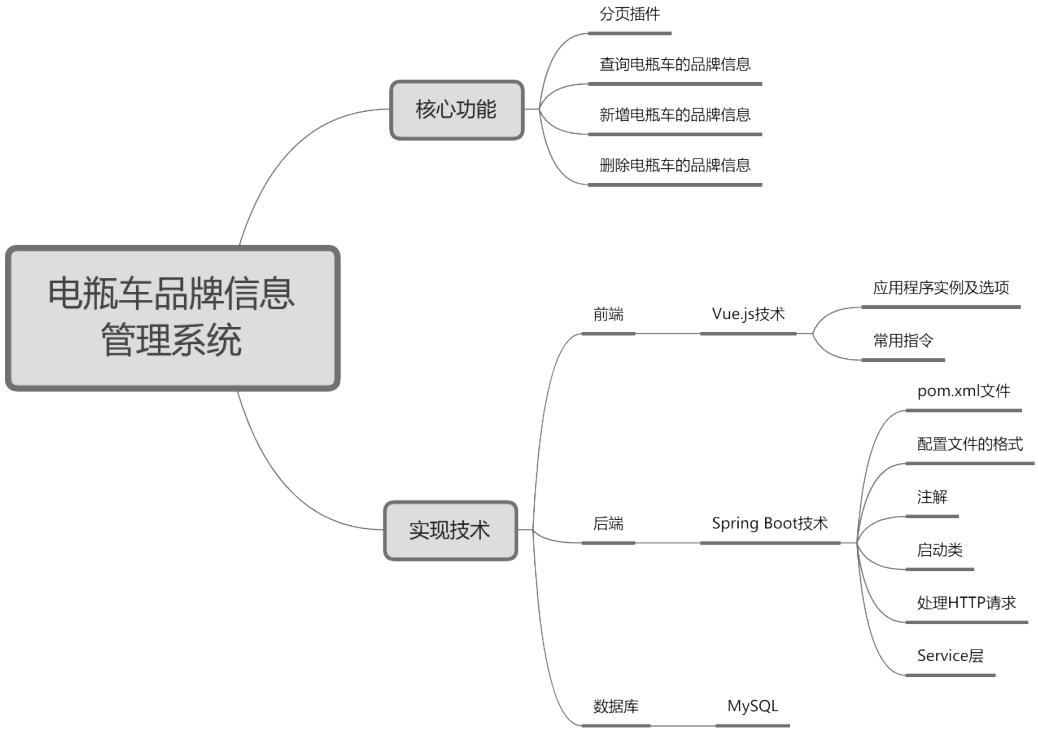
第 1 章

电瓶车品牌信息管理系统

——Vue.js + Spring Boot + MySQL

电瓶车又被称作电动自行车或电动助力车，是一种结合了传统自行车和电动车技术的交通工具。电瓶车作为一种环保、节能、便捷的交通工具，不仅适合短途出行，而且可以作为公共交通的接驳工具，可以减少人们对汽车的依赖，缓解城市交通拥堵和空气污染问题。本章将开发一个简单的全栈项目——电瓶车品牌信息管理系统，用于管理电瓶车的品牌信息。其中，本项目的前端将使用 Vue.js 予以实现，后端将使用 Spring Boot 予以实现。此外，本项目用于存储数据的工具是 MySQL 数据库。

本项目的核心功能及实现技术如下：



1.1 开 发 背 景

随着科学技术的进步，人们的环保意识不断提高，电瓶车市场不断扩大，不断有新的制造商加入。同时，为了更好地满足消费者的需求，扩大市场占有率，各品牌纷纷推出新款车型，市场竞争日趋激烈。为了在市场竞争中脱颖而出，电瓶车制造商需要不断加大研发投入，不断推出更多符合市场需求的产品，不断提升产品质量和服务水平。例如，雅迪、爱玛等传统品牌通过提升产品性能和智能化水平来巩固市场地

位；小牛电动、九号等新兴品牌通过推出符合年轻人口味的产品来聚焦高端市场。在这样的背景下，本章将开发一个全栈项目，即电瓶车品牌信息管理系统。该系统是基于对电瓶车的品牌信息进行管理的网络平台。前端负责把由后端实现的、用于查看、新增、删除电瓶车品牌信息等功能呈现在用户的浏览器上，进而达到与用户进行交互的目的；后端则负责处理由前端发送的请求（例如查看、新增或者删除电瓶车品牌信息），并根据这个请求执行相应的业务逻辑，最终把处理结果返回前端。

电瓶车品牌信息管理系统将实现以下目标：

- ☑ 页面简洁、功能明确、操作方便；
- ☑ 用户可以查看各个电瓶车的品牌信息（如品牌名称、品牌评分、好评率和品牌介绍等）；
- ☑ 用户可以新增电瓶车的品牌信息；
- ☑ 用户可以删除某个电瓶车的品牌信息。

1.2 系 统 设 计

1.2.1 开发环境

本项目的开发及运行环境如下：

- ☑ 操作系统：推荐 Windows 10、11 及以上版本，兼容 Windows7（SP1）。
- ☑ 开发工具：IntelliJ IDEA。
- ☑ 前端实现技术：HTML5、CSS3、JavaScript、Vue.js。
- ☑ 后端实现技术：Java EE、Spring Boot。
- ☑ 数据库：MySQL 8.0。
- ☑ Web 服务器：Tomcat 9.0 及以上版本。

1.2.2 业务流程

启动项目后，打开浏览器，访问 http://localhost:8080/pages/bikes.html，即可看到电瓶车品牌信息管理系统的主页面。

在主页面上，程序将分页显示 10 条数据，共分两页，每一页最多显示 7 条数据，用户通过分页导航可以随意切换并访问这两个分页的数据。

用户在单击“新增电瓶车品牌信息”按钮后，程序将弹出一个窗口，用户在这个窗口中依次输入电瓶车的品牌名称、品牌评分、好评率和品牌介绍等信息，单击“确定”按钮，即可完成新增电瓶车品牌信息的操作。

在主页面上显示的每一条数据的后面，都有一个“删除”按钮，用户先单击某一个“删除”按钮，再单击删除提示窗口中的“确定”按钮，程序将删除对应的电瓶车品牌信息。

电瓶车品牌信息管理系统的业务流程如图 1.1 所示。

1.2.3 功能结构

本项目的功能结构已经在章首页中给出，作为基于对电瓶车的品牌信息进行管理的网络平台，本项目实现的具体功能如下：

- ☑ 分页插件：用于显示数据总数、分页数、与某个页面对应的数据和分页导航等信息。
- ☑ 查询电瓶车的品牌信息：查询数据库中所有电瓶车的品牌名称、品牌评分、好评率和品牌介绍等信息。

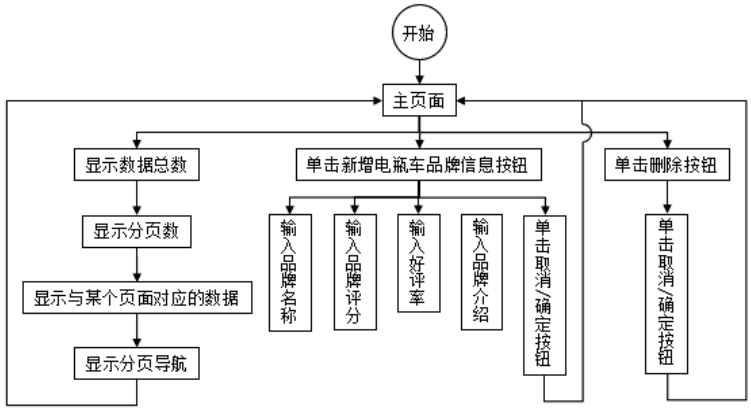


图 1.1　电瓶车品牌信息管理系统的业务流程图

☑ 新增电瓶车的品牌信息：向数据库添加新的电瓶车品牌信息。
☑ 删除电瓶车的品牌信息：从数据库删除某一条电瓶车品牌信息。

1.3　前端技术准备

在实际开发中，Spring Boot 和 Vue.js 的结合被广泛应用于构建 Web 应用程序。Spring Boot 和 Vue.js 的结合不仅使得前、后端能够并行开发以缩短开发周期，而且使得前、后端能够分离设计以方便维护。此外，Spring Boot 和 Vue.js 都具有丰富的 API，便于程序开发人员解决在程序开发过程中遇到的问题。本节将先简单介绍本项目所使用的前端核心技术 Vue.js，下一节再介绍本项目所使用的后端核心技术 Spring Boot。

对于一个全栈项目而言，前端指的是 Web 应用程序中与用户进行交互的部分，它主要负责把由后端提供的数据和实现的功能呈现在用户的浏览器上。前端通常由 HTML、CSS 和 JavaScript 予以构建。Vue.js（简称为 Vue）是一个开源的、非常受欢迎的 JavaScript 框架，是一套用于构建用户界面的渐进式框架。与其他重量级框架不同的是，它只关注视图层，采用自底向上增量开发的设计。Vue.js 的核心目标之一，是通过尽可能简单的 API 实现响应的数据绑定和可组合的视图组件。它不仅容易上手，还非常容易与其他库或已有项目进行整合。Vue.js 实际上是一个用于开发 Web 前端界面的库，其本身具有响应式编程和组件化的特点。所谓响应式编程，即保持状态和视图的同步。响应式编程允许将相关模型的变化自动反映到视图上，反之亦然。和其他前端框架一样，Vue.js 同样采用“一切都是组件”的理念，即将一个网页分割成多个可复用的组件。下面将对本项目中用到的 Vue.js 中的重点知识进行必要介绍，以确保读者可以顺利完成本项目。

