商城端和商城后台管理系统都会用到分页功能。在讲解具体的功能模块开发之前，先介绍一下分页功能的知识点。

分页是网站系统非常重要也十分常用的功能。在MVC开发模式下开发人员通常把它放在ModelAndView对象中，然后在页面代码中循环遍历列表数据并渲染到网站页面上。它也可以通过接口返回实现，即在前端通过Ajax调用接口数据并通过插件来实现分页数据的渲染和翻页功能。这两种方式的实现代码本书都会讲解。在商城项目中有多个页面会用到分页功能，不同的页面会使用不同的交互方式，但它们的分页原理都是一样的。

## 19.1 分页功能介绍

分页存在于在各类电商网站、新闻网站、音乐网站，以及各类后台管理系统中。这里笔者整理了一些常用网站的分页展示效果。

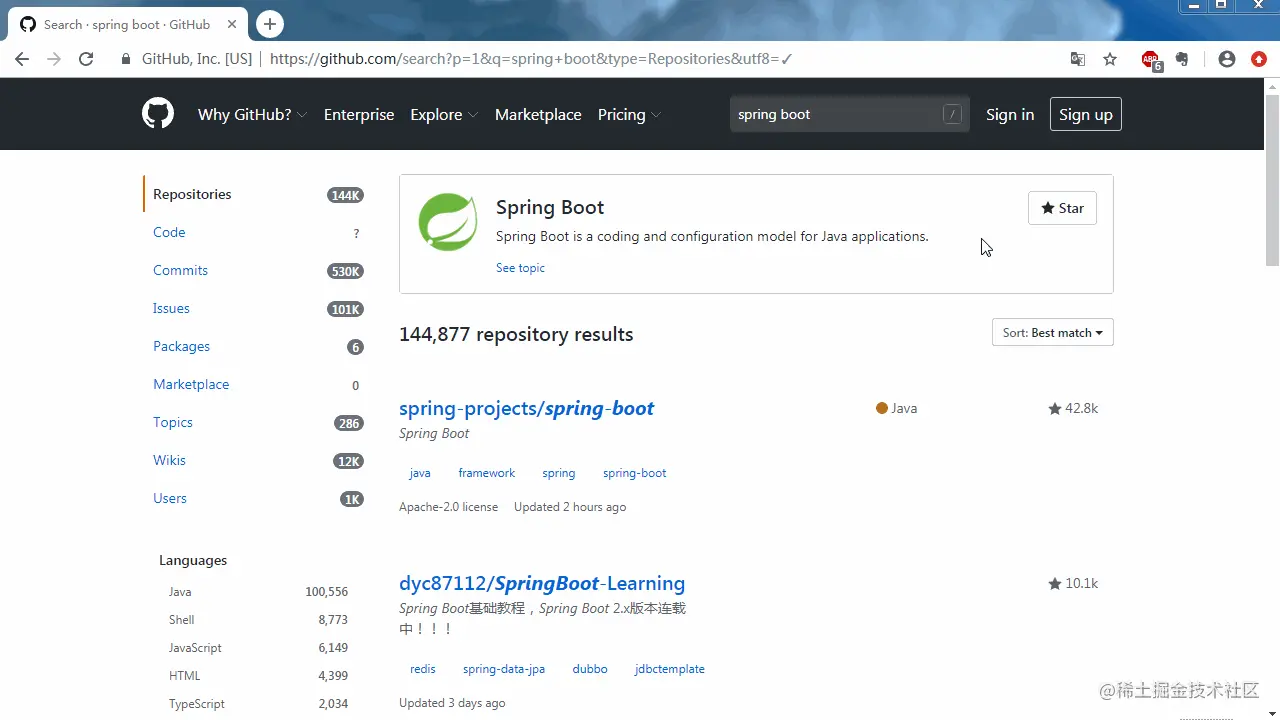
### 19.1.1 百度分页

在百度首页搜索框中输入 "java" 查询相关信息之后跳转到搜索结果页面，页面中大致会有 10 条左右的数据列表，此时展示的是第 1 页的数据，如果想看后面的搜索内容点击页面下方的分页信息即可，比如点击第 6 页或者下一页的按钮就可以看到更多的信息了。



