

SpringBoot+Vue前后端分离

前后端分离

前后端分离就是将一个应用的前端代码和后端代码分开写，为什么要这样做？

如果不使用前后端分离的方式，会有哪些问题？

传统的Java Web开发中，前端使用JSP开发，JSP不是由后端开发者来独立完成的。

前端—>HTML静态页面—>后端—>JSP

这种开发方式效率极低，可以使用前后端分离的方式进行开发，就可以完美地解决这一问题。

前端只需要独立编写客户端代码，后端也需要独立编写服务端代码提供数据接口即可。

前端通过Ajax请求来访问后端的数据接口，将Model展示到View中即可。

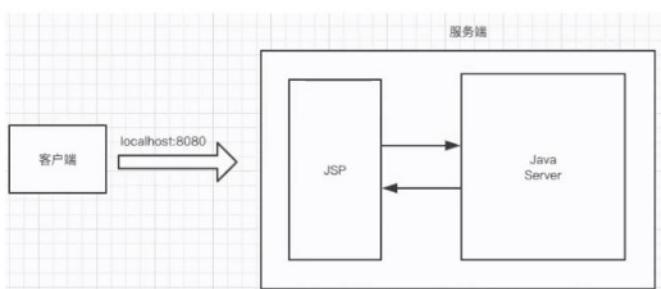
前后端开发者只需要提前约定好接口文档（URL、参数、数据类型...），然后分别独立开发即可，前端可以造假数据进行测试，完全不需要依赖于后端，最后完成前后端集成即可，真正实现了前后端应用的解耦合

单体—>前端应用 + 后端应用

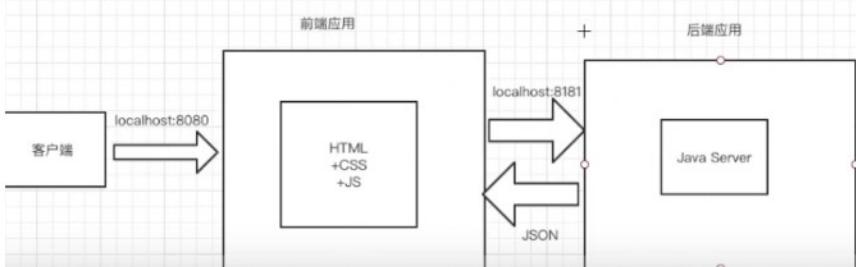
前端应用：负责数据展示和用户交互。

后端应用：负责提供数据处理接口。

前端HTML—>Ajax—>RESTful后端数据接口。



前后端分离的结构（相当于有两个服务端）



前后端分离就是将一个单体应用拆分成两个独立的应用，前端应用和后端应用以JSON格式进行数据交互。

实现技术

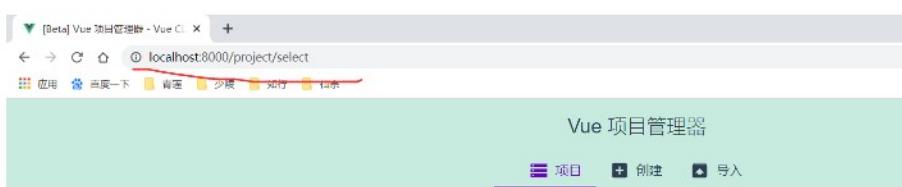
Spring Boot + Vue

使用Spring Boot进行后端应用开发，使用Vue进行前端应用开发。

前端 使用 Vue项目管理器创建 Vue项目

dos命令窗口输入 vue ui

```
C:\Users\白麒麟-HAO>vue ui
◆ Starting GUI...
◆ Ready on http://localhost:8000
```



点击创建然后选择创建项目在哪个文件夹下，输入项目名称，手动配置，然后选择router、vuex，去掉代码校验linter/formatter，点击创建项目成功后

项目仪表盘

欢迎来到新项目！

可以下面步骤测试下是否能够运行

任务

serve 编译和热更新 (用于开发环境)

点击输出会看到有两个地址，在浏览器中打开即可看到启动后的项目

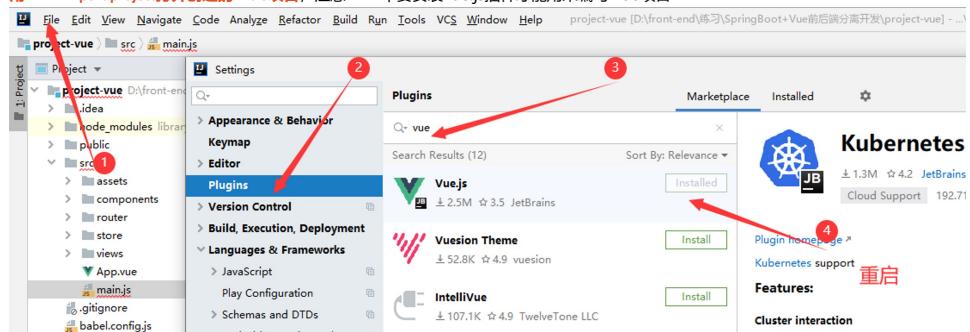
运行成功

如果需要关掉Vue项目管理器，只需在dos命令窗口下按 Ctrl + C 并输入 Y 回车即可

```
C:\Users\白麟飘尘HAO>vue ui
⚡ Starting GUI...
⚡ Ready on http://localhost:8000
终止批处理操作吗(Y/N)? Y
C:\Users\白麟飘尘HAO>
```



用IDEA import project 打开创建的 Vue项目，注意IDEA中要安装vue.js插件才能用来编写 vue项目



打开 js 文件发现报错，因为IDEA中JS的版本较低，改成ES6即可

```

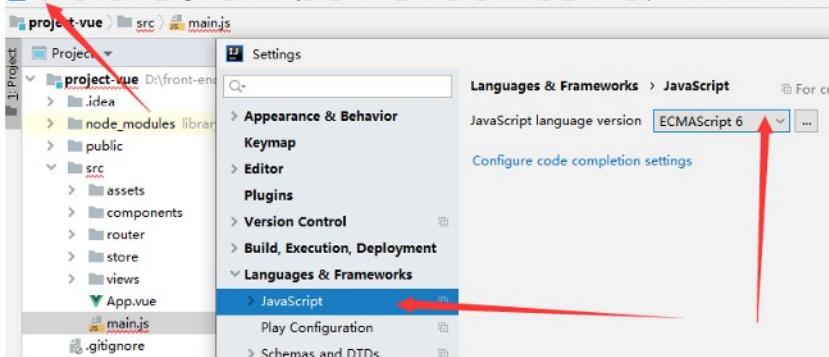
import Vue from 'vue'
import App from './App.vue'
import router from './router'
import store from './store'

Vue.config.productionTip = false

new Vue({
  router,
  store,
  render: h => h(App)
}).$mount('#app')