1.3.1　应用程序实例及选项

每个 Vue.js 的应用都需要创建一个应用程序的实例对象并挂载到指定 DOM 上。在 Vue.js 3.0 中，创建一个应用程序实例的语法格式如下：

```
Vue.createApp(App)
```

createApp()是一个全局 API，它接收一个根组件选项对象作为参数。选项对象中包括数据、方法、生命周期钩子函数等。创建应用程序实例后，可以调用实例的 mount()方法，将应用程序实例的根组件挂载到指定的 DOM 元素上。这样，该 DOM 元素中的所有数据变化都会被 Vue.js 所监控，从而实现数据的双向绑定。例如，要绑定的 DOM 元素的 id 属性值为 app，创建一个应用程序实例并绑定到该 DOM 元素的代

码如下：

```
Vue.createApp(App).mount('#app')
```

下面分别对组件选项对象中的几个常用选项进行介绍。

1. 数据

在组件选项对象中有一个 data 选项，该选项是一个函数，Vue.js 在创建组件实例时会调用该函数。data()函数可以返回一个数据对象，应用程序实例本身会代理数据对象中的所有数据。例如，创建一个根组件实例 vm，在实例的 data 选项中定义一个数据，代码如下：

```
<div id="app">
        <h2>{{text}}</h2>
</div>
<script src="https://unpkg.com/vue@next"></script>
<script type="text/javascript">
        //创建应用程序实例
        const vm = Vue.createApp({
                //返回数据对象
                data(){
                        return {
                                text: '千里之行，始于足下。'                    //定义数据
                        }
                }
        //装载应用程序实例的根组件
        }).mount('#app');
</script>
```

上述代码中，将创建的根组件实例赋值给变量 vm，在实际开发中并不要求一定要将根组件实例赋值给某个变量。

2. 方法

在创建的应用程序实例中，通过 methods 选项可以定义方法。应用程序实例本身也会代理 methods 选项中的所有方法，因此也可以像访问 data 数据那样来调用方法。例如，在根组件实例的 methods 选项中定义一个 showInfo()方法，代码如下：

```
<div id="app">
        <p>{{showInfo()}}</p>
</div>
<script src="https://unpkg.com/vue@next"></script>
<script type="text/javascript">
        //创建应用程序实例
        const vm = Vue.createApp({
                //返回数据对象
                data(){
                        return {
                                text : '静以修身，俭以养德。',
                                author : ' —— 诸葛亮'
                        }
                },
                methods : {
                        showInfo : function(){
                                return this.text + this.author;                 //连接字符串
                        }
                }
        //装载应用程序实例的根组件
        }).mount('#app');
</script>
```

3. 生命周期钩子

每个应用程序实例在创建时都有一系列的初始化步骤。例如，创建数据绑定、编译模板、将实例挂载到 DOM 并在数据变化时触发 DOM 更新、销毁实例等。在这个过程中会运行一些叫作生命周期钩子的函

数，通过这些钩子函数可以定义业务逻辑。应用程序实例中几个主要的生命周期钩子函数说明如下：

- ☑ beforeCreate：在实例初始化之后，数据观测和事件/监听器配置之前调用。
- ☑ created：在实例创建之后进行调用，此时尚未开始 DOM 编译。在需要初始化处理一些数据时会比较有用。
- ☑ beforeMount：在挂载开始之前进行调用，此时 DOM 还无法操作。
- ☑ mounted：在 DOM 文档渲染完毕之后进行调用。相当于 JavaScript 中的 window.onload()方法。
- ☑ beforeUpdate：在数据更新时进行调用，适合在更新之前访问现有的 DOM，例如手动移除已添加的事件监听器。
- ☑ updated：在数据更改导致的虚拟 DOM 被重新渲染时进行调用。
- ☑ beforeDestroy：在销毁实例前进行调用，此时实例仍然有效，可以解绑一些使用 addEventListener 监听的事件等。
- ☑ destroyed：在实例被销毁之后进行调用。

这里通过一个示例来了解 Vue.js 内部的运行机制，代码如下：

```
<div id="app">
    <p>{{text}}</p>
</div>
<script src="https://unpkg.com/vue@next"></script>
<script type="text/javascript">
    //创建应用程序实例
    const vm = Vue.createApp({
        //返回数据对象
        data(){
            return {
                text : '山不在高，有仙则名。'
            }
        },
        beforeCreate : function(){
            console.log('beforeCreate');
        },
        created : function(){
            console.log('created');
        },
        beforeMount : function(){
            console.log('beforeMount');
        },
        mounted : function(){
            console.log('mounted');
        },
        beforeUpdate : function(){
            console.log('beforeUpdate');
        },
        updated : function(){
            console.log('updated');
        }
    //装载应用程序实例的根组件
    }).mount('#app');
    setTimeout(function(){
        vm.text = "水不在深，有龙则灵。";
    },2000);
</script>
```

在浏览器控制台中运行上述代码，页面渲染完成后，结果如图 1.2 所示。

经过两秒钟后调用 setTimeout()方法，修改 text 的内容，触发 beforeUpdate 和 updated 钩子函数，结果如图 1.3 所示。

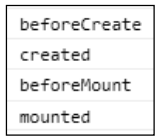


图 1.2　页面渲染后的效果

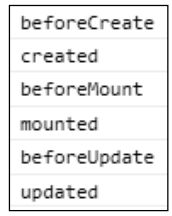


图 1.3　页面最终效果

1.3.2　常用指令

在 Vue.js 中，为了实现渲染视图的功能，指令是必不可少的。例如，在视图中经常需要通过条件判断控制 DOM 的显示状态，这时就需要使用 v-if、v-else、v-else-if 等指令。下面将对 Vue.js 中的常用指令进行介绍。

1. v-if 指令

v-if 指令可以根据表达式的值来判断是否输出 DOM 元素及其包含的子元素。如果表达式的值为 true，则输出 DOM 元素及其包含的子元素；否则，将 DOM 元素及其包含的子元素移除。

v-if 是一个指令，必须将它添加到一个元素上，根据表达式的结果判断是否输出该元素。如果需要对一组元素进行判断，需要使用<template>元素作为包装元素，并在该元素上使用 v-if，最后的渲染结果里不会包含<template>元素。

例如，根据表达式的结果判断是否输出一组单选按钮，代码如下：

```
<div id="app">
        <template v-if="show">
                <input type="radio" value="手机">手机
                <input type="radio" value="电脑">电脑
                <input type="radio" value="家电">家电
                <input type="radio" value="家具">家具
        </template>
</div>
<script src="https://unpkg.com/vue@next"></script>
<script type="text/javascript">
        //创建应用程序实例
        const vm = Vue.createApp({
                //返回数据对象
                data(){
                        return {
                                show : true
                        }
                }
        //装载应用程序实例的根组件
        }).mount('#app');
</script>
```

2. v-else 指令

v-else 指令的作用相当于 JavaScript 中的 else 语句，可以将该指令配合 v-if 指令一起使用。

例如，输出用户的年龄，并判断该年龄是否小于 18。如果是，则输出用户未成年；否则输出用户已成年。代码如下：

```
<div id="app">
        <p>Tom 的年龄是{{age}}</p>
        <p v-if="age<18">Tom 未成年</p>
        <p v-else>Tom 已成年</p>
</div>
<script src="https://unpkg.com/vue@next"></script>
<script type="text/javascript">
        //创建应用程序实例
```

```
    const vm = Vue.createApp({
        //返回数据对象
        data(){
            return {
                age: 20
            }
        }
    //装载应用程序实例的根组件
    }).mount('#app');
</script>
```

3. v-else-if 指令

v-else-if 指令的作用相当于 JavaScript 中的 else if 语句，使用该指令可以进行更多的条件判断，不同的条件对应不同的输出结果。

例如，输出数据对象中的属性 m 和 n 的值，并根据比较两个属性的值，输出比较的结果。代码如下：

```
<div id="app">
    <p>m 的值是{{m}}</p>
    <p>n 的值是{{n}}</p>
    <p v-if="m<n">m 小于 n</p>
    <p v-else-if="m===n">m 等于 n</p>
    <p v-else>m 大于 n</p>
</div>
<script src="https://unpkg.com/vue@next"></script>
<script type="text/javascript">
    //创建应用程序实例
    const vm = Vue.createApp({
        //返回数据对象
        data(){
            return {
                m: 16,
                n: 16
            }
        }
    //装载应用程序实例的根组件
    }).mount('#app');
</script>
```

4. v-show 指令

v-show 指令是根据表达式的值判断是否显示或隐藏 DOM 元素。当表达式的值为 true 时，元素将被显示；当表达式的值为 false 时，元素将被隐藏，此时为元素添加了一个内联样式 style="display:none"。与 v-if 指令不同，使用 v-show 指令的元素，无论表达式的值为 true 还是 false，该元素都始终会被渲染并保留在 DOM 中。绑定值的改变只是简单地切换元素的 CSS 属性 display。

注意

v-show 指令不支持<template>元素，也不支持 v-else 指令。

5. v-for 指令

Vue.js 提供了列表渲染的功能，可将数组或对象中的数据循环渲染到 DOM 中。在 Vue.js 中，列表渲染使用的是 v-for 指令，其效果类似于 JavaScript 中的遍历操作。

v-for 指令将根据接收到的数组中的数据重复渲染相应的 DOM 元素。该指令需要使用 item in items 形式的语法。其中，items 为数据对象中的数组名称，item 为数组元素的别名，通过别名可以获取当前数组遍历的每个元素。

例如，可以使用 v-for 指令将<li>标签循环渲染，输出数组中存储的职位名称，代码如下：

```
<div id="app">
    <ul>
        <li v-for="item in items">{{item.position}}</li>
```

```
        </ul>
</div>
<script src="https://unpkg.com/vue@next"></script>
<script type="text/javascript">
        const vm = Vue.createApp({
                data(){
                        return {
                                items : [                                                    //定义职位数组
                                        { position : '前端工程师'},
                                        { position : '一二线运维'},
                                        { position : '项目经理'}
                                ]
                        }
                }
        }).mount('#app');
</script>
```

