公司于 1995 年 5 月推出的 Java 面向对象程序设计语言和 Java 平台的总称。由 James Gosling和同事们共同研发，并在 1995 年正式推出。

后来 Sun 公司被 Oracle （甲骨文）公司收购，Java 也随之成为 Oracle 公司的产品。

Java分为三个体系：

- JavaSE（J2SE）（Java2 Platform Standard Edition，java平台标准版）

- JavaEE(J2EE)(Java 2 Platform,Enterprise Edition，java平台企业版)
- JavaME(J2ME)(Java 2 Platform Micro Edition，java平台微型版)。

2005 年 6 月，JavaOne 大会召开，SUN 公司公开 Java SE 6。此时，Java 的各种版本已经更名，以取消其中的数字"2"：J2EE 更名为 Java EE，J2SE 更名为Java SE，J2ME 更名为 Java ME。

Java语言的特点：

- **Java 语言是简单的：**

  Java 语言的语法与 C 语言和 C++ 语言很接近，使得大多数程序员很容易学习和使用。另一方面，Java 丢弃了 C++ 中很少使用的、很难理解的、令人迷惑的那些特性，如操作符重载、多继承、自动的强制类型转换。特别地，Java 语言不使用指针，而是引用。并提供了自动分配和回收内存空间，使得程序员不必为内存管理而担忧。

- **Java 语言是面向对象的：**

  Java 语言提供类、接口和继承等面向对象的特性，为了简单起见，只支持类之间的单继承，但支持接口之间的多继承，并支持类与接口之间的实现机制（关键字为 implements）。Java 语言全面支持动态绑定，而 C++语言只对虚函数使用动态绑定。总之，Java语言是一个纯的面向对象程序设计语言。

- **Java语言是分布式的：**

  Java 语言支持 Internet 应用的开发，在基本的 Java 应用编程接口中有一个网络应用编程接口（java net），它提供了用于网络应用编程的类库，包括 URL、URLConnection、Socket、ServerSocket 等。Java 的 RMI（远程方法激活）机制也是开发分布式应用的重要手段。

- Java 语言是健壮的：

  Java 的强类型机制、异常处理、垃圾的自动收集等是 Java 程序健壮性的重要保证。对指针的丢弃是 Java 的明智选择。Java 的安全检查机制使得 Java 更具健壮性。

- **Java语言是安全的：**

  Java通常被用在网络环境中，为此，Java 提供了一个安全机制以防恶意代码的攻击。除了Java 语言具有的许多安全特性以外，Java 对通过网络下载的类具有一个安全防范机制（类 ClassLoader），如分配不同的名字空间以防替代本地的同名类、字节代码检查，并提供安全管理机制（类 SecurityManager）让 Java 应用设置安全哨兵。

- **Java 语言是体系结构中立的：**

  Java 程序（后缀为 java 的文件）在 Java 平台上被编译为体系结构中立的字节码格式（后缀为 class 的文件），然后可以在实现这个 Java 平台的任何系统中运行。这种途径适合于异构的网络环境和软件的分发。

- **Java 语言是可移植的：**

  这种可移植性来源于体系结构中立性，另外，Java 还严格规定了各个基本数据类型的长度。Java 系统本身也具有很强的可移植性，Java 编译器是用 Java 实现的，Java 的运行环境是用 ANSI C 实现的。

- **Java 语言是解释型的：**

  如前所述，Java 程序在 Java 平台上被编译为字节码格式，然后可以在实现这个 Java 平台的任何系统中运行。在运行时，Java 平台中的 Java 解释器对这些字节码进行解释执行，执行过程中需要的类在联接阶段被载入到运行环境中。

- **Java 是高性能的：**

  与那些解释型的高级脚本语言相比，Java 的确是高性能的。事实上，Java 的运行速度随着 JIT(Just-In-Time)编译器技术的发展越来越接近于 C++。

- **Java 语言是多线程的：**

  在 Java 语言中，线程是一种特殊的对象，它必须由 Thread 类或其子（孙）类来创建。通常有两种方法来创建线程：其一，使用型构为 Thread(Runnable) 的构造子类将一个实现了 Runnable 接口的对象包装成一个线程，其二，从 Thread 类派生出子类并重写 run 方法，使用该子类创建的对象即为线程。值得注意的是 Thread 类已经实现了 Runnable 接口，因此，任何一个线程均有它的 run 方法，而 run 方法中包含了线程所要运行的代码。线程的活动由一组方法来控制。Java 语言支持多个线程的同时执行，并提供多线程之间的同步机制（关键字为 synchronized）。

- **Java 语言是动态的：**

  Java 语言的设计目标之一是适应于动态变化的环境。Java 程序需要的类能够动态地被载入到运行环境，也可以通过网络来载入所需要的类。这也有利于软件的升级。另外，Java 中的类有一个运行时刻的表示，能进行运行时刻的类型检查。

Java学习文档

基础的语法可以网上找个教程自己练练手，如果想系统的学习，那买本Java相关的书籍从头到尾看一遍（多动手）

Java教程： **https://www.runoob.com/java/java-intro.html**

SpringBoot

SpringBoot指得是一种开发框架，是基于Java语言开发的，这里大家不要把Java跟SpringBoot搞混淆了。Java是一种语言，而SpringBoot是基于Java这门语言开发的框架，主要是帮助开发者更加快速、高效的开发项目。下面学长简单介绍下什么是SpringBoot，如果想了解更详细的学习资料，大家可以网上百度下相关的教程。

什么是SpringBoot?

Spring Boot是由Pivotal团队提供的全新框架，其设计目的是用来简化新Spring应用的初始搭建以及开发过程。该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。通过这种方式，Spring Boot致力于在蓬勃发展的快速应用开发领域(rapid application development)成为领导者。

SpringBoot基于Spring4.0设计，不仅继承了Spring框架原有的优秀特性，而且还通过简化配置来进一步简化了Spring应用的整个搭建和开发过程。另外SpringBoot通过集成大量的框架使得依赖包的版本冲突，以及引用的不稳定性等问题得到了很好的解决。
SpringBoot所具备的特征有：
（1）可以创建独立的Spring应用程序，并且基于其Maven或Gradle插件，可以创建可执行的JARs和WARs；
（2）内嵌Tomcat或Jetty等Servlet容器；
（3）提供自动配置的“starter”项目对象模型（POMS）以简化Maven配置；
（4）尽可能自动配置Spring容器；
（5）提供准备好的特性，如指标、健康检查和外部化配置；
（6）绝对没有代码生成，不需要XML配置。

SpringBoot框架中还有两个非常重要的策略：开箱即用和约定优于配置。开箱即用，Outofbox，是指在开发过程中，通过在MAVEN项目的pom文件中添加相关依赖包，然后使用对应注解来代替繁琐的XML配置文件以管理对象的生命周期。这个特点使得开发人员摆脱了复杂的配置工作以及依赖的管理工作，更加专注于业务逻辑。约定优于配置，Convention over configuration，是一种由SpringBoot本身来配置目标结构，由开发者在结构中添加信息的软件设计范式。这一特点虽降低了部分灵活性，增加了BUG定位的复杂性，但减少了开发人员需要做出决定的数量，同时减少了大量的XML配置，并且可以将代码编译、测试和打包等工作自动化。

SpringBoot学习文档

官方文档： **https://spring.io/quickstart**

springboot入门教程： **https://www.liaoxuefeng.com/wiki/1252599548343744/1266265175882464**

Vue

咱们这个项目是前后端分离的项目。前面已经说过了，后端是基于Java语言和SpringBoot后端开发框架开发的，现在讲得前端部分就是基于Vue开发的，学长先简单介绍一下Vue，想深入学习的，多百度多动手。

什么是Vue?

在解释什么是Vue之前，学长想先跟大家解释下Vue、HTML、JavaScript、CSS这几项技术间的关系。说到web前端，大家都知道是用HTML和js还是css样式去开发的，下面学长从维基百科摘录了这些技术名词的官方定义：

HTML技术： 超文本标记语言（英语：HyperText Markup Language，简称：HTML）是一种用于创建网页的标准标记语言。HTML是一种基础技术，常与CSS、JavaScript一起被众多网站用于设计网页、网页应用程序以及移动应用程序的用户界面[3]。网页浏览器可以读取HTML文件，并将其渲染成可视化网页。HTML描述了一个网站的结构语义随着线索的呈现，使之成为一种标记语言而非编程语言。

HTML元素是构建网站的基石。HTML允许嵌入图像与对象，并且可以用于创建交互式表单，它被用来结构化信息——例如标题、段落和列表等等，也可用来在一定程度上描述文档的外观和语义。HTML的语言形式为尖括号包围的HTML元素（如），浏览器使用HTML标签和脚本来诠释网页内容，但不会将它们显示在页面上。

HTML可以嵌入如JavaScript的脚本语言，它们会影响HTML网页的行为。网页浏览器也可以引用层叠样式表（CSS）来定义文本和其它元素的外观与布局。维护HTML和CSS标准的组织万维网联盟（W3C）鼓励人们使用CSS替代一些用于表现的HTML元素。

JavaScript技术： JavaScript（通常缩写为JS）是一种高级的、解释型的编程语言[8]。JavaScript是一门基于原型、头等函数的语言[9]，是一门多范式的语言，它支持面向对象程序设计，指令式编程，以及函数式编程。它提供语法来操控文本、数组、日期以及正则表达式等，不支持I/O，比如网络、存储和图形等，但这些都可以由它的宿主环境提供支持。它已经由ECMA（欧洲电脑制造商协会）通过ECMAScript实现语言的标准化[8]。它被世界上的绝大多数网站所使用，也被世界主流浏览器（Chrome、IE、Firefox、Safari、Opera）支持。

JavaScript与Java在名字或语法上都有很多相似性，但这两门编程语言从设计之初就有很大的不同，JavaScript的语言设计主要受到了Self（一种基于原型的编程语言）和Scheme（一门函数式编程语言）的影响[9]。在语法结构上它又与C语言有很多相似（例如if条件语句、switch语句、while循环、do-while循环等）[10]。

在客户端，JavaScript在传统意义上被实现为一种解释语言，但在最近[何时？]，它已经可以被即时编译（JIT）执行。随着最新的HTML5和CSS3语言标准的推行它还可用于游戏、桌面和移动应用程序的开发和在服务器端网络环境运行 如Node.js。

CSS技术： 层叠样式表（英语：Cascading Style Sheets，缩写：CSS；又称串样式列表、级联样式表、串接样式表、阶层式样式表）是一种用来为结构化文档（如HTML文档或XML应用）添加样式（字体、间距和颜色等）的计算机语言，由W3C定义和维护。CSS3现在已被大部分现代浏览器支持，而下一版的CSS4仍在开发中。

为什么要使用Vue?

Vue是一款友好的、多用途且高性能的JavaScript框架，使用vue可以创建可维护性和可测试性更强的代码库，Vue允许可以将一个网页分割成可复用的组件，每个组件都包含属于自己的HTML、CSS、JavaScript，以用来渲染网页中相应的地方，所以越来越多的前端开发者使用vue。

> vue是JavaScript封装成的框架， 是一套用于构建用户界面的渐进式JavaScript框架，能实现强大的功能。
> Vue.js的目标是通过尽可能简单的API实现响应的数据绑定和组合的视图组件。

javaScript是运行在浏览器端的脚本语言，JavaScript主要解决的是前端与用户交互的问题，包括使用交互与数据交互，JavaScript是浏览器解释执行的。

Vue (读音 /vjuː/，类似于 view) 是一套用于构建用户界面的渐进式JavaScript框架。与其他重量级框架不同的是，Vue 采用自底向上增量开发的设计。Vue 的核心库只关注视图层，并且非常容易学习，非常容易与其它库或已有项目整合。另一方面，Vue 完全有能力驱动采用单文件组件和Vue生态系统支持的库开发的复杂单页应用。

Vue.js 的目标是通过尽可能简单的 API 实现响应的数据绑定和组合的视图组件。

Vue.js 自身不是一个全能框架——它只聚焦于视图层。因此它非常容易学习，非常容易与其它库或已有项目整合。另一方面，在与相关工具和支持库一起使用时，Vue.js 也能完美地驱动复杂的单页应用

Vue学习文档

Vue.js官方文档： **https://cn.vuejs.org/v2/guide/**

Element

什么是element?

Element-UI是饿了么前端团队推出的一款基于Vue.js 2.0 的桌面端UI框架。是一套为开发者、设计师和产品经理准备的基于Vue的桌面端组件库，是网站快速成型的工具。 具备一致性（与现实生活一致，与界面中一致），反馈（控制反馈，页面反馈，可视化编程），效率（简化流程，清晰明确，帮助用户识别），可控（用户决策，结果可控）这四大设计原则。

element学习文档

官方开发文档： **https://element.eleme.cn/#/zh-CN/component/installation**

MySQL

关系型数据库我们用的是mysql，版本是MySQL8。咱们这个项目中，MySQL主要用于存储数据，提供数据新增、数据查询、数据修改、数据删除的功能。（也就是我们常说的增删改查）学长先简单介绍下MySQL，想学习更多MySQL的请百度相关MySQL教程。

什么是MySQL?

为了让大家更好的理解mysql，需要先理解 **数据库** 和 **SQL** 两个概念，你如果都懂，那你可以跳过本节。本节将从三个方面去介绍MySQL：

1. 什么是数据库?

我们每天都在不知不觉的用数据库。

- 当你想听你喜欢的歌曲，你打开你的手机中的曲目，其实你已经在用**数据库**了。
- 当你拍照并且上传到社交网站，你的照片墙就是**数据库**。
- 当你预览电子商城，你就是在使用商城的**数据库**。

数据库随时随地的存在，并且使用，简单的说，数据库就是收集数据的结构。数据涉及很多，例如一个产品属于种类，并且有自己的数据标签，这就是为什么要用关系型数据。在关系数据库，我们建模数据包括 *产品，品类，标签* 等等，所有这些都用一个表格，包含行和列，就像**Excel中的电子表格**。

一个表格与其他表格构成关系，一对一，或者一对多，因为我们要处理大量数据，所以需要定义数据库，表格等，我们更一步的将数据变成信息。

这样SQL就应运而生！

2.SQL 一种数据库语言

SQL- 是structured query language简称

SQL 是一种标准的数据库语言。ANSI/SQL 有专门的标准。

SQL 包含以下3个功能：

1. 数据创建语句，能够帮助你定义数据库和对象，例如表，视图，触发器，存储过程；
2. 数据操纵语言，能够更新数据，**查询数据**；
3. 数据控制语言，帮你管理数据权限。

那么，你明白数据库和SQL，回答一下几个问题。

3.MySQL是什么?

MySQL是数据库管理系统，能够帮助你管理关系型数据库，并且是开源的，意味着这是免费的，如果必要，你可以修改源代码。

尽管MySQL是开源软件，你需要买社区版才能得到专项服务。

MySQL 对比Oracle和 SQL server 有非常大的优势。.

- MySQL 可以在几乎所有平台上运营UNIX, Linux, Windows，小到你可以安装服务器在自己的pc中，而且，可靠，可拓展，运行速度飞快。
- 如果你开发web或者webapp，mysql 是明智的选择，因为他拥有LAMP堆栈， 包含Linux, Apache, MySQL, 和 PHP。

MySQL学习文档

MySQL教程： **https://www.runoob.com/mysql/mysql-tutorial.html**

Redis

Redis是内存数据库（或者称为缓存数据库）。在本项目中主要用于登陆和权限相关。本项目中权限是基于Spring-Security和spring-session实现的，session相关数据存储在Redis中。当然，Redis的作用远不止这些，你可以作为缓存使用，Redis强大的性能和丰富的数据结构可以帮你做很多事情。本节将简单介绍下Redis，如果想深入学习，可以自行百度，网上的教程非常丰富。

什么是Redis?

Redis 是完全开源的，遵守 BSD 协议，是一个高性能的 key-value 数据库。

Redis 与其他 key - value 缓存产品有以下三个特点：

- Redis支持数据的持久化，可以将内存中的数据保存在磁盘中，重启的时候可以再次加载进行使用。

- Redis不仅仅支持简单的key-value类型的数据，同时还提供list，set，zset，hash等数据结构的存储。
- Redis支持数据的备份，即master-slave模式的数据备份。

Redis 优势

- 性能极高 – Redis能读的速度是110000次/s,写的速度是81000次/s 。
- 丰富的数据类型 – Redis支持二进制案例的 Strings, Lists, Hashes, Sets 及 Ordered Sets 数据类型操作。
- 原子 – Redis的所有操作都是原子性的，意思就是要么成功执行要么失败完全不执行。单个操作是原子性的。多个操作也支持事务，即原子性，通过MULTI和EXEC指令包起来。
- 丰富的特性 – Redis还支持 publish/subscribe, 通知, key 过期等等特性。

Redis与其他key-value存储有什么不同?

- Redis有着更为复杂的数据结构并且提供对他们的原子性操作，这是一个不同于其他数据库的进化路径。Redis的数据类型都是基于基本数据结构的同时对程序员透明，无需进行额外的抽象。
- Redis运行在内存中但是可以持久化到磁盘，所以在对不同数据集进行高速读写时需要权衡内存，因为数据量不能大于硬件内存。在内存数据库方面的另一个优点是，相比在磁盘上相同的复杂的数据结构，在内存中操作起来非常简单，这样Redis可以做很多内部复杂性很强的事情。同时，在磁盘格式方面他们是紧凑的以追加的方式产生的，因为他们并不需要进行随机访问。

Redis学习文档

菜鸟教程： **https://www.runoob.com/redis/redis-intro.html**

官网： **https://redis.io/docs/**

maven

maven主要是用于管理项目依赖（依赖你可以理解为你项目中需要用到的jar包）的，其特点就是pom.xml文件，在pom.xml文件里你可以定义相关的依赖。这玩意其实非常简单，玩儿个几次基本上就入门了，本节会简单介绍下maven，想学习更多内容的自行百度，多动手。

什么是maven?

在了解Maven之前，我们先来看看一个Java项目需要的东西。首先，我们需要确定引入哪些依赖包。例如，如果我们需要用到 **commons logging**，我们就必须把commons logging的jar包放入classpath。如果我们还需要 **log4j**，就需要把log4j相关的jar包都放到classpath中。这些就是依赖包的管理。

其次，我们要确定项目的目录结构。例如，`src` 目录存放Java源码，`resources` 目录存放配置文件，`bin` 目录存放编译生成的 `.class` 文件。

此外，我们还需要配置环境，例如JDK的版本，编译打包的流程，当前代码的版本号。

最后，除了使用Eclipse这样的IDE进行编译外，我们还必须能通过命令行工具进行编译，才能够让项目在一个独立的服务器上编译、测试、部署。

这些工作难度不大，但是非常琐碎且耗时。如果每一个项目都自己搞一套配置，肯定会一团糟。我们需要的是一个标准化的Java项目管理和构建工具。

Maven就是是专门为Java项目打造的管理和构建工具，它的主要功能有：

- 提供了一套标准化的项目结构；
- 提供了一套标准化的构建流程（编译，测试，打包，发布……）；

- 提供了一套依赖管理机制。

Maven项目结构

一个使用Maven管理的普通的Java项目，它的目录结构默认如下：

```
a-maven-project
├── pom.xml
├── src
│   ├── main
│   │   ├── java
│   │   └── resources
│   └── test
│       ├── java
│       └── resources
└── target
```

项目的根目录`a-maven-project`是项目名，它有一个项目描述文件`pom.xml`，存放Java源码的目录是`src/main/java`，存放资源文件的目录是`src/main/resources`，存放测试源码的目录是`src/test/java`，存放测试资源的目录是`src/test/resources`，最后，所有编译、打包生成的文件都放在`target`目录里。这些就是一个Maven项目的标准目录结构。

所有的目录结构都是约定好的标准结构，我们千万不要随意修改目录结构。使用标准结构不需要做任何配置，Maven就可以正常使用。

我们再来看最关键的一个项目描述文件`pom.xml`，它的内容长得像下面：

```
<project ...>
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.itranswarp.learnjava</groupId>
  <artifactId>hello</artifactId>
  <version>1.0</version>
  <packaging>jar</packaging>
  <properties>
        ...
  </properties>
  <dependencies>
        <dependency>
            <groupId>commons-logging</groupId>
            <artifactId>commons-logging</artifactId>
            <version>1.2</version>
        </dependency>
  </dependencies>
</project>
```

其中，`groupId`类似于Java的包名，通常是公司或组织名称，`artifactId`类似于Java的类名，通常是项目名称，再加上`version`，一个Maven工程就是由`groupId`，`artifactId`和`version`作为唯一标识。我们在引用其他第三方库的时候，也是通过这3个变量确定。例如，依赖`commons-logging`：

```
<dependency>
    <groupId>commons-logging</groupId>
    <artifactId>commons-logging</artifactId>
    <version>1.2</version>
</dependency>
```

使用`<dependency>`声明一个依赖后，Maven就会自动下载这个依赖包并把它放到classpath中。

安装Maven

要安装Maven，可以从Maven官网下载最新的Maven 3.6.x，然后在本地解压，设置几个环境变量：

```
M2_HOME=/path/to/maven-3.6.x
PATH=$PATH:$M2_HOME/bin
```

Windows可以把`%M2_HOME%\bin`添加到系统Path变量中。

然后，打开命令行窗口，输入`mvn -version`，应该看到Maven的版本信息：

```
┌──────────────────────────────────────────────────────────────────────────────┐
│Command Prompt                                       - ▫ x  │
├──────────────────────────────────────────────────────────────────────────────┤
│Microsoft Windows [Version 10.0.0]                          │
│(c) 2015 Microsoft Corporation. All rights reserved.        │
│                                                            │
│C:\> mvn -version                                           │
│Apache Maven 3.6.0 (97c98ec64a1fdfee7767ce5ffb20918...)     │
│Maven home: C:\Users\liaoxuefeng\maven                      │
│Java version: ...                                           │
│...                                                         │
│C:\> _                                                      │
│                                                            │
│                                                            │
└──────────────────────────────────────────────────────────────────────────────┘
```

如果提示命令未找到，说明系统PATH路径有误，需要修复后再运行。

小结

Maven是一个Java项目的管理和构建工具：

- Maven使用`pom.xml`定义项目内容，并使用预设的目录结构；
- 在Maven中声明一个依赖项可以自动下载并导入classpath；
- Maven使用`groupId`，`artifactId`和`version`唯一定位一个依赖。

maven学习文档

入门教程： https://www.runoob.com/maven/maven-tutorial.html

mybatis

大学里常说的SSM框架，指的是spring、springMVC、mybatis，这里面的M就是mybatis，主要用于操作数据库的这么一个开发框架。目前SSM里面的spring和SpringMVC基本被SpringBoot锁替代，换句话说，SpringBoot就是在spring的基础上升级的，本质上是一个公司开发的东西。本节我们简单介绍下mybatis，想学习更多资料的同学自行百度相关教程。

什么是mybatis?

MyBatis是一个Java持久化框架，它通过XML描述符或注解把对象与存储过程或SQL语句关联起来，映射成数据库内对应的纪录。

MyBatis是在Apache许可证 2.0下分发的自由软件，是iBATIS 3.0的分支版本，其维护团队也包含iBATIS的初创成员。

与其他对象关系映射框架不同，MyBatis没有将Java对象与数据库表关联起来，而是将Java方法与SQL语句关联。MyBatis允许用户充分利用数据库的各种功能，例如存储过程、视图、各种复杂的查询以及某数据库的专有特性。如果要对遗留数据库、不规范的数据库进行操作，或者要完全控制SQL的执行，MyBatis是一个不错的选择。

与JDBC相比，MyBatis简化了相关代码：SQL语句在一行代码中就能执行。MyBatis提供了一个映射引擎，声明式的把SQL语句执行结果与对象树映射起来。通过使用一种内建的类XML表达式语言，或者使用Apache Velocity集成的插件，SQL语句可以被动态的生成。

MyBatis与Spring Framework和Google Guice集成，这使开发者免于依赖性问题。

MyBatis支持声明式数据缓存（declarative data caching）。当一条SQL语句被标记为“可缓存”后，首次执行它时从数据库获取的所有数据会被存储在一段高速缓存中，今后执行这条语句时就会从高速缓存中读取结果，而不是再次命中数据库。MyBatis提供了基于 Java HashMap 的默认缓存实现，以及用于与OSCache、Ehcache、Hazelcast和Memcached连接的默认连接器。MyBatis还提供API供其他缓存实现使用。

mybatis学习文档

mybatis官方文档： https://mybatis.org/mybatis-3/zh/index.html

http

http是一种网络协议，本项目是前后端分离的项目，那前端项目和后端项目是怎么进行数据传输的呢？其实就是基于http协议进行数据传输的。举个例子，你准备登陆系统，此时你需要填写用户名、密码、输入验证码信息，这时候前端需要把这些信息发送给后端服务，后端进行用户名和密码的校验。就是这么一个简单的登陆流程，那数据怎么传输到后端呢？其实就是http协议进行网络数据的传输。http协议用起来也比较简单，但是网络相关的知识还是比较庞杂的，这个大家感兴趣的话自行百度相关教程。

什么是http?

协议概述

HTTP是一个客户端（用户）和服务端（网站）之间请求和应答的标准，通常使用 **TCP协议** 。通过使用 **网页浏览器** 、 **网络爬虫** 或者其它的工具，客户端发起一个HTTP请求到服务器上指定端口（默认 **端口** 为80）。我们称这个客户端为用户代理程序（user agent）。应答的服务器上存储着一些资源，比如HTML文件和图像。我们称这个应答服务器为源服务器（origin server）。在用户代理和源服务器中间可能存在多个“中间层”，比如 **代理服务器** 、 **网关** 或者 **隧道** （tunnel）。

尽管 **TCP/IP** 协议是互联网上最流行的应用，但是在HTTP协议中并没有规定它必须使用或它支持的层。事实上HTTP可以在任何互联网协议或其他网络上实现。HTTP假定其下层协议提供可靠的传输。因此，任何能够提供这种保证的协议都可以被其使用，所以其在TCP/IP协议族使用TCP作为其传输层。

通常，由HTTP客户端发起一个请求，创建一个到服务器指定端口（默认是80端口）的TCP连接。HTTP服务器则在那个端口监听客户端的请求。一旦收到请求，服务器会向客户端返回一个状态，比如"HTTP/1.1 200 OK"，以及返回的内容，如请求的文件、错误消息、或者其它信息。

请求方法

HTTP/1.1协议中共定义了八种方法（也叫“动作”）来以不同方式操作指定的资源，下面介绍常用的几种方法：

- GET

  向指定的资源发出“显示”请求。使用GET方法应该只用在读取资料，而不应当被用于产生“ **副作用** ”的操作中，例如在 **网络应用程序** 中。其中一个原因是GET可能会被 **网络爬虫** 等随意访问。参见 **安全方法** 。浏览器直接发出的GET只能由一个url触发。GET上要在url之外带一些参数就只能依靠url上附带querystring。

- HEAD

  与GET方法一样，都是向服务器发出指定资源的请求。只不过服务器将不传回资源的本文部分。它的好处在于，使用这个方法可以在不必传输全部内容的情况下，就可以获取其中“关于该资源的信息”（元信息或称元数据）。

- POST

  向指定资源提交数据，请求服务器进行处理（例如提交表单或者上传文件）。数据被包含在请求本文中。这个请求可能会创建新的资源或修改现有资源，或二者皆有。每次提交，表单的数据被浏览器用编码到HTTP请求的body里。浏览器发出的POST请求的body主要有两种格式，一种是application/x-www-form-urlencoded用来传输简单的数据，大概就是"key1=value1&key2=value2"这样的格式。另外一种是传文件，会采用multipart/form-data格式。采用后者是因为application/x-www-form-urlencoded的编码方式对于文件这种二进制的数据非常低效。

- PUT

  向指定资源位置上传其最新内容。

- DELETE

  请求服务器删除Request-URI所标识的资源。

http学习文档

入门教程： **https://www.runoob.com/http/http-tutorial.html**

easyExcel

本项目中有大量的Excel导出功能，目前是基于阿里巴巴开源的easyExcel来实现的，下面我们简单介绍下easyExcel

什么是easyExcel?