```

JavaScript改成6之后就不再报错



IDEA中启动 Vue应用，其内嵌了一个服务器

```

Terminal: Local
Microsoft Windows [版本 10.0.18362.1139]
(c) 2019 Microsoft Corporation. 保留所有权利。

D:\front-end\练习\SpringBoot+Vue前后端分离开发\project-vue>npm run serve

> project-vue@0.1.0 serve D:\front-end\练习\SpringBoot+Vue前后端分离开发\project-vue
> vue-cli-service serve

[ 100%] Starting development server...
10% building 0/1 modules 1 active ..._modules\webpack-dev-server\client\index.js?http://192.
10% building 1/2 modules 1 active ..._modules\webpack\hot\dev-server.js D:\front-end\练习\Sp
10% building 2/5 modules 3 active ...modules\cache-loader\dist\cjs.js?ref--12-0ID:\front-end

```

怎么停止启动的项目，按下 Ctrl + C 终止即可

```

Note that the development build is not optimized.
To create a production build, run npm run build.

终止批处理操作吗(Y/N)? y

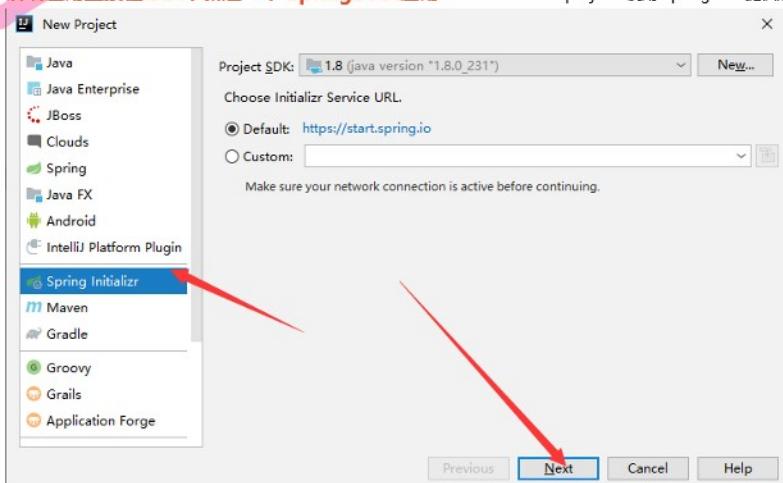
D:\front-end\练习\SpringBoot+Vue前后端分离开发\project-vue>

```

补充： IDEA进入一个项目后想 import另一个项目，可以file --> close project回到初次打开IDEA时界面再选择 import project (不要选择 Open project)

前端应用搭建好之后，需要与后端应用进行交互，接下来启动后端应用

后端应用直接在IDEA中新建一个 SpringBoot应用 file --> new --> project 使用 Springboot提供的一个模板



New Project

Project Metadata

Group: com.baichen
Artifact: project-springboot
Type: Maven Project (Generate a Maven based project archive.)
Language: Java
Packaging: Jar
Java Version: 8
Version: 0.0.1-SNAPSHOT
Name: project-springboot
Description: Demo project for Spring Boot
Package: com.baichen.projectspringboot

Previous Next Cancel Help

next

New Project

Dependencies

Developer Tools
Web
Template Engines
Security
SQL
NoSQL
Messaging
I/O
Ops
Observability
Testing
Spring Cloud
Spring Cloud Security
Spring Cloud Tools
Spring Cloud Config
Spring Cloud Discovery
Spring Cloud Routing
Spring Cloud Circuit Breaker
Spring Cloud Tracing
Spring Cloud Messaging

选择SpringBoot版本 Spring Boot 2.3.4

选上这四个Dependencies

Liquibase Migration
Flyway Migration
JOOQ Access Layer
IBM DB2 Driver
Apache Derby Database
H2 Database
HyperSQL Database
MS SQL Server Driver
 MySQL Driver
Oracle Driver
PostgreSQL Driver

MySQL Driver
MySQL JDBC and R2DBC driver.

Selected Dependencies

Developer Tools
Lombok
Web
Spring Web
SQL
Spring Data JPA
MySQL Driver

Previous Next Cancel Help

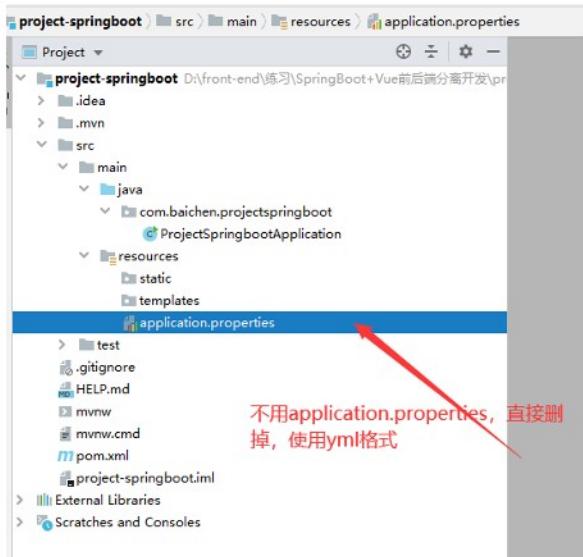
Next 直接选择 Finish

New Project

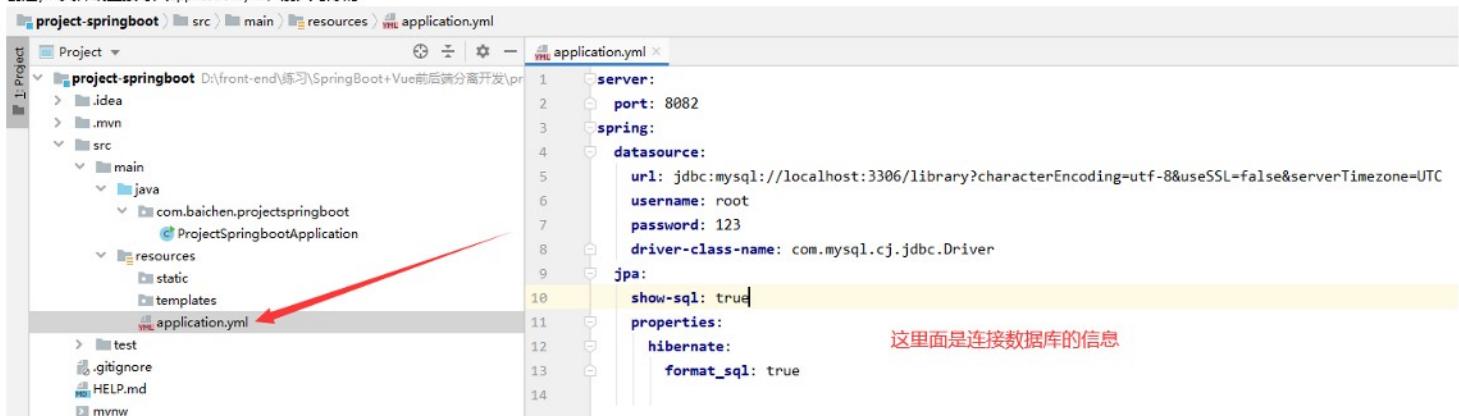
Project name: project-springboot
Project location: D:\front-end\练习\SpringBoot+Vue前后端分离开发\project-springboot