在使用 v-for 指令遍历数组时，还可以指定一个参数作为当前数组元素的索引，语法格式为(item,index) in items。其中，items 为数组名称，item 为数组元素的别名，index 为数组元素的索引。

例如，可以使用 v-for 指令将<li>标签循环渲染，输出数组中存储的职位名称和相应的索引，代码如下：

```
<div id="app">
        <ul>
                <li v-for="(item,index) in items">{{index}} - {{item.position}}</li>
        </ul>
</div>
<script src="https://unpkg.com/vue@next"></script>
<script type="text/javascript">
        const vm = Vue.createApp({
                data(){
                        return {
                                items : [                                                    //定义职位数组
                                        { position : '前端工程师'},
                                        { position : '一二线运维'},
                                        { position : '项目经理'}
                                ]
                        }
                }
        }).mount('#app');
</script>
```

1.4 后端技术准备

对于一个全栈项目而言，后端主要由服务器端和数据库组成，它负责处理由前端发送的请求，与数据库进行交互并执行相应的业务逻辑，把处理结果返回前端。Spring Boot 是当下非常流行的一个后端框架，它是在 Spring 的基础上发展而来的、全新的、开源的框架。开发 Spring Boot 的主要动机是简化部署 Web 项目的配置过程。那么，Spring Boot 是如何以更简单的、更灵活的方式开发 Web 项目的呢？Spring Boot 通过自动配置机制简化了 Web 开发流程，程序开发人员只需要通过依赖注入即可获取所需对象，无须手动管理工厂类。它采用约定优先于配置的原则，程序开发人员只需要在配置文件中定义必要的参数，其余配置均采用合理的默认值。即使不进行任何显式配置，项目也能基于内置的默认设置正常启动。Spring Boot 自带 Tomcat 服务器，在项目启动的过程中可以自动完成所有资源的部署操作。Spring Boot 项目启动的速度很快，即使包含庞大的依赖库，也能够在几秒钟内完成部署和启动。下面将对本项目用到的 Spring Boot 中的重点知识进行必要介绍，以确保读者可以顺利完成本项目。

1.4.1 pom.xml 文件

pom.xml 文件是 Maven 构建项目的核心配置文件，程序开发人员可以在此文件中为项目添加新的依

赖，添加在<dependencies>标签内部，作为其子标签，格式如下：

```
<dependency>
    <groupId>所属团队</groupId>
    <artifactId>项目 ID</artifactId>
    <version>版本号</version>
    <scope>使用范围（可选）</scope>
</dependency>
```

<dependency>是<dependencies>的子标签，dependencies 是 dependency 的复数形式。

例如，Spring Boot 项目自带的 Web 依赖和 JUnit 单元测试依赖，其在 pom.xml 文件中的代码如下：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

程序开发人员只需要仿照这种格式在<dependencies>标签内部添加其他依赖，而后保存 pom.xml 文件，Maven 就会自动下载依赖中的 JAR 文件并自动将其引入项目中。

1.4.2 配置文件的格式

程序开发人员在配置 Spring Boot 项目的过程中，会在配置文件中配置该项目所需的数据信息。这些数据信息被称作“配置信息”。那么，配置信息都包含哪些内容呢？在实际开发中，配置信息的内容非常丰富，这里仅举例予以说明。

- ☑ Tomcat 服务器。
- ☑ 数据库的连接信息，即用于连接数据库的用户名和密码。
- ☑ Spring Boot 项目的启动端口。
- ☑ 第三方系统或者接口的调用密钥信息。
- ☑ 打印用于发现和定位问题的日志。

Spring Boot 支持多种格式的配置文件，最常用的是 properties 格式（默认格式）和比较新颖的 yml 格式。下面将分别介绍这两种格式的特点。

1. properties 格式

properties 格式是经典的键值对文本格式。也就是说，如果某一个配置文件的格式是 properties 格式，那么这个配置文件的文本格式为键值对。键值对的语法非常简单，具体如下：

```
key=value
```

在上述格式中，“=”左侧为键（key），“=”右侧为值（value）。在配置文件中，每个键独占一行。如果多个键之间存在层级关系，就需要使用“父键.子键”的格式予以表示。例如，在配置文件中，为一个有三层关系的键赋值的语法如下：

```
key1.key2.key3=value
```

例如，启动 Spring Boot 项目的 Tomcat 端口号为 8080，那么在这个项目的 application.properties 文件中就能够找到如下内容：

```
server.port=8080
```

启动这个项目后，即可在控制台看到如下一行日志：

```
Tomcat started on port(s): 8080 (http) with context path ''
```

这行日志表明 Tomcat 根据配置开启的是 8080 端口。

在 application.properties 文件中，“#”被称作注释符号，用于向其中添加注释信息。例如：

```
# Tomcat 端口
server.port=8080
```

application.properties 文件不支持中文。如果程序开发人员在 application.properties 文件中编写中文，IntelliJ IDEA 会自动将其转化为 Unicode 码，将鼠标指针悬停在 Unicode 码处可以看到对应的中文。

2. yml 格式

yml 是 YAML 的缩写，它是一种可读性高、用于表达数据序列化的文本格式。对于 yml 格式的配置文件，其文本格式也是键值对。只不过，键值对的语法与 Python 语言中的键值对的语法非常相似，具体如下：

```
key: value
```

注意

英文格式的“:”与值之间必须有至少一个空格。

在上述格式中，英文格式的“:”左侧为键（key），英文格式的“:”右侧为值（value）。需要注意的是，英文格式的“:”与值之间只能用空格缩进，不能用 tab 缩进；空格数量表示各层的层级关系。例如，在配置文件中，为一个有三层关系的键赋值的语法如下：

```
key1:
  key2:
    key3: value
```

在 properties 格式的配置文件，即使父键相同，在为每一个父键的子键赋值时也要单独占一行，还要把父键写完整，例如：

```
com.mr.strudent.name=tom
com.mr.strudent.age=21
```

但是在 yml 格式的配置文件中，只需要编写一次父键，并保证两个子键缩进关系相同即可。例如，把上述 properties 格式的键值对修改为 yml 格式的键值对的语句如下：

```
com:
  mr:
    student:
      name: Tom
      age: 21
```

对于 Spring Boot 项目的配置文件，不论是采用 properties 格式，还是采用 yml 格式，都由程序开发人员自行决定。但是，在同一个 Spring Boot 项目中，应尽量只使用一种格式的配置文件。否则，这个 Spring Boot 项目中的 yml 格式的配置文件将被忽略。

1.4.3 注解

在给出注解的概念之前，须明确什么是元数据。所谓元数据，指的是用于描述数据的数据。下面结合某个配置文件里的一行信息，举例说明什么是元数据。

```
<string name="app_name">AnnotionProject</string>
```

上述信息中的数据 app_name 用于描述数据 AnnotionProject，数据 app_name 就是元数据。

那么，什么是注解呢？注解又被称作标注，是一种被加入源码的、具有特殊语法的元数据。需要特别说明的是：

☑ 注解仅是元数据，和业务逻辑无关。
☑ 虽然注解不属于程序本身，但是可以对程序作出解释。
☑ 应用程序中的类、方法、变量、参数、包等程序元素都可以被注解。

在理解了“什么是注解”后，再来了解一下在应用程序中注解的应用体现在哪些方面。

☑ 在编译时进行格式检查。例如，如果被@Override 标记的方法不是父类的某个方法，编译器就会报错。
☑ 减少配置。依据代码的依赖性，使用注解替代配置文件。
☑ 减少重复工作。在程序开发的过程中，通过注解减少对某个方法的调用次数。

Spring Boot 是一个支持海量注解的框架，其自带的常用注解如表 1.1 所示。

表 1.1 Spring Boot 的常用注解

注　解	标注位置	功　能
@Autowired	成员变量	自动注入依赖
@Bean	方法	用@Bean 标注方法等价于 XML 中配置的 bean，用于注册 Bean
@Component	类	用于注册组件。当不清楚注册类属于哪个模块时就用这个注解
@ComponentScan	类	开启组件扫描器
@Configuration	类	声明配置类
@ConfigurationProperties	类	用于加载额外的 properties 配置文件
@Controller	类	声明控制器类
@ControllerAdvice	类	可用于声明全局异常处理类和全局数据处理类
@EnableAutoConfiguration	类	开启项目的自动配置功能
@ExceptionHandler	方法	用于声明处理全局异常的方法
@Import	类	用于导入一个或者多个 @Configuration 注解标注的类
@ImportResource	类	用于加载 xml 配置文件
@PathVariable	方法参数	让方法参数从 URL 中的占位符中取值
@Qualifier	成员变量	与@Autowired 配合使用，当 Spring 容器中有多个类型相同的 Bean 时，可以用@Qualifier("name")来指定注入哪个名称的 Bean
@RequestMapping	方法	指定方法可以处理哪些 URL 请求
@RequestParam	方法参数	让方法参数从 URL 参数中取值
@Resource	成员变量	与@Autowired 功能类似，但有 name 和 type 两个参数，可根据 Spring 配置的 bean 的名称进行注入
@ResponseBody	方法	表示方法的返回结果直接写入 HTTP response body 中。如果返回值是字符串，则直接在网页上显示该字符串
@RestController	类	相当于@Controller 和@ResponseBody 的合集，表示这个控制器下的所有方法都被@ResponseBody 标注
@Service	服务的实现类	用于声明服务的实现类
@SpringBootApplication	主类	用于声明项目主类
@Value	成员变量	动态注入，支持“#{}”与“${}”表达式

这些注解的编码位置是非常灵活的。当注解用于标注类、成员变量和方法时，注解的编码位置可以在

成员变量的上边，例如：

```
@Autowired
private String name;
```

也可以在成员变量的左边，例如：

```
@Autowired private String name;
```

在 Spring Boot 常用注解中，需要特别说明的是，使用@RequestParam 能够标注方法中的参数。例如：

```
@RequestMapping("/user")
@ResponseBody
public String getUser(@RequestParam Integer id) {
    return "success";
}
```