EasyExcel是一个基于Java的简单、省内存的读写Excel的开源项目。在尽可能节约内存的情况下支持读写百M的Excel。 github地址: https://github.com/alibaba/easyexcel

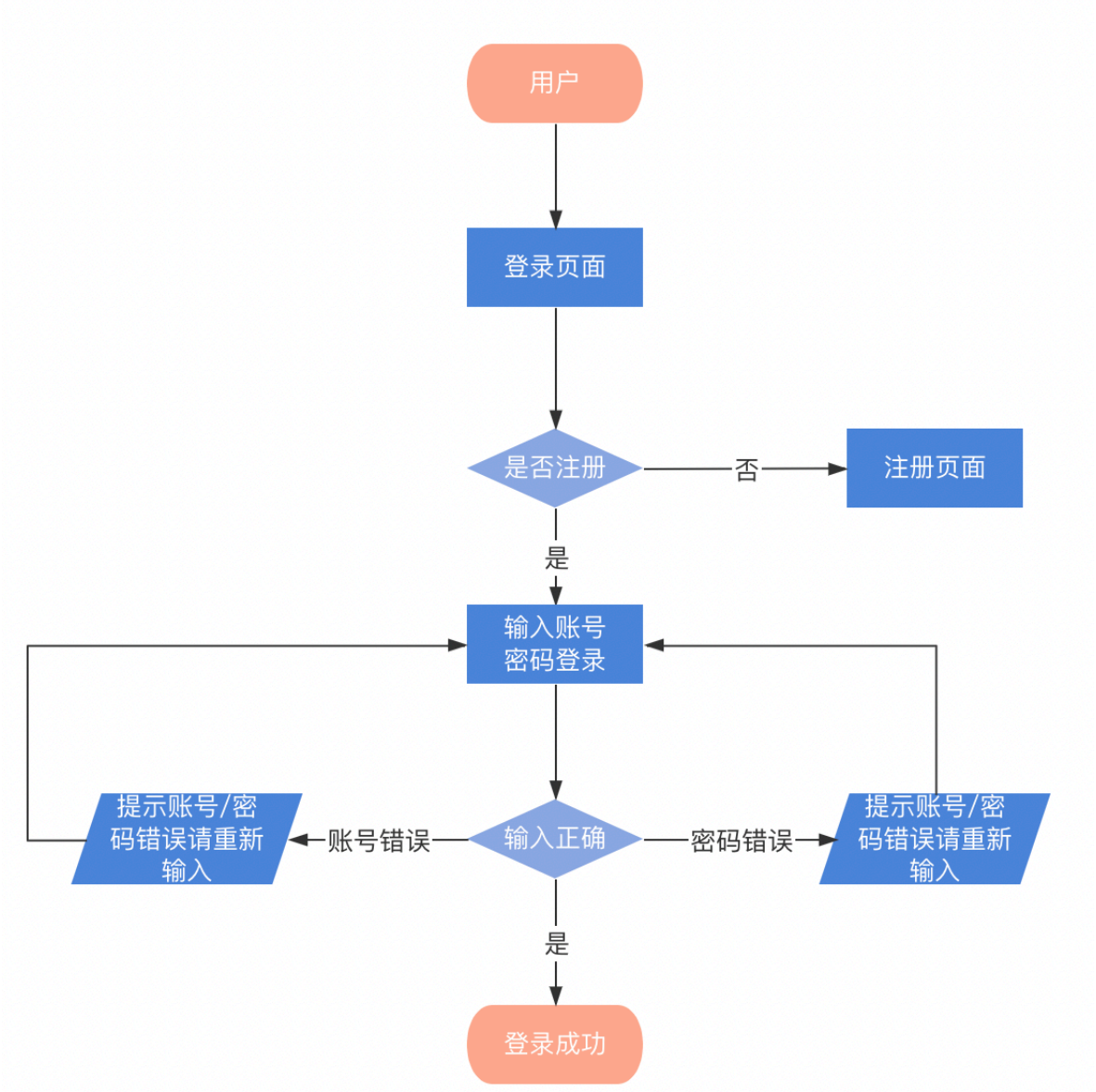
easyExcel学习文档

官方文档： https://www.yuque.com/easyexcel/doc/easyexcel

系统设计

【用户&角色】详细设计

用户登陆流程



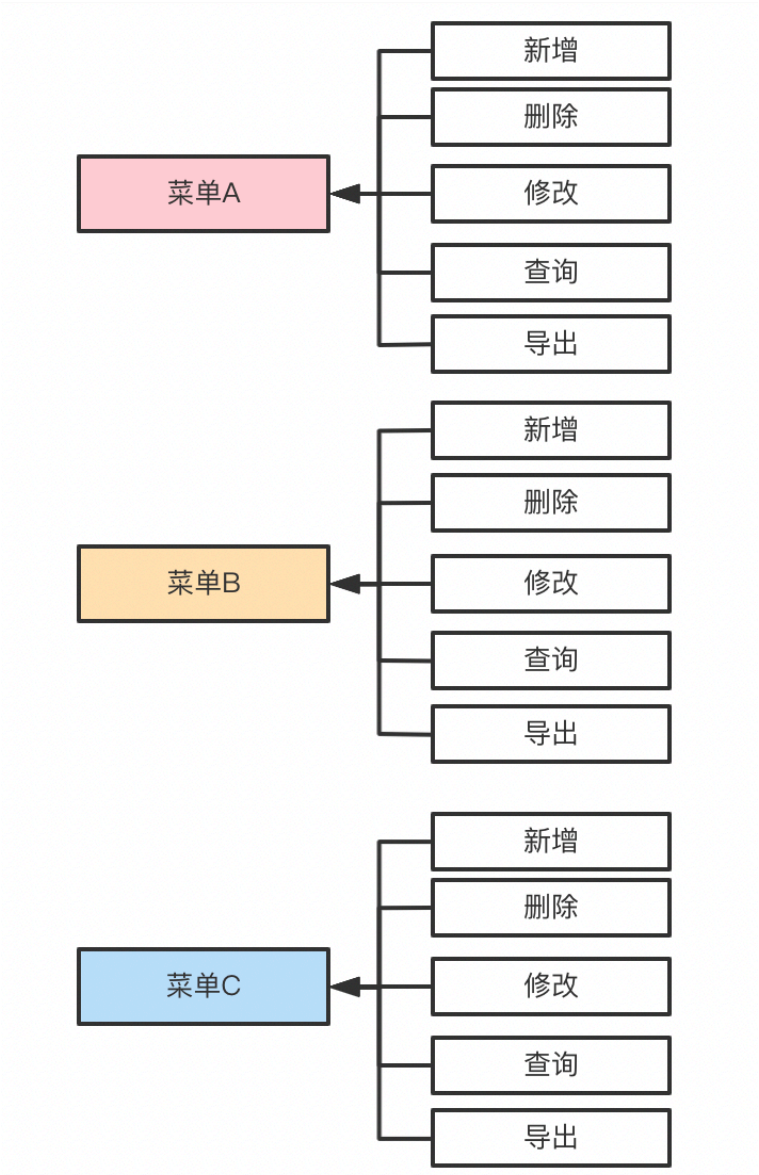
角色和权限的关系

权限：用大白话说就是对资源的访问权限，那资源是什么呢？资源就是菜单，比如【用户管理菜单】那菜单还有很多按钮，比如【新增】、【删除】、【修改】、【查询】、【导出】等按钮，这些都是资源。所以总结下来就是对各种菜单、各种按钮的访问/操作权限。

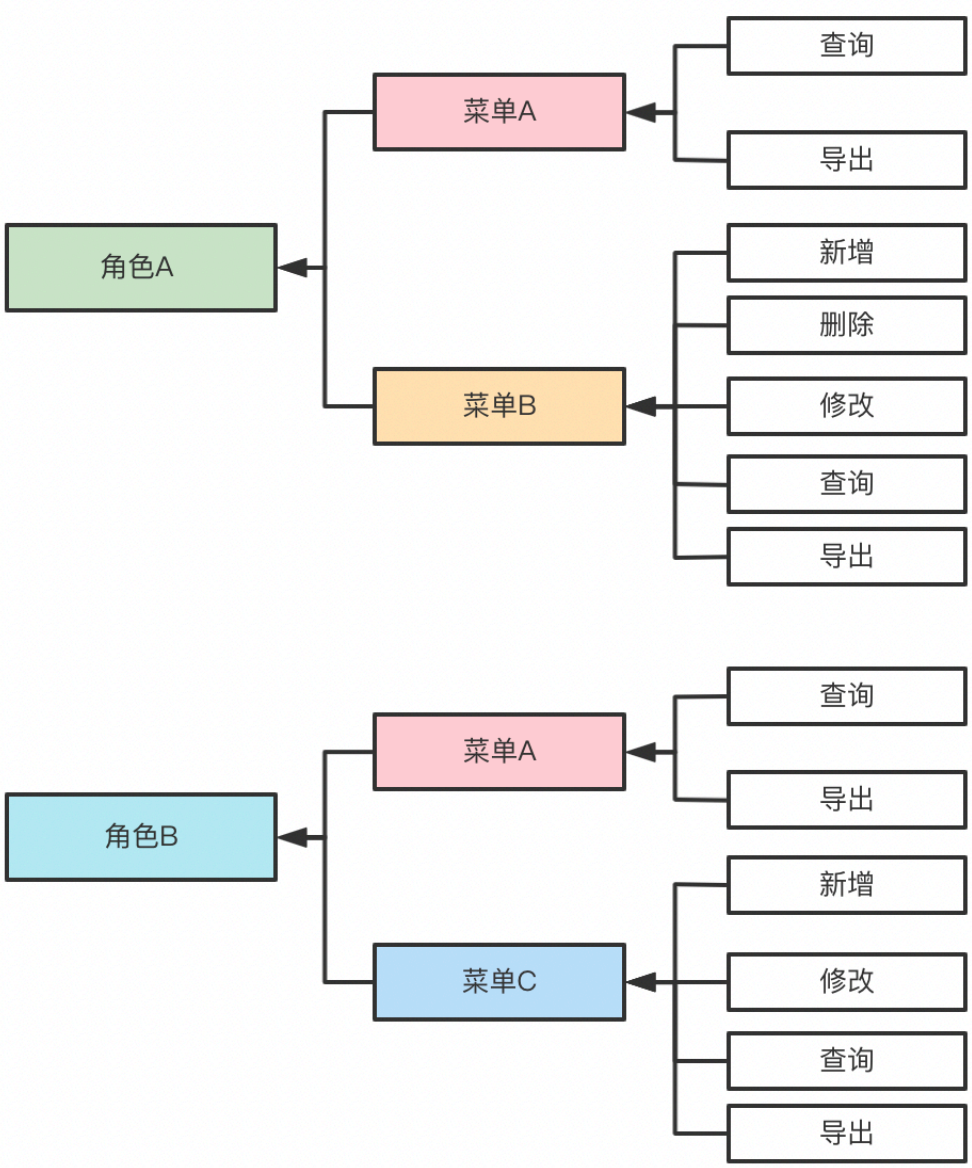
角色：那角色就是定义了一组的权限集合，比如把【用户管理菜单】的【新增】、【删除】和【日志管理菜单】的【查询】权限放在一块儿，这就是角色，指的是一组权限的集合。

下面画图给大家再详细聊一聊：

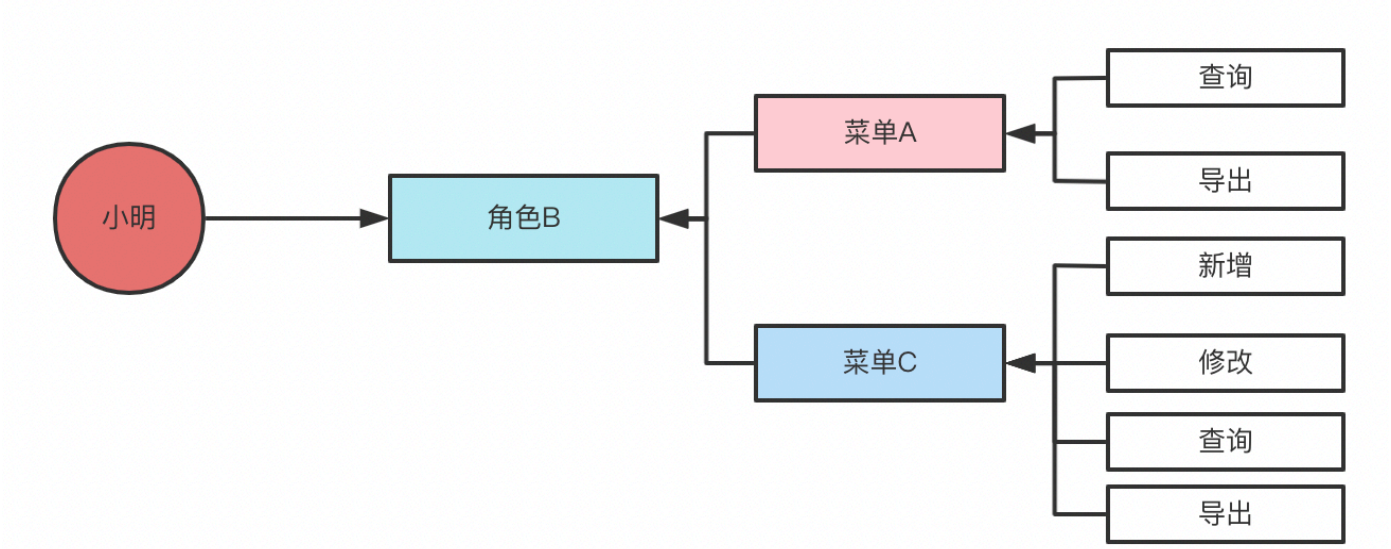
1、下面这张图画了三个模块，【菜单A】、【菜单B】、【菜单C】，每个菜单都有【新增】、【删除】、【修改】、【查询】、【导出】等按钮，这就是资源的模型



2、下面这幅图定义了两组权限集合，也就是前面说的角色。【角色A】、【角色B】，这里两个角色拥有的菜单和权限集合是不一样的，这个大家看清楚，所以这里的角色对应的都是系统真实的用户群体，因为每个用户群体可以操作的菜单和按钮是不一样的，也就有了不同的角色。



3、下面这幅图是用户和角色的关系，我把拥有一组特殊权限集合的角色分配给某个用户，那么这个用户就拥有了这个角色对应的所有权限，你可以理解角色就是权限的便捷操作，快速赋值权限。比如用户【小明】分配了【角色B】，那【小明】登陆系统的时候就只会展示【角色B】拥有的【菜单】和对应的【按钮】，系统其他的菜单和按钮就看不到了。大家结合图好好理解下，学长通过三张图，按理说是把权限、角色、用户间的关系是讲得非常透彻了，大家好好理解下。



用户密码

数据库里面存的密码都是加密的，比如小明的密码是123456，那数据库里面存的肯定不是123456，不能直接存明文，这样万一数据库数据泄露了，那用户的密码岂不是全部泄露了，是不是这个理儿？

所以用户密码必然是加密的，基于md5加密，比如123456加密后可能就是一串很长的字符串，这是为了密码安全。

【用户角色管理】详细设计

表结构设计

```
CREATE TABLE `role` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键',
  `name` varchar(256) DEFAULT NULL COMMENT '角色名称',
  `description` varchar(1024) DEFAULT NULL COMMENT '备注',
  `authorities` varchar(2048) DEFAULT NULL COMMENT '权限列表',
  `create_time` bigint NOT NULL DEFAULT '0' COMMENT '创建时间',
  `update_time` bigint DEFAULT '0' COMMENT '更新时间',
  `type` int DEFAULT '1' COMMENT '角色类型;1:自定义角色 2:系统默认角色',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8mb3 COMMENT='用户角色表'
```

后端代码设计

后端代码都放在api目录下，本节将详细介绍【用户角色管理】的后端代码设计

DO层（Domain Object）

数据领域对象，对象属性跟数据库字段一一对应。

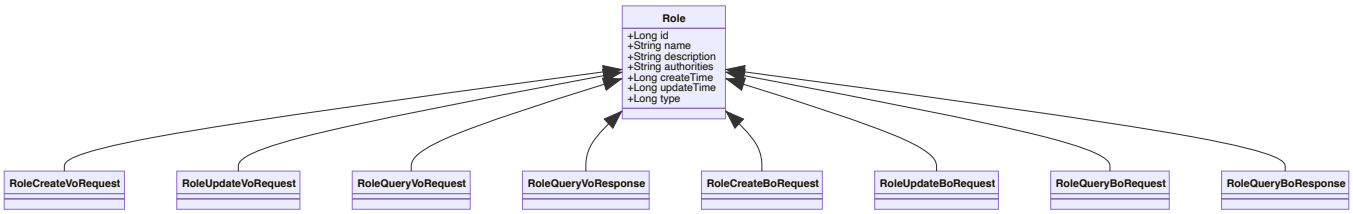
数据库表名称：role

类型：Java类

类名：RoleDo.java

代码位置：domain/src/main/java/com/senior/domain/model/RoleDo.java

类图如下：



Mapper层（mapper.xml）

Mapper.xml主要是写一些SQL语句，最常见的就是增删改查，这些SQL语句都是写在Mapper.xml里面的

数据库表名称：role

类型：xml文件

类名：RoleMapper.xml

代码位置：service/base/src/main/resources/mybatis/mappers/RoleMapper.xml

DAO层（Data Access Object）

持久层操作对象，结合RoleDo对象，对role表的数据进行增删改查操作。

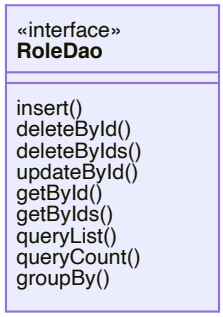
数据库表名称：role

类型：Java接口

类名：RoleDao.java

代码位置：service/base/src/main/java/com/senior/dao/RoleDao.java

类图如下：



Service层（业务逻辑层）

业务逻辑层，负责对数据的处理。如果没有逻辑处理任务，此层只做单纯的数据传递作用，而后又到了DAO层。

数据库表名称：role

类型：Java接口

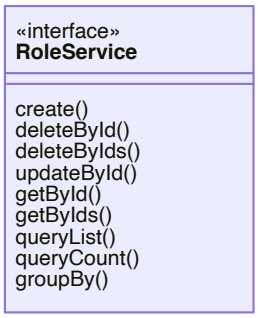
类名：RoleService.java

代码位置：service/business/src/main/java/com/senior/service/RoleService.java

接口实现类：RoleServiceImpl.java

接口实现类代码位置：service/business/src/main/java/com/senior/service/impl/RoleServiceImpl.java

类图如下：



Controller层（控制层）

控制层，主要是提供一些接口（增删改查接口）。本项目是前后端分离的项目，前端跟后端是基于http协议实现数据传输的。接口的代码其实就放在Controller这一层，这一层主要负责提供http接口，然后就是组装前端传输的数据，再将组装的数据调用Service层。

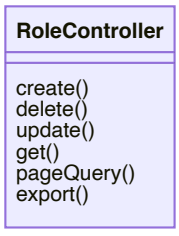
数据库表名称：role

类型：Java类

类名：RoleController.java

代码位置：console/api/src/main/java/com/senior/console/api/controller/RoleController.java

类图如下：



前端代码设计

前端是基于Vue开发的，框架的话是基于饿了么开源的element框架。代码都放在console/web目录下，本节将详细介绍【用户角色管理】的前端代码设计

列表页面

代码路径：console/web/src/page/role/Role.vue

代码阅读技巧：方法都放在methods里面，这里面还有很多关键字，比如created、computed、watch、methods、data等，这些都是Vue的语法，大家如果看不懂，需要去学习下Vue的基础语法，还是比较简单的。

主要功能：列表页的功能主要是展示【用户角色】的数据，包括页面上的一些按钮，比如新增、修改、详情、删除、导出等

详情&新增页面

代码路径：console/web/src/page/role/RoleDialog.vue

代码阅读技巧：跟列表页一样。

主要功能：提供【用户角色】新增和查看【用户角色】详情的功能

API定义

代码路径：console/web/src/api/Role.js

主要功能：本项目是前后端分离的项目，前面我们说过好多次了，前后端直接的数据交互是基于http协议来实现的。那么API这一块的代码就是定义的后端接口的地址，用户在页面上通过填写各种数据，或者点击某个按钮时，此时将组装好的数据通过API这层封装好的http接口，就可以把数据传输给后端的Controller层了，里面的接口大致上就是【用户角色管理】的增删改查操作。

流程图

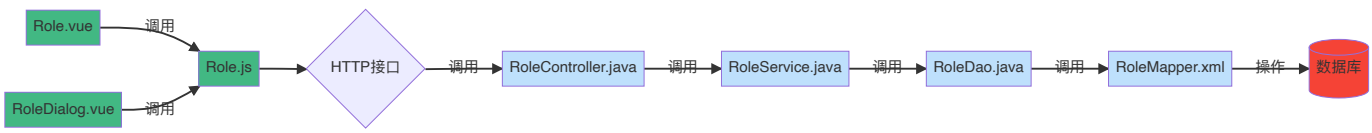
代码调用链

绿色部分是前端代码文件

蓝色部分是后端代码文件

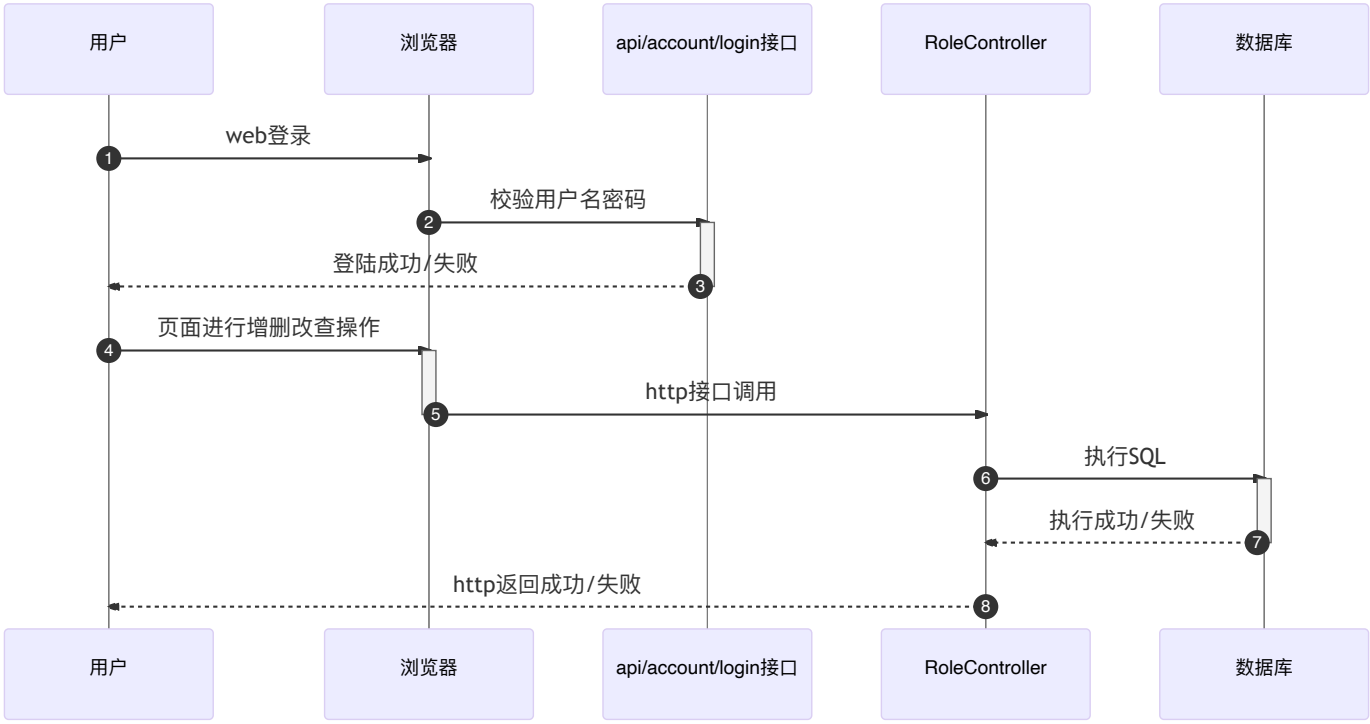
前端跟后端之间的调用是通过http接口来实现的，这里的http指的是http协议

红色部分是数据库，最终的操作都是对数据库的增删改查操作



时序图

详细介绍用户从【登录系统】到【用户角色增删改查操作】这个过程中系统运行时序图，每一步都标了顺序。



增删改查功能设计

前面跟大家详细介绍了后端设计和前端设计，以及前后端之间是通过http协议进行交互的逻辑也跟大家讲解过，相信大家看完学长前面的讲解后，对这个模块已经有了一个大致的认识，那本节就是打铁趁热的一个模块，通过画图的方式，把前后端整个的调用流程给大家串起来。