More Settings

Previous Finish Cancel Help



创建yml文件或直接导入application.yml, 别人写好的

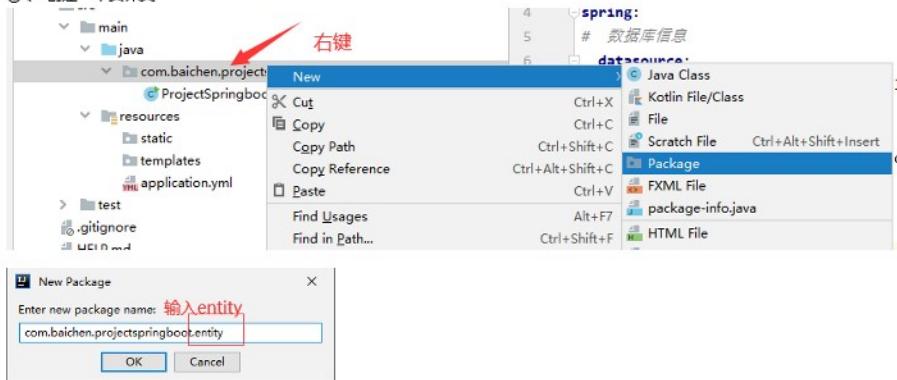


创建一个数据库 library, 创建表 book

book @ library (localhost_3306) - 表								
	对象	视图	函数	用户	其它	查询	备份	自动运行
200								
localhost								
localhost_3306								
book								
datacube								
food								
hotel								
information_schema								
library								
book								
1 书名	1 书名	author	publish	pages	price	bookcaseid	abled	
2 作者	2 作者	author	publish	pages	price	bookcaseid	abled	
3 页数	3 页数	pages	price	bookcaseid	abled			
4 价格	4 价格	price	bookcaseid	abled				
5 书名	5 书名	bookname	author	pages	price	bookcaseid	abled	
6 作者	6 作者	author	bookname	pages	price	bookcaseid	abled	
7 页数	7 页数	pages	bookname	author	price	bookcaseid	abled	
8 价格	8 价格	price	bookname	author	pages	bookcaseid	abled	
9 我们仨	9 我们仨	bookname	author	pages	price	bookcaseid	abled	
10 情亲	10 情亲	bookname	author	pages	price	bookcaseid	abled	
11 水浒传	11 水浒传	bookname	author	pages	price	bookcaseid	abled	
12 三国演义	12 三国演义	bookname	author	pages	price	bookcaseid	abled	
13 红楼梦	13 红楼梦	bookname	author	pages	price	bookcaseid	abled	
14 西游记	14 西游记	bookname	author	pages	price	bookcaseid	abled	

怎么连接数据库, 使用 SpringDataJPA

①、创建一个实体类



②、创建 Book这个类用来跟表进行一个绑定



```

project-springboot > src > main > java > com > baichen > projectspringboot > entity > Book.java
Project application.yml Book.java
1 package com.baichen.projects.springboot.entity;
2
3 public class Book {
4 }
5

```

③、怎么跟数据表绑定起来，加 @Entity注解，Book类就可以跟数据表进行一个绑定，根据表名和类名的映射关系

输入 @Entity 回车

```

3 import lombok.Data;
4
5 import javax.persistence.Entity;
6
7 @Entity
8 // 加@Data: Lombok的注解(先引入Lombok)，自动生成各种各样的Get和Set方法
9 @Data

```

④、接下来是属性的对应，实体类中的每一个对象表中数据映射起来

```

12 @Id
13     private Integer id;
14     private String name;
15     private String author;

```

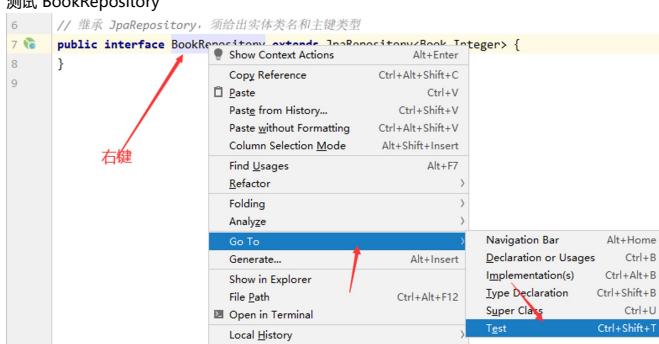
⑤、在 repository中创建一个叫 BookRepository的接口，此接口不用定义方法，直接继承 JpaRepository (继承 JpaRepository，须给出实体类名和主键类型)

```

project-springboot D:\front-end\练习\SpringBoot+Vu > src > main > java > com > baichen > projectspringboot > repository > BookRepository.java
Project application.yml Book.java BookRepository.java pom.xml
1 package com.baichen.projects.springboot.repository;
2
3 import com.baichen.projects.springboot.entity.Book;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 // 继承 JpaRepository，须给出实体类名和主键类型
7 public interface BookRepository extends JpaRepository<Book, Integer> {
8 }
9

```

测试 BookRepository



下一步直接点 OK就行，在test文件夹下会自动生成如下的一个类

```

project-springboot > src > test > java > com > baichen > projectspringboot > repository > BookRepositoryTest.java
Project application.yml Book.java BookRepository.java BookRepositoryTest.java ProjectSpringbootApplicationTests.java
1 package com.baichen.projects.springboot.repository;
2
3 import static org.junit.jupiter.api.Assertions.*;
4
5 class BookRepositoryTest {
6 }
7

```

测试类先添加一个 @SpringBootTest注解，在方法上加 @Test注解，点击方法左边绿色按钮 run或 debug测试

```

3 import org.junit.jupiter.api.Test;
4 import org.springframework.beans.factory.annotation.Autowired;
5 import org.springframework.boot.test.context.SpringBootTest;
6
7 // import static org.junit.jupiter.api.Assertions.*;
8
9 @SpringBootTest
10 class BookRepositoryTest {
11     // 测试 BookRepository, 所以先将其注入进来加注解自动注入
12     @Autowired
13     private BookRepository bookRepository;
14
15     @Test
16     void findAll(){
17         System.out.println(bookRepository.findAll());
18     }