1.4.4 启动类

使用注解能够启动一个 Spring Boot 项目，这是因为在每一个 Spring Boot 项目中都有一个启动类，并且启动类必须被@SpringBootApplication 注解标注，进而能够调用用于启动一个 Spring Boot 项目的 SpringApplication.run()方法。

在本项目中，com.mr 包下的 RunApplication 类就是启动类。代码如下：

```
package com.mr;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class RunApplication {                                    //启动类
    public static void main(String[] args) {                     //主方法
        SpringApplication.run(RunApplication.class, args);
    }
}
```

@SpringBootApplication 注解虽然重要，但使用起来非常简单，因为这个注解是由多个功能强大的注解整合而成的。打开@SpringBootApplication 注解的源码可以看到它被很多其他注解标注，其中最核心的 3 个注解如下：

- ☑ @SpringBootConfiguration 注解，让项目采用基于 Java 注解的配置方式，而不是传统的 XML 文件配置。当然，如果程序开发人员编写了传统的 XML 配置文件，Spring Boot 也是能够读取这些 XML 文件并识别里面的内容的。
- ☑ @EnableAutoConfiguration 注解，开启自动配置。这样 Spring Boot 在启动的时候就可以自动加载所有配置文件和配置类了。
- ☑ @ComponentScan 注解，启用组件扫描器。这样项目才能自动发现并创建各个组件的 Bean，包括 Web 控制器（@Controller）、服务（@Service）、配置类（@Configuration）和其他组件（@Component）。

一个项目可以有多个启动类，但这样的代码毫无意义。一个项目应该只使用一次@SpringBootApplication 注解。

1.4.5 处理 HTTP 请求

在开发 Spring Boot 项目的过程中，Spring Boot 的典型应用是处理 HTTP 请求。所谓处理 HTTP 请求，

就是 Spring Boot 把用户通过 URL 地址发送的请求交给不同的业务代码进行处理的过程。

Spring Boot 提供了用于声明控制器类的@Controller 注解。也就是说，在 Spring Boot 项目中，把被@Controller 注解标注的类称作控制器类。控制器类在 Spring Boot 项目中发挥的作用是处理用户发送的 HTTP 请求。Spring Boot 会把不同的用户请求交给不同的控制器进行处理，而控制器则会把处理后得到的结果反馈给用户。

说明

控制器（Controller）定义了应用程序的行为，它负责对用户发送的请求进行解释，并把这些请求映射成相应的行为。

因为@Controller 注解本身被@Component 注解标注，所以控制器类属于组件。这说明在启动 Spring Boot 项目时，控制器类会被扫描器自动扫描。这样，程序开发人员就可以在控制器类中注入 Bean。例如，在控制器中注入 Environment 环境组件，代码如下：

```
@Controller
public class TestController {
    @Autowired
    Environment env;
}
```

Spring Boot 提供了用于映射 URL 地址的@RequestMapping 注解。@RequestMapping 注解可以标注类和方法。如果一个类或者方法被@RequestMapping 注解标注，那么这个类或者方法就能够处理用户通过@RequestMapping 注解映射的 URL 地址发送的请求。

注意

@Controller 注解要结合@RequestMapping 注解一起使用。

@RequestMapping 有几个常用属性，下面主要对 value 属性进行介绍。

value 属性是@RequestMapping 注解的默认属性，用于指定映射的 URL 地址。在单独使用 value 属性时，value 属性可以被隐式调用。调用 value 属性的语法如下：

```
@RequestMapping("test")
@RequestMapping("/test")
@RequestMapping(value= "/test")
@RequestMapping(value={"/test"})
```

上面这 4 种语法所映射的 URL 地址均为“域名/test”。其中，域名指的是当前 Spring Boot 项目所在的域。如果在 IntelliJ IDEA 中启动一个 Spring Boot 项目，那么域名就是 127.0.0.1:8080。

@RequestMapping 注解映射的 URL 地址可以是多层的。例如：

```
@RequestMapping("/shop/books/computer")
```

上述代码映射的完整的 URL 地址是 http://127.0.0.1:8080/shop/books/computer。需要特别注意的是，这个 URL 地址中的任何一层都是不可或缺的，否则将引发 404 错误。

@RequestMapping 注解允许一个方法同时映射多个 URL 地址。其语法如下：

```
@RequestMapping(value = { "/address1", "/address2", "/address3", ..... })
```

1.4.6 Service 层

Spring Boot 中的 Service 层是业务逻辑层，其作用是处理业务需求，封装业务方法，执行 DAO 层中用

于访问、处理数据的操作。Service 层通常由一个接口和这个接口的实现类组成。其中，Service 层的接口可以在 Controller 层中被调用，用于实现数据的传递和处理；Service 层的实现类须使用@Service 注解予以标注。

在 Spring Boot 中，把被@Service 注解标注的类称作服务类。@Service 注解属于 Component 组件，可以被 Spring Boot 的组件扫描器扫描到。当启动 Spring Boot 项目时，服务类的对象会被自动地创建，并被注册成 Bean。

Service 层的实现过程如图 1.4 所示，具体实现过程如下：

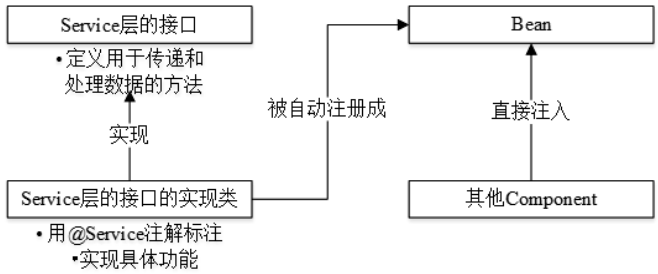


图 1.4　Service 层的实现过程

（1）定义一个 Service 层的接口，在这个接口中定义用于传递和处理数据的方法。例如，定义一个 Service 层的接口 ProductService，代码如下：

```
public interface ProductService {
    …                                          //省略用于传递和处理数据的方法
}
```

（2）定义一个 Service 层的接口的实现类，使用@Service 注解予以标注。这个实现类的作用有两个：一个作用是实现 Service 层的接口中的业务方法；另一个作用是执行 DAO 层中用于访问、处理数据的操作。例如，使用@Service 注解标注实现 ProductService 接口的 ProductServiceImpl 类，代码如下：

```
@Service
public class ProductServiceImpl implements ProductService {
    …                                          //省略用于实现接口的业务方法和用于执行访问处理数据的操作的代码
}
```

（3）在服务类的对象被自动地创建并被注册成 Bean 之后，其他 Component 组件即可直接注入这个 Bean。

1.5　数据库设计

数据库是一个用于存储数据的仓库。通过数据库管理系统，可以有效地存储、组织和管理数据。本项目使用的是 MySQL 数据库。MySQL 数据库是一个中小型、关系型数据库管理系统，由于其体积小、速度快、总体成本低，尤其是开放源码这一特点，许多大中小型网站都为了降低网站运营成本而选择 MySQL 作为后端数据库。此外，MySQL 可以称得上是目前运行速度最快的 SQL 语言数据库管理系统。SQL 语言，即结构化查询语言，它是世界上最流行的、标准化的数据库语言。在实际开发中，MySQL 不仅为多种编程语言提供了 API，而且支持多线程，同时还优化了 SQL 查询算法，极大地提高了性能、可扩展性、可用性，从而满足用户进行业务访问和业务处理的需求。本项目在 MySQL 数据库中创建一个名为 db_e-bike 的库，在这个库中创建与电瓶车品牌信息对应的表。

电瓶车品牌信息表的名称为 bike，主要用于存储电瓶车的品牌编号、品牌名称、品牌评分、好评率、品牌介绍等，其结构如表 1.2 所示。

表 1.2 bike 表结构

字段名称	数据类型	长度	是否主键	说明
id	INT		主键	电瓶车的品牌编号
brand_name	VARCHAR	20		电瓶车的品牌名称
brand_rating	VARCHAR	10		电瓶车的品牌评分
favorable_rate	VARCHAR	10		电瓶车的好评率
brand_intro	VARCHAR	255		电瓶车的品牌介绍

1.6 后端依赖配置和公共模块设计

在开发 Spring Boot 项目的过程中，程序开发人员不仅需要为当前项目手动添加依赖，而且需要为当前项目手动添加配置信息，还需要为当前项目设计公共模块。

依赖是指当前项目所需的外部库或者模块，通常以 jar 包的形式存在。一个项目中可以包含多个依赖，这些依赖通过 Maven 等工具进行管理。

配置信息是用于为当前项目设置各种参数的信息，它通常被存储在配置文件中，以便在启动当前项目时读取并应用这些参数。

公共模块是一组预先构建的代码模块。这些模块可以被共享和复用，因此它们有助于提高开发效率，减少重复工作，并且使得系统的维护和扩展变得更加容易。

下面将依次对本项目后端的依赖配置和公共模块进行介绍。

1.6.1 添加依赖和配置信息

1. 在 pom.xml 文件中添加依赖

因为本项目把 Maven 作为项目构建工具，而 pom.xml 文件是 Maven 构建项目的核心配置文件，所以需要在 pom.xml 文件中为本项目添加依赖，这些依赖会被添加到 pom.xml 文件中的<dependencies>标签内部。代码如下：

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.6.3</version>
    </parent>

    <groupId>com.gs</groupId>
    <artifactId>springboot_crud</artifactId>
    <version>0.0.1-SNAPSHOT</version>

    <properties>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>

        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.4.3</version>
```

```
        </dependency>

        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
            <version>1.2.6</version>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>

        <!--lombok-->
        <dependency>
            <groupId>org.projectlombok</groupId>
            <artifactId>lombok</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>
</project>
```

2. 在 application.yml 文件中添加配置信息

本项目采用的是 yml 格式的配置文件，并在 application.yml 文件中添加如下配置信息：