按钮在Role.vue文件中，可以通过中文关键字去搜索，找到具体的位置。

具体调用路径如下：（**部分方法名称可能存在细微的差异，这个大家具体看代码，以代码中的方法名为准**）

讲解调用路径时学长会按照**文件名**+'^'+**方法名**的方式讲解，这样大家就可以很快的在项目中找到相关的代码了

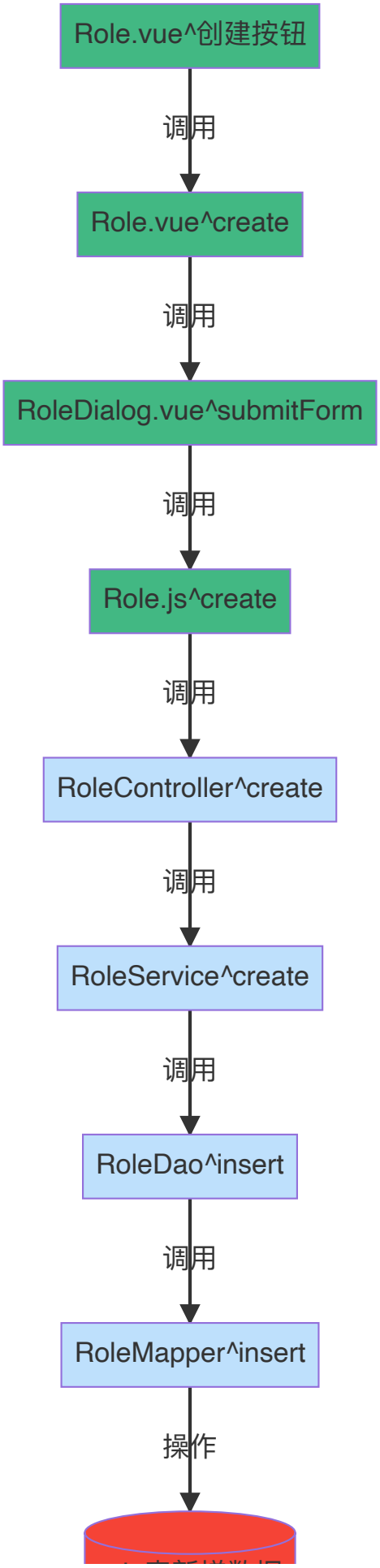
为了让大家更好地区分前后端代码的职责，学长通过不同的颜色来区分前后端代码，让大家一目了然！

绿色部分是前端代码文件

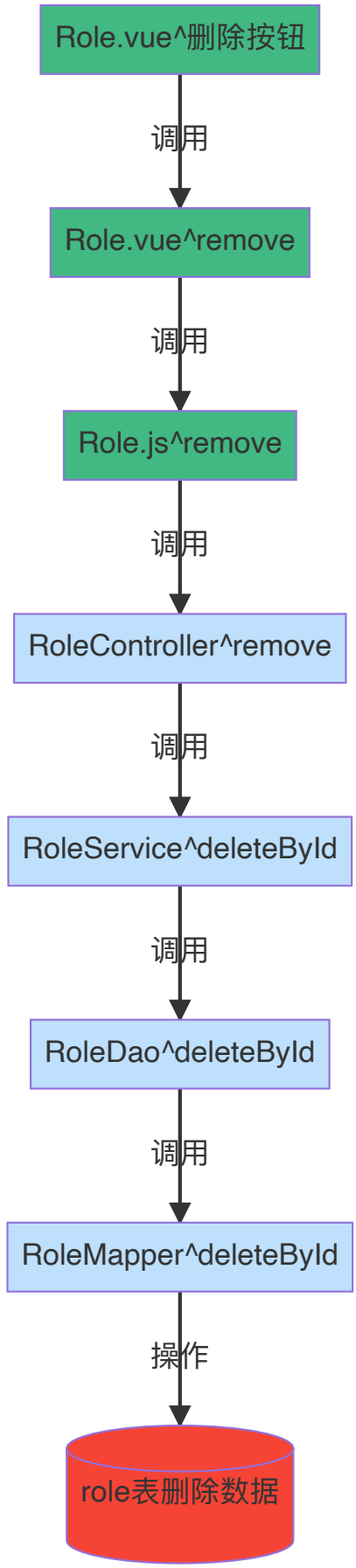
蓝色部分是后端代码文件

红色部分是数据库对应的表名

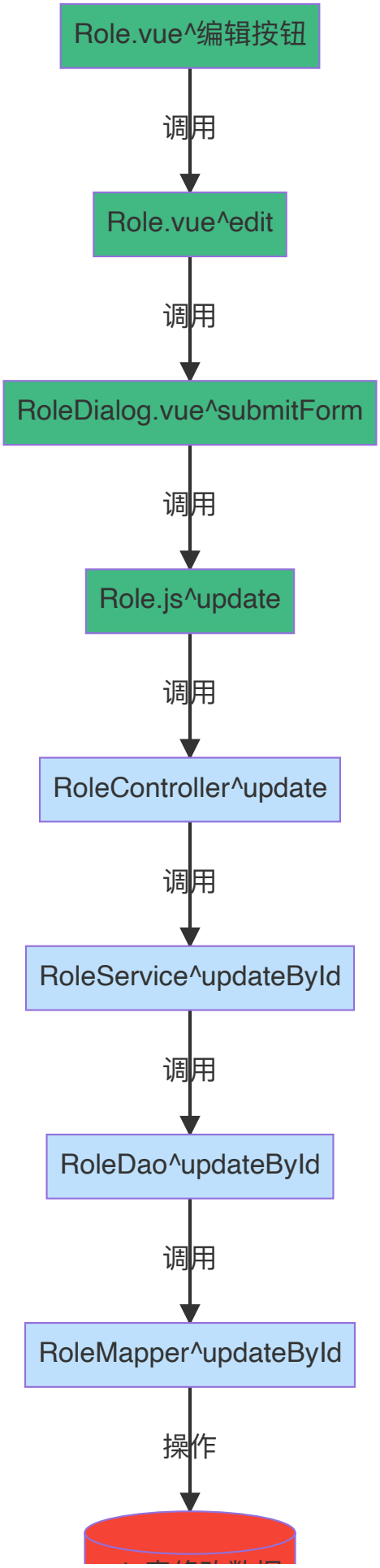
新增流程


Role.vue^创建按钮
调用
Role.vue^create
调用
RoleDialog.vue^submitForm
调用
Role.js^create
调用
RoleController^create
调用
RoleService^create
调用
RoleDao^insert
调用
RoleMapper^insert
操作

删除流程

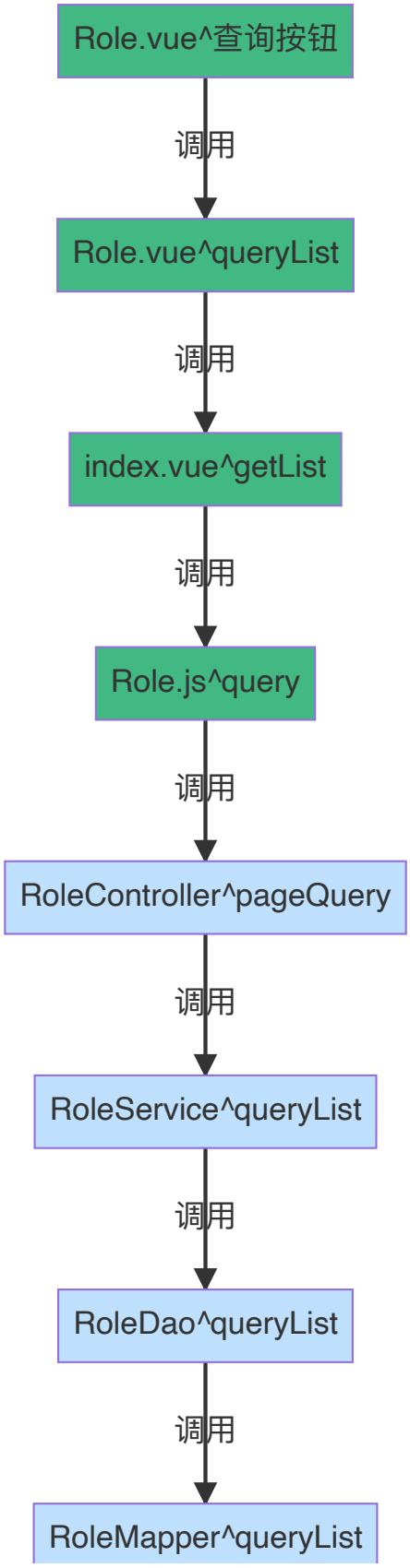

Role.vue^删除按钮
调用
Role.vue^remove
调用
Role.js^remove
调用
RoleController^remove
调用
RoleService^deleteById
调用
RoleDao^deleteById
调用
RoleMapper^deleteById
操作
role表删除数据

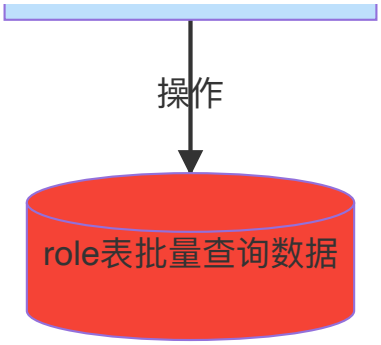
修改流程


Role.vue^编辑按钮
调用
Role.vue^edit
调用
RoleDialog.vue^submitForm
调用
Role.js^update
调用
RoleController^update
调用
RoleService^updateById
调用
RoleDao^updateById
调用
RoleMapper^updateById
操作


role表修改数据

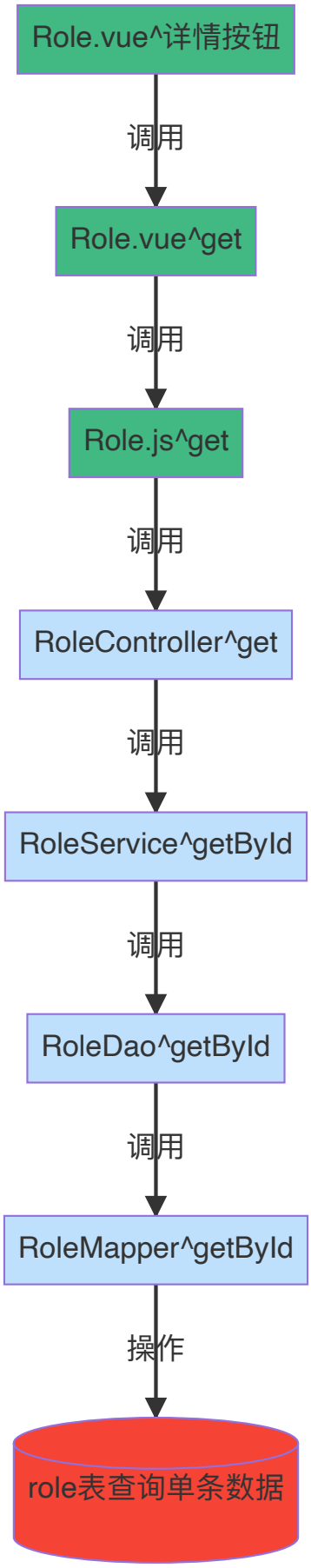
批量查询流程


Role.vue^查询按钮
调用
Role.vue^queryList
调用
index.vue^getList
调用
Role.js^query
调用
RoleController^pageQuery
调用
RoleService^queryList
调用
RoleDao^queryList
调用
RoleMapper^queryList

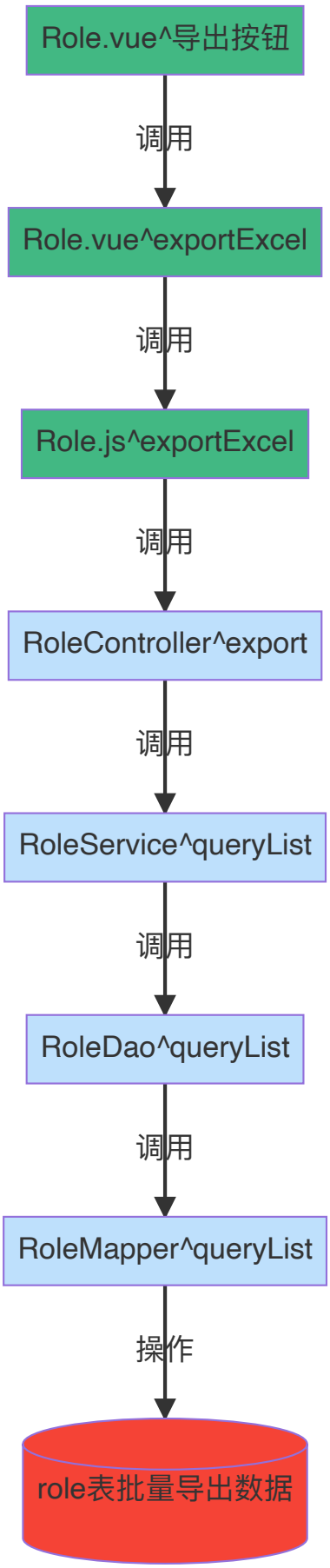


单条查询流程

这个功能在页面不一定有，但是学长先把流程给大家画出来。


Role.vue^详情按钮
调用
Role.vue^get
调用
Role.js^get
调用
RoleController^get
调用
RoleService^getById
调用
RoleDao^getById
调用
RoleMapper^getById
操作
role表查询单条数据

导出流程


Role.vue^导出按钮
调用
Role.vue^exportExcel
调用
Role.js^exportExcel
调用
RoleController^export
调用
RoleService^queryList
调用
RoleDao^queryList
调用
RoleMapper^queryList
操作
role表批量导出数据

【账号管理】详细设计

表结构设计

```
CREATE TABLE `account` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键',
  `passport` varchar(16) NOT NULL COMMENT '账号',
  `name` varchar(6) NOT NULL COMMENT '名称',
  `password` varchar(64) NOT NULL COMMENT '密码',
  `status` int NOT NULL DEFAULT '0' COMMENT '用户状态',
  `role_id` bigint NOT NULL COMMENT '角色',
  `email` varchar(32) NOT NULL COMMENT '邮箱',
  `phone` varchar(16) NOT NULL COMMENT '手机号',
  `address` varchar(64) DEFAULT NULL COMMENT '地址',
  `sex` int NOT NULL DEFAULT '0' COMMENT '性别 0：女；1：男',
  `description` varchar(128) DEFAULT NULL COMMENT '备注',
  `last_login_time` bigint DEFAULT NULL COMMENT '最近登录时间',
  `create_time` bigint NOT NULL DEFAULT '0' COMMENT '创建时间',
  `update_time` bigint DEFAULT NULL COMMENT '更新时间',
  `department_id` bigint DEFAULT NULL COMMENT '部门',
  PRIMARY KEY (`id`),
  UNIQUE KEY `passport_UNIQUE` (`passport`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8mb3 COMMENT='账号'
```

后端代码设计

后端代码都放在api目录下，本节将详细介绍【账号管理】的后端代码设计

DO层（Domain Object）

数据领域对象，对象属性跟数据库字段一一对应。

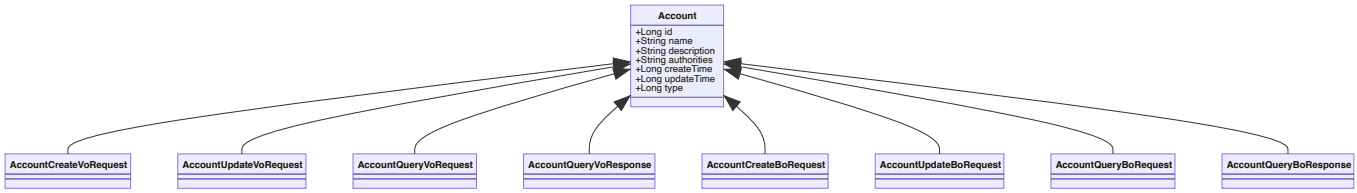
数据库表名称：account

类型：Java类

类名：AccountDo.java

代码位置：domain/src/main/java/com/senior/domain/model/AccountDo.java

类图如下：



Mapper层（mapper.xml）

Mapper.xml主要是写一些SQL语句，最常见的就是增删改查，这些SQL语句都是写在Mapper.xml里面的

数据库表名称：account

类型：xml文件

类名：AccountMapper.xml

代码位置：service/base/src/main/resources/mybatis/mappers/AccountMapper.xml

DAO层（Data Access Object）

持久层操作对象，结合AccountDo对象，对account表的数据进行增删改查操作。

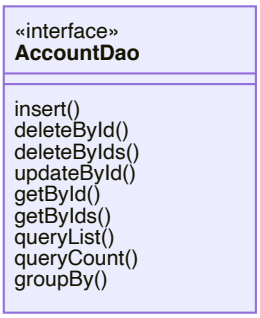
数据库表名称：account

类型：Java接口

类名：AccountDao.java

代码位置：service/base/src/main/java/com/senior/dao/AccountDao.java

类图如下：



Service层（业务逻辑层）

业务逻辑层，负责对数据的处理。如果没有逻辑处理任务，此层只做单纯的数据传递作用，而后又到了DAO层。

数据库表名称：account

类型：Java接口

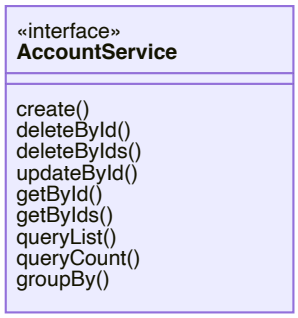
类名：AccountService.java

代码位置：service/business/src/main/java/com/senior/service/AccountService.java

接口实现类：AccountServiceImpl.java

接口实现类代码位置：service/business/src/main/java/com/senior/service/impl/AccountServiceImpl.java

类图如下：



Controller层（控制层）

控制层，主要是提供一些接口（增删改查接口）。本项目是前后端分离的项目，前端跟后端是基于http协议实现数据传输的。接口的代码其实就放在Controller这一层，这一层主要负责提供http接口，然后就是组装前端传输的数据，再将组装的数据调用Service层。

数据库表名称：account

类型：Java类

类名：AccountController.java

代码位置：console/api/src/main/java/com/senior/console/api/controller/AccountController.java

类图如下：

前端代码设计

前端是基于Vue开发的，框架的话是基于饿了么开源的element框架。代码都放在console/web目录下，本节将详细介绍【账号管理】的前端代码设计

列表页面

代码路径：console/web/src/page/account/Account.vue

代码阅读技巧：方法都放在methods里面，这里面还有很多关键字，比如created、computed、watch、methods、data等，这些都是Vue的语法，大家如果看不懂，需要去学习下Vue的基础语法，还是比较简单的。

主要功能：列表页的功能主要是展示【账号】的数据，包括页面上的一些按钮，比如新增、修改、详情、删除、导出等

详情&新增页面

代码路径：console/web/src/page/account/AccountDialog.vue

代码阅读技巧：跟列表页一样。

主要功能：提供【账号】新增和查看【账号】详情的功能

API定义

代码路径：console/web/src/api/Account.js

主要功能：本项目是前后端分离的项目，前面我们说过好多次了，前后端直接的数据交互是基于http协议来实现的。那么API这一块的代码就是定义的后端接口的地址，用户在页面上通过填写各种数据，或者点击某个按钮时，此时将组装好的数据通过API这层封装好的http接口，就可以把数据传输给后端的Controller层了，里面的接口大致上就是【账号管理】的增删改查操作。

流程图

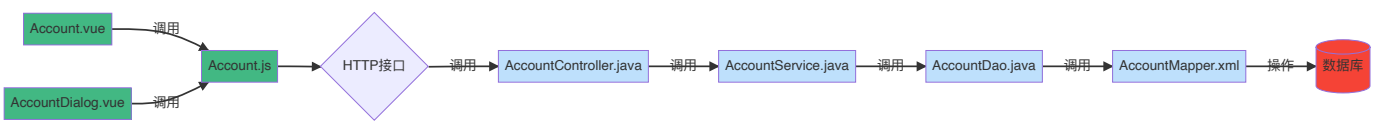
代码调用链

绿色部分是前端代码文件

蓝色部分是后端代码文件

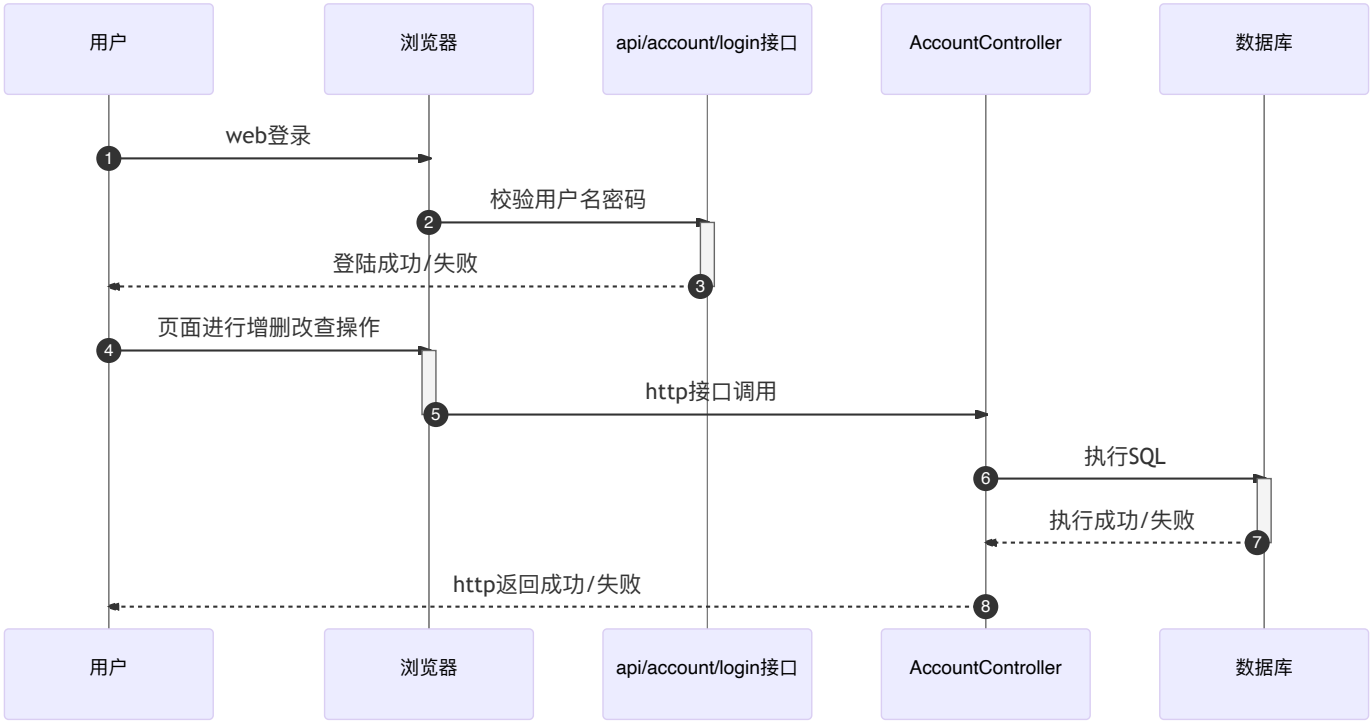
前端跟后端之间的调用是通过http接口来实现的，这里的http指的是http协议

红色部分是数据库，最终的操作都是对数据库的增删改查操作



时序图

详细介绍用户从【登录系统】到【账号增删改查操作】这个过程中系统运行时序图，每一步都标了顺序。



增删改查功能设计

前面跟大家详细介绍了后端设计和前端设计，以及前后端之间是通过http协议进行交互的逻辑也跟大家讲解过，相信大家看完学长前面的讲解后，对这个模块已经有了一个大致的认识，那本节就是打铁趁热的一个模块，通过画图的方式，把前后端整个的调用流程给大家串起来。

按钮在Account.vue文件中，可以通过中文关键字去搜索，找到具体的位置。

具体调用路径如下：（**部分方法名称可能存在细微的差异，这个大家具体看代码，以代码中的方法名为准**）

讲解调用路径时学长会按照**文件名**+'^'+**方法名**的方式讲解，这样大家就可以很快的在项目中找到相关的代码了

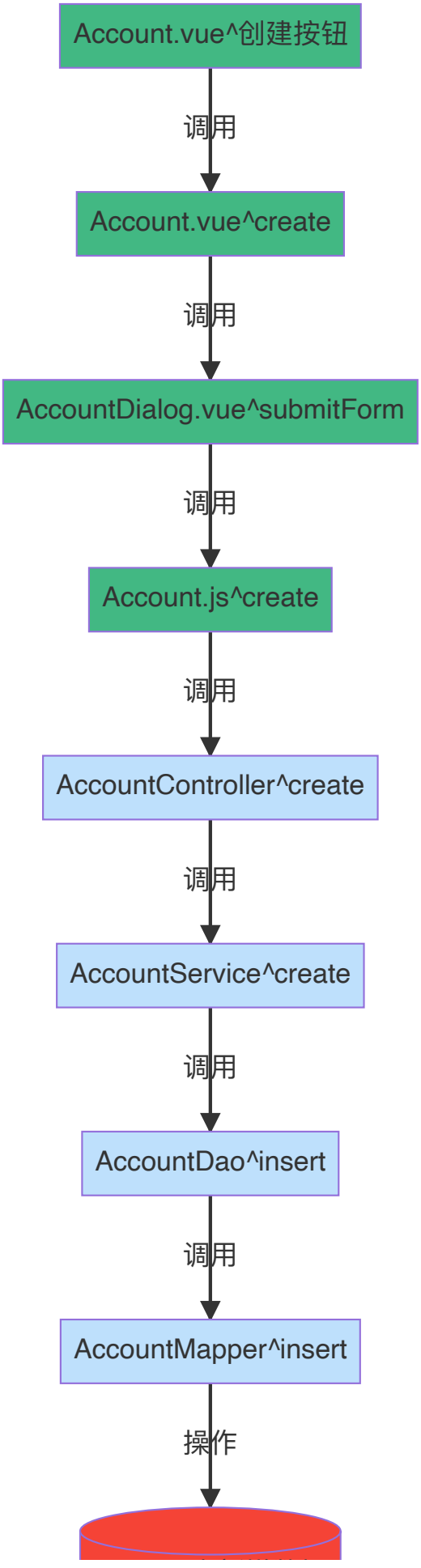
为了让大家更好地区分前后端代码的职责，学长通过不同的颜色来区分前后端代码，让大家一目了然！