```

测试结果

表示测试成功了

```

Run: BookRepositoryTest.findAll ×
Tests passed: 1 of 1 test - 245 ms
2020-10-21 10:34:20.645 INFO 10500 --- [main] c.b.p.repository.BookRepositoryTest
Hibernate:
    select
        book0_.id as id1_0_,
        book0_.author as author2_0_,
        book0_.name as name3_0_
    from
        book book0_
[Book(id=1, name=解忧杂货店, author=东野圭吾), Book(id=2, name=追风筝的人, author=卡勒德·胡赛尼), Book(id=3, name=百年孤独, author=加西亚·马尔克斯)]

```

这里输出了 sql语句

在实体类 Book 中只映射了 id、name、author，所以这里只输出了这三种数据

测试打印的结果中有 sql语句，是因为在 application.yml中有如下配置，注释掉这部分内容则不会输出 sql语句

```

jpa:
# 打印 SQL
show-sql: true
properties:
  hibernate:
    # 格式化 SQL, 会自动换行
    format_sql: true

```

⑥、测试没问题后，目前还是后端内部映射，接下来方法放入 Controller中，使得 Web可以调用，

```

package com.baichen.projectspringboot.controller;

import com.baichen.projectspringboot.entity.Book;
import com.baichen.projectspringboot.repository.BookRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

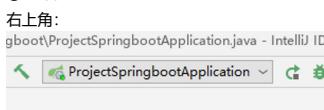
import java.util.List;

@RestController
@RequestMapping("/book")
public class BookHandler {
    @Autowired
    private BookRepository bookRepository;

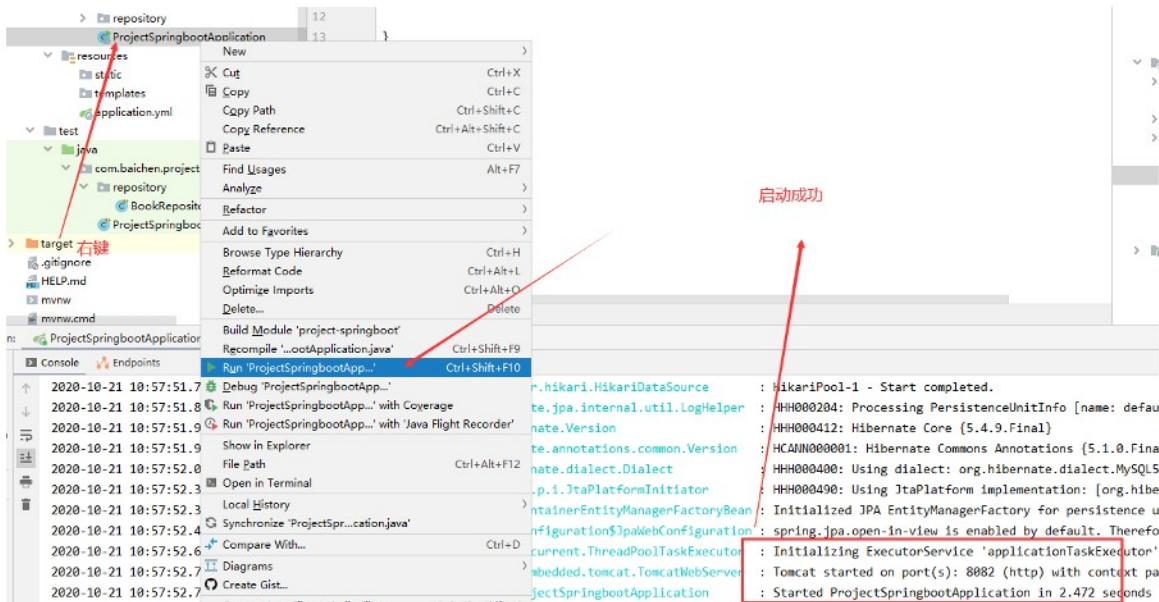
    // 提供一个方法，返回 Book集合，注意引入自己写的
    @GetMapping("/all")
    public List<Book> findAll(){
        return bookRepository.findAll();
    }
}

```

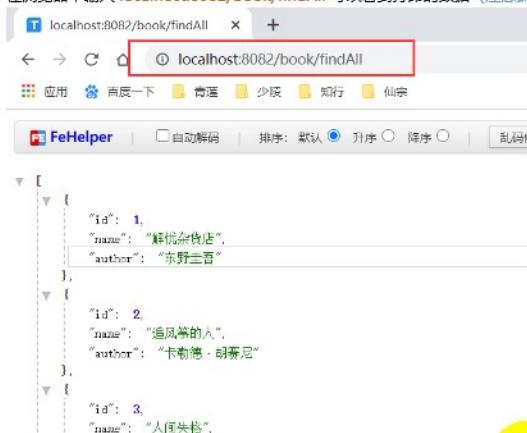
⑦、启动后端项目



或如下方式：



在浏览器中输入 `localhost:8082/book/findAll` 可以看到打印的数据 (注意端口号不是 8080, 为了防止冲突, 自己设置过了)



前后端应用都弄好了之后, 接下来只要实现前后端数据交互

前后端通过 Ajax 实现对接, 后端写好之后启动起来放在那不用管

新建一个组件 Book.vue, 并使用 mock 数据, 接下来用后端数据

```

App.vue x index.js x Book.vue x
1 <template>
2   <div>
3     <table>
4       <thead>
5         <tr>
6           <th>编号</th>
7           <th>书名</th>
8           <th>作者</th>
9         </tr>
10        <tr v-for="item in bookList">
11          <td>{{item.id}}</td>
12          <td>{{item.name}}</td>
13          <td>{{item.author}}</td>
14        </tr>
15      </table>
16    </div>
17  </template>
18
19  <script>
20    export default {
21      name: "Book",
22      data(){
23        return{
24          // mock 数据
25          bookList:[
26            {
27              id:1,
28              name:'毛泽东选集',
29              author:'毛泽东'
30            },
31          ]
32        }
33      }
34    }
35  </script>

```

vue 里面请求 ajax 用 axios。首先安装, 使用命令 `vue add axios`

D:\front-end\练习\SpringBoot+Vue前后端分离开发\project-vue vue add axios

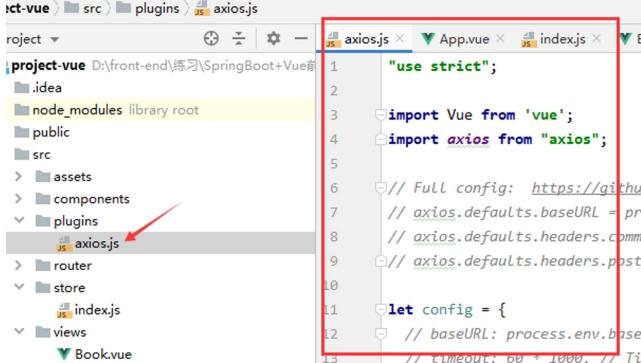
Installing vue-cli-plugin-axios...

[.....] / rollbackFailedOptional: verb npm-session 0ffb25578cbaefb0

54 packages are looking for funding
run `npm fund` for details

↳ Running completion hooks...