```
server:
  port: 8080
spring:
  datasource:
    druid:
      driver-class-name: com.mysql.cj.jdbc.Driver
      url: jdbc:mysql://localhost:3306/db_e-bike?useUnicode=true&characterEncoding=UTF-8&serverTimezone=GMT%2b8
      username: root
      password: root

mybatis-plus:
  global-config:
    db-config:
      id-type: auto
  configuration:
    log-impl: org.apache.ibatis.logging.stdout.StdOutImpl
```

1.6.2 工具类设计

将一些反复调用的代码封装成工具类，不仅可以提高开发效率，还可以提高代码的可读性。本项目具有两个工具类，分别是全局异常处理类和通用返回类。下面将分别介绍这两个类。

1. 全局异常处理类

当一个 Spring Boot 项目没有对用户触发的异常进行拦截时，用户触发的异常就会触发最底层异常。在实际开发中，程序开发人员必须对最底层异常进行拦截。

拦截全局最底层异常的方式非常简单，只需在全局异常处理类中单独写一个“兜底”的、用于处理异常的方法，并使用@ExceptionHandler(Exception.class)注解予以标注。

本项目的全局异常处理类是 com.mr.controller.util 工具包下的 ProjectExceptionAdvice 类，该类的代码如下：

```
package com.mr.controller.util;

import org.springframework.web.bind.annotation.ExceptionHandler;

public class ProjectExceptionAdvice {
    //拦截所有的异常信息
    @ExceptionHandler
    public R doException(Exception ex){
        ex.printStackTrace();
        return new R("服务器故障，请稍后再试！");
    }
}
```

2. 通用返回类

在实际开发过程中，需要编写很多个控制器。虽然在这些控制器中的方法各不相同，但是这些控制器的作用都是先让后端处理由前端发送的请求，再把由后端返回的结果传递给前端。程序开发人员习惯把由后端返回的所有结果都统一封装成一个类，并把这个类称作“通用返回类”，同时定义这个类为 R 类，这样由后端传递给前端的结果的类型就都是 R 类型了。也就是说，在控制器中，R 类不仅接收了由后端处理的结果，而且被传递给前端，进而统一了返回的类型。

在本项目的 R 类中，包含了 3 个私有的属性，它们分别是表示 Boolean 型对象的 bool、表示实体类对象的 obj 和表示字符串信息的 str。为了方便外部类访问这 3 个私有的属性，需要为它们添加 Getter/Setter 方法。

此外，在 R 类中还包含了 1 个无参构造方法和 4 个有参构造方法，这 4 个有参构造方法分别为只含有 Boolean 型对象的 bool 的构造方法、含有 Boolean 型对象的 bool 和实体类对象的 obj 的构造方法、含有 Boolean 型对象的 bool 和字符串信息的 str 的构造方法、只含有字符串信息的 str 的构造方法。com.mr.controller.util 工具包下的 R 类的代码如下：

```
package com.mr.controller.util;

public class R {                                          //通用返回值类
    private Boolean bool;                                 //Boolean 型对象
    private Object obj;                                   //实体类对象
    private String str;                                   //字符串信息

    //为通用返回值类添加无参构造方法和有参构造方法
    public R() {
    }

    public R(Boolean flag) {
        this.bool = flag;
    }

    public R(Boolean flag, Object data) {
        this.bool = flag;
        this.obj = data;
    }

    public R(Boolean flag, String msg) {
        this.bool = flag;
        this.str = msg;
    }
```

```
    public R(String msg) {
        this.str = msg;
    }

    //分别为上述的 3 个属性添加 Getter/Setter 方法
    public Boolean getFlag() {
        return bool;
    }

    public void setFlag(Boolean flag) {
        this.bool = flag;
    }

    public Object getData() {
        return obj;
    }

    public void setData(Object data) {
        this.obj = data;
    }

    public String getMsg() {
        return str;
    }

    public void setMsg(String msg) {
        this.str = msg;
    }
}
```

1.6.3 实体类设计

实体类又称数据模型类。顾名思义，实体类是一种专门用于保存数据模型的类。每一个实体类都对应着一种数据模型，通常会将类的属性与数据表的字段相对应。虽然实体类的属性都是私有的，但是通过每一个属性的 Getter/Setter 方法，外部类就能够获取或修改实体类的某一个属性值。实体类通常都会提供无参构造方法，并根据具体情况确定是否提供有参构造方法。

本项目只有一个实体类，这个实体类对应的是 com.mr.pojo 包下的 Bike.java 文件，表示电瓶车类。电瓶车类中的品牌编号、品牌名称、品牌评分、好评率、品牌介绍这 5 个属性，与 db_e-bike 库的 bike 表中的 5 个字段相对应。为了方便外部类访问这 5 个私有的属性，需要为它们添加 Getter/Setter 方法。com.mr.pojo 包下的 Bike 类代码如下：

```
package com.mr.pojo;

public class Bike {
    private Integer id;                                    //编号
    private String brandName;                              //品牌名称
    private String brandRating;                            //品牌评分
    private String favorableRate;                          //好评率
    private String brandIntro;                             //品牌介绍
    //分别为上述的 5 个属性添加 Getter/Setter 方法
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getBrandName() {
        return brandName;
    }
```

```
    public void setBrandName(String brandName) {
        this.brandName = brandName;
    }

    public String getBrandRating() {
        return brandRating;
    }

    public void setBrandRating(String brandRating) {
        this.brandRating = brandRating;
    }

    public String getFavorableRate() {
        return favorableRate;
    }

    public void setFavorableRate(String favorableRate) {
        this.favorableRate = favorableRate;
    }

    public String getBrandIntro() {
        return brandIntro;
    }

    public void setBrandIntro(String brandIntro) {
        this.brandIntro = brandIntro;
    }
}
```

1.6.4 DAO 层设计

本项目的 DAO 层使用了 MyBatisPlus。MyBatisPlus 简称 MP，是一个 MyBatis 的增强工具。MyBatisPlus 在 MyBatis 的基础上只做增强不做改变，专为简化开发、提高开发效率而生。

何以体现 MyBatisPlus 能够简化开发、提高开发效率呢？当使用 MyBatis 时，在编写 Mapper 接口后，不仅需要手动编写对数据执行增、删、改、查等操作的方法，还需要手动编写与每个方法对应的 SQL 语句。例如，使用 MyBatis 读取 t_people 表中的数据，并把读取的数据封装在实体对象中。为此，创建 PeopleMapper 接口作为映射器，在映射器中实现以下 3 个业务：

- ☑ 向 t_people 表添加一个新人员，该人员的数据如下：小丽，女性，20 岁；
- ☑ 将小丽的年龄修改为 19 岁；
- ☑ 删除小丽的所有数据。

PeopleMapper 接口的代码如下：

```
import org.apache.ibatis.annotations.Delete;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Update;
public interface PeopleMapper {
    @Insert("insert into t_people(name,gender,age) values('小丽','女',20)")
    boolean addXiaoLi();

    @Update("update t_people set age = 19 where name = '小丽'")
    boolean updateXiaoLi();

    @Delete("delete from t_people where name = '小丽'")
    boolean delXiaoLi();
}
```

当使用 MyBatisPlus 时，只需要创建 Mapper 接口并继承 BaseMapper 接口，此时当前的 Mapper 接口就会获得由 BaseMapper 接口提供的对数据执行增、删、改、查等操作的方法。也就是说，在创建 Mapper 接口后，既不需要手动编写对数据执行增、删、改、查等操作的方法，也不需要手动编写与每个方法对应的 SQL 语句，从而实现简化开发、提高开发效率的目的。在使用 MyBatisPlus 的情况下，可以将上述代码修改如下：

```
import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.mr.po.People;

@Mapper
@Repository
public interface PeopleMapper extends BaseMapper<People> {

}
```

简而言之，当使用 MyBatisPlus 时，创建的 Mapper 接口是一个空接口。

本项目的 BikeDao 接口就是 DAO 层。因为 BikeDao 接口继承了 BaseMapper 接口，所以 BikeDao 接口是一个空接口。BikeDao 接口的代码如下：

```
package com.mr.dao;

import com.baomidou.mybatisplus.core.mapper.BaseMapper;
import com.mr.pojo.Bike;
import org.apache.ibatis.annotations.Mapper;
import org.springframework.stereotype.Repository;

@Mapper
@Repository
public interface BikeDao extends BaseMapper<Bike> {                    // Dao 层