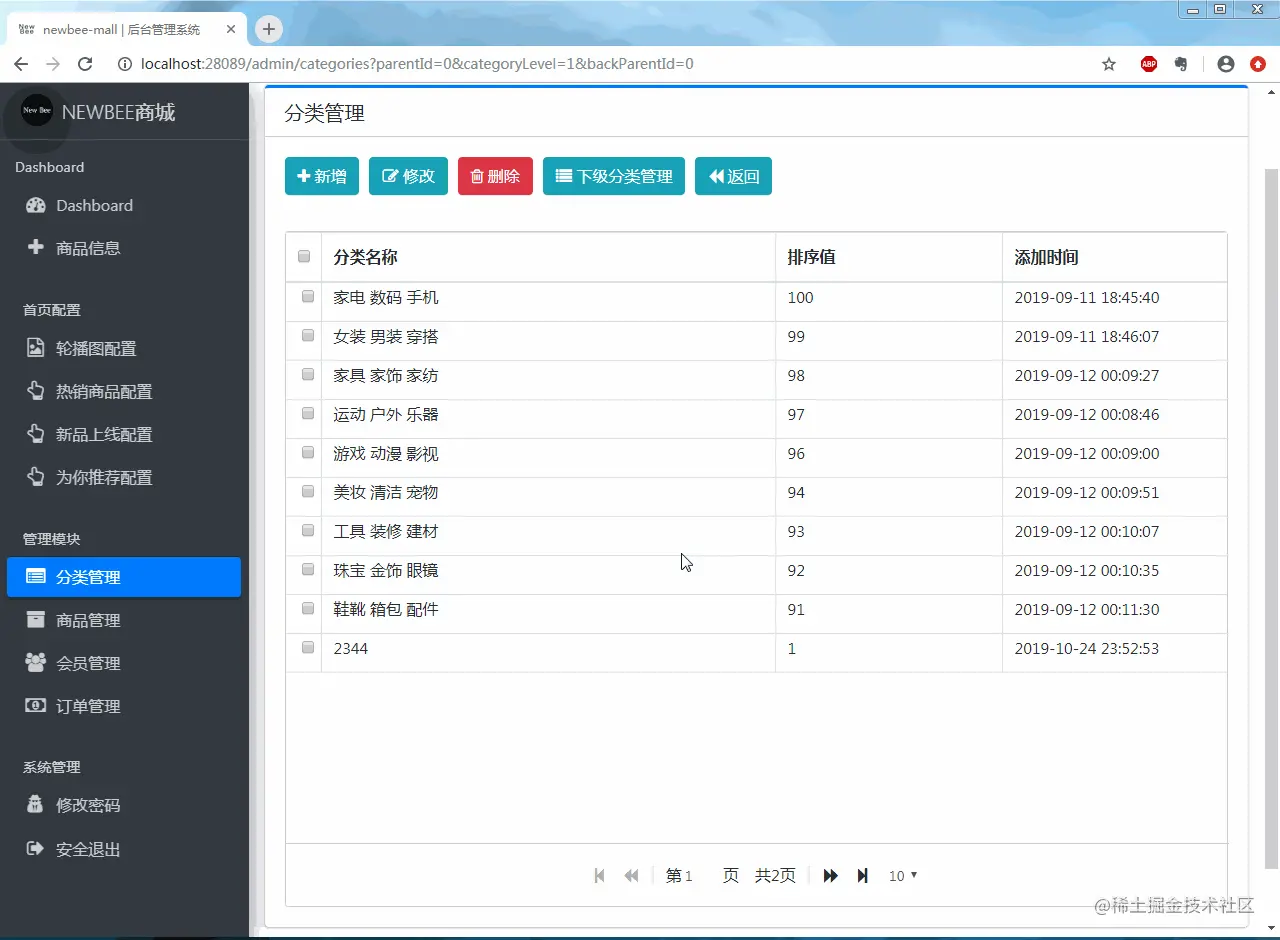
### 19.1.2 GitHub 分页功能

在 GitHub 页面的搜索框中输入 "spring boot" 查询相关代码仓库，搜索结果页面如下，页面中有 10 条左右的仓库数据列表，当前展示的是第 1 页的数据，如果想看后面的搜索内容点击页面下方的分页信息即可，比如点击第 2 页或者第 5 页的分页按钮就可以看到更多的关于 spring boot 的代码仓库了。