绿色部分是前端代码文件

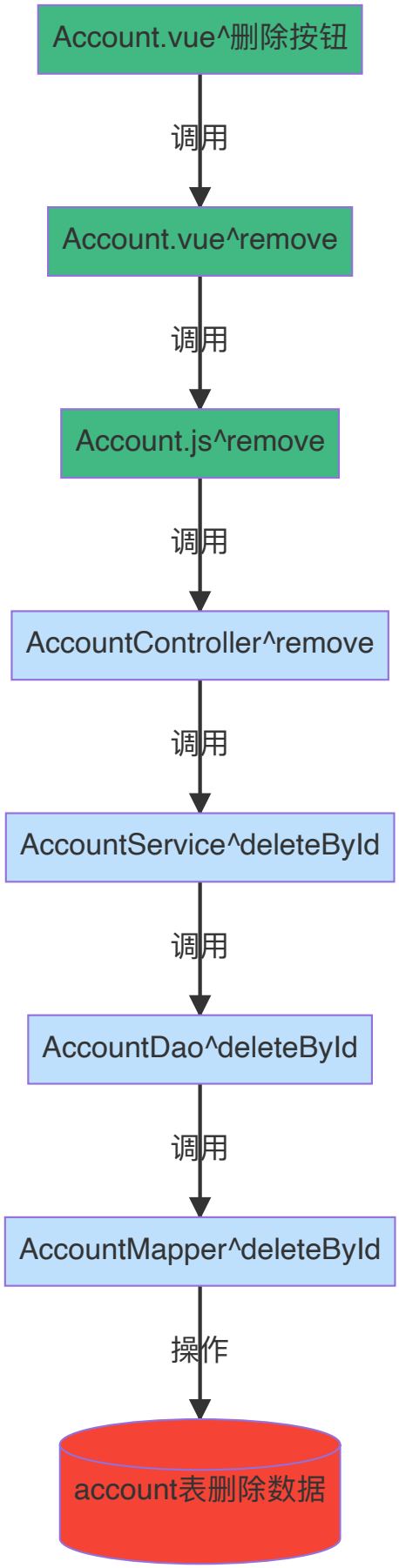
蓝色部分是后端代码文件

红色部分是数据库对应的表名

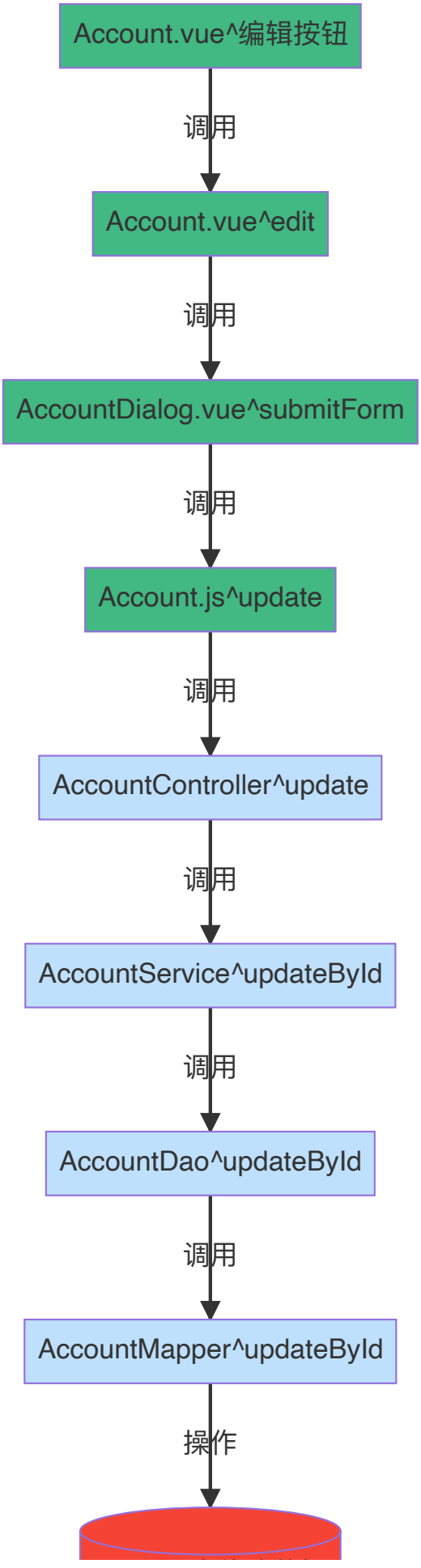
新增流程


Account.vue^创建按钮
调用
Account.vue^create
调用
AccountDialog.vue^submitForm
调用
Account.js^create
调用
AccountController^create
调用
AccountService^create
调用
AccountDao^insert
调用
AccountMapper^insert
操作

删除流程

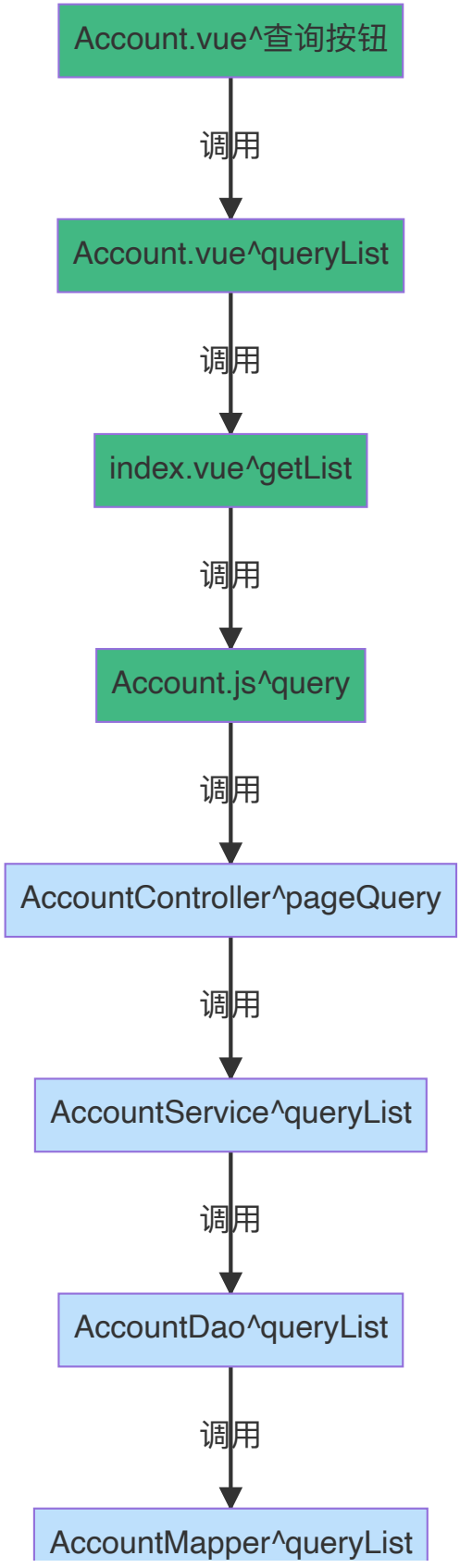

Account.vue^删除按钮
调用
Account.vue^remove
调用
Account.js^remove
调用
AccountController^remove
调用
AccountService^deleteById
调用
AccountDao^deleteById
调用
AccountMapper^deleteById
操作
account表删除数据

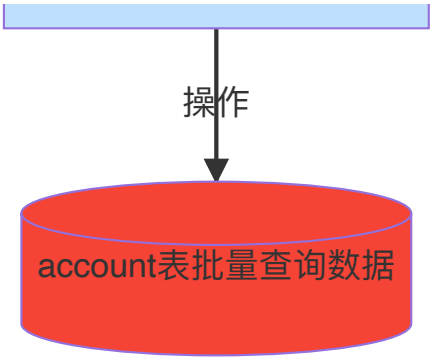
修改流程


Account.vue^编辑按钮
调用
Account.vue^edit
调用
AccountDialog.vue^submitForm
调用
Account.js^update
调用
AccountController^update
调用
AccountService^updateById
调用
AccountDao^updateById
调用
AccountMapper^updateById
操作


account表修改数据

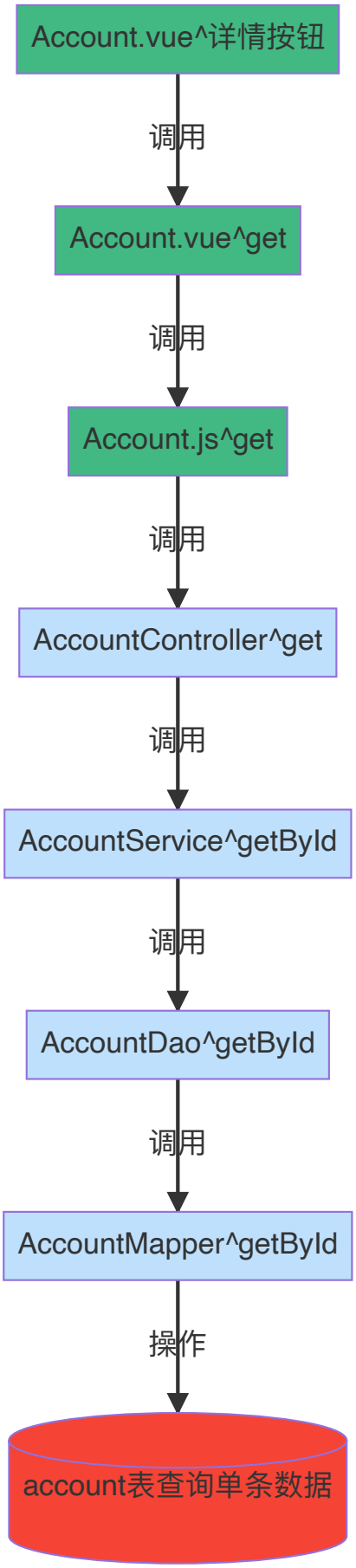
批量查询流程


Account.vue^查询按钮
调用
Account.vue^queryList
调用
index.vue^getList
调用
Account.js^query
调用
AccountController^pageQuery
调用
AccountService^queryList
调用
AccountDao^queryList
调用
AccountMapper^queryList

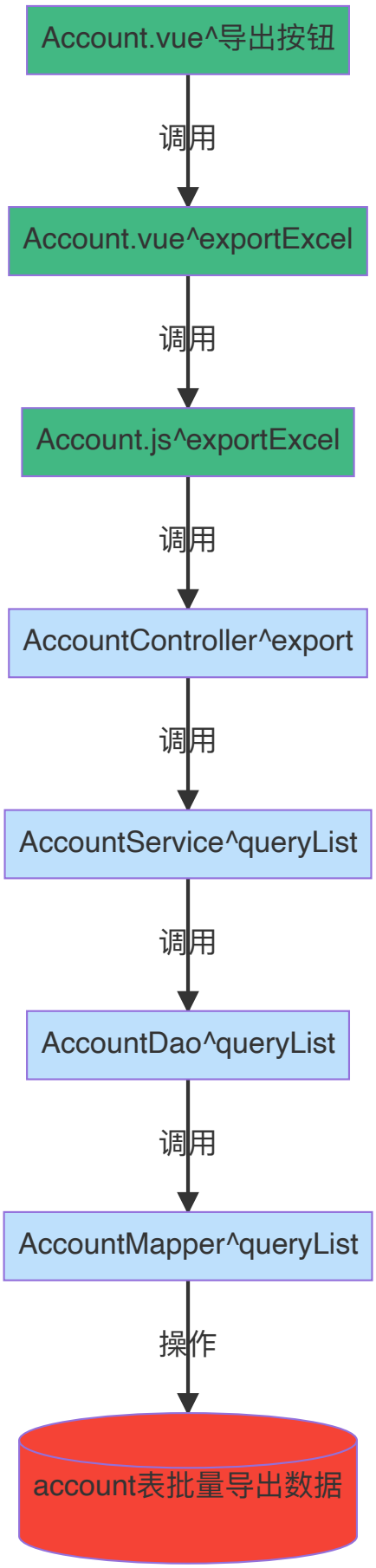


单条查询流程

这个功能在页面不一定有，但是学长先把流程给大家画出来。


Account.vue^详情按钮
调用
Account.vue^get
调用
Account.js^get
调用
AccountController^get
调用
AccountService^getById
调用
AccountDao^getById
调用
AccountMapper^getById
操作
account表查询单条数据

导出流程


Account.vue^导出按钮
调用
Account.vue^exportExcel
调用
Account.js^exportExcel
调用
AccountController^export
调用
AccountService^queryList
调用
AccountDao^queryList
调用
AccountMapper^queryList
操作
account表批量导出数据

【部门管理】详细设计

表结构设计

```
CREATE TABLE `department` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键',
  `name` varchar(255) DEFAULT NULL COMMENT '部门',
  `description` text COMMENT '备注',
  `create_time` bigint NOT NULL COMMENT '创建时间',
  `update_time` bigint DEFAULT NULL COMMENT '修改时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8mb3 COMMENT='部门'
```

后端代码设计

后端代码都放在api目录下，本节将详细介绍【部门管理】的后端代码设计

DO层（Domain Object）

数据领域对象，对象属性跟数据库字段一一对应。

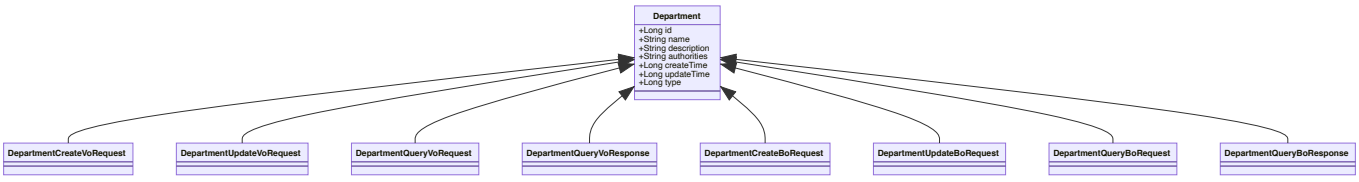
数据库表名称：department

类型：Java类

类名：DepartmentDo.java

代码位置：domain/src/main/java/com/senior/domain/model/DepartmentDo.java

类图如下：



Mapper层（mapper.xml）

Mapper.xml主要是写一些SQL语句，最常见的就是增删改查，这些SQL语句都是写在Mapper.xml里面的

数据库表名称：department

类型：xml文件

类名：DepartmentMapper.xml

代码位置：service/base/src/main/resources/mybatis/mappers/DepartmentMapper.xml

DAO层（Data Access Object）

持久层操作对象，结合DepartmentDo对象，对department表的数据进行增删改查操作。

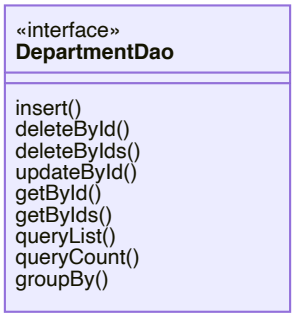
数据库表名称：department

类型：Java接口

类名：DepartmentDao.java

代码位置：service/base/src/main/java/com/senior/dao/DepartmentDao.java

类图如下：



Service层（业务逻辑层）

业务逻辑层，负责对数据的处理。如果没有逻辑处理任务，此层只做单纯的数据传递作用，而后又到了DAO层。

数据库表名称：department

类型：Java接口

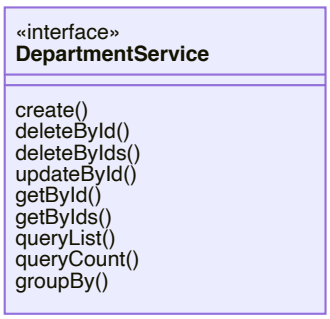
类名：DepartmentService.java

代码位置：service/business/src/main/java/com/senior/service/DepartmentService.java

接口实现类：DepartmentServiceImpl.java

接口实现类代码位置：service/business/src/main/java/com/senior/service/impl/DepartmentServiceImpl.java

类图如下：



Controller层（控制层）

控制层，主要是提供一些接口（增删改查接口）。本项目是前后端分离的项目，前端跟后端是基于http协议实现数据传输的。接口的代码其实就放在Controller这一层，这一层主要负责提供http接口，然后就是组装前端传输的数据，再将组装的数据调用Service层。

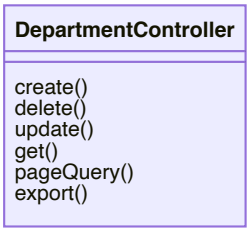
数据库表名称：department

类型：Java类

类名：DepartmentController.java

代码位置：console/api/src/main/java/com/senior/console/api/controller/DepartmentController.java

类图如下：



前端代码设计

前端是基于Vue开发的，框架的话是基于饿了么开源的element框架。代码都放在console/web目录下，本节将详细介绍【部门管理】的前端代码设计

列表页面

代码路径：console/web/src/page/department/Department.vue

代码阅读技巧：方法都放在methods里面，这里面还有很多关键字，比如created、computed、watch、methods、data等，这些都是Vue的语法，大家如果看不懂，需要去学习下Vue的基础语法，还是比较简单的。

主要功能：列表页的功能主要是展示【部门】的数据，包括页面上的一些按钮，比如新增、修改、详情、删除、导出等

详情&新增页面

代码路径：console/web/src/page/department/DepartmentDialog.vue

代码阅读技巧：跟列表页一样。

主要功能：提供【部门】新增和查看【部门】详情的功能

API定义

代码路径：console/web/src/api/Department.js

主要功能：本项目是前后端分离的项目，前面我们说过好多次了，前后端直接的数据交互是基于http协议来实现的。那么API这一块的代码就是定义的后端接口的地址，用户在页面上通过填写各种数据，或者点击某个按钮时，此时将组装好的数据通过API这层封装好的http接口，就可以把数据传输给后端的Controller层了，里面的接口大致上就是【部门管理】的增删改查操作。

流程图

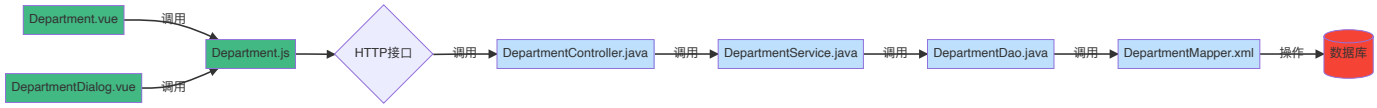
代码调用链

绿色部分是前端代码文件

蓝色部分是后端代码文件

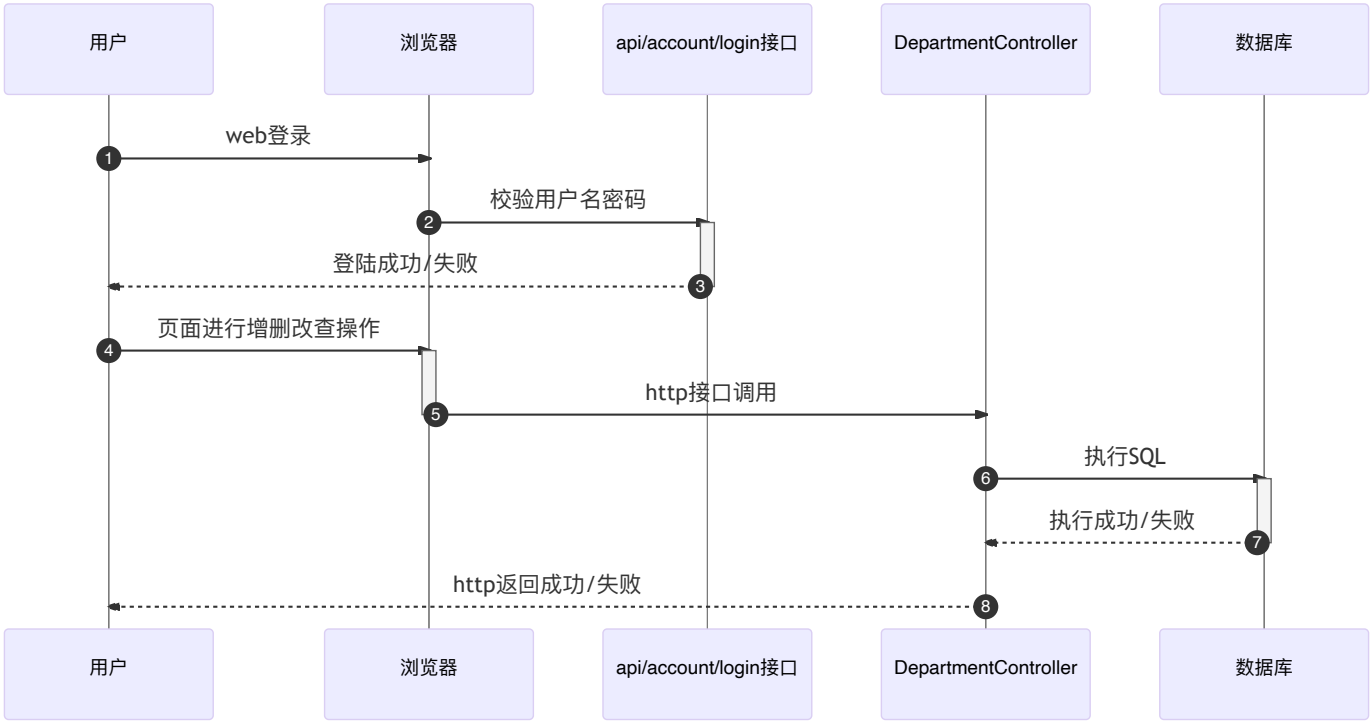
前端跟后端之间的调用是通过http接口来实现的，这里的http指的是http协议

红色部分是数据库，最终的操作都是对数据库的增删改查操作



时序图

详细介绍用户从【登录系统】到【部门增删改查操作】这个过程中系统运行时序图，每一步都标了顺序。



增删改查功能设计

前面跟大家详细介绍了后端设计和前端设计，以及前后端之间是通过http协议进行交互的逻辑也跟大家讲解过，相信大家看完学长前面的讲解后，对这个模块已经有了一个大致的认识，那本节就是打铁趁热的一个模块，通过画图的方式，把前后端整个的调用流程给大家串起来。

按钮在Department.vue文件中，可以通过中文关键字去搜索，找到具体的位置。

具体调用路径如下：（**部分方法名称可能存在细微的差异，这个大家具体看代码，以代码中的方法名为准**）

讲解调用路径时学长会按照**文件名**+'^'+**方法名**的方式讲解，这样大家就可以很快的在项目中找到相关的代码了

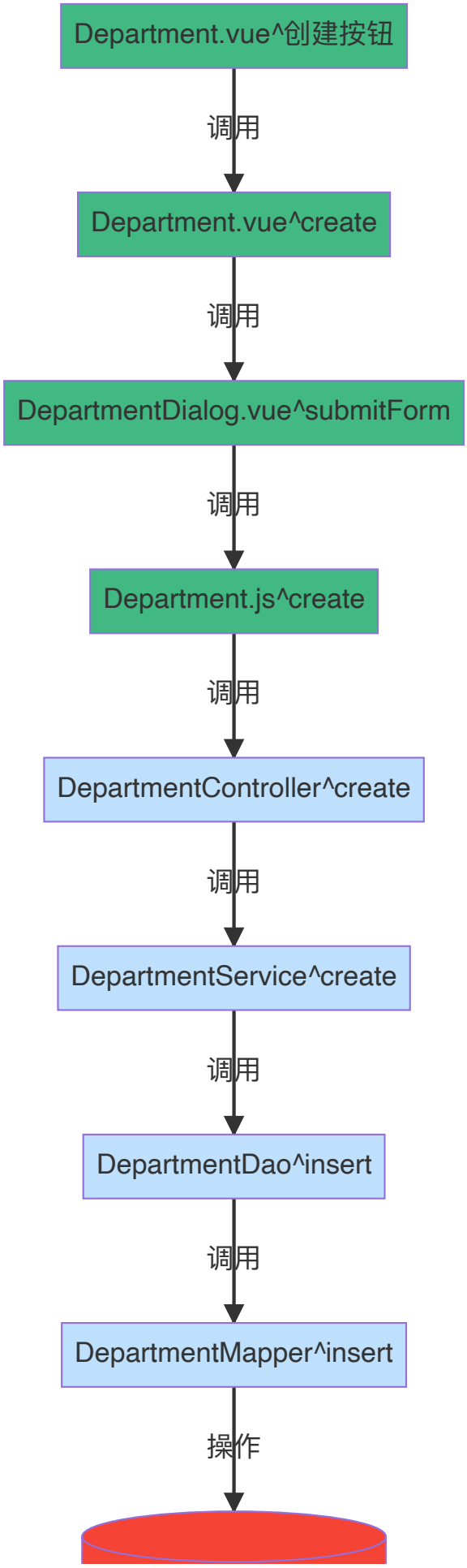
为了让大家更好地区分前后端代码的职责，学长通过不同的颜色来区分前后端代码，让大家一目了然！

绿色部分是前端代码文件

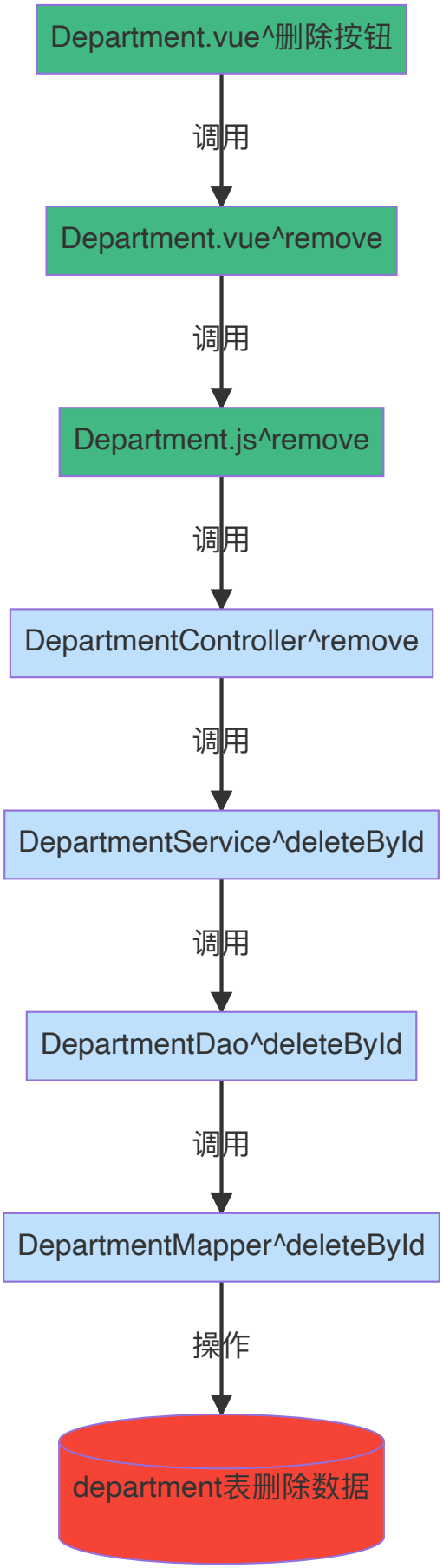
蓝色部分是后端代码文件

红色部分是数据库对应的表名

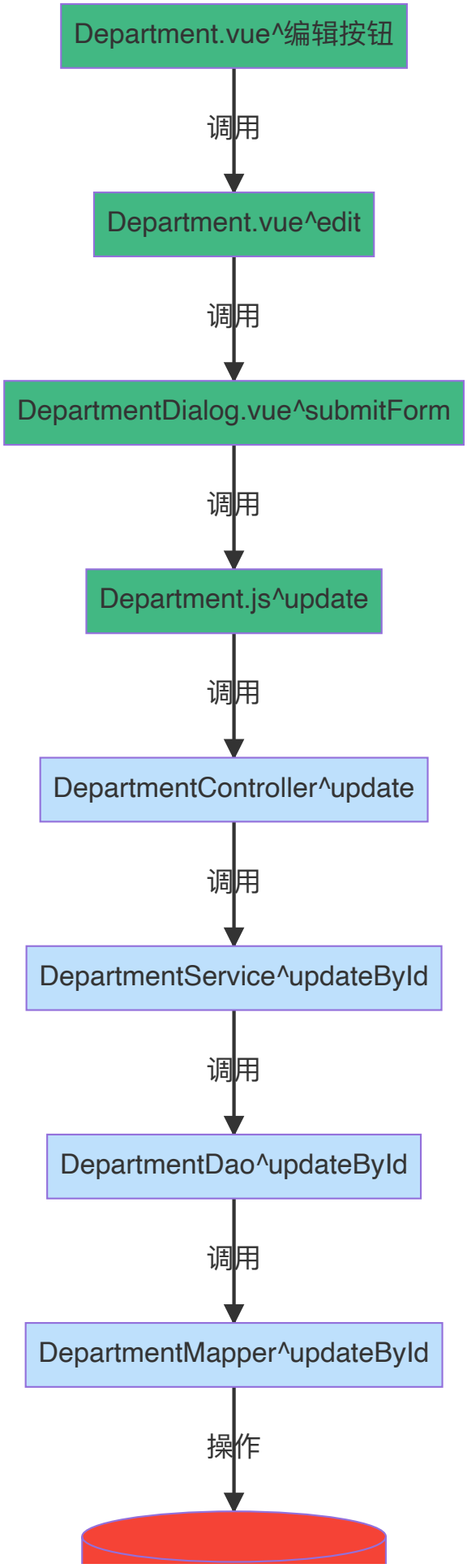
新增流程


Department.vue^创建按钮
调用
Department.vue^create
调用
DepartmentDialog.vue^submitForm
调用
Department.js^create
调用
DepartmentController^create
调用
DepartmentService^create
调用
DepartmentDao^insert
调用
DepartmentMapper^insert
操作