}
```

为了让读者能够更深入地理解 BaseMapper 接口，这里给出 BaseMapper 接口的相关代码：

```
package com.baomidou.mybatisplus.core.mapper;

import com.baomidou.mybatisplus.core.conditions.Wrapper;
import com.baomidou.mybatisplus.core.metadata.IPage;
import java.io.Serializable;
import java.util.Collection;
import java.util.List;
import java.util.Map;
import org.apache.ibatis.annotations.Param;

public interface BaseMapper<T> extends Mapper<T> {
    int insert(T entity);
    int deleteById(Serializable id);
    int deleteByMap(@Param("cm") Map<String, Object> columnMap);
    int delete(@Param("ew") Wrapper<T> queryWrapper);
    int deleteBatchIds(@Param("coll") Collection<? extends Serializable> idList);
    int updateById(@Param("et") T entity);
    int update(@Param("et") T entity, @Param("ew") Wrapper<T> updateWrapper);
    T selectById(Serializable id);
    List<T> selectBatchIds(@Param("coll") Collection<? extends Serializable> idList);
    List<T> selectByMap(@Param("cm") Map<String, Object> columnMap);
    T selectOne(@Param("ew") Wrapper<T> queryWrapper);
    Integer selectCount(@Param("ew") Wrapper<T> queryWrapper);
    List<T> selectList(@Param("ew") Wrapper<T> queryWrapper);
    List<Map<String, Object>> selectMaps(@Param("ew") Wrapper<T> queryWrapper);
    List<Object> selectObjs(@Param("ew") Wrapper<T> queryWrapper);
    <E extends IPage<T>> E selectPage(E page, @Param("ew") Wrapper<T> queryWrapper);
    <E extends IPage<Map<String, Object>>> E selectMapsPage(E page, @Param("ew") Wrapper<T> queryWrapper);
}
```

1.7 分页插件模块设计

在主页面上，本项目通过分页插件显示数据总数、分页数、与某个页面对应的数据和分页导航等信息。程序分页显示 10 条数据，共分两页，每一页最多显示 7 条数据，用户通过分页导航可以随意切换并

访问这两个分页的数据，如图 1.5 所示。下面将介绍分页插件模块的实现过程。

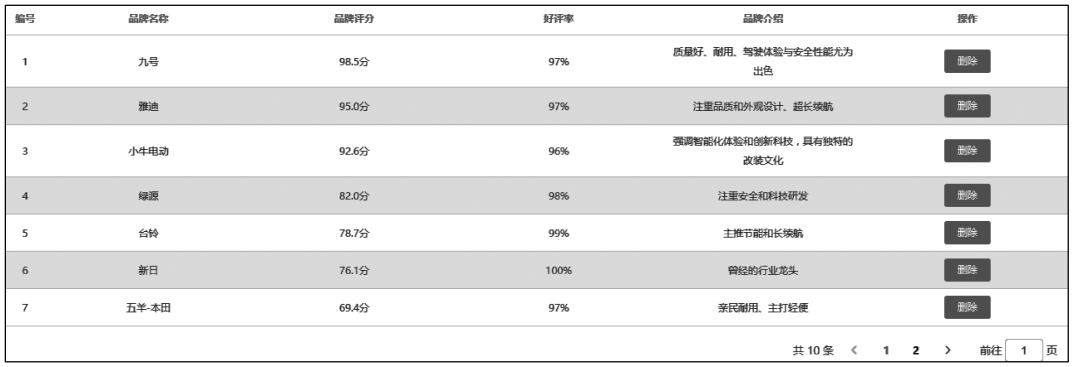


图 1.5　分页插件的效果图

1.7.1　前端设计

本项目中的 bikes.html 即为主页面。在初始化主页面时，主页面还没有来得及从数据库中获取数据，此时数据总数为 0，并且电瓶车的品牌名称、品牌评分、好评率和品牌介绍的值均为空的字符串。因此，在 bikes.html 中初始化分页插件相关数据的代码如下：

```
<script src="../js/vue.js"></script>
<script>
    var vue = new Vue({
        el: '#app',
        data: {
            dataList: [],                                   //当前页面要展示的列表数据
            dialogFormVisible: false,                       //添加表单是否可见
            dialogFormVisible4Edit: false,                  //编辑表单是否可见
            formData: {},                                   //表单数据
            rules: {                                        //校验规则
                brandName: [{required: true, message: '品牌名称为必填项', trigger: 'blur'}],
                brandRating: [{required: true, message: '品牌评分为必填项', trigger: 'blur'}],
                favorableRate: [{required: true, message: '好评率为必填项', trigger: 'blur'}],
                textarea: [{required: true, message: '品牌介绍为必填项', trigger: 'blur'}]
            },
            pagination: {                                   //分页插件的相关数据
                currentPage: 1,                             //当前页码
                pageSize: 7,                                //每页显示的记录数
                total: 0,                                   //总记录数
                brandName: "",
                brandRating: "",
                favorableRate: "",
                brandIntro: ""
            }
        },

        //钩子函数，Vue 对象初始化完成后自动执行
        created() {
            //调用查询全部数据
            this.getAll();
        },
        //省略 methods 选项的代码
    })
</script>
```

1.7.2　后端设计

com.mr.config 包下的 PageConfig 类为分页插件配置类，这个类可以为本项目配置分页插件，进而分页

显示与每个页面对应的数据。下面将介绍分页插件的出处及其配置过程。

在介绍分页插件的出处之前，首先了解一下 MyBatis 插件机制。所谓 MyBatis 插件机制，指的是 MyBatis 插件会拦截 Executor、StatementHandler、ParameterHandler 和 ResultSetHandler 这 4 个接口的方法，为了执行自定义的拦截逻辑，需要先利用 JDK 动态代理机制为这些接口的实现类创建代理对象，再执行代理对象的方法。

- ☑ Executor：MyBatis 的内部执行器，它负责调用 StatementHandler 操作数据库，并把结果集通过 ResultSetHandler 予以自动映射。
- ☑ StatementHandler：MyBatis 直接让数据库执行 SQL 脚本的对象。
- ☑ ParameterHandler：MyBatis 为了实现 SQL 带入参数而设置的对象。
- ☑ ResultSetHandler：MyBatis 把 ResultSet 集合映射成 POJO 的接口对象。

MyBatisPlus 依据 MyBatis 插件机制，为程序开发人员提供了 PaginationInnerInterceptor、BlockAttackInnerInterceptor、OptimisticLockerInnerInterceptor 等常用的插件，以便在实际开发中使用。不难发现，这些插件都实现了 InnerInterceptor 接口。

- ☑ PaginationInnerInterceptor：用于实现自动分页的插件。
- ☑ BlockAttackInnerInterceptor：用于防止全表更新与删除的插件
- ☑ OptimisticLockerInnerInterceptor：用于实现乐观锁的插件。

在明确分页插件的出处后，下面将介绍分页插件的配置过程。因为 PageConfig 类是分页插件配置类，所以须使用@Configuration 注解标注 PageConfig 类。在 PageConfig 类中，有一个用于返回 MybatisPlus 插件对象的 mybatisPlusInterceptor()方法。在这个方法中，首先创建一个 MybatisPlus 插件对象，然后让这个 MybatisPlus 插件对象实现自动分页的功能。com.mr.config 包下的 PageConfig 类的代码如下：

```
package com.mr.config;

import com.baomidou.mybatisplus.extension.plugins.MybatisPlusInterceptor;
import com.baomidou.mybatisplus.extension.plugins.inner.PaginationInnerInterceptor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class PageConfig {                                                    //分页插件配置类
    @Bean
    public MybatisPlusInterceptor mybatisPlusInterceptor() {
        MybatisPlusInterceptor interceptor = new MybatisPlusInterceptor();   //MybatisPlus 插件对象
        interceptor.addInnerInterceptor(new PaginationInnerInterceptor());   //配置分页插件
        return interceptor;
    }
}
```

1.8 查询电瓶车品牌信息模块设计

在 db_e-bike 库的 bike 表中一共有 10 条电瓶车的品牌信息，这些信息将被分页显示在主页面上。其中，主页面上的第 1 个分页显示了 7 条数据，如图 1.6 所示；主页面上的第 2 个分页显示了 3 条数据，如图 1.7 所示。下面将介绍查询电瓶车品牌信息模块的实现过程。

1.8.1 前端设计

如图 1.6 和图 1.7 所示，电瓶车的品牌信息都显示在了一个表格模型中，这个表格模型的表头分别为编号、品牌名称、品牌评分、好评率、品牌介绍和操作。在分页插件的作用下，这些信息会被分页显示在主页面上。在第 1.7.1 节中，分页插件的相关数据（如当前页码、每页显示的记录数、总记录数等）虽然

编号	品牌名称	品牌评分	好评率	品牌介绍	操作
1	九号	98.5分	97%	质量好、耐用、驾驶体验与安全性能尤为出色	删除
2	雅迪	95.0分	97%	注重品质和外观设计、超长续航	删除
3	小牛电动	92.6分	96%	强调智能化体验和创新科技，具有独特的改装文化	删除
4	绿源	82.0分	98%	注重安全和科技研发	删除
5	台铃	78.7分	99%	主推节能和长续航	删除
6	新日	76.1分	100%	曾经的行业龙头	删除
7	五羊-本田	69.4分	97%	亲民耐用、主打轻便	删除

图 1.6　主页面第 1 个分页的效果图

编号	品牌名称	品牌评分	好评率	品牌介绍	操作
1	小刀	65.6分	92%	爬坡很猛	删除
2	爱玛	63.1分	99%	智能化和舒适性方面表现突出	删除
3	凤凰	51.7分	91.2%	品质过硬、价格亲民	删除

图 1.7　主页面第 2 个分页的效果图

已经被初始化，但是需要以电瓶车的品牌信息为依据予以重置。代码如下：

```
<el-table size="small" current-row-key="id" :data="dataList" stripe highlight-current-row>
    <el-table-column type="index" align="center" label="编号"></el-table-column>
    <el-table-column prop="brandName" label="品牌名称" align="center"></el-table-column>
    <el-table-column prop="brandRating" label="品牌评分" align="center"></el-table-column>
    <el-table-column prop="favorableRate" label="好评率" align="center"></el-table-column>
    <el-table-column prop="brandIntro" label="品牌介绍" align="center"></el-table-column>
    <el-table-column label="操作" align="center">
        <template slot-scope="scope">
            <el-button type="danger" size="mini" @click="handleDelete(scope.row)">删除</el-button>
        </template>
    </el-table-column>
</el-table>

<!--分页插件-->
<div class="pagination-container">
    <el-pagination
            class="pagiantion"
            @current-change="handleCurrentChange"
            :current-page="pagination.currentPage"
            :page-size="pagination.pageSize"
            layout="total, prev, pager, next, jumper"
            :total="pagination.total">
    </el-pagination>
</div>
```

Vue.js 在创建组件实例时会调用 getAll()方法和 handleCurrentChange()方法。getAll()方法用于分页查询，通过对 db_e-bike 库的 bike 表执行查询操作，获取其中所有的电瓶车品牌信息（10 条），并以此为依据确定分页插件的“当前页码”“每页显示的记录数”和“总记录数”，即“当前页码”为 1、“每页显示的记录数”为 7、“总记录数”为 10。因此，10 条电瓶车品牌信息将被分页插件分为 2 页，第 1 个分页有 7 条数据，第 2 个分页有 3 条数据。handleCurrentChange()方法用于切换页面，既可以从第 1 个分页切换至第 2 个分页，也可以从第 2 个分页切换至第 1 个分页。代码如下：