### 19.1.3 商城后台管理系统分页

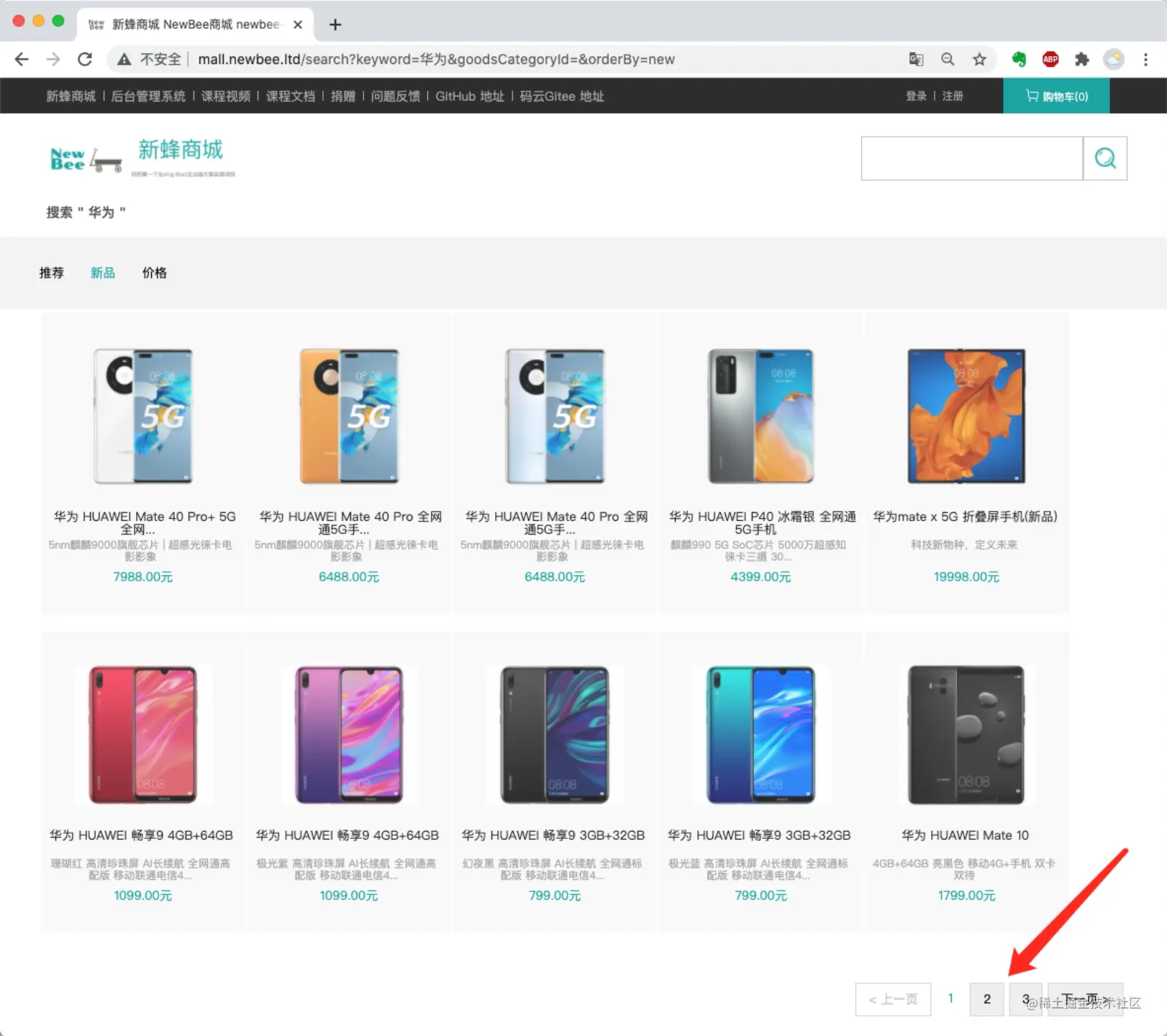
商城后台管理系统中的大部分功能模块中，数据列表页面也并没有把所有的内容都展示出来，而是通过分页功能分别展示出来，比如分类数据和订单数据的管理，如下所示：



### 19.1.4 商品搜索页分页

通常商城系统的商品数量非常大，不可能在一个页面中把商品数据全部展示出来，所以在搜索结果页面也需要加入分页功能的设计。

以新蜂商城为例，在搜索框输入关键字“华为”，查询出与华为相关的商品，然后可以在搜索结果页面看到对应的商品。该页面也用分页功能展示了部分商品，每一页展示10条商品数据，可以通过页面底部的数字或翻页按钮来查看更多的商品信息，如下图所示。



当然，商城系统还有其他功能模块也用到了分页功能，在后续开发过程中都会逐一讲解。

## 19.2 分页的作用

分页功能作为各类网站和系统不可或缺的部分，其作用有5个。

（1）减少系统资源的消耗

数据查询结果是放在系统内存中的，如果在数据量很大的情况下一次性将所有内容都显示出来，会占用过多内存，而通过分页可以减少这种系统资源的消耗。

（2）提高数据库的查询性能

后端服务与数据库间通过网络传输数据，一次传输10条数据结果集与一次传输2万条数据结果集相比，肯定是前者消耗的网络资源更少。

（3）提升页面的访问速度

浏览器与后端服务间的传输也是通过网络进行的。因为数据包的大小有差别，返回10条数据明显要比返回2万条数据速度更快。

（4）符合用户的浏览习惯

以搜索结果或者商品展示为例，用户通常只查看展示在前面的部分数据，而将所有数据都展示出来不符合用户浏览习惯。

（5）适配页面的排版

由于设备屏幕的大小固定，一个屏幕能够展示的信息有限，如果一次展示太多的数据，无论排版还是页面美观度都会被影响。分页的作用也是为了适配页面的排版。

## 19.3 分页功能的设计

### 19.3.1 前端分页功能设计

* GitHub 分页按钮



* 百度 分页按钮



通过这种前端页面分页展示区的设计可以看出，分页展示区比较重要的3个参数：页码展示、当前页码、每页条数。

此外，有些页面也会加上首页、尾页、跳转页码等功能。开发人员根据功能需要和页面设计可以自行增加或删减。

### 19.3.2 后端分页功能设计

分页功能在前端页面的执行是渲染数据和分页信息展示，在后端页面则需要按照前端传输而来的请求，将分页所需的数据正确地查询出来并返回给前端。两端的侧重点并不相同。比如，前端需要展示所有页码，而后端则只需要提供总页数即可，不需要对总页数进行其他操作。再比如前端需要根据用户的操作记录当前页码的参数，以便对页码信息进行调整和限制，而后端只需要接收前端传输过来的页码并进行相应判断和查询即可。