✓ Successfully invoked generator for plugin: vue-cli-plugin-axios



```
use strict";
import Vue from 'vue';
import axios from "axios";
// Full config: https://github.com/axios/axios#full-config
// axios.defaults.baseURL = process.env.baseURL || '';
// axios.defaults.headers.common['Authorization'] = `Bearer ${process.env.token}`;
// axios.defaults.headers.post['Content-Type'] = 'application/json';
let config = {
  // baseURL: process.env.baseURL || '',
  // timeout: 60 * 1000, // Timeout
  // headers: { ... }
};
export default config;
```

在页面初始化的过程中获取数据

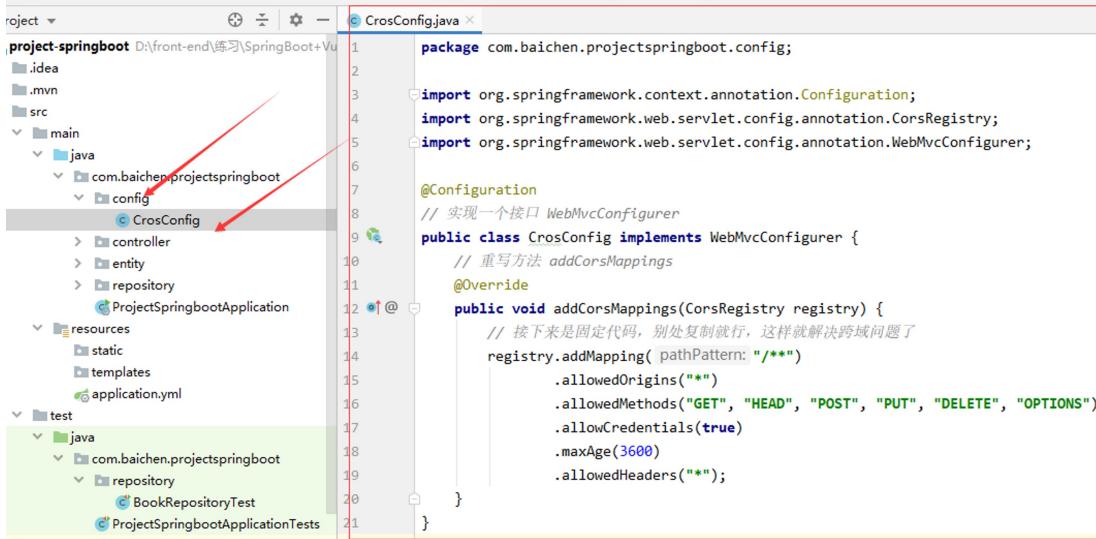


```
created() {
  axios.get('http://localhost:8082/book/findAll').then( res => {
    console.log(res)
  })
}
```

此时控制台报错，跨域问题，因为前端地址是 <http://localhost:8080/>，请求后端的地址 <http://localhost:8082/book/findAll>，端口号不同，跨服务，需要跨域

[HMR] Waiting for update signal from WDS...
Access to XMLHttpRequest at 'http://localhost:8082/book/findAll' from origin 'http://localhost:8080' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
GET http://localhost:8082/book/findAll net::ERR_FAILED
Uncaught (in promise) Error: Network Error
at createError (createError.js:22:83:16)
at XMLHttpRequest.handleError (xhr.js:50d:69)

跨域问题前后端都可以解决，这里用 SpringBoot 后端解决：添加一个配置类



```
package com.baichen.projectspringboot.config;

import org.springframework.context.annotation.Configuration;
import org.springframework.web.servlet.config.annotation.CorsRegistry;
import org.springframework.web.servlet.config.annotation.WebMvcConfigurer;

@Configuration
// 实现一个接口 WebMvcConfigurer
public class CrosConfig implements WebMvcConfigurer {
    // 重写方法 addCorsMappings
    @Override
    public void addCorsMappings(CorsRegistry registry) {
        // 接下来是固定代码，别处复制就行，这样就解决跨域问题了
        registry.addMapping( pathPattern: "/**" )
            .allowedOrigins("*")
            .allowedMethods("GET", "HEAD", "POST", "PUT", "DELETE", "OPTIONS")
            .allowCredentials(true)
            .maxAge(3600)
            .allowedHeaders("*");
    }
}
```

重新启动后端，前端刷新一下，可以看到控制台成功获取数据



```
[HMR] Waiting for update signal from WDS...
[object Object]
  ↗ [data: Array(16), status: 200, statusText: "", headers: {}, config: {}]
  ↗ config: {transformRequest: {}, transformResponse: {}, timeout: 0, xsrfCookieName: "XSRF-TOKEN", adapter: f, ...}
  ↗ data: (16) [{}]
  ↗ headers: {content-type: "application/json"}
  ↗ request: XMLHttpRequest {readyState: 4, timeout: 0, withCredentials: false, upload: XMLHttpRequestUpload, onreadystatechange: f, ...}
  ↗ status: 200
  ↗ statusText: ""
  ↗ __proto__: Object
```

把调用接口获取到的数据赋值，刷新

```

48   created() {
49     axios.get('http://localhost:8082/book/findAll').then( res => {
50       // console.log(res)
51       this.bookList = res.data;
52       // 因为使用的箭头函数，这里的 this指的就是当前的 vue对象
53     })
54   }

```

此时页面中就是数据库中的真实数据，而不是在前端代码中 mock的数据了



编号	书名	作者
1	解忧杂货店	东野圭吾
2	追风筝的人	卡勒德·胡赛尼
3	人间失格	太宰治
4	这就是二十四节气	高春香
5	白夜行	东野圭吾
6	挪威的森林	村上春树
7	暖暖心绘本	米拉弗特华
8	天才江左疯子在右	高铭
9	我们仨	杨绛
10	活着	余华
11	水浒传	施耐庵
12	三国演义	罗贯中
13	红楼梦	曹雪芹
14	西游记	吴承恩
15	测试测试	
16	Spring Boot	张三

使用 Element-ui，安装配置查看官网 <https://element.eleme.cn/#/zh-CN/component/installation>

或者另外一篇笔记。

ElementUI后台管理系统主要的标签：

```

el-container: 构建整个页面框架;
el-aside: 构建左侧菜单;
el-menu: 左侧菜单内容，常用属性:
  default-openeds: 设置默认展开的菜单，通过菜单的index值来关联;
  default-active: 默认选中的菜单，也是通过菜单的index值来关联，注意加单引号，多个要用数组
el-submenu: 可展开的菜单，常用属性:
  index: 菜单的下标，文本类型，不能是数值类型
  template: 用来对应（设置）el-submenu的菜单名;
  i: 设置图标，通过class属性设置;
el-menu-item: 菜单的子节点，不可再展开，常用属性:
  index: 菜单的下标，文本类型，不能是数值类型