删除流程

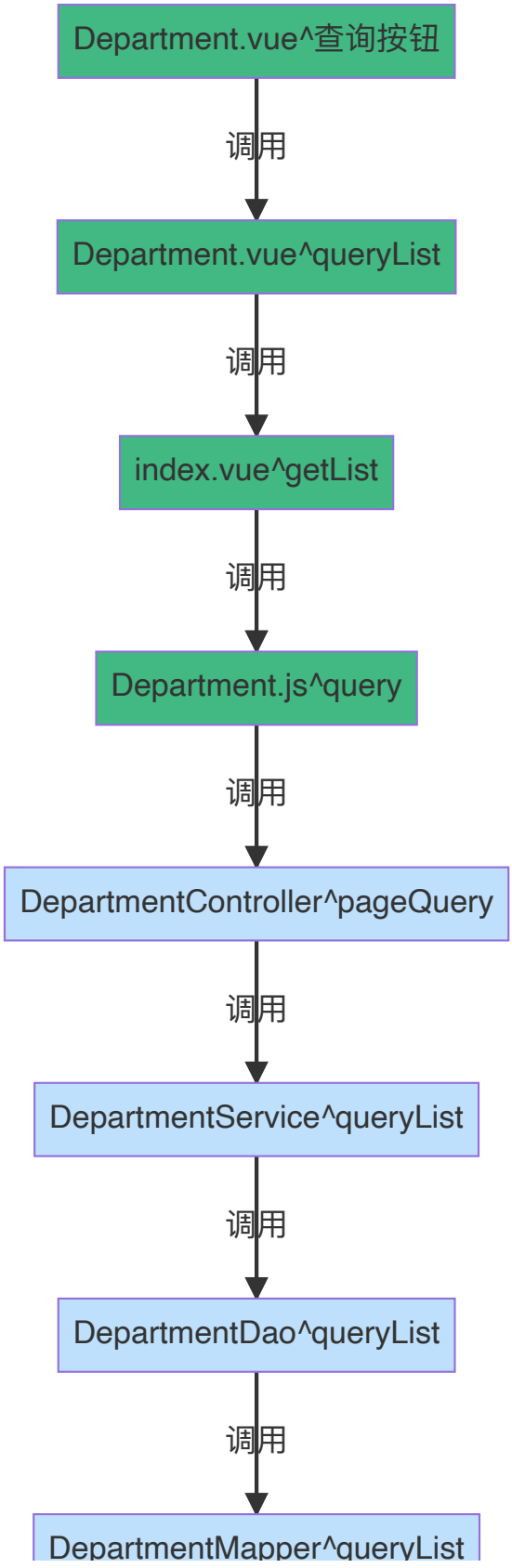

Department.vue^删除按钮
调用
Department.vue^remove
调用
Department.js^remove
调用
DepartmentController^remove
调用
DepartmentService^deleteById
调用
DepartmentDao^deleteById
调用
DepartmentMapper^deleteById
操作
department表删除数据

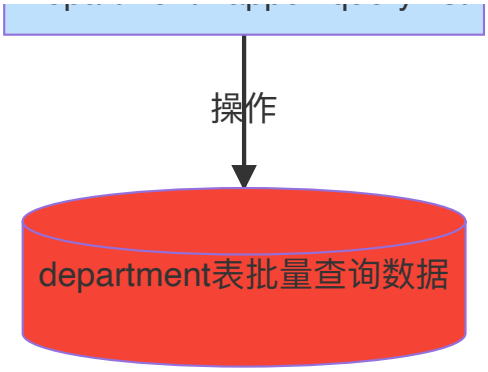
修改流程


Department.vue^编辑按钮
调用
Department.vue^edit
调用
DepartmentDialog.vue^submitForm
调用
Department.js^update
调用
DepartmentController^update
调用
DepartmentService^updateById
调用
DepartmentDao^updateById
调用
DepartmentMapper^updateById
操作


department表修改数据

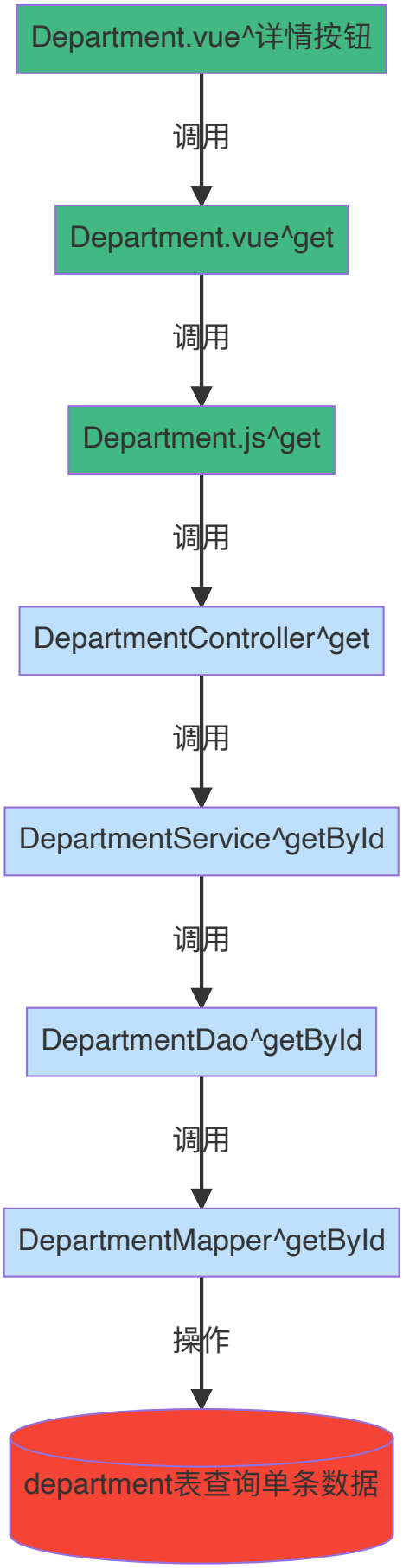
批量查询流程


Department.vue^查询按钮
调用
Department.vue^queryList
调用
index.vue^getList
调用
Department.js^query
调用
DepartmentController^pageQuery
调用
DepartmentService^queryList
调用
DepartmentDao^queryList
调用
DepartmentMapper^queryList

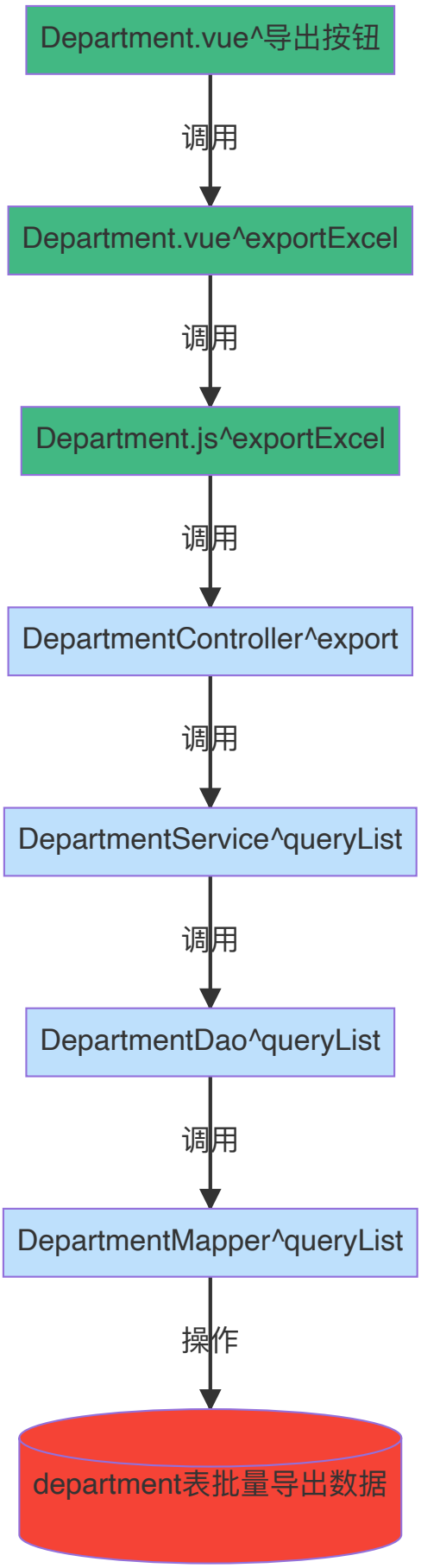


单条查询流程

这个功能在页面不一定有，但是学长先把流程给大家画出来。


Department.vue^详情按钮
调用
Department.vue^get
调用
Department.js^get
调用
DepartmentController^get
调用
DepartmentService^getById
调用
DepartmentDao^getById
调用
DepartmentMapper^getById
操作
department表查询单条数据

导出流程


Department.vue^导出按钮
调用
Department.vue^exportExcel
调用
Department.js^exportExcel
调用
DepartmentController^export
调用
DepartmentService^queryList
调用
DepartmentDao^queryList
调用
DepartmentMapper^queryList
操作
department表批量导出数据

【报修管理】详细设计

表结构设计

```
CREATE TABLE `repair` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键',
  `status` int NOT NULL COMMENT '状态;1:未处理 2:处理中 3: 已完成',
  `content` text NOT NULL COMMENT '内容',
  `room_id` bigint NOT NULL COMMENT '会议室',
  `account_id` bigint NOT NULL COMMENT '报修人',
  `create_time` bigint NOT NULL COMMENT '创建时间',
  `update_time` bigint DEFAULT NULL COMMENT '修改时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8mb3 COMMENT='报修'
```

后端代码设计

后端代码都放在api目录下，本节将详细介绍【报修管理】的后端代码设计

DO层（Domain Object）

数据领域对象，对象属性跟数据库字段一一对应。

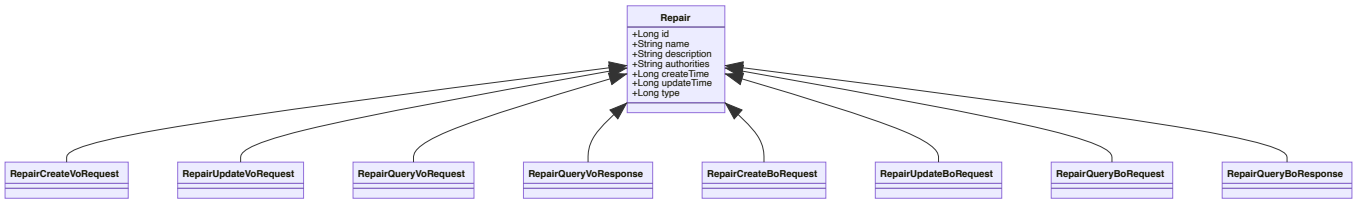
数据库表名称：repair

类型：Java类

类名：RepairDo.java

代码位置：domain/src/main/java/com/senior/domain/model/RepairDo.java

类图如下：



Mapper层（mapper.xml）

Mapper.xml主要是写一些SQL语句，最常见的就是增删改查，这些SQL语句都是写在Mapper.xml里面的

数据库表名称：repair

类型：xml文件

类名：RepairMapper.xml

代码位置：service/base/src/main/resources/mybatis/mappers/RepairMapper.xml

DAO层（Data Access Object）

持久层操作对象，结合RepairDo对象，对repair表的数据进行增删改查操作。

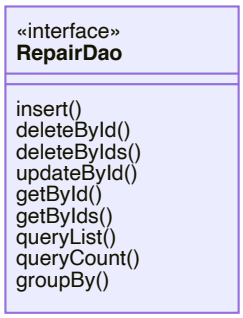
数据库表名称：repair

类型：Java接口

类名：RepairDao.java

代码位置：service/base/src/main/java/com/senior/dao/RepairDao.java

类图如下：



Service层（业务逻辑层）

业务逻辑层，负责对数据的处理。如果没有逻辑处理任务，此层只做单纯的数据传递作用，而后又到了DAO层。

数据库表名称：repair

类型：Java接口

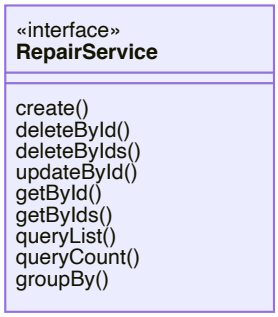
类名：RepairService.java

代码位置：service/business/src/main/java/com/senior/service/RepairService.java

接口实现类：RepairServiceImpl.java

接口实现类代码位置：service/business/src/main/java/com/senior/service/impl/RepairServiceImpl.java

类图如下：



Controller层（控制层）

控制层，主要是提供一些接口（增删改查接口）。本项目是前后端分离的项目，前端跟后端是基于http协议实现数据传输的。接口的代码其实就放在Controller这一层，这一层主要负责提供http接口，然后就是组装前端传输的数据，再将组装的数据调用Service层。

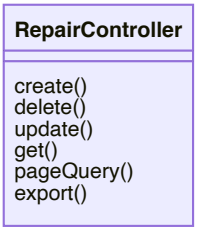
数据库表名称：repair

类型：Java类

类名：RepairController.java

代码位置：console/api/src/main/java/com/senior/console/api/controller/RepairController.java

类图如下：



前端代码设计

前端是基于Vue开发的，框架的话是基于饿了么开源的element框架。代码都放在console/web目录下，本节将详细介绍【报修管理】的前端代码设计

列表页面

代码路径：console/web/src/page/repair/Repair.vue

代码阅读技巧：方法都放在methods里面，这里面还有很多关键字，比如created、computed、watch、methods、data等，这些都是Vue的语法，大家如果看不懂，需要去学习下Vue的基础语法，还是比较简单的。

主要功能：列表页的功能主要是展示【报修】的数据，包括页面上的一些按钮，比如新增、修改、详情、删除、导出等

详情&新增页面

代码路径：console/web/src/page/repair/RepairDialog.vue

代码阅读技巧：跟列表页一样。

主要功能：提供【报修】新增和查看【报修】详情的功能

API定义

代码路径：console/web/src/api/Repair.js

主要功能：本项目是前后端分离的项目，前面我们说过好多次了，前后端直接的数据交互是基于http协议来实现的。那么API这一块的代码就是定义的后端接口的地址，用户在页面上通过填写各种数据，或者点击某个按钮时，此时将组装好的数据通过API这层封装好的http接口，就可以把数据传输给后端的Controller层了，里面的接口大致上就是【报修管理】的增删改查操作。

流程图

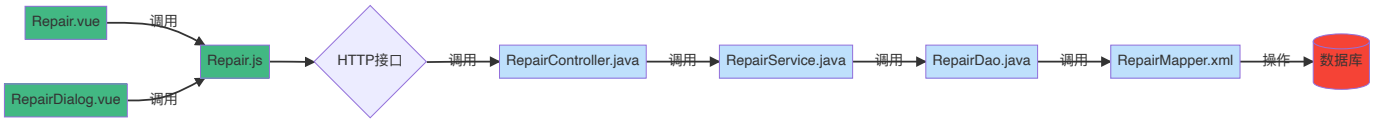
代码调用链

绿色部分是前端代码文件

蓝色部分是后端代码文件

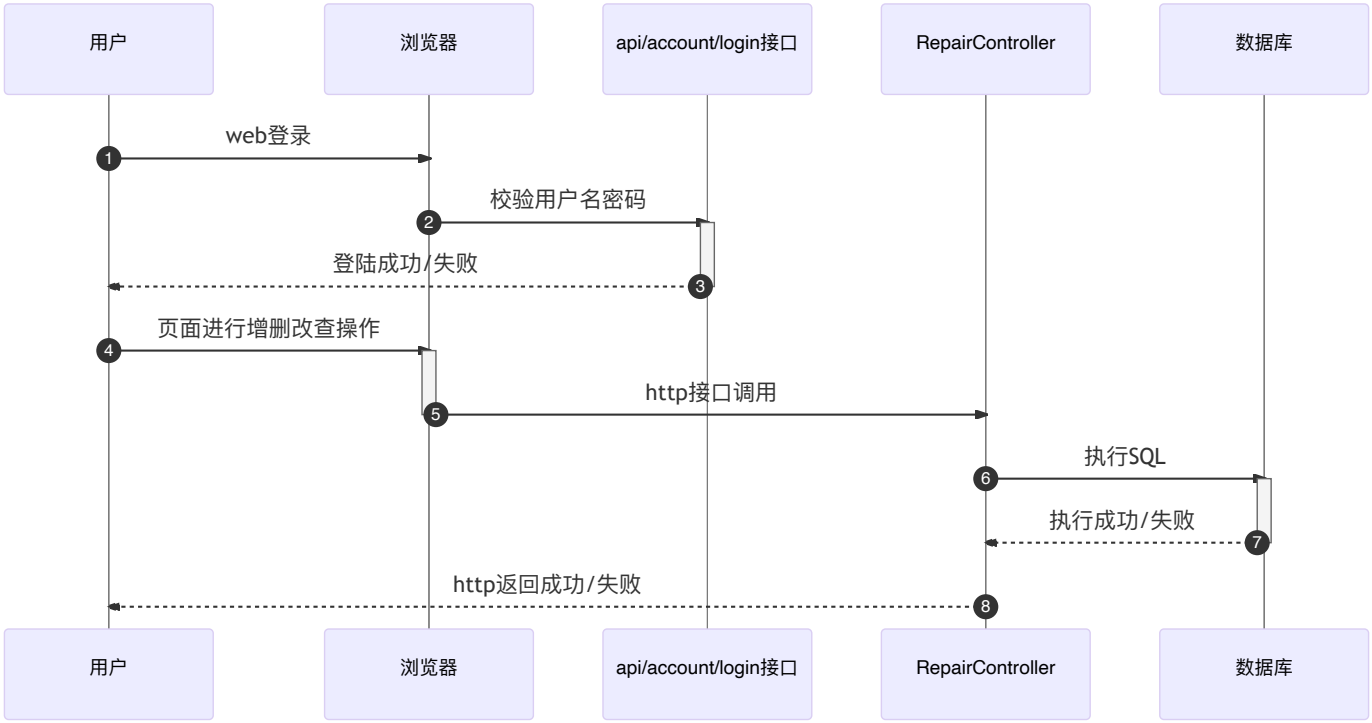
前端跟后端之间的调用是通过http接口来实现的，这里的http指的是http协议

红色部分是数据库，最终的操作都是对数据库的增删改查操作



时序图

详细介绍用户从【登录系统】到【报修增删改查操作】这个过程中系统运行时序图，每一步都标了顺序。



增删改查功能设计

前面跟大家详细介绍了后端设计和前端设计，以及前后端之间是通过http协议进行交互的逻辑也跟大家讲解过，相信大家看完学长前面的讲解后，对这个模块已经有了一个大致的认识，那本节就是打铁趁热的一个模块，通过画图的方式，把前后端整个的调用流程给大家串起来。

按钮在Repair.vue文件中，可以通过中文关键字去搜索，找到具体的位置。

具体调用路径如下：（**部分方法名称可能存在细微的差异，这个大家具体看代码，以代码中的方法名为准**）

讲解调用路径时学长会按照**文件名**+'^'+**方法名**的方式讲解，这样大家就可以很快的在项目中找到相关的代码了

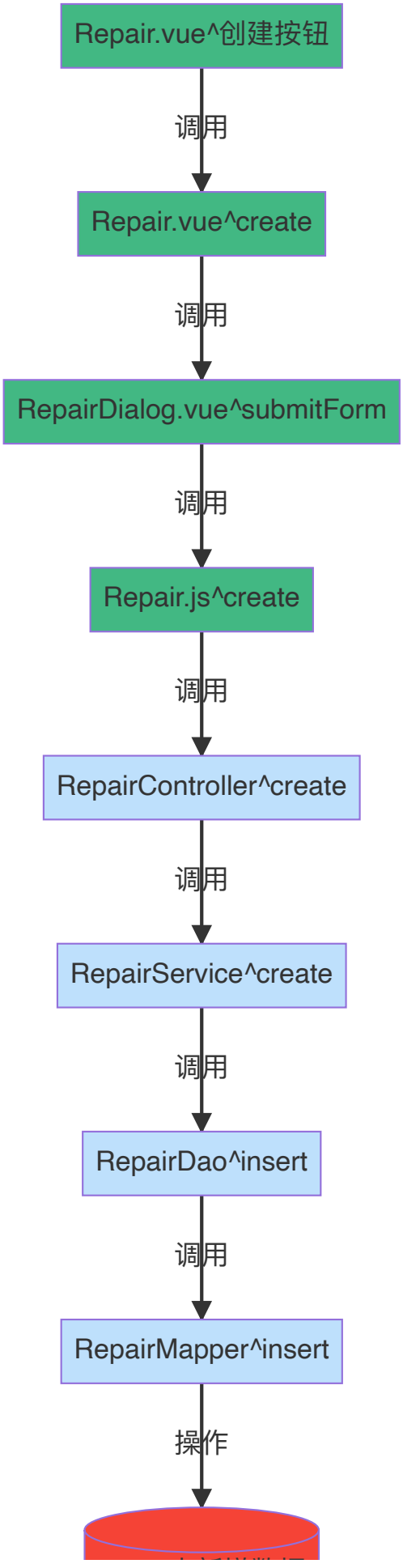
为了让大家更好地区分前后端代码的职责，学长通过不同的颜色来区分前后端代码，让大家一目了然！

绿色部分是前端代码文件

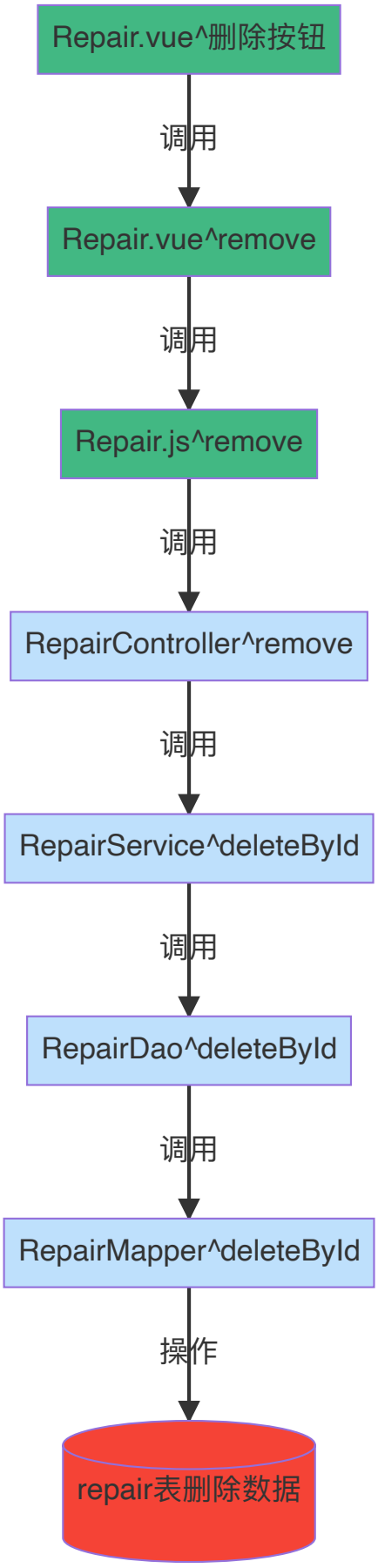
蓝色部分是后端代码文件

红色部分是数据库对应的表名

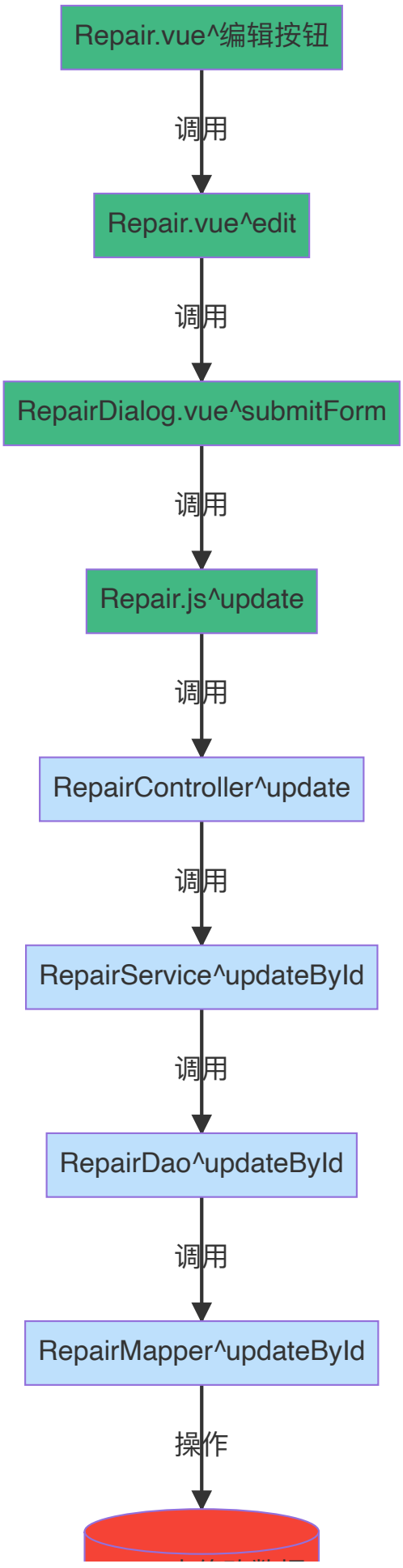
新增流程


Repair.vue^创建按钮
调用
Repair.vue^create
调用
RepairDialog.vue^submitForm
调用
Repair.js^create
调用
RepairController^create
调用
RepairService^create
调用
RepairDao^insert
调用
RepairMapper^insert
操作