后端分页必不可少的两个参数：页码、条数（每页）。

在实现分页功能时，使用不同的数据库实现方式也不同。因为不同数据库实现分页功能的关键字有些差别，比如SQL Server的top关键字、Oracle的rownum关键字、MySQL的limit关键字。limit关键字如下所示：

//下面是mysql的实现语句：

**select** \* **from** tb\_xxxx limit 10,20

分页功能的最终实现就是通过页码和条数，确定数据库需要查询的数据。比如查询第1页且每页20条的数据，就是查询数据库中从第1到20条的数据；查询第4页且每页10条的数据就是查询数据库中第30到40条的数据。因此，对于后端代码的实现来说，页码和条数两个参数就显得特别重要，缺少这两个参数查询逻辑就不成立，分页数据也就无从查起。

此外，为了前端分页区的展示，还要将数据总量或者总页数返回给前端。数据总量是必不可少的，而总页数可以计算出来（即数据总量除以每页条数）。数据总量的获取方式：

**select** count(\*) **from** tb\_xxxx

之后将数据封装，并返回给前端即可。

## 19.4 分页功能的编码实现

这里实现的分页功能是一个通用的分页接口。笔者将分页数据封装到一个返回结果对象中，并通过JSON格式返回，目的是让读者理解分页功能的主要逻辑和简单的代码实现。

接下来将结合tb\_user表进行简单的查询与分页功能的实现，即在前端请求对应的页数时，返回那一页的所有数据。

### 19.4.1 新增测试数据

为了演示分页功能，需要在表中多增加一些数据，如果数量太少就无法充分模拟分页功能。

执行SQL向数据库中新增一些测试数据，代码如下所示：

*-- ------------------------------ Table structure for tb\_user-- ----------------------------***DROP** **TABLE** IF **EXISTS** `tb\_user`;

**CREATE** **TABLE** `tb\_user` (

`id` **INT**(11) **NOT** **NULL** AUTO\_INCREMENT COMMENT '主键',

`name` **VARCHAR**(100) **NOT** **NULL** **DEFAULT** '' COMMENT '登录名',

`password` **VARCHAR**(100) **NOT** **NULL** **DEFAULT** '' COMMENT '密码',

**PRIMARY** KEY (`id`)

) ENGINE=INNODB AUTO\_INCREMENT=1 **DEFAULT** CHARSET=utf8;

*-- ------------------------------ Records of tb\_user-- ----------------------------***INSERT** **INTO** `tb\_user` **VALUES** (1, 'admin', 'e10adc3949ba59abbe56e057f20f883e');**INSERT** **INTO** `tb\_user` **VALUES** (2, 'test2', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (3, 'test3', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (4, 'test4', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (5, 'test5', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (6, 'test6', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (7, 'test7', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (8, 'test8', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (9, 'test9', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (10, 'test10', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (11, 'test11', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (12, 'test12', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (13, 'test13', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (14, 'test14', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (15, 'test15', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (16, 'test16', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (17, 'test17', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (18, 'test18', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (19, 'test19', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (20, 'admin2', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (21, 'admin3', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (22, 'admin4', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (23, 'admin5', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (24, 'admin6', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (25, 'admin7', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (26, 'admin8', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (27, 'admin9', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (28, 'admin10', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (29, 'admin11', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (30, 'admin12', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (31, 'admin13', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (32, 'admin14', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (33, 'admin15', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (34, 'admin16', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (35, 'admin17', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (36, 'admin18', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (37, 'admin19', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (38, 'admin011', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (39, 'admin02', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (40, 'admin03', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (41, 'admin04', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (42, 'admin05', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (43, 'admin06', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (44, 'admin07', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (45, 'admin08', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (46, 'admin09', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (47, 'admin010', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (48, 'admin011', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (49, 'admin012', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (50, 'admin013', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (51, 'admin014', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (52, 'admin015', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (53, 'admin016', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (54, 'admin017', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (55, 'admin018', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (56, 'admin019', '098f6bcd4621d373cade4e832627b4f6');**INSERT** **INTO** `tb\_user` **VALUES** (57, 'ZHENFENG13', '77c9749b451ab8c713c48037ddfbb2c4');**INSERT** **INTO** `tb\_user` **VALUES** (58, '213312', 'eqwfasdfa');**INSERT** **INTO** `tb\_user` **VALUES** (59, '14415143', '51435135');**INSERT** **INTO** `tb\_user` **VALUES** (60, 'shisan', 'e10adc3949ba59abbe56e057f20f883e');**INSERT** **INTO** `tb\_user` **VALUES** (61, 'zhangsan', 'fcea920f7412b5da7be0cf42b8c93759');**INSERT** **INTO** `tb\_user` **VALUES** (62, 'test-user1', '3d0faa930d336ba748607ab7076ebce2');**INSERT** **INTO** `tb\_user` **VALUES** (63, '3123213213', '6fdce2f14f4baf2d666fa13dfd8d1945');