```
<script>
    var vue = new Vue({
        //省略初始化分页插件的代码（详见第 1.7.1 节）
```

```
        methods: {
            //分页查询
            getAll() {
                param = "?"
                param += "brandName="+this.pagination.brandName;
                param += "&brandRating="+this.pagination.brandRating;
                param += "&favorableRate="+this.pagination.favorableRate;
                param += "&brandIntro="+this.pagination.brandIntro;
                //发送异步请求
                axios.get("/bikes/"+this.pagination.currentPage+"/"+this.pagination.pageSize+param).then((res) => {
                    this.pagination.pagesize = res.data.data.size;
                    this.pagination.currentPage = res.data.data.current;
                    this.pagination.total = res.data.data.total;
                    this.dataList = res.data.data.records;
                });
            },
            //切换页码
            handleCurrentChange(currentPage) {
                //修改页面值为当前选中页码值
                this.pagination.currentPage = currentPage;
                //执行查询
                this.getAll();
            },
            //省略其他方法代码
        }
    })
</script>
```

1.8.2 后端设计

查询模块的后端设计主要包括查询模块的控制器类设计和服务类设计。下面将分别介绍如何设计查询模块的控制器类和服务类。

1. 控制器类设计

本项目的 BikeController 类为控制器类，被@RestController 注解标注。在 BikeController 类中，定义了两个方法，它们分别是 getAll()方法和 getPage()方法。其中，getAll()方法用于查询所有的电瓶车品牌信息；getPage()方法用于获取显示数据的某个分页。getAll()方法和 getPage()方法的代码分别如下：

```
@GetMapping
public R getAll() {
    return new R(true, bikeService.list());                    //查询所有电瓶车的品牌信息
}

@GetMapping("{currentPage}/{pageSize}")
public R getPage(@PathVariable int currentPage, @PathVariable int pageSize, Bike bike) {
    IPage<Bike> page = bikeService.getPage(currentPage, pageSize, bike);    //获取显示数据的某个分页
    //如果当前页码值大于总页码值，那么重新执行操作，使最大页码值为当前页码
    if (currentPage > page.getPages()) {
        page = bikeService.getPage((int) page.getPages(), pageSize, bike);
    }
    return new R(true, page);
}
```

@GetMapping 用于处理 get 请求，通常在查询数据时使用，@GetMapping 的语法如下：

```
@GetMapping("path")
```

@GetMapping 等价于处理 get 请求的@RequestMapping，@RequestMapping 的语法如下：

```
@RequestMapping(value = "path" , method = RequestMethod.GET)
```

2. 服务类设计

本项目的 BikeService 接口为服务接口。在 BikeService 接口中，定义了一个用于获取显示数据的某个分页的 getPage()方法。在 getPage()方法中，有 3 个参数，它们分别为 int 类型的表示“当前页码”的

currentPage、int 类型的表示“每页显示的记录数”的 pageSize、Bike 类型的表示“电瓶车对象（内含电瓶车品牌信息）”的 bike。此外，该方法具有返回值，返回值是显示“电瓶车对象（内含电瓶车品牌信息）”的某个分页。getPage()方法的代码如下：

```
IPage<Bike> getPage(int currentPage, int pageSize, Bike bike);                    //获取用于显示数据的某个分页
```

本项目的 BikeServiceImpl 类是 BikeService 接口的实现类，被@Service 注解标注，即服务类。在 BikeServiceImpl 类中，重写了 BikeService 接口中的 getPage()方法。该方法的主要作用是调用 DAO 层（数据访问对象）执行数据库操作。重写后的 getPage()方法的代码如下：

```
@Override
public IPage<Bike> getPage(int currentPage, int pageSize, Bike bike) {                    //获取用于显示数据的某个分页
    LambdaQueryWrapper<Bike> lqw = new LambdaQueryWrapper<Bike>();
    lqw.like(Strings.isNotEmpty(bike.getBrandName()), Bike::getBrandName, bike.getBrandName());
    lqw.like(Strings.isNotEmpty(bike.getBrandRating()), Bike::getBrandRating, bike.getBrandRating());
    lqw.like(Strings.isNotEmpty(bike.getFavorableRate()), Bike::getFavorableRate, bike.getFavorableRate());
    lqw.like(Strings.isNotEmpty(bike.getBrandIntro()), Bike::getBrandIntro, bike.getBrandIntro());
    IPage page = new Page(currentPage,pageSize);
    bikeDao.selectPage(page,lqw);
    return page;
}
```

在上述代码中，LambdaQueryWrapper 是 MyBatisPlus 中的一个功能类，用于构建 lambda 表达式风格的查询条件，它提供了类型安全的条件构造器，可以减少编写字段名称的错误。

1.9 新增电瓶车品牌信息模块设计

如图 1.8 所示，在主页面的头部有一个“新增电瓶车品牌信息”按钮。用户单击这个按钮，程序将弹出“新增电瓶车品牌信息”窗口，如图 1.9 所示。在这个窗口中，用户依次输入电瓶车的品牌名称、品牌评分、好评率和品牌介绍等信息，单击“确定”按钮，完成新增电瓶车品牌信息的操作。下面将介绍新增电瓶车品牌信息模块的实现过程。

图 1.8 主页面头部的效果图

图 1.9 “新增电瓶车品牌信息”窗口的效果图

1.9.1 前端设计

bikes.html 是本项目的主页面，因此在 bikes.html 的头部添加“新增电瓶车品牌信息”按钮，代码如下：

```
<div class="filter-container">
    <el-button type="primary" class="butT" @click="handleCreate()">新增电瓶车品牌信息</el-button>
</div>
```

用户单击“新增电瓶车品牌信息”按钮，程序将弹出“新增电瓶车品牌信息”窗口。在这个窗口中，包含 4 个标签、3 个文本框、1 个文本域、1 个“取消”按钮和 1 个“确定”按钮。代码如下：

```
<div class="add-form">
    <el-dialog title="新增电瓶车品牌信息" :visible.sync="dialogFormVisible">
        <el-form ref="dataAddForm" :model="formData" :rules="rules" label-position="right"
                 label-width="100px">
            <el-row>
                <el-col :span="12">
                    <el-form-item label="品牌名称" prop="brandName">
                        <el-input v-model="formData.brandName"/>
                    </el-form-item>
                </el-col>
            </el-row>

            <el-row>
                <el-col :span="12">
                    <el-form-item label="品牌评分" prop="brandRating">
                        <el-input v-model="formData.brandRating"/>
                    </el-form-item>
                </el-col>
                <el-col :span="12">
                    <el-form-item label="好评率" prop="favorableRate">
                        <el-input v-model="formData.favorableRate"/>
                    </el-form-item>
                </el-col>
            </el-row>

            <el-row>
                <el-col :span="24">
                    <el-form-item label="品牌介绍">
                        <el-input v-model="formData.brandIntro" type="textarea"></el-input>
                    </el-form-item>
                </el-col>
            </el-row>
        </el-form>

        <div slot="footer" class="dialog-footer">
            <el-button @click="cancel()">取消</el-button>
            <el-button type="primary" @click="handleAdd()">确定</el-button>
        </div>
    </el-dialog>
</div>
```

Vue.js 在创建组件实例时会分别调用 handleCreate()方法、resetForm()方法、handleAdd()方法和 cancel()方法。其中，handleCreate()方法用于显示“新增电瓶车品牌信息”窗口；resetForm()方法用于重置表格模型中的数据；handleAdd()方法的作用是，用户在“新增电瓶车品牌信息”窗口中依次输入电瓶车的品牌名称、品牌评分、好评率和品牌介绍等信息，单击“确定”按钮，如果操作成功，那么表格模型中的数据将被重置，进而新增的电瓶车品牌信息会显示在第 2 个分页上；cancel()方法的作用是，如果用户单击窗口中的“取消”按钮，那么窗口将被关闭，并弹出“当前操作取消”的信息。代码如下：

```
<script>
    var vue = new Vue({
        //省略初始化分页插件的代码（详见第 1.7.1 节）
        methods: {
            //省略第 1.8.1 节中的代码
```

```
            //弹出添加窗口
            handleCreate() {
                this.dialogFormVisible = true;
                this.resetForm();
            },
            //重置表单
            resetForm() {
                this.formData = {};
            },
            //添加
            handleAdd() {
                axios.post("/bikes", this.formData).then((res) => {
                    //判断添加是否成功
                    if (res.data.flag) {
                        //关闭弹窗
                        this.dialogFormVisible = false;
                        this.$message.success(res.data.msg);
                    } else {
                        this.$message.error(res.data.msg);
                    }
                }).finally(() => {
                    //重新加载数据
                    this.getAll();
                });
            },
            //取消
            cancel() {
                this.dialogFormVisible = false;
                this.dialogFormVisible4Edit = false;
                this.$message.info("当前操作取消");
            },
            //省略其他方法的代码
        }
    })
</script>
```

1.9.2 后端设计

新增模块的后端设计主要包括新增模块的控制器类设计和服务类设计。下面将分别介绍如何设计新增模块的控制器类和服务类。

1. 控制器类设计

在 BikeController 类（控制器类）中，定义了一个 insert()方法，该方法用于判断是否成功地执行了新增电瓶车品牌信息的操作。如果操作成功，就返回“添加成功”信息；否则，就返回“添加失败”信息。insert()方法的代码如下：