删除流程

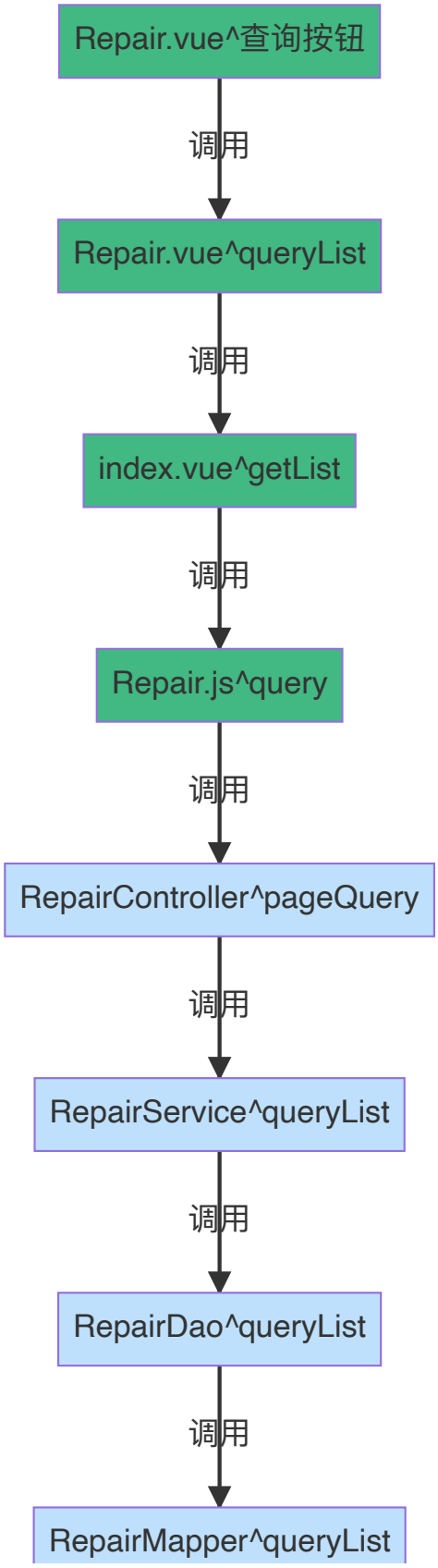

Repair.vue^删除按钮
调用
Repair.vue^remove
调用
Repair.js^remove
调用
RepairController^remove
调用
RepairService^deleteById
调用
RepairDao^deleteById
调用
RepairMapper^deleteById
操作
repair表删除数据

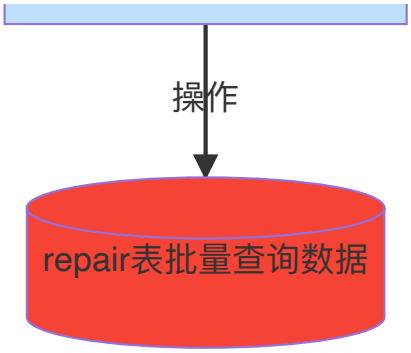
修改流程


Repair.vue^编辑按钮
调用
Repair.vue^edit
调用
RepairDialog.vue^submitForm
调用
Repair.js^update
调用
RepairController^update
调用
RepairService^updateById
调用
RepairDao^updateById
调用
RepairMapper^updateById
操作


repair表修改数据

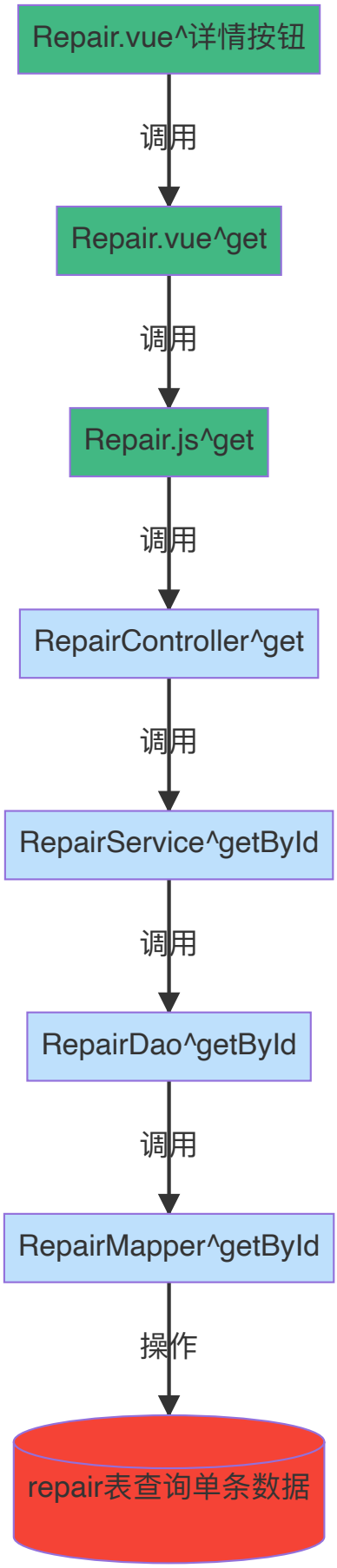
批量查询流程


Repair.vue^查询按钮
调用
Repair.vue^queryList
调用
index.vue^getList
调用
Repair.js^query
调用
RepairController^pageQuery
调用
RepairService^queryList
调用
RepairDao^queryList
调用
RepairMapper^queryList

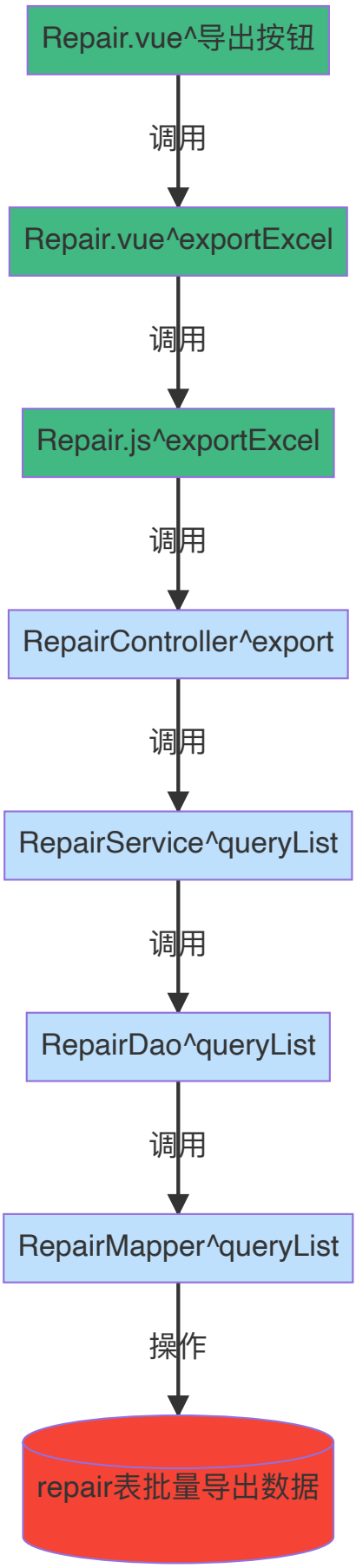


单条查询流程

这个功能在页面不一定有，但是学长先把流程给大家画出来。


Repair.vue^详情按钮
调用
Repair.vue^get
调用
Repair.js^get
调用
RepairController^get
调用
RepairService^getById
调用
RepairDao^getById
调用
RepairMapper^getById
操作
repair表查询单条数据

导出流程


Repair.vue^导出按钮
调用
Repair.vue^exportExcel
调用
Repair.js^exportExcel
调用
RepairController^export
调用
RepairService^queryList
调用
RepairDao^queryList
调用
RepairMapper^queryList
操作
repair表批量导出数据

【预约记录管理】详细设计

表结构设计

```
CREATE TABLE `reserve` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键',
  `room_id` bigint NOT NULL COMMENT '会议室',
  `content` text COMMENT '内容',
  `account_id` bigint NOT NULL COMMENT '预约人',
  `reserve_start_time` bigint NOT NULL COMMENT '开始时间',
  `reserve_end_time` bigint NOT NULL COMMENT '结束时间',
  `create_time` bigint NOT NULL COMMENT '创建时间',
  `update_time` bigint DEFAULT NULL COMMENT '修改时间',
  `meeting_id` bigint NOT NULL COMMENT '会议',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8mb3 COMMENT='预约记录'
```

后端代码设计

后端代码都放在api目录下，本节将详细介绍【预约记录管理】的后端代码设计

DO层（Domain Object）

数据领域对象，对象属性跟数据库字段一一对应。

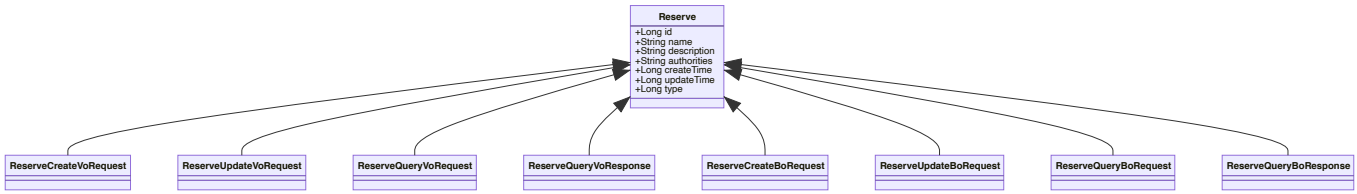
数据库表名称：reserve

类型：Java类

类名：ReserveDo.java

代码位置：domain/src/main/java/com/senior/domain/model/ReserveDo.java

类图如下：



Mapper层（mapper.xml）

Mapper.xml主要是写一些SQL语句，最常见的就是增删改查，这些SQL语句都是写在Mapper.xml里面的

数据库表名称：reserve

类型：xml文件

类名：ReserveMapper.xml

代码位置：service/base/src/main/resources/mybatis/mappers/ReserveMapper.xml

DAO层（Data Access Object）

持久层操作对象，结合ReserveDo对象，对reserve表的数据进行增删改查操作。

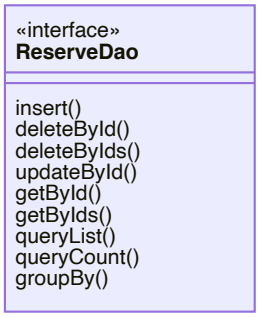
数据库表名称：reserve

类型：Java接口

类名：ReserveDao.java

代码位置：service/base/src/main/java/com/senior/dao/ReserveDao.java

类图如下：



Service层（业务逻辑层）

业务逻辑层，负责对数据的处理。如果没有逻辑处理任务，此层只做单纯的数据传递作用，而后又到了DAO层。

数据库表名称：reserve

类型：Java接口

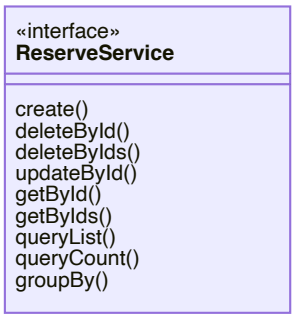
类名：ReserveService.java

代码位置：service/business/src/main/java/com/senior/service/ReserveService.java

接口实现类：ReserveServiceImpl.java

接口实现类代码位置：service/business/src/main/java/com/senior/service/impl/ReserveServiceImpl.java

类图如下：



Controller层（控制层）

控制层，主要是提供一些接口（增删改查接口）。本项目是前后端分离的项目，前端跟后端是基于http协议实现数据传输的。接口的代码其实就放在Controller这一层，这一层主要负责提供http接口，然后就是组装前端传输的数据，再将组装的数据调用Service层。

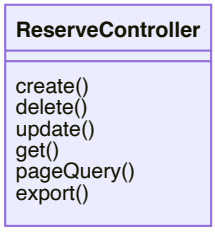
数据库表名称：reserve

类型：Java类

类名：ReserveController.java

代码位置：console/api/src/main/java/com/senior/console/api/controller/ReserveController.java

类图如下：



前端代码设计

前端是基于Vue开发的，框架的话是基于饿了么开源的element框架。代码都放在console/web目录下，本节将详细介绍【预约记录管理】的前端代码设计

列表页面

代码路径：console/web/src/page/reserve/Reserve.vue

代码阅读技巧：方法都放在methods里面，这里面还有很多关键字，比如created、computed、watch、methods、data等，这些都是Vue的语法，大家如果看不懂，需要去学习下Vue的基础语法，还是比较简单的。

主要功能：列表页的功能主要是展示【预约记录】的数据，包括页面上的一些按钮，比如新增、修改、详情、删除、导出等

详情&新增页面

代码路径：console/web/src/page/reserve/ReserveDialog.vue

代码阅读技巧：跟列表页一样。

主要功能：提供【预约记录】新增和查看【预约记录】详情的功能

API定义

代码路径：console/web/src/api/Reserve.js

主要功能：本项目是前后端分离的项目，前面我们说过好多次了，前后端直接的数据交互是基于http协议来实现的。那么API这一块的代码就是定义的后端接口的地址，用户在页面上通过填写各种数据，或者点击某个按钮时，此时将组装好的数据通过API这层封装好的http接口，就可以把数据传输给后端的Controller层了，里面的接口大致上就是【预约记录管理】的增删改查操作。

流程图

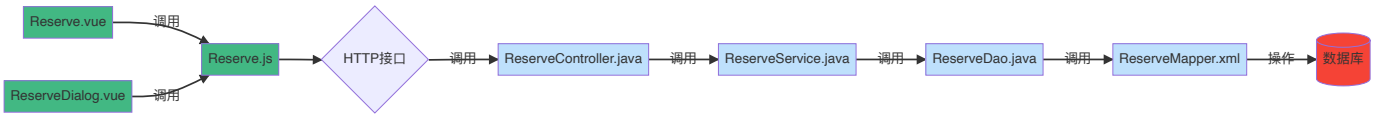
代码调用链

绿色部分是前端代码文件

蓝色部分是后端代码文件

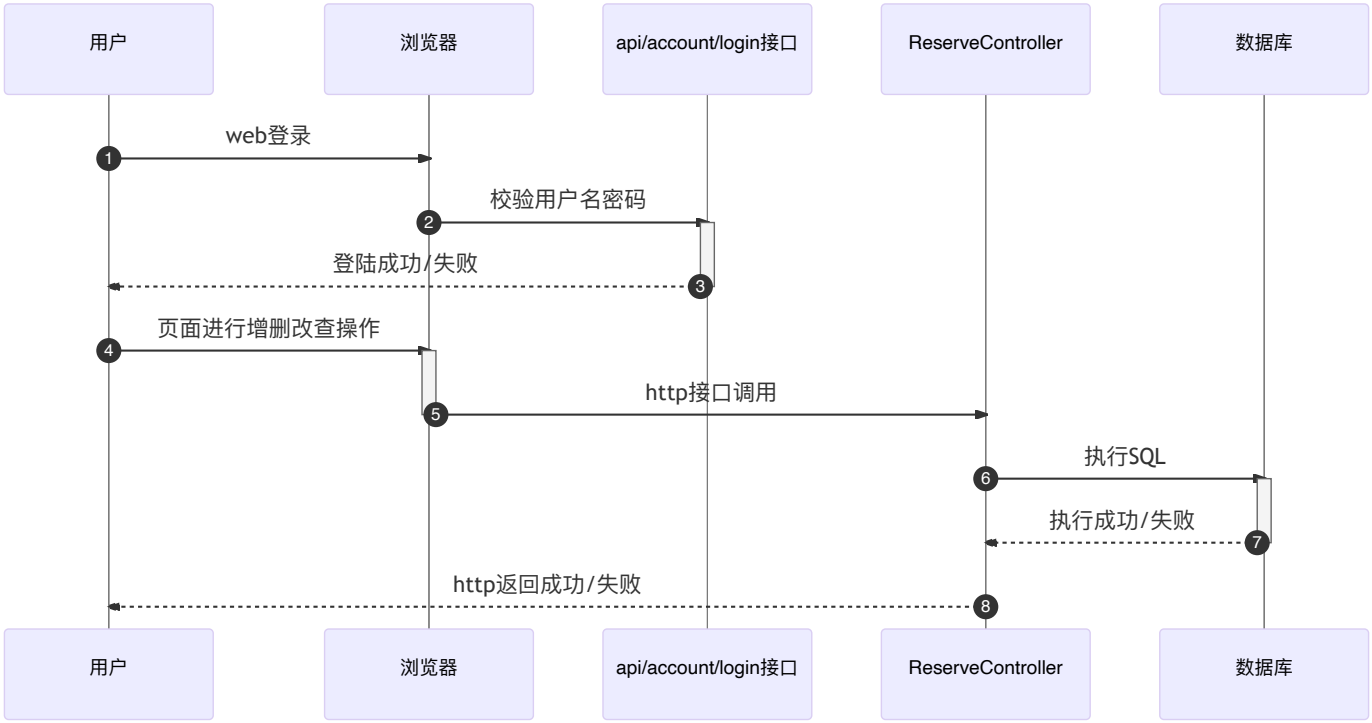
前端跟后端之间的调用是通过http接口来实现的，这里的http指的是http协议

红色部分是数据库，最终的操作都是对数据库的增删改查操作



时序图

详细介绍用户从【登录系统】到【预约记录增删改查操作】这个过程中系统运行时序图，每一步都标了顺序。



增删改查功能设计

前面跟大家详细介绍了后端设计和前端设计，以及前后端之间是通过http协议进行交互的逻辑也跟大家讲解过，相信大家看完学长前面的讲解后，对这个模块已经有了一个大致的认识，那本节就是打铁趁热的一个模块，通过画图的方式，把前后端整个的调用流程给大家串起来。

按钮在Reserve.vue文件中，可以通过中文关键字去搜索，找到具体的位置。

具体调用路径如下：（**部分方法名称可能存在细微的差异，这个大家具体看代码，以代码中的方法名为准**）

讲解调用路径时学长会按照**文件名**+'^'+**方法名**的方式讲解，这样大家就可以很快的在项目中找到相关的代码了

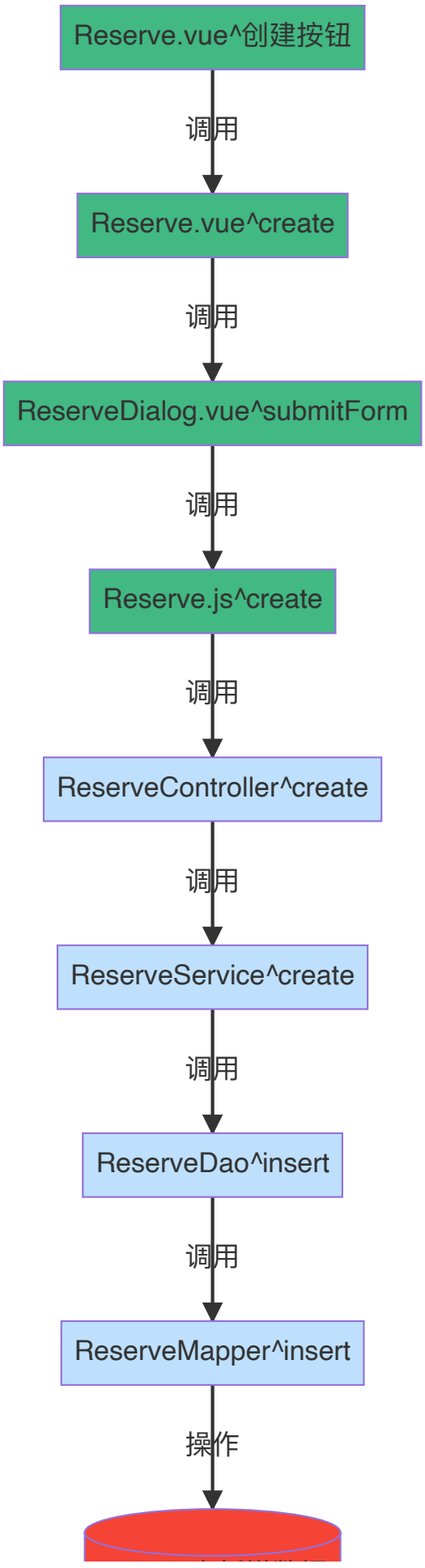
为了让大家更好地区分前后端代码的职责，学长通过不同的颜色来区分前后端代码，让大家一目了然！

绿色部分是前端代码文件

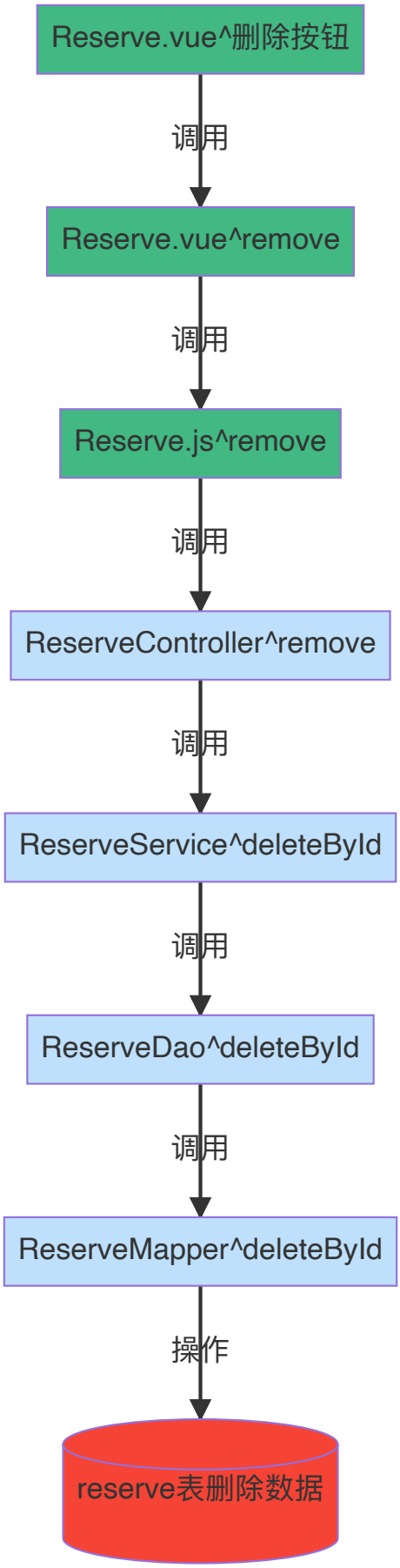
蓝色部分是后端代码文件

红色部分是数据库对应的表名

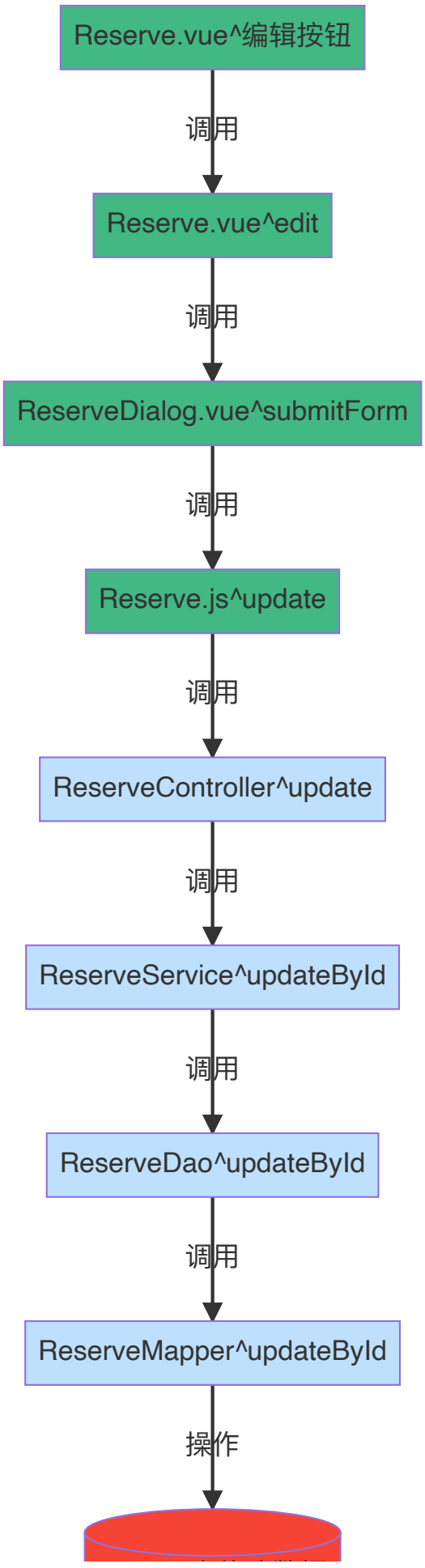
新增流程


Reserve.vue^创建按钮
调用
Reserve.vue^create
调用
ReserveDialog.vue^submitForm
调用
Reserve.js^create
调用
ReserveController^create
调用
ReserveService^create
调用
ReserveDao^insert
调用
ReserveMapper^insert
操作

删除流程

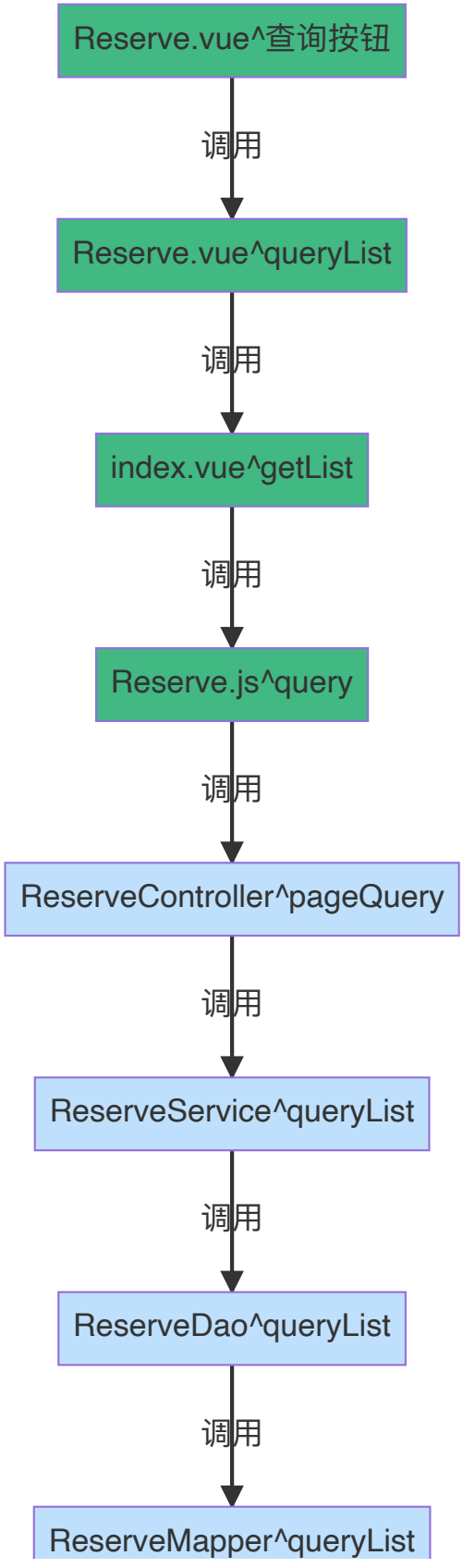

Reserve.vue^删除按钮
调用
Reserve.vue^remove
调用
Reserve.js^remove
调用
ReserveController^remove
调用
ReserveService^deleteById
调用
ReserveDao^deleteById
调用
ReserveMapper^deleteById
操作
reserve表删除数据