### 19.4.2 分页功能返回结果的封装

在之前的文章中，返回结果的数据格式都比较简单，如果遇到复杂的数据类型则需要修改代码。比如这里的分页数据结果，就需要返回多个字段。在实际的项目开发工作中开发人员通常会将返回结果进行抽象并封装成一个常用的Java类。

新建util包并在util包中新建Result通用结果类，代码如下所示：

**package** ltd.newbee.mall.util;

**import** java.io.Serializable;

*/\*\**

*\* @author 13*

*\* @qq交流群 796794009*

*\* @email 2449207463@qq.com*

*\* @link https://github.com/newbee-ltd*

*\*/***public** **class** **Result**<T> **implements** **Serializable** {

**private** **static** **final** **long** serialVersionUID = 1L;

**private** **int** resultCode;

**private** String message;

**private** T data;

**public** **Result**() {

}

**public** **Result**(**int** resultCode, String message) {

this.resultCode = resultCode;

this.message = message;

}

**public** **int** **getResultCode**() {

**return** resultCode;

}

**public** **void** **setResultCode**(**int** resultCode) {

this.resultCode = resultCode;

}

**public** String **getMessage**() {

**return** message;

}

**public** **void** **setMessage**(String message) {

this.message = message;

}

**public** T **getData**() {

**return** data;

}

**public** **void** **setData**(T data) {

this.data = data;

}

**@Override**

**public** String **toString**() {

**return** "Result{" +

"resultCode=" + resultCode +

", message='" + message + '\'' +

", data=" + data +

'}';

}

}

后端接口返回的数据会根据以上格式进行数据封装，包括业务码、返回信息、实际的数据结果。这个格式是开发人员自行设置的，如果有其他更好的方案也可以进行适当的调整。如要为了返回结果的数据统一，前端接收该结果后需要对数据进行解析，并通过业务码进行相应的逻辑操作，再获取data属性中的数据并进行页面渲染或者进行信息提示。

实际返回的数据格式示例如下所示。

* 对象列表数据：

{

"resultCode": 200,

"message": "SUCCESS",

"data": [{

"id": 2,

"name": "user1",

"password": "123456"

}, {

"id": 1,

"name": "13",

"password": "12345"

}]}

* 单条对象数据

{

"resultCode": 200,

"message": "SUCCESS",

"data": {

"id": 2,

"name": "user1",

"password": "123456"

}}

* 简单对象数据

{

"resultCode": 200,

"message": "SUCCESS",

"data": **true**}

以上代码分别是不同格式的数据返回，在后端进行业务处理后程序将会返回给前端一串JSON格式的数据。resultCode等于200表示数据请求成功，该字段也可以自行定义，比如 0、1001、500等。message值为SUCCESS，也可以自行定义返回信息，比如获取成功、列表数据查询成功等字符串。这些都需要与前端约定好，一个resultCode码只能表示一种含义。而data属性使用泛型定义，因此在data字段中的数据可以是一个对象数组，也可以是一个字符串、数字等类型，它根据不同的业务返回不同的结果，在接下来的实践内容中都会以这种方式返回数据。

分页结果数据也做了统一的封装，根据前文的分析，分页结果一般需要以下几个字段：总记录数、每页记录数、总页数、当前页数和实际的列表数据。

在util包中新建PageResult类，PageResult分页结果集的数据格式定义如下所示：

**package** ltd.newbee.mall.util;

**import** java.io.Serializable;**import** java.util.List;

*/\*\**

*\* 分页工具类*

*\**

*\* @author 13*

*\* @qq交流群 796794009*

*\* @email 2449207463@qq.com*

*\* @link https://github.com/newbee-ltd*

*\*/***public** **class** **PageResult** **implements** **Serializable** {

*//总记录数*

**private** **int** totalCount;

*//每页记录数*

**private** **int** pageSize;

*//总页数*

**private** **int** totalPage;

*//当前页数*

**private** **int** currPage;

*//列表数据*

**private** List<?> list;

*/\*\**

*\* 分页*

*\**

*\* @param list 列表数据*

*\* @param totalCount 总记录数*

*\* @param pageSize 每页记录数*

*\* @param currPage 当前页数*

*\*/*

**public** **PageResult**(List<?> list, **int** totalCount, **int** pageSize, **int** currPage) {

this.list = list;

this.totalCount = totalCount;

this.pageSize = pageSize;

this.currPage = currPage;

this.totalPage = (**int**) Math.ceil((**double**) totalCount / pageSize);

}

**public** **int** **getTotalCount**() {

**return** totalCount;

}

**public** **void** **setTotalCount**(**int** totalCount) {

this.totalCount = totalCount;