```



像上面这种菜单，包括其子菜单，一般在路由里面去设置，这样方便设置权限以及每个菜单点击进去页面不同，动态加载

Vue router动态构建左侧菜单

· 导航 1

- 页面1
- 页面2

· 导航 2

- 页面3
- 页面4
- 页面5

```

src > router > index.js
App.vue <-- Home.vue <-- PageOne.vue <-- index.js <-- Book.vue <-->
D:\front-end练习\Sp... modules library root
  - sets
  - components
    - HelloWorld.vue
  - uugins
  - uter
  - ore
  - ewns
    - Book.vue
    - Home.vue
    - PageFive.vue
    - PageFour.vue
    - PageOne.vue
    - PageThree.vue
    - PageTwo.vue
  - op.vue
  - ain.js
  - nore
  - l.config.js
  - age.json
  - age-lock.json
  - ct-vue.iml
  - ME.md
  - Libraries
  - s and Consoles

1 import Vue from 'vue'
2 import VueRouter from 'vue-router'
3 import Home from "../views/Home";
4 // 引入组件之后通过 routes映射路由
5
6 Vue.use(VueRouter)
7
8 const routes = [
9   {
10     path: '/',
11     name: '导航 1',
12     component: Home,
13     children: [
14       {
15         path: '/one',
16         name: '页面 1',
17         component: () => import('../views/PageOne.vue')
18       },
19       {
20         path: '/two',
21         name: '页面 2',
22         component: () => import('../views/PageTwo.vue')
23       }
24     ],
25   },
26   {
27     path: '/navigation',
28     name: '导航 2',
29     component: Home,
30     children: [
31       {
32         path: '/three',
33         name: '页面 3',
34         component: () => import('../views/PageThree.vue')
35       },
36       {
37         path: '/four',
38         name: '页面 4',
39         component: () => import('../views/PageFour.vue')
40       }
41     ]
42   }
43 ]
44
45 // 动态读取 router里面的菜单, 上面的方法不可取 -->
46 <el-menu router>
47   <!-- $router.options获取路由配置文件下的数组 routes -->
48   <!-- 注意设置 index, 其是文本类型, 不能是数值类型, 拼个空字符串 -->
49   <el-submenu v-for="(item,index) in $router.options.routes" :index="index + ''">
50     <template slot="title"><i class="el-icon-message"></i><{{ item.name }}></template>
51     <el-menu-item v-for="(menuItem,menuIndex) in item.children" :index="menuItem.path">
52       {{ menuItem.name }}
53     </el-menu-item>
54   </el-submenu>
55 </el-menu>
56 </el-aside>
57
58 <el-main>
59   <router-view></router-view>
60 </el-main>

```

菜单 menu 与 router的绑定

1. 标签添加 `router`属性, 不需要等于什么
2. 在页面中添加 `<router-view>`标签, 它是一个容器, 动态渲染所选择的 router
3. `<el-menu-item>`标签的 index 就是要跳转的 router,

设置打开页面就是页面 1

```

9   {
10     path: '/',
11     name: '导航 1',
12     redirect: '/one',
13     component: Home,
14     children: [
15       {
16         path: '/one',
17         name: '页面 1',
18         component: () => import('../views/PageOne.vue')
19       },
20     ]
21   }

```

localhost:8080/one

应用 百度一下 香蕉 少矮 知行 仙宗

导航 1

这是页面 1

页面 1

页面 2

导航 2

给 `<el-menu-item>` 添加 `:class="$route.path==menuItem.path"`

`$route.path` 获取的是浏览器地址

前后端一起实现一个分页效果 用 `findAll` 中的 `Pageable` 重载

```
// 根据前端传来的 page、size 参数查询，实现分页效果，显然这里不能用 findAll
@GetMapping("/findAll")
public List<Book> findAll(){
    return bookRepository.findAll();
}
@Handler > findAll()
    findAll() List
    findAll(Pageable pageable) List
    findAll(Example<S> example) Page
    findAll(Example<S> example, Sort sort) Page
31 // 返回的是 Page，不是 List，/{page}/{size} 表示通过前端传参：用 PathVariable 把前端传来的参数取出来
32 public Page<Book> findAll(@PathVariable("page") Integer page, @PathVariable("size") Integer size){
33
34     // SpringBootJpa 提供了一个分页的方法，调用即可：首先拿到 Pageable 对象，表示分页信息；把 pageable 传给 findAll 方法
35     // Pageable 是一个接口，用其 PageRequest 实现类：用 .of(page, size) 方法传递页码信息，page 表示多少页，size 表示每页多少数据
36     Pageable pageable = PageRequest.of( page-1, size );
37     return bookRepository.findAll(pageable);
38 }
默认页码是从 0 开始的，所以这里减 1
```

后端接口好了，前端处理数据

```
62 created(){
63     // 获取第一页的数据，共 6 条数据
64     axios.get('http://localhost:8082/book/findAll/1/6').then( res => {
65         console.log(res)
66     })
67 }, ...
```

[HMR] Waiting for update signal from WDS...

△ [vue-router] Duplicate named routes definition: { name: "页面 4", path: "/five" }

```
▼ (data: {}, status: 200, statusText: "", headers: {}, config: {}, ...)
  ▶ config: {transformRequest: {}, transformResponse: {}, timeout: 0, xsrfCookieName: "XSRF-TOKEN", adapter: f, ...}
  ▶ data:
    ▶ content: Array(6)
      ▶ 0: {id: 1, name: "解忧杂货店", author: "东野圭吾"}
      ▶ 1: {id: 2, name: "追风筝的人", author: "卡勒德·胡赛尼"}
      ▶ 2: {id: 3, name: "人世间", author: "太宰治"}
      ▶ 3: {id: 4, name: "这就是二十四节气", author: "高春香"}
      ▶ 4: {id: 5, name: "白夜行", author: "东野圭吾"}
      ▶ 5: {id: 6, name: "挪威的森林", author: "村上春树"}
```