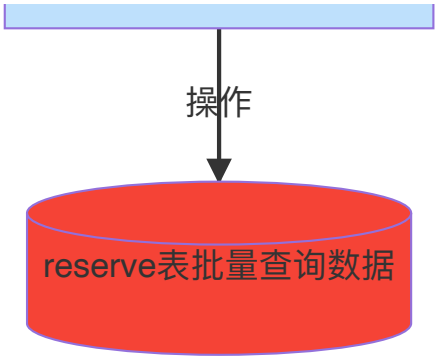
修改流程


Reserve.vue^编辑按钮
调用
Reserve.vue^edit
调用
ReserveDialog.vue^submitForm
调用
Reserve.js^update
调用
ReserveController^update
调用
ReserveService^updateById
调用
ReserveDao^updateById
调用
ReserveMapper^updateById
操作


reserve表修改数据

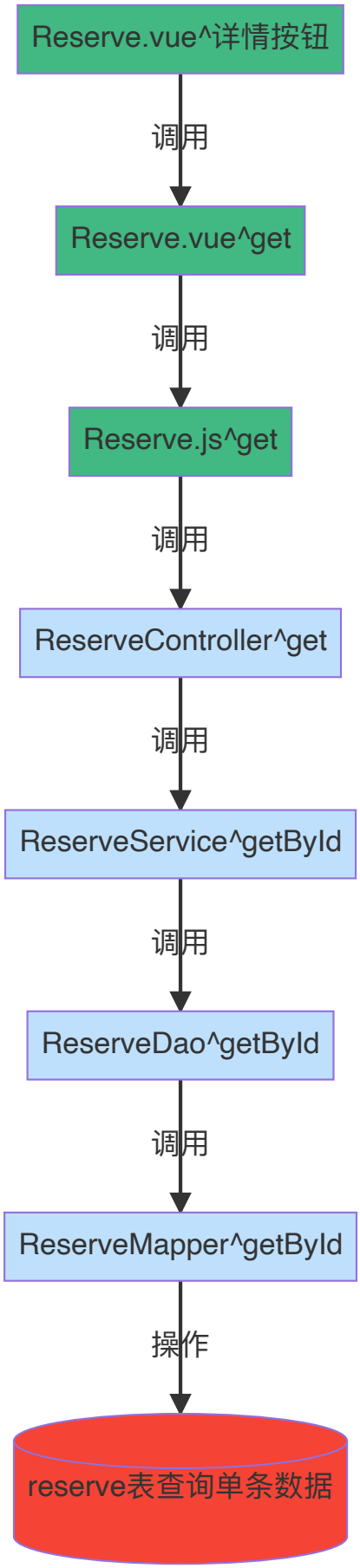
批量查询流程


Reserve.vue^查询按钮
调用
Reserve.vue^queryList
调用
index.vue^getList
调用
Reserve.js^query
调用
ReserveController^pageQuery
调用
ReserveService^queryList
调用
ReserveDao^queryList
调用
ReserveMapper^queryList

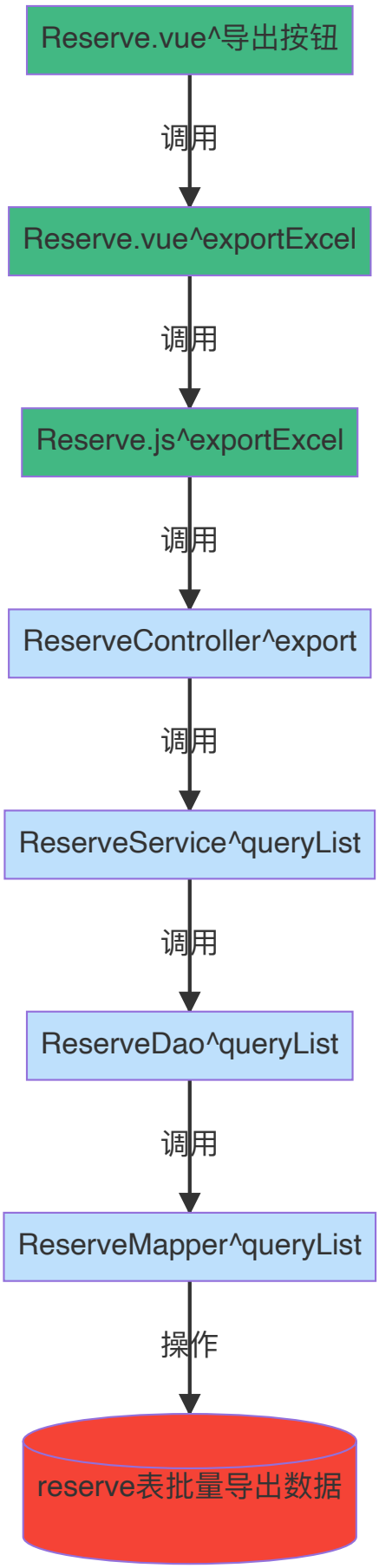


单条查询流程

这个功能在页面不一定有，但是学长先把流程给大家画出来。


Reserve.vue^详情按钮
调用
Reserve.vue^get
调用
Reserve.js^get
调用
ReserveController^get
调用
ReserveService^getById
调用
ReserveDao^getById
调用
ReserveMapper^getById
操作
reserve表查询单条数据

导出流程


Reserve.vue^导出按钮
调用
Reserve.vue^exportExcel
调用
Reserve.js^exportExcel
调用
ReserveController^export
调用
ReserveService^queryList
调用
ReserveDao^queryList
调用
ReserveMapper^queryList
操作
reserve表批量导出数据

【会议室管理】详细设计

表结构设计

```
CREATE TABLE `room` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键',
  `name` varchar(255) NOT NULL COMMENT '会议室',
  `room_image_ids` varchar(64) NOT NULL COMMENT '实景照片',
  `description` text COMMENT '备注',
  `create_time` bigint NOT NULL COMMENT '创建时间',
  `update_time` bigint DEFAULT NULL COMMENT '修改时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8mb3 COMMENT='会议室'
```

后端代码设计

后端代码都放在api目录下，本节将详细介绍【会议室管理】的后端代码设计

DO层（Domain Object）

数据领域对象，对象属性跟数据库字段一一对应。

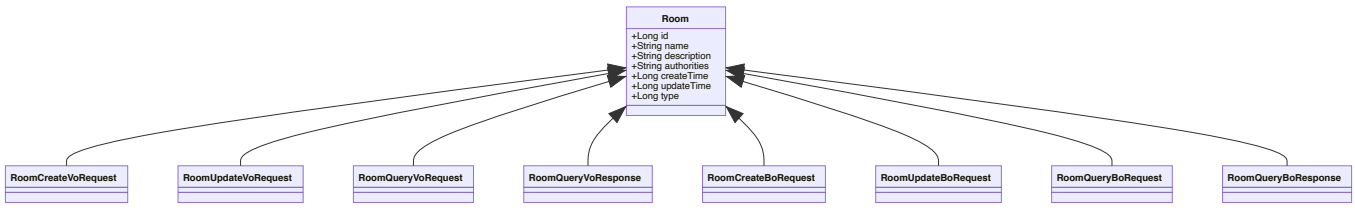
数据库表名称：room

类型：Java类

类名：RoomDo.java

代码位置：domain/src/main/java/com/senior/domain/model/RoomDo.java

类图如下：



Mapper层（mapper.xml）

Mapper.xml主要是写一些SQL语句，最常见的就是增删改查，这些SQL语句都是写在Mapper.xml里面的

数据库表名称：room

类型：xml文件

类名：RoomMapper.xml

代码位置：service/base/src/main/resources/mybatis/mappers/RoomMapper.xml

DAO层（Data Access Object）

持久层操作对象，结合RoomDo对象，对room表的数据进行增删改查操作。

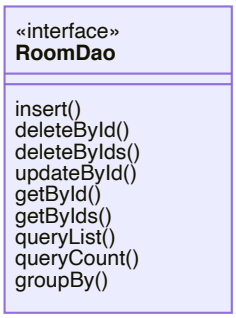
数据库表名称：room

类型：Java接口

类名：RoomDao.java

代码位置：service/base/src/main/java/com/senior/dao/RoomDao.java

类图如下：



Service层（业务逻辑层）

业务逻辑层，负责对数据的处理。如果没有逻辑处理任务，此层只做单纯的数据传递作用，而后又到了DAO层。

数据库表名称：room

类型：Java接口

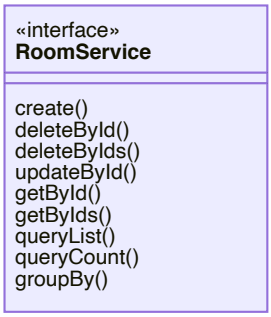
类名：RoomService.java

代码位置：service/business/src/main/java/com/senior/service/RoomService.java

接口实现类：RoomServiceImpl.java

接口实现类代码位置：service/business/src/main/java/com/senior/service/impl/RoomServiceImpl.java

类图如下：



Controller层（控制层）

控制层，主要是提供一些接口（增删改查接口）。本项目是前后端分离的项目，前端跟后端是基于http协议实现数据传输的。接口的代码其实就放在Controller这一层，这一层主要负责提供http接口，然后就是组装前端传输的数据，再将组装的数据调用Service层。

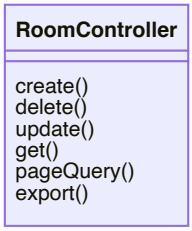
数据库表名称：room

类型：Java类

类名：RoomController.java

代码位置：console/api/src/main/java/com/senior/console/api/controller/RoomController.java

类图如下：



前端代码设计

前端是基于Vue开发的，框架的话是基于饿了么开源的element框架。代码都放在console/web目录下，本节将详细介绍【会议室管理】的前端代码设计

列表页面

代码路径：console/web/src/page/room/Room.vue

代码阅读技巧：方法都放在methods里面，这里面还有很多关键字，比如created、computed、watch、methods、data等，这些都是Vue的语法，大家如果看不懂，需要去学习下Vue的基础语法，还是比较简单的。

主要功能：列表页的功能主要是展示【会议室】的数据，包括页面上的一些按钮，比如新增、修改、详情、删除、导出等

详情&新增页面

代码路径：console/web/src/page/room/RoomDialog.vue

代码阅读技巧：跟列表页一样。

主要功能：提供【会议室】新增和查看【会议室】详情的功能

API定义

代码路径：console/web/src/api/Room.js

主要功能：本项目是前后端分离的项目，前面我们说过好多次了，前后端直接的数据交互是基于http协议来实现的。那么API这一块的代码就是定义的后端接口的地址，用户在页面上通过填写各种数据，或者点击某个按钮时，此时将组装好的数据通过API这层封装好的http接口，就可以把数据传输给后端的Controller层了，里面的接口大致上就是【会议室管理】的增删改查操作。

流程图

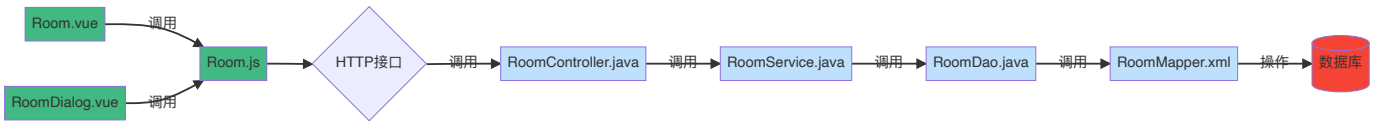
代码调用链

绿色部分是前端代码文件

蓝色部分是后端代码文件

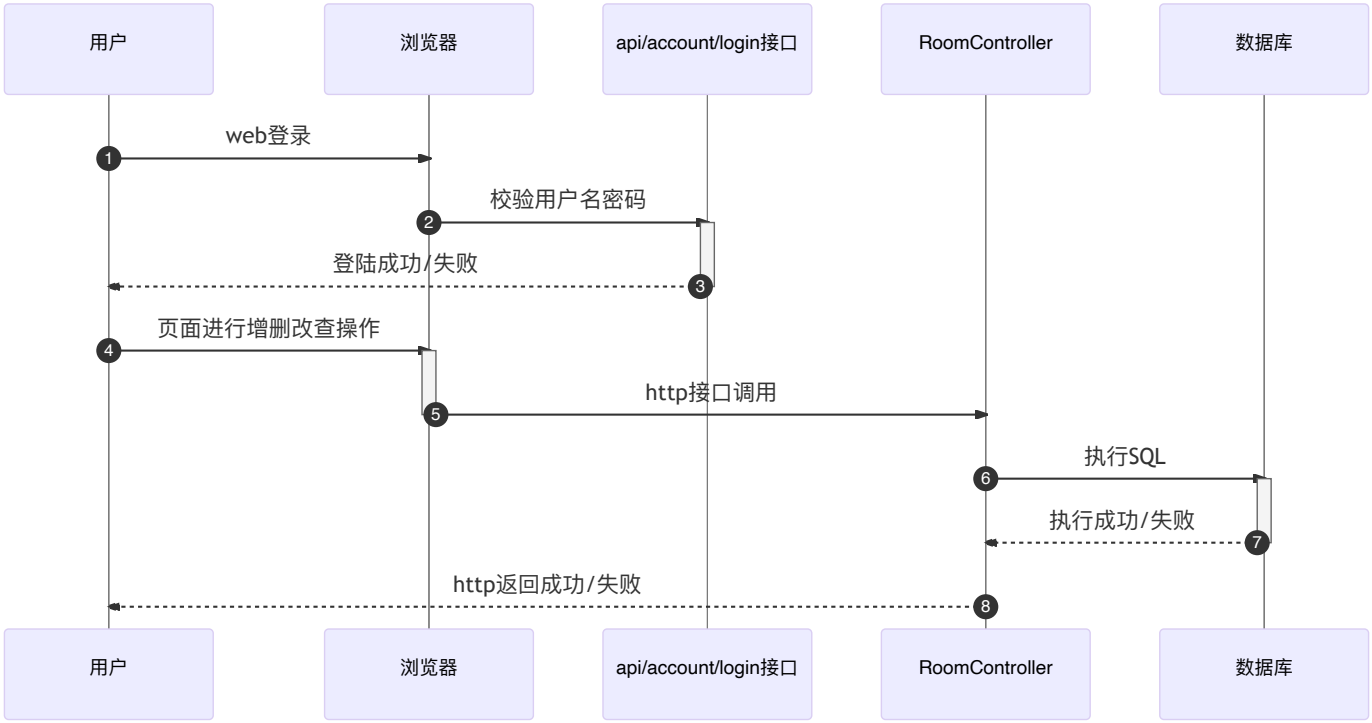
前端跟后端之间的调用是通过http接口来实现的，这里的http指的是http协议

红色部分是数据库，最终的操作都是对数据库的增删改查操作



时序图

详细介绍用户从【登录系统】到【会议室增删改查操作】这个过程中系统运行时序图，每一步都标了顺序。



增删改查功能设计

前面跟大家详细介绍了后端设计和前端设计，以及前后端之间是通过http协议进行交互的逻辑也跟大家讲解过，相信大家看完学长前面的讲解后，对这个模块已经有了一个大致的认识，那本节就是打铁趁热的一个模块，通过画图的方式，把前后端整个的调用流程给大家串起来。

按钮在Room.vue文件中，可以通过中文关键字去搜索，找到具体的位置。

具体调用路径如下：（**部分方法名称可能存在细微的差异，这个大家具体看代码，以代码中的方法名为准**）

讲解调用路径时学长会按照**文件名**+'^'+**方法名**的方式讲解，这样大家就可以很快的在项目中找到相关的代码了

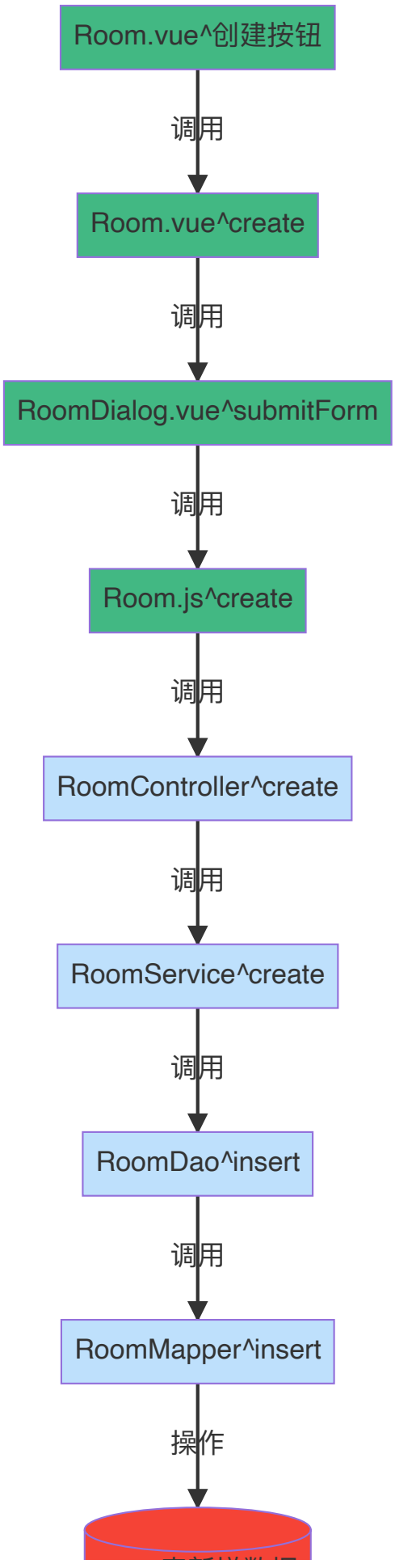
为了让大家更好地区分前后端代码的职责，学长通过不同的颜色来区分前后端代码，让大家一目了然！

绿色部分是前端代码文件

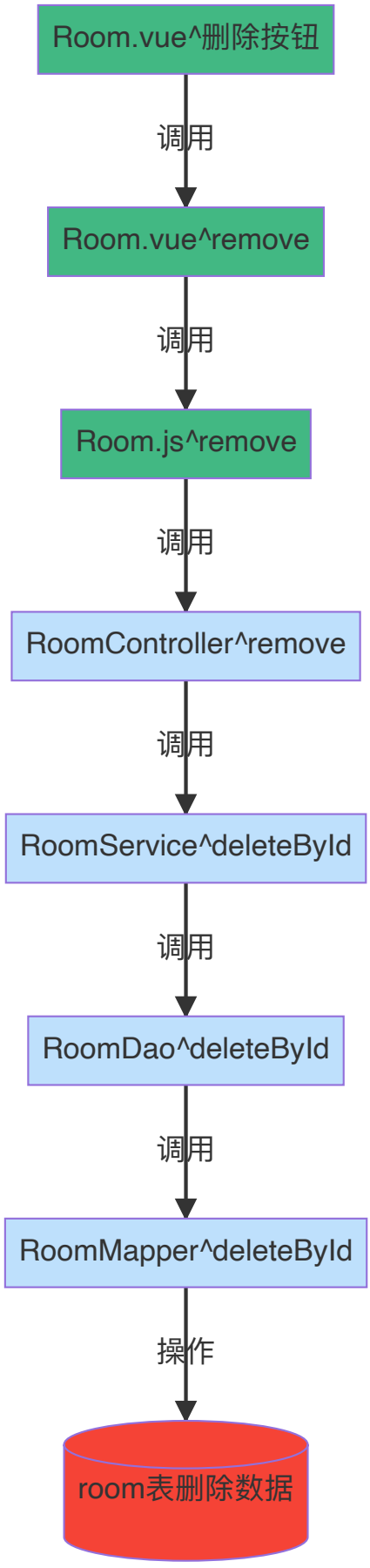
蓝色部分是后端代码文件

红色部分是数据库对应的表名

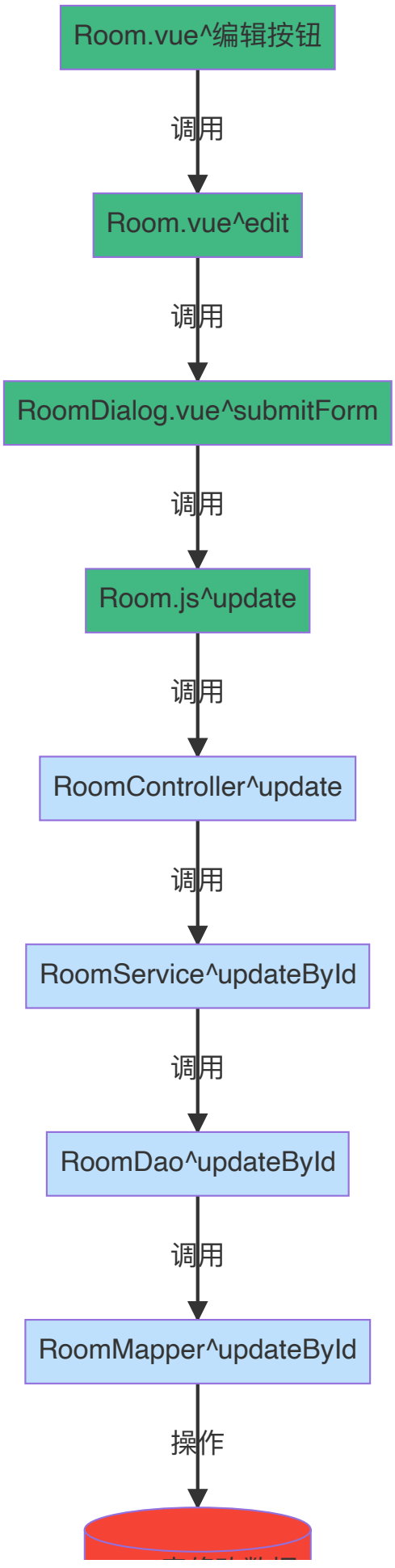
新增流程


Room.vue^创建按钮
调用
Room.vue^create
调用
RoomDialog.vue^submitForm
调用
Room.js^create
调用
RoomController^create
调用
RoomService^create
调用
RoomDao^insert
调用
RoomMapper^insert
操作

删除流程

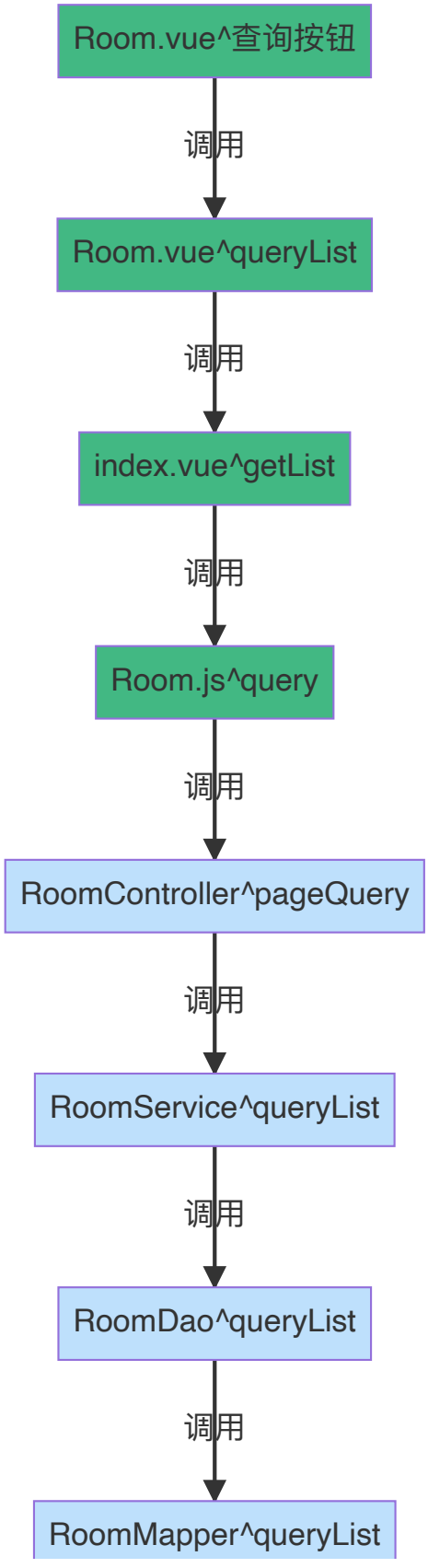

Room.vue^删除按钮
调用
Room.vue^remove
调用
Room.js^remove
调用
RoomController^remove
调用
RoomService^deleteById
调用
RoomDao^deleteById
调用
RoomMapper^deleteById
操作
room表删除数据

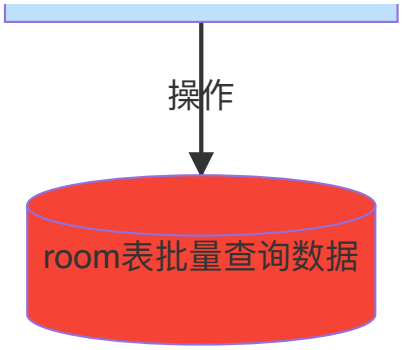
修改流程


Room.vue^编辑按钮
调用
Room.vue^edit
调用
RoomDialog.vue^submitForm
调用
Room.js^update
调用
RoomController^update
调用
RoomService^updateById
调用
RoomDao^updateById
调用
RoomMapper^updateById
操作


room表修改数据

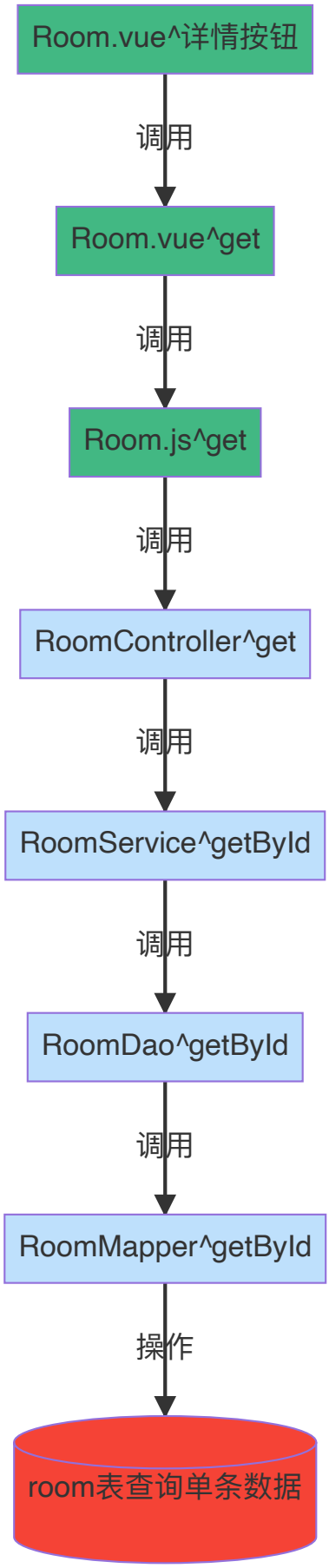
批量查询流程


Room.vue^查询按钮
调用
Room.vue^queryList
调用
index.vue^getList
调用
Room.js^query
调用
RoomController^pageQuery
调用
RoomService^queryList
调用
RoomDao^queryList
调用
RoomMapper^queryList