}

**public** **int** **getPageSize**() {

**return** pageSize;

}

**public** **void** **setPageSize**(**int** pageSize) {

this.pageSize = pageSize;

}

**public** **int** **getTotalPage**() {

**return** totalPage;

}

**public** **void** **setTotalPage**(**int** totalPage) {

this.totalPage = totalPage;

}

**public** **int** **getCurrPage**() {

**return** currPage;

}

**public** **void** **setCurrPage**(**int** currPage) {

this.currPage = currPage;

}

**public** List<?> getList() {

**return** list;

}

**public** **void** **setList**(List<?> list) {

this.list = list;

}

}

## 19.5 分页功能代码的具体实现

### 19.5.1 数据层分页代码的实现

由于tb\_user的实体类和Mapper文件已经创建，所以新增对应的方法即可。在UserDao接口中新增两个方法findUsers()和getTotalUser()，代码如下所示：

*/\*\**

*\* 返回分页数据列表*

*\**

*\* @param pageUtil*

*\* @return*

*\*/*

List<User> **findUsers**(PageQueryUtil pageUtil);

*/\*\**

*\* 返回数据总数*

*\**

*\* @param pageUtil*

*\* @return*

*\*/***int** **getTotalUser**(PageQueryUtil pageUtil);

该段代码表示分别根据传入的分页参数获取分页数据列表并返回数据数量。

然后在UserMapper.xml文件中新增这两个方法的映射语句，代码如下所示：

*<!-- 查询用户列表 -->*<select id="findUsers" parameterType="Map" resultMap="UserResult">

select id,name,password from tb\_user

order by id desc

<if test="start!=null and limit!=null">

limit #{start},#{limit}

</if></select>

*<!-- 查询用户总数 -->*<select id="getTotalUser" parameterType="Map" resultType="int">

select count(\*) from tb\_user</select>

### 19.5.2 业务层分页代码的实现

新建service包，并新增业务类UserService，代码如下所示：

**package** ltd.newbee.mall.service;

**import** ltd.newbee.mall.dao.UserDao;**import** ltd.newbee.mall.entity.User;**import** ltd.newbee.mall.util.PageQueryUtil;**import** ltd.newbee.mall.util.PageResult;**import** org.springframework.beans.factory.annotation.Autowired;**import** org.springframework.stereotype.Service;

**import** java.util.List;

**@Servicepublic** **class** **UserService** {

**@Autowired**

**private** UserDao userDao;

**public** PageResult **getUserPage**(PageQueryUtil pageUtil) {

*// 当前页码中的数据列表*

List<User> users = userDao.findUsers(pageUtil);

*// 数据总条数，用于计算分页数据*

**int** total = userDao.getTotalUser(pageUtil);

*// 分页信息封装*

**PageResult** pageResult = **new** **PageResult**(users, total, pageUtil.getLimit(), pageUtil.getPage());

**return** pageResult;

}

}

首先根据当前页面和每页条数查询当前页的数据集合，然后调用select count(\*)语句查询数据的总条数用于计算分页数据，最后将获取的数据封装到PageResult对象中并返回给Controller层。

### 19.5.3 控制层分页代码的实现

在controller包中新建PageTestController类，用于实现分页请求的处理并返回查询结果，代码如下所示：

**package** ltd.newbee.mall.controller;

**import** ltd.newbee.mall.service.UserService;**import** ltd.newbee.mall.util.PageResult;**import** ltd.newbee.mall.util.Result;**import** ltd.newbee.mall.util.PageQueryUtil;**import** org.springframework.beans.factory.annotation.Autowired;**import** org.springframework.util.StringUtils;**import** org.springframework.web.bind.annotation.RequestMapping;**import** org.springframework.web.bind.annotation.RequestMethod;**import** org.springframework.web.bind.annotation.RequestParam;**import** org.springframework.web.bind.annotation.RestController;

**import** java.util.Map;

*/\*\**

*\* @author 13*

*\* @qq交流群 796794009*

*\* @email 2449207463@qq.com*

*\* @link https://github.com/newbee-ltd*

*\*/***@RestController@RequestMapping("/users")public** **class** **PageTestController** {

**@Autowired**

**private** UserService userService;

*/\*\**

*\* 分页功能测试*

*\*/*

**@RequestMapping(value = "/list", method = RequestMethod.GET)**

**public** Result **list**(**@RequestParam** Map<String, Object> params) {

**Result** result = **new** **Result**();

**if** (StringUtils.isEmpty(params.get("page")) || StringUtils.isEmpty(params.get("limit"))) {

*// 返回错误码*

result.setResultCode(500);

*// 错误信息*

result.setMessage("参数异常！");

**return** result;

}

*// 封装查询参数*

**PageQueryUtil** queryParamList = **new** **PageQueryUtil**(params);

*// 查询并封装分页结果集*

**PageResult** userPage = userService.getUserPage(queryParamList);

*// 返回成功码*

result.setResultCode(200);

result.setMessage("查询成功");