渲染到页面就好

前后端一起实现添加数据的功能

```
3 <!-- 校验之前，输入框数据要双向绑定，用户输入数据时内容要绑定到 Vue 对象，通过 :model(用来跟对象绑定，例如 ruleForm 对象) -->
4 <!-- 表单绑定后，里面的 input 要跟绑定对象中的属性绑定，通过 v-model，例如 ruleForm 对象中的 bookName、author 属性 -->
5 <!-- :rules 是用来绑定校验规则的，再通过 prop -->
6 <el-form :model="ruleForm" :rules="rules" ref="ruleForm" label-width="100px" class="demo-ruleForm">
7     <el-form-item label="书名" prop="bookName">
8         <el-input v-model="ruleForm.bookName"></el-input>
9     </el-form-item>
10    <el-form-item label="作者" prop="author">
11        <el-input v-model="ruleForm.author"></el-input>
12    </el-form-item>
13
14    <el-form-item>
15        <el-button type="primary" @click="submitForm('ruleForm')">提交</el-button>
16        <el-button @click="resetForm('ruleForm')">重置</el-button>
17    </el-form-item>
18 </el-form>
19 </div>
20 </template>
21
22 <script>
23     export default {
24         name: 'addBook',
```

```

4     <!-- 校验之前，输入框数据要双向绑定，用户输入数据时内容要绑定到 Vue 对象，通过 :model(用来跟对象绑定，例如 ruleForm 对象) -->
5     <!-- 表单绑定后，里面的 input 要跟绑定对象中的属性绑定，通过 v-model，例如 ruleForm 对象中的 bookName、author 属性 -->
6     <!-- :rules 是用来绑定校验规则的，再通过 prop -->
7     <el-form :model="ruleForm" :rules="rules" ref="ruleForm" label-width="100px" class="demo-ruleForm">
8         <el-form-item label="书名" prop="bookName">
9             <el-input v-model="ruleForm.bookName"></el-input>
10        <el-form-item label="作者" prop="author">
11            <el-input v-model="ruleForm.author"></el-input>
12        </el-form-item>
13
14        <el-form-item>
15            <el-button type="primary" @click="submitForm('ruleForm')">提交</el-button>
16            <el-button @click="resetForm('ruleForm')">重置</el-button>
17        </el-form-item>
18    </el-form>
19 </div>
20 </template>
21
22 <script>
23     export default {
24         name: 'AddBook',
25         data() {
26             return {
27                 ruleForm: {
28                     bookName: '',
29                     author: ''
30                 },
31                 rules: {
32                     bookName: [
33                         { required: true, message: '请输入书名', trigger: 'blur' },
34                         { min: 1, max: 100, message: '长度在 1 到 100 个字符', trigger: 'blur' }
35                     ],
36                     author: [
37                         { required: true, message: '请输入作者名', trigger: 'blur' },
38                         { min: 2, max: 25, message: '长度在 2 到 25 个字符', trigger: 'blur' }
39                     ]
40                 }
41             }
42         }
43     }
44 </script>

```

Element UI 表单数据校验

定义 rules 对象，在 rules 对象中设置表单各个选项的校验规则

```

rules: {
    name: [
        { required: true, message: 'error', trigger: 'blur' },
        { min: 3, max: 5, message: '长度在 3 到 5 个字符', trigger: 'blur' }
    ]
}

```

required: true, 是否为必填项

message: 'error', 提示信息

trigger: 'blur', 触发事件

页面效果：



先写后端接口，进入 repository，BookRepository 继承了 JpaRepository，其有继承了其他，他们中的方法 BookRepository 都可以用，而这里面有 save 方法，可以直接用

```

package com.baichen.projectspringboot.repository;

import com.baichen.projectspringboot.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;

// 承接 JpaRepository，必须给出来体类名和主键类型
public interface BookRepository extends JpaRepository<Book, Integer>

```

```

/**
 * JPA specific extension of {@link org.springframework.data.repository.Repository}.
 *
 * @author Oliver Gierke
 * @author Christoph Strobl
 * @author Mark Paluch
 */
@NoRepositoryBean
public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>, Qi

```

```

34     */
35     @NoRepositoryBean
36     public interface JpaRepository<T, ID> extends PagingAndSortingRepository<T, ID>, JpaRepository<T, ID> {
37     /*
38
39     * DataJpa repository /> PagingAndSortingRepository
40     */
41     // file, bytecode version: 52.0 (Java 8)
42
43

```

按住 Ctrl键点击

repositoryBean
lic interface PagingAndSortingRepository<T, ID> extends CrudRepository<T, ID> {
Iterable<T> findAll(Sort var1);

Multiple implementations

ngframework / data / repository / CrudRepository

```

1  /**
2  *
3  *
4  * package org.springframework.data.repository;
5  *
6  * import java.util.Optional;
7  *
8  * @NoRepositoryBean
9  * public interface CrudRepository<T, ID> extends Repository<T, ID> {
10 *     <S extends T> S save(S var1);
11 * }

```

这样测试会在数据库的book表中添加一条数据，id因为是主键，要实现自增，所以实体类中加可以实现自增的注解

测试save方法

```

23     void save(){
24         // 添加两个属性，把对象的引用赋给了变量
25         Book book = new Book();
26         book.setName("Spring Boot");
27         book.setAuthor("王二");
28         Book book1 = bookRepository.save(book);
29         System.out.println(book1);
30     }

```

这样测试会在数据库的book表中添加一条数据，id因为是主键，要实现自增，所以实体类中加可以实现自增的注解

Springboot D:\front-end\练习\SpringBoot\src\main\java\com\baichen\projectspringboot\in\java\com\baichen\projectspringboot\>\config\>\controller\>\entity\> Book

```

4  import javax.persistence.Entity;
5  import javax.persistence.GeneratedValue;
6  import javax.persistence.GenerationType;
7  import javax.persistence.Id;
8  @Entity
9  // 加 @Data: Lombok的注解(先引入Lombok)，自动生成各种各样的 Get 和 Set 方法
10 @Data
11 public class Book {
12     @Id
13     @GeneratedValue(strategy = GenerationType.IDENTITY)
14     private Integer id;
15     // 因为数据库中 id 是自增的，不然测试会报错 org.springframework.orm.JpaSystemException:
16     // @GeneratedValue(strategy = GenerationType.IDENTITY): 自增注解, MySQL 设置自增就是这个
17     // 因为数据库中 id 是自增的，不然测试会报错 org.springframework.orm.JpaSystemException:
18     // @GeneratedValue(strategy = GenerationType.IDENTITY): 自增注解, MySQL 设置自增就是这个

```

测试成功之后，需要写 Controller，这样 Web 可以调用后端接口

启动后端

前端调用接口

```

39     @PostMapping("/save")
40     // 需要传一个 Book 对象，前端是通过 JSON 格式传数据，需要用 @RequestBody 注解映射才能将 JSON 转成 Java 对象
41     public String save(@RequestBody Book book){
42         Book result = bookRepository.save(book);
43         if(result != null){ // 条件为真，则保存成功，前端可基于此判断得知保存的结果成功与否
44             return "success";
45         }else {
46             return "error";
47         }
48     }

```

在页面填写内容点击提交，发现 success

* 书名

* 作者

```

▼ Object
  ▼ config:
    ▶ adapter: f xhrAdapter(config)
    data: {"bookName": "保罗策兰诗选", "author": "孟明")
    ▶ headers: {Accept: "application/json, text/plain, /"}, Content-Type: "application/json; charset=utf-8"
    maxContentLength: -1
    method: "post"
    timeout: 0
  ▶ transformRequest: {0: f}
  ▶ transformResponse: {0: f}
  url: "http://localhost:8082/book/save"
  validateStatus: f validateStatus(status)
  xsrfCookieName: "XSRF-TOKEN"
  xsrfHeaderName: "X-XSRF-TOKEN"
  ▶ __proto__: Object
  data: "success"
  ▶ headers:

```

id	name	author	publish	pages	price	bookcaseid	abled
1	解忧杂货店	东野圭吾	电子工业出版社	102	27.30	9	1
2	追风筝的人	卡勒德·胡塞尼	中信出版社	330	26.00	1	1
3	人间失格	太宰治	作家出版社	150	17.30	1	1
14	西游记	吴承恩	三联出版社	300	60.00	3	1
21	(Null)	孟明	(Null)	(Null)	(Null)	(Null)	(Null)

但是书名“保罗策兰诗选”并没有加入到数据库中，排查问题是前后端参数名定义不一致造成的，后端用的 name，前端页面用的 bookName，所以修改前端书名的参数名称为 name，再添加即成功。（注意：）

```

<el-form-item label="书名" prop="name">
  <el-input v-model="bookInfo.name"></el-input>
</el-form-item>

```

这两个参数名要一样

23 梵高手稿 梵高 (Null) (Null) (Null) (Null)

// 调用保存接口把数据传到后端进行保存，传递的是对象，直接放在请求地址后面，逗号隔开

```

axios.post('http://localhost:8082/book/save', this.bookInfo).then( res => {
  if(res.data == 'success'){
    _this.$confirm('《' + _this.bookInfo.name + '》添加成功', '消息', {
      distinguishCancelAndClose: true,
      confirmButtonText: '查询图书',
      cancelButtonText: '关闭',
    })
    .then(() => {
      // 点击查询图书跳转到查询图书的页面
      _this.$router.push('/query')
    })
  }else{
    this.$message({
      showClose: true,
      message: '提交失败！！',
      type: 'warning'
    });
  }
})

```

实现修改数据的功能

修改是先通过把 id 传给后端，后端 findById 查找到数据之后再修改

在查询图书页面点击修改，携带 id 跳转到修改图书页面（页面与添加图书页面一样）

```

55   edit(row){
56     // console.log(row) // 页面中点哪行数据，row就是当前这行对象，带着图书的 id跳转修改图书页面
57     // this.$router.push('/edit') // 这样写只是跳转，携带对象用下面的写法
58     this.$router.push({
59       path:'/edit',
60       query:{
61         id:row.id
62       }
63     })
64   },

```

然后修改图书页面拿到 id，在生命周期 created 中通过调用后端接口 findById 拿到该 id 的图书信息

后端先测试 findById 方法并写 controller

```

32   @Test
33   void findById(){
34     bookRepository.findById(1);
35   }

```

按住Ctrl点击查看findById是什么对象

```

10  @NoRepositoryBean
11  public interface CrudRepository<T, ID> extends Repository<T, ID> {
12    <S extends T> S save(S var1);
13
14    <S extends T> Iterable<S> saveAll(Iterable<S> var1);
15
16    Optional<T> findById(ID var1);
17
18    boolean existsById(ID var1);
19

```

是 Optional 对象

Optional 是 JDK8 加的特性，相当于给实体类又包了一层，可以抛掉实体类是空的时候报的控制异常

所以在 findById 后面再用 get() 方法，get 后面对应取值一个 Book 对象

```

36   void findById(){
37     bookRepository.findById(1).get();
38   }

```

m get()

Book

```

32     @Test
33     void findById(){
34         Book book =bookRepository.findById(1).get();
35         System.out.println(book);
36     }

```

```

32     @Test
33     void findById(){
34         Book book =bookRepository.findById(1).get();
35         System.out.println(book);
36     }

```

BookRepositoryTest > findById()

id 为 1 时, 查询出来了图书信息, 测试成功

写 controller

```

50     @GetMapping("/findById/{id}")
51     public Book findById(@PathVariable("id") Integer id){
52         return bookRepository.findById(id).get();
53     }

```

启动后端, 浏览器上简单测试下获取到了数据

localhost:8082/book/findById/1

FeHelper | 自动解码 | 排序: 默认 | 升序 | 降序 | 乱码修正 | 元数据 | 折叠所有 | 下载JSON

```

{
    "id": 1,
    "name": "解忧杂货店",
    "author": "东野圭吾"
}

```

接下来在修改图书页面中改了数据之后, 点击修改需要调用一个后端 update接口

编号	4
* 书名	这就是二十四节气
这里点击重置会清掉 id, 使得最终修改时不能用 /save 接口, 必须另写一个 /update	
* 作者	高春香
<input type="button" value="修改"/> <input type="button" value="重置"/>	

```

38     @Test
39     void update(){
40         Book book = new Book();
41         book.setId(129);
42         book.setName("测试测试");
43         // 没有 update方法, 还是调用save方法, 其既可以保存又可以修改, 看 key是否存在, 无主键时保存, 有主键时修改
44         Book book2 = bookRepository.save(book);
45         System.out.println(book2);

```

BookRepositoryTest > update()

Tests passed: 1 of 1 test – 185 ms

where
book0_.id=?
Hibernate:
insert
into
book
(author, name)
values
(?,?)
Book(id=39, name=测试测试, author=null)

因为数据库 book表中此时 id最大为 38, 没有 id=129的图书信息, 所以 save方法保存, 而不是修改, id自增为 39

```

55 // 修改注解 PutMapping, 删除专用注解 DeleteMapping
56 @PutMapping("/update")
57 public String update(@RequestBody Book book){
58     Book result = bookRepository.save(book);
59     if(result != null){
60         return "success";
61     }else {
62         return "error";
63     }
64 }

```

然后在修改图书页面把请求方法和请求地址改一下，把添加成功改为修改成功就好了

实现删除数据的功能

后端，测试删除方法，可以看到数据库中 id 为 11 的图书被删除了

48	@Test	9 我们仨	杨绛	生活·读书·新知三联书店	89	17.20	7	1
49	void delete(){	10 活着	余华	作家出版社	100	100.00	6	1
50	bookRepository.deleteById(11);	12 三国演义	罗贯中	三联出版社	300	50.00	2	1
51	}	13 红楼梦	曹雪芹	三联出版社	300	50.00	5	1
52	}	14 西游记	吴承恩	三联出版社	300	60.00	3	1

```

66 @DeleteMapping("/deleteById/{id}")
67 public void deleteById(@PathVariable("id") Integer id){
68     bookRepository.deleteById(id);
69 }

```

前端

```

66 // 删除, 根据 id 调用后端删除接口
67 deleteBook(row){
68     axios.delete('http://localhost:8082/book/deleteById/' + row.id).then( res => {
69         this.$confirm('`' + row.name + '`删除成功!', '消息', {
70             confirmButtonText: '确定'
71         })
72         .then(() => {
73             // 动态刷新一下页面, 使用以下前端语法
74             window.location.reload();
75         })
76     })
77 }

```