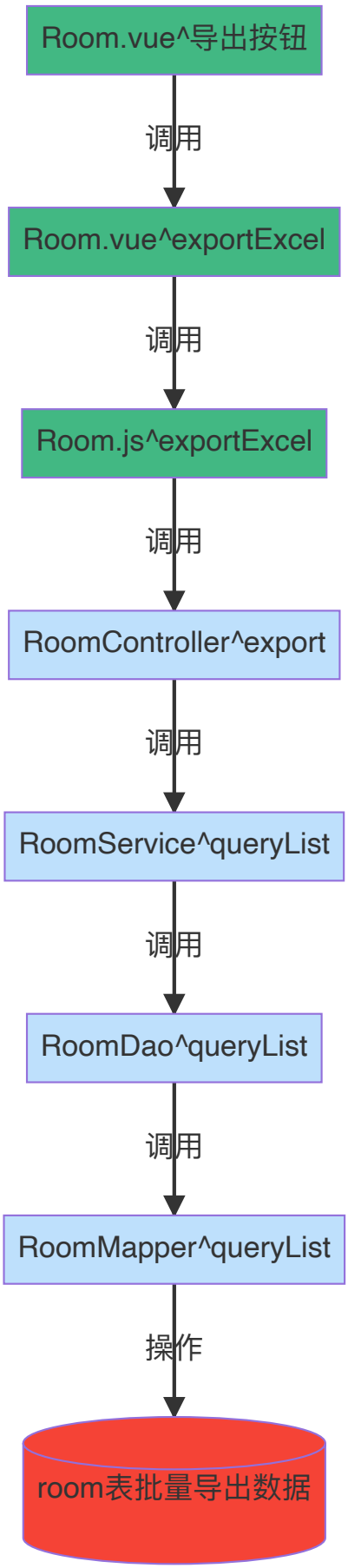


单条查询流程

这个功能在页面不一定有，但是学长先把流程给大家画出来。


Room.vue^详情按钮
调用
Room.vue^get
调用
Room.js^get
调用
RoomController^get
调用
RoomService^getById
调用
RoomDao^getById
调用
RoomMapper^getById
操作
room表查询单条数据

导出流程


Room.vue^导出按钮
调用
Room.vue^exportExcel
调用
Room.js^exportExcel
调用
RoomController^export
调用
RoomService^queryList
调用
RoomDao^queryList
调用
RoomMapper^queryList
操作
room表批量导出数据

【会议管理】详细设计

表结构设计

```
CREATE TABLE `meeting` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '主键',
  `name` varchar(255) NOT NULL COMMENT '会议名称',
  `content` text NOT NULL COMMENT '内容',
  `room_id` bigint NOT NULL COMMENT '会议室',
  `account_ids` varchar(512) NOT NULL COMMENT '人员名单',
  `create_time` bigint NOT NULL COMMENT '创建时间',
  `update_time` bigint DEFAULT NULL COMMENT '修改时间',
  PRIMARY KEY (`id`)
) ENGINE=InnoDB  DEFAULT CHARSET=utf8mb3 COMMENT='会议'
```

后端代码设计

后端代码都放在api目录下，本节将详细介绍【会议管理】的后端代码设计

DO层（Domain Object）

数据领域对象，对象属性跟数据库字段一一对应。

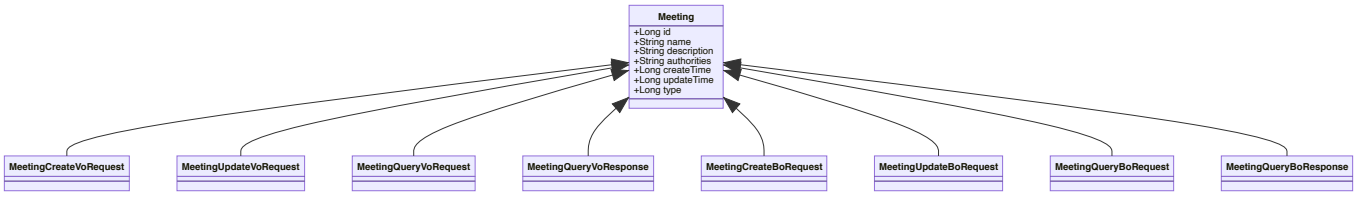
数据库表名称：meeting

类型：Java类

类名：MeetingDo.java

代码位置：domain/src/main/java/com/senior/domain/model/MeetingDo.java

类图如下：



Mapper层（mapper.xml）

Mapper.xml主要是写一些SQL语句，最常见的就是增删改查，这些SQL语句都是写在Mapper.xml里面的

数据库表名称：meeting

类型：xml文件

类名：MeetingMapper.xml

代码位置：service/base/src/main/resources/mybatis/mappers/MeetingMapper.xml

DAO层（Data Access Object）

持久层操作对象，结合MeetingDo对象，对meeting表的数据进行增删改查操作。

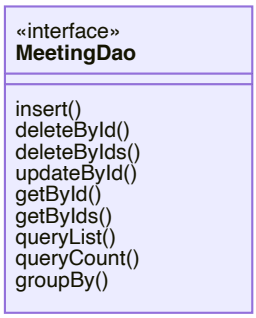
数据库表名称：meeting

类型：Java接口

类名：MeetingDao.java

代码位置：service/base/src/main/java/com/senior/dao/MeetingDao.java

类图如下：



Service层（业务逻辑层）

业务逻辑层，负责对数据的处理。如果没有逻辑处理任务，此层只做单纯的数据传递作用，而后又到了DAO层。

数据库表名称：meeting

类型：Java接口

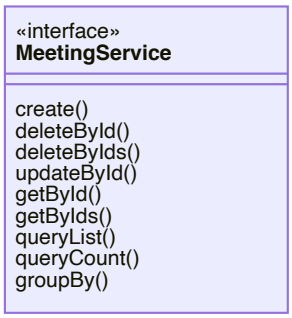
类名：MeetingService.java

代码位置：service/business/src/main/java/com/senior/service/MeetingService.java

接口实现类：MeetingServiceImpl.java

接口实现类代码位置：service/business/src/main/java/com/senior/service/impl/MeetingServiceImpl.java

类图如下：



Controller层（控制层）

控制层，主要是提供一些接口（增删改查接口）。本项目是前后端分离的项目，前端跟后端是基于http协议实现数据传输的。接口的代码其实就放在Controller这一层，这一层主要负责提供http接口，然后就是组装前端传输的数据，再将组装的数据调用Service层。

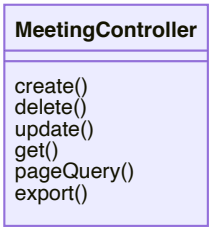
数据库表名称：meeting

类型：Java类

类名：MeetingController.java

代码位置：console/api/src/main/java/com/senior/console/api/controller/MeetingController.java

类图如下：



前端代码设计

前端是基于Vue开发的，框架的话是基于饿了么开源的element框架。代码都放在console/web目录下，本节将详细介绍【会议管理】的前端代码设计

列表页面

代码路径：console/web/src/page/meeting/Meeting.vue

代码阅读技巧：方法都放在methods里面，这里面还有很多关键字，比如created、computed、watch、methods、data等，这些都是Vue的语法，大家如果看不懂，需要去学习下Vue的基础语法，还是比较简单的。

主要功能：列表页的功能主要是展示【会议】的数据，包括页面上的一些按钮，比如新增、修改、详情、删除、导出等

详情&新增页面

代码路径：console/web/src/page/meeting/MeetingDialog.vue

代码阅读技巧：跟列表页一样。

主要功能：提供【会议】新增和查看【会议】详情的功能

API定义

代码路径：console/web/src/api/Meeting.js

主要功能：本项目是前后端分离的项目，前面我们说过好多次了，前后端直接的数据交互是基于http协议来实现的。那么API这一块的代码就是定义的后端接口的地址，用户在页面上通过填写各种数据，或者点击某个按钮时，此时将组装好的数据通过API这层封装好的http接口，就可以把数据传输给后端的Controller层了，里面的接口大致上就是【会议管理】的增删改查操作。

流程图

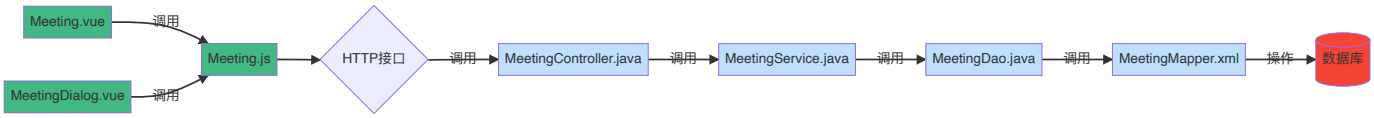
代码调用链

绿色部分是前端代码文件

蓝色部分是后端代码文件

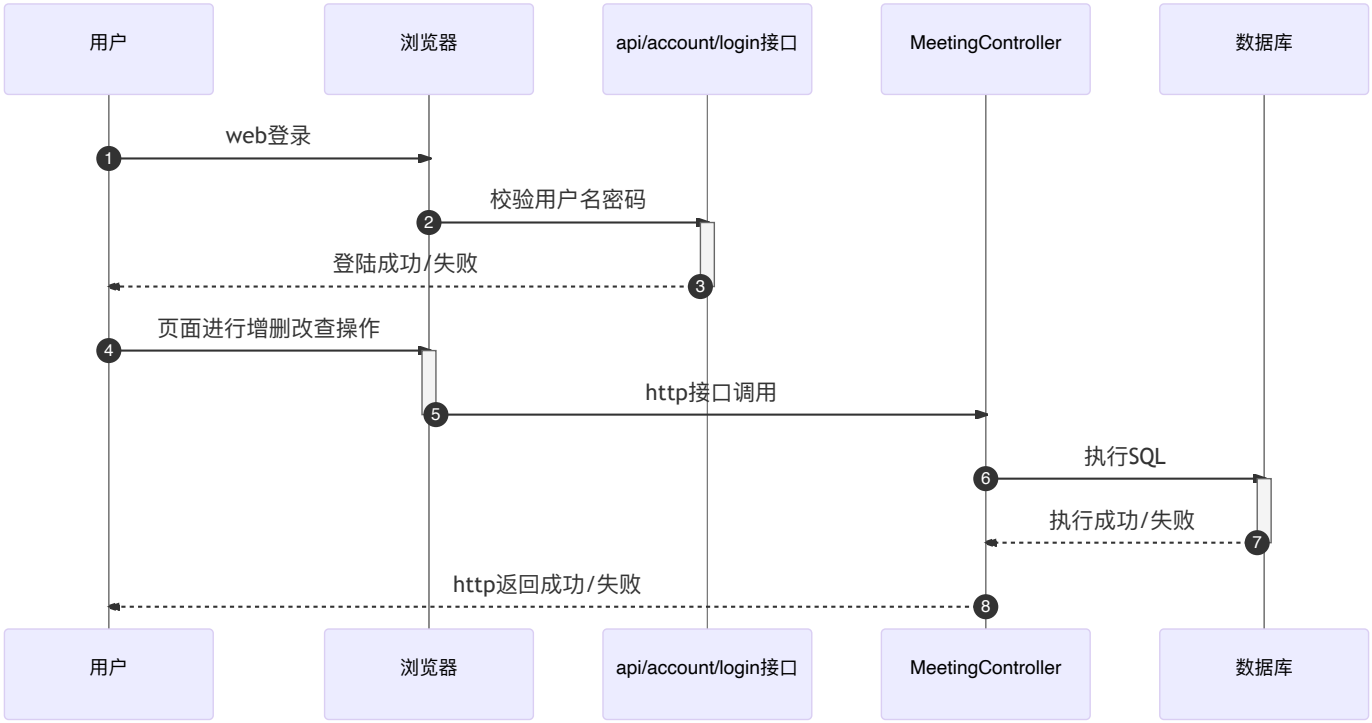
前端跟后端之间的调用是通过http接口来实现的，这里的http指的是http协议

红色部分是数据库，最终的操作都是对数据库的增删改查操作



时序图

详细介绍用户从【登录系统】到【会议增删改查操作】这个过程中系统运行时序图，每一步都标了顺序。



增删改查功能设计

前面跟大家详细介绍了后端设计和前端设计，以及前后端之间是通过http协议进行交互的逻辑也跟大家讲解过，相信大家看完学长前面的讲解后，对这个模块已经有了一个大致的认识，那本节就是打铁趁热的一个模块，通过画图的方式，把前后端整个的调用流程给大家串起来。

按钮在Meeting.vue文件中，可以通过中文关键字去搜索，找到具体的位置。

具体调用路径如下：（**部分方法名称可能存在细微的差异，这个大家具体看代码，以代码中的方法名为准**）

讲解调用路径时学长会按照**文件名**+'^'+**方法名**的方式讲解，这样大家就可以很快的在项目中找到相关的代码了

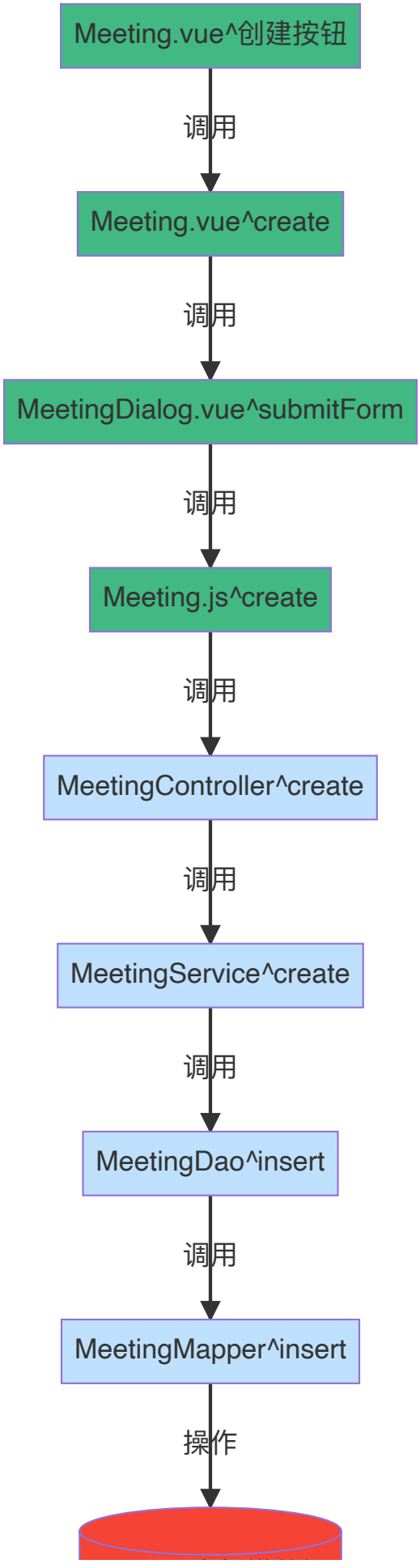
为了让大家更好地区分前后端代码的职责，学长通过不同的颜色来区分前后端代码，让大家一目了然！

绿色部分是前端代码文件

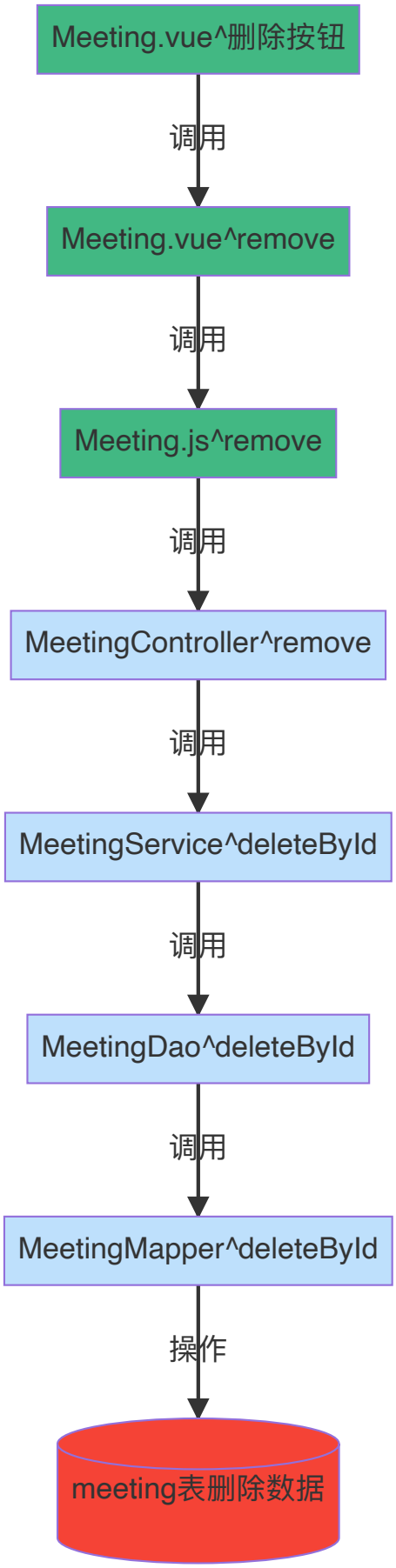
蓝色部分是后端代码文件

红色部分是数据库对应的表名

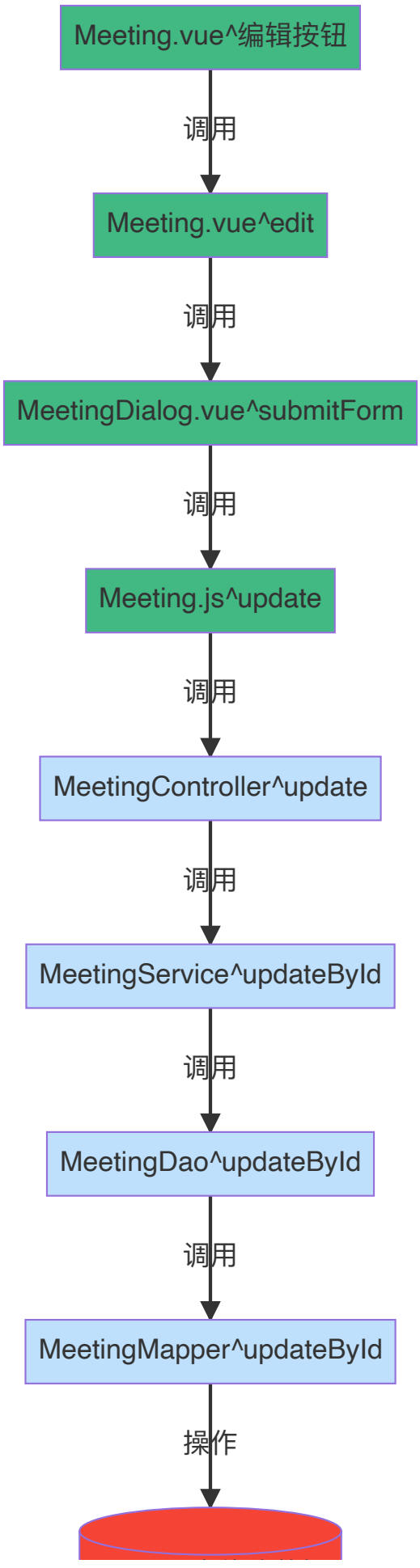
新增流程


Meeting.vue^创建按钮
调用
Meeting.vue^create
调用
MeetingDialog.vue^submitForm
调用
Meeting.js^create
调用
MeetingController^create
调用
MeetingService^create
调用
MeetingDao^insert
调用
MeetingMapper^insert
操作

删除流程

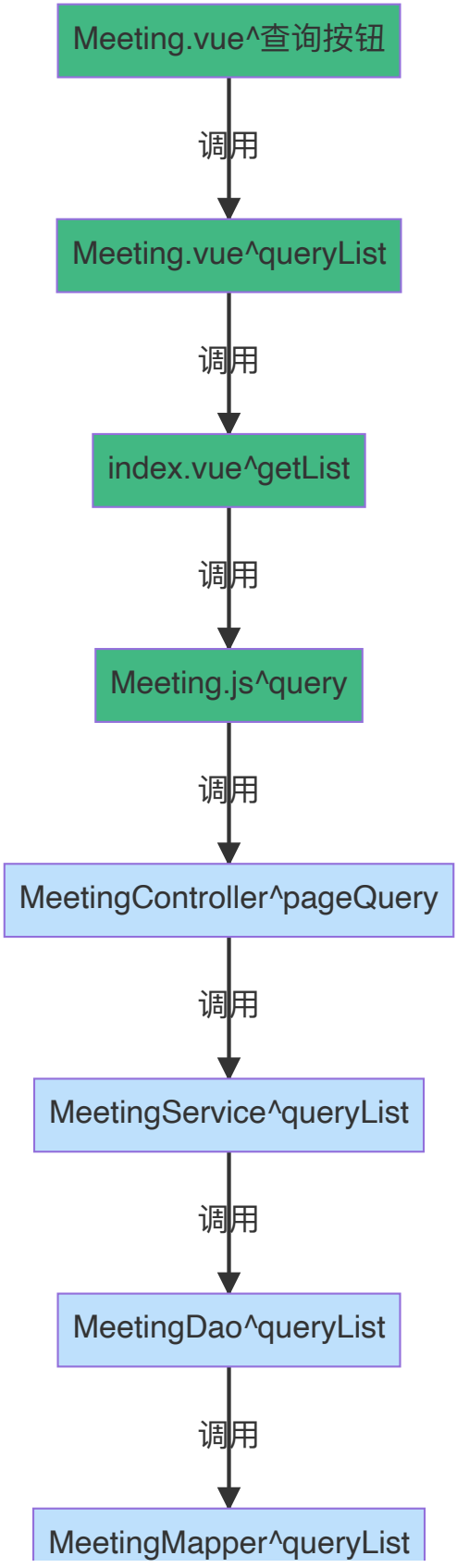

Meeting.vue^删除按钮
调用
Meeting.vue^remove
调用
Meeting.js^remove
调用
MeetingController^remove
调用
MeetingService^deleteById
调用
MeetingDao^deleteById
调用
MeetingMapper^deleteById
操作
meeting表删除数据

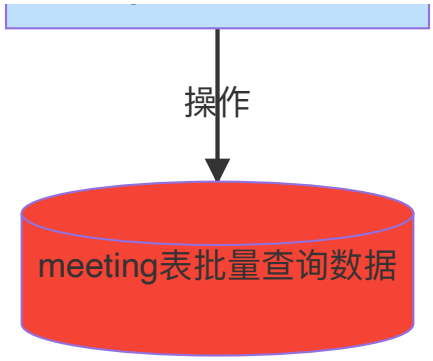
修改流程


Meeting.vue^编辑按钮
调用
Meeting.vue^edit
调用
MeetingDialog.vue^submitForm
调用
Meeting.js^update
调用
MeetingController^update
调用
MeetingService^updateById
调用
MeetingDao^updateById
调用
MeetingMapper^updateById
操作


meeting表修改数据

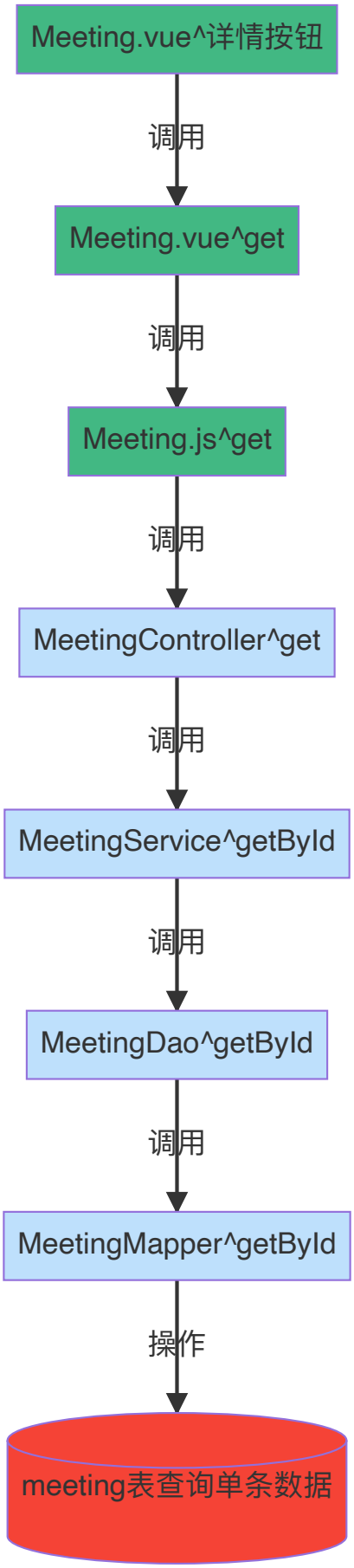
批量查询流程


Meeting.vue^查询按钮
调用
Meeting.vue^queryList
调用
index.vue^getList
调用
Meeting.js^query
调用
MeetingController^pageQuery
调用
MeetingService^queryList
调用
MeetingDao^queryList
调用
MeetingMapper^queryList

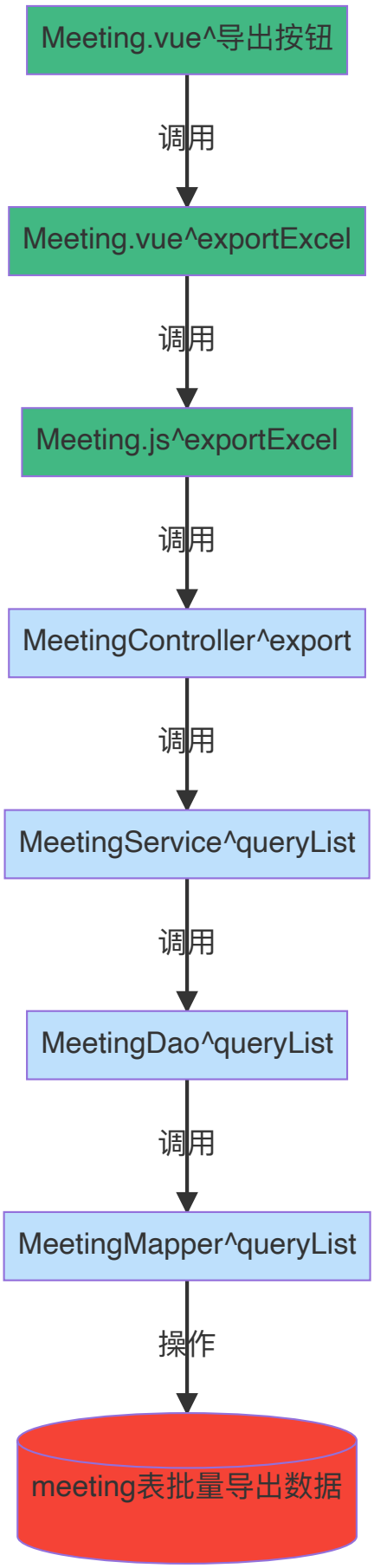


单条查询流程

这个功能在页面不一定有，但是学长先把流程给大家画出来。


Meeting.vue^详情按钮
调用
Meeting.vue^get
调用
Meeting.js^get
调用
MeetingController^get
调用
MeetingService^getById
调用
MeetingDao^getById
调用
MeetingMapper^getById
操作
meeting表查询单条数据

导出流程


Meeting.vue^导出按钮
调用
Meeting.vue^exportExcel
调用
Meeting.js^exportExcel
调用
MeetingController^export
调用
MeetingService^queryList
调用
MeetingDao^queryList
调用
MeetingMapper^queryList
操作
meeting表批量导出数据