*// 返回分页数据*

result.setData(userPage);

**return** result;

}

}

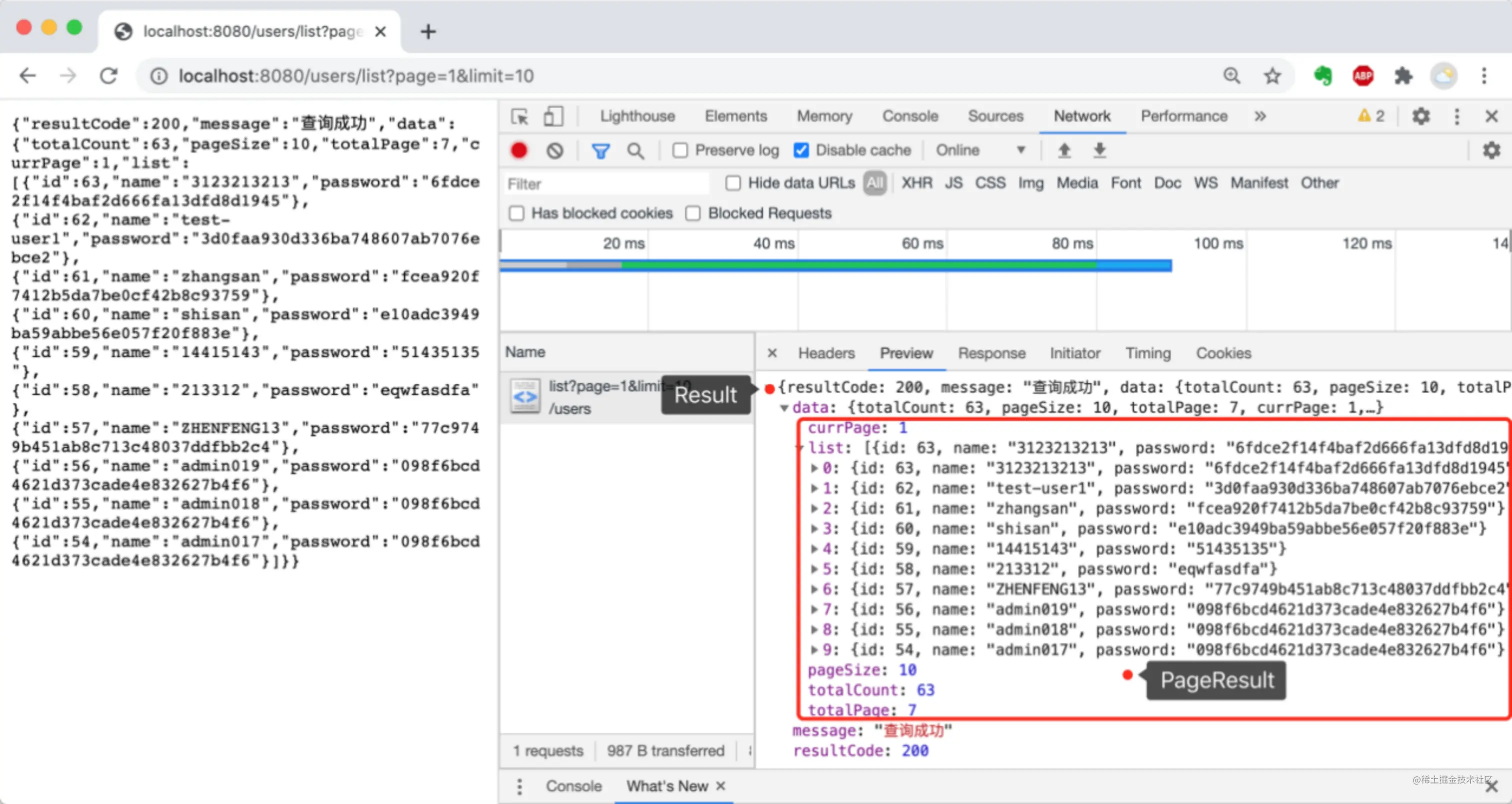
通过后端代码可以看出，分页功能的交互流程：前端将所需页码和条数参数传输给后端，后端在接收分页请求后对分页参数进行计算，并利用MySQL的limit关键字查询对应的记录，在查询结果被封装后返回给前端。在TestUserControler类上使用的是@RestController注解，该注解相当于@ResponseBody＋@Controller的组合注解。

### 19.5.4 分页功能测试

在编码完成后启动Spring Boot项目。在启动成功后打开浏览器，在地址栏输入如下地址：

[http://localhost:8080/users/list?page=1&limit=10](https://link.juejin.cn/?target=http://localhost:8080/users/list?page=1&limit=10" \o "http://localhost:8080/users/list?page=1&limit=10" \t "/home/erfenjiao/文档\\x/_blank)