```
@PostMapping
public R insert(@RequestBody Bike bike) {
    boolean flag = bikeService.insertBike(bike);                    //是否成功执行新增电瓶车品牌信息的操作
    return new R(flag, flag ? "添加成功" : "添加失败");
}
```

@PostMapping 用于处理 post 请求，通常在新增数据时使用，@PostMapping 的语法如下：

```
@PostMapping("path")
```

@PostMapping 等价于处理 post 请求的@RequestMapping，@RequestMapping 的语法如下：

```
@RequestMapping(value = "path", method = RequestMethod.POST)
```

2. 服务类设计

在 BikeService 接口（服务接口）中，定义了一个 insertBike()方法，该方法用于判断是否成功地执行了新增电瓶车品牌信息的操作。在 insertBike()方法中，有 1 个参数，即 Bike 类型的表示“电瓶车对象（内含电瓶车品牌信息）”的 bike。此外，该方法具有返回值，返回值是一个布尔值。insertBike()方法的代码如下：

```
boolean insertBike(Bike bike);                                    //是否执行新增电瓶车品牌信息的操作
```

在 BikeServiceImpl 类（服务类）中，重写了 BikeService 接口中的 insertBike()方法。该方法的主要作用是调用 DAO 层（数据访问对象）执行数据库操作。重写后的 insertBike()方法的代码如下：

```
@Override
public boolean insertBike(Bike bike) {                            //是否成功执行新增电瓶车品牌信息的操作
    return bikeDao.insert(bike)>0;
}
```

1.10 删除电瓶车品牌信息模块设计

如图 1.6 和图 1.7 所示，在主页面上显示的每一条数据的后面，都有一个“删除”按钮，用户先单击某一个“删除”按钮，再单击删除提示窗口中的“确定”按钮（如图 1.10 所示），程序将删除与这个“删除”按钮对应的电瓶车品牌信息，以完成删除电瓶车品牌信息的操作。下面将介绍删除电瓶车品牌信息模块的实现过程。

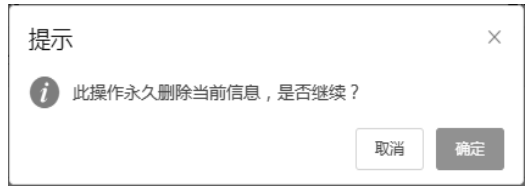


图 1.10 删除提示窗口的效果图

1.10.1 前端设计

在第 1.8.1 节中，已经设计完成了表格模型中的表头。在表头操作中，已经添加了“删除”按钮。为了让“删除”按钮发挥作用，Vue.js 在创建组件实例时会调用 handleDelete()方法：用户在单击“删除”按钮后，如果继续单击删除提示窗口中的“确定”按钮，程序将删除与这个“删除”按钮对应的电瓶车品牌信息，并且表格模型中的数据会被重置；如果单击删除提示窗口中的“取消”按钮，那么窗口将被关闭，并弹出“取消操作”的信息。代码如下：

```
<script>
    var vue = new Vue({
        //省略初始化分页插件的代码（详见第 1.7.1 节）
        methods: {
            // 省略第 1.8.1 节和第 1.9.1 节中的代码
            // 删除
            handleDelete(row) {
                this.$confirm("此操作永久删除当前信息，是否继续？", "提示", {type: "info"}).then((res) => {
                    axios.delete("/bikes/" + row.id).then((res) => {
                        if (res.data.flag) {
                            this.$message.success("删除成功");
                        } else {
                            this.$message.error("数据同步失败，自动刷新");
                        }
                    }).finally(() => {
                        //重新加载数据
                        this.getAll();
                    });
                }).catch(() => {
                    this.$message.info("取消操作");
                });
            },
        }
    })
</script>
```

1.10.2 后端设计

删除模块的后端设计主要包括删除模块的控制器类设计和服务类设计。下面将分别介绍如何设计删除模块的控制器类和服务类。

1. 控制器类设计

在 BikeController 类（控制器类）中，定义了一个用于根据电瓶车的品牌编号删除电瓶车品牌信息的 delete()方法。delete()方法的代码如下：

```
@DeleteMapping("{id}")
public R delete(@PathVariable Integer id) {
        return new R(bikeService.deleteBike(id));                    //删除电瓶车的品牌信息
}
```

@DeleteMapping 用于处理 delete 请求，通常在删除数据时使用，@DeleteMapping 的语法如下：

```
@DeleteMapping("path")
```

@GetMapping 等价于处理 delete 请求的@RequestMapping，@RequestMapping 的语法如下：

```
@RequestMapping(value = "path",method = RequestMethod.DELETE)
```

2. 服务类设计

在 BikeService 接口（服务接口）中，定义了一个 deleteBike()方法，该方法用于判断是否成功地执行了删除电瓶车品牌信息的操作。在 insertBike()方法中，有 1 个参数，即 Integer 类型的表示“电瓶车的品牌编号”的 id。此外，该方法具有返回值，返回值是一个布尔值。deleteBike()方法的代码如下：

```
boolean deleteBike(Integer id);                                      //是否执行删除电瓶车品牌信息的操作
```

在 BikeServiceImpl 类（服务类）中，重写了 BikeService 接口中的 deleteBike()方法。该方法的主要作用是调用 DAO 层（数据访问对象）来执行数据库操作。重写后的 deleteBike()方法的代码如下：

```
@Override
public boolean deleteBike(Integer id) {                              //是否成功执行删除电瓶车品牌信息的操作
    return bikeDao.deleteById(id)>0;
}
```

1.11 项 目 运 行

通过前述步骤，我们设计并完成了“电瓶车品牌信息管理系统”项目的开发。下面运行本项目，检验一下我们的开发成果。如图 1.11 所示，在 IntelliJ IDEA 中，单击▶快捷图标，即可运行本项目。

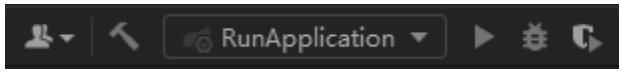


图 1.11 IntelliJ IDEA 的快捷图标

成功运行本项目，打开浏览器，访问 http://localhost:8080/pages/bikes.html，即可看到如图 1.12 所示的电瓶车品牌信息管理系统的主页面。

主页面一共显示 10 条数据。因为一个分页只能显示 7 条数据，所以需要两个分页，即第 1 个分页显示 7 条数据，第 2 个分页显示 3 条数据。用户通过分页导航可以随意切换并访问这两个分页的数据。

用户在单击“新增电瓶车品牌信息”按钮后，需要先在弹出的窗口中依次输入电瓶车的品牌名称、品牌评分、好评率和品牌介绍等信息，再单击“确定”按钮，进而完成新增电瓶车品牌信息的操作。

在主页面上显示的每一条数据的后面，都有一个“删除”按钮，用户先单击某一个“删除”按钮，再单击删除提示窗口中的“确定”按钮，进而完成删除电瓶车品牌信息的操作。

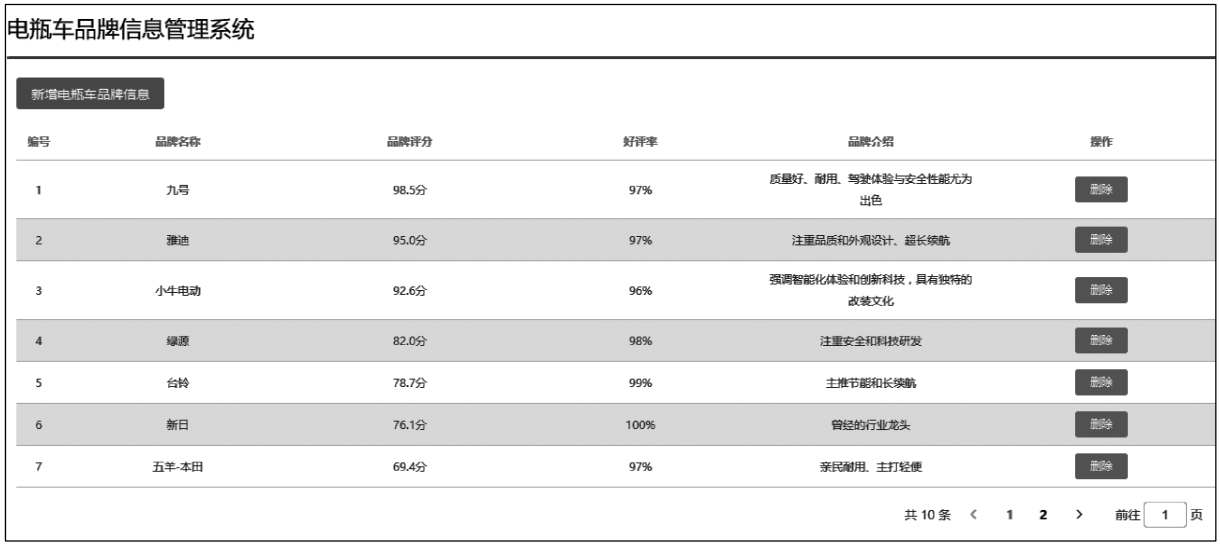


图 1.12　电瓶车品牌信息管理系统的主页面

这样，我们就成功地检验了本项目的运行。

本项目比较简单，虽然只应用了单表查询，但是麻雀虽小，五脏俱全。本项目使用 Vue.js 实现了前端页面、使用 Spring Boot 实现了后端业务逻辑。Vue.js 在创建组件实例时会调用特定的方法，进而达到渲染页面的目的。Spring Boot 在处理业务逻辑时层次分明、结构清晰，Controller 层主要负责具体的业务模块流程的控制，Service 层主要负责业务模块的逻辑应用设计，DAO 层主要负责与数据库进行联络。

1.12　源 码 下 载

虽然本章详细地讲解了如何编码实现“电瓶车品牌信息管理系统”项目的各个功能，但给出的代码都是代码片段，而非源码。为了方便读者学习，本书提供了完整的项目源码，扫描右侧二维码即可下载。

源码下载