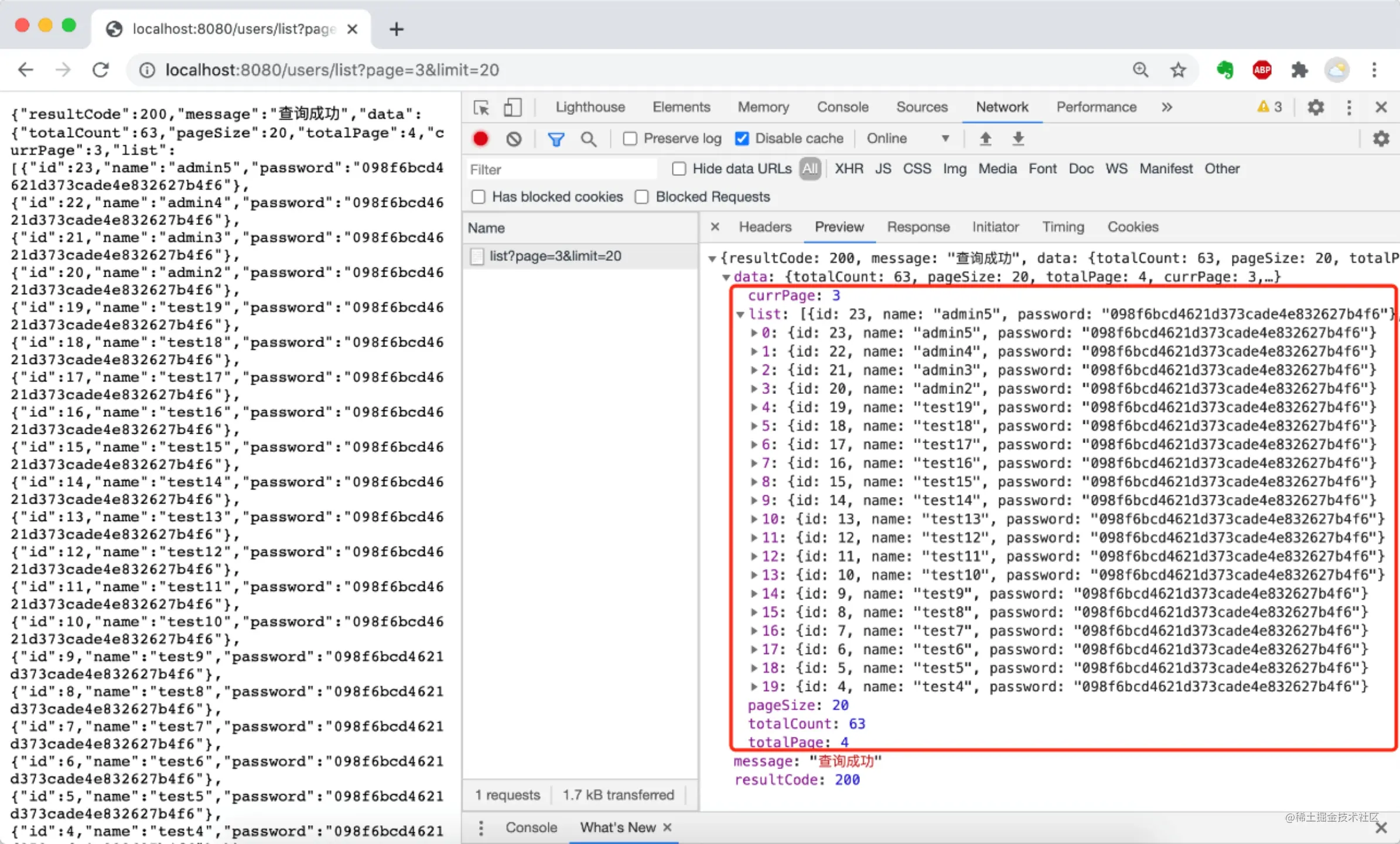
请求参数分别为1和10，表示以每页10条的数据查询第1页的数据。在结果页面中可以看到分页数据的结果返回。接着按住F12键或者右键进入浏览器调试模式，通过开发人员工具查看返回的分页数据结果。在NetWork面板中可以看到这次请求的信息，返回格式是标准的Result对象格式，其中包含resultCode、message和data三个字段。其中data对象中的数据格式为PageResult对象的数据格式。currPage表示当前页，数值为1。后端封装的第1页的10条数据都在list字段中。测试结果及返回的分页数据格式如下图所示。



接下来通过改变page参数和limit参数的值进行分页功能的测试。比如以每页20条的数据查询第3页的数据，则请求URL为：

[http://localhost:8080/users/list?page=3&limit=20](https://link.juejin.cn/?target=http://localhost:8080/users/list?page=3&limit=20" \o "http://localhost:8080/users/list?page=3&limit=20" \t "/home/erfenjiao/文档\\x/_blank)

得到的结果如下图所示。返回的数据格式依然没变，只是具体字段的值有些改动，比如currPage、totalPage、list字段。因为请求的参数不同，返回的分页数据肯定也会有所改动。



读者可以结合该过程进行代码的学习和理解，并自行对比不同参数结果返回的不同。