在线商城开发文档

项目展示

（1）项目结构图如图 9-1 所示。

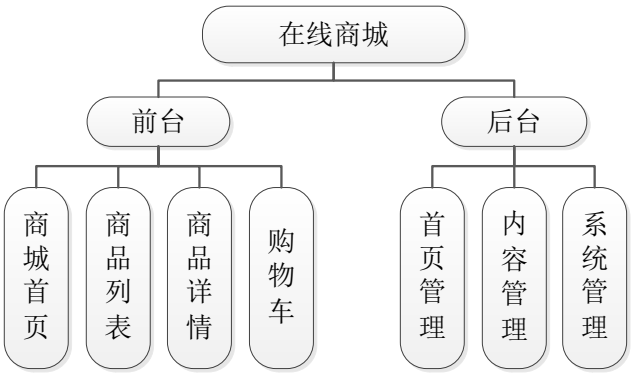

图 9-1　项目结构图

（2）前台首页效果如图 9-2 所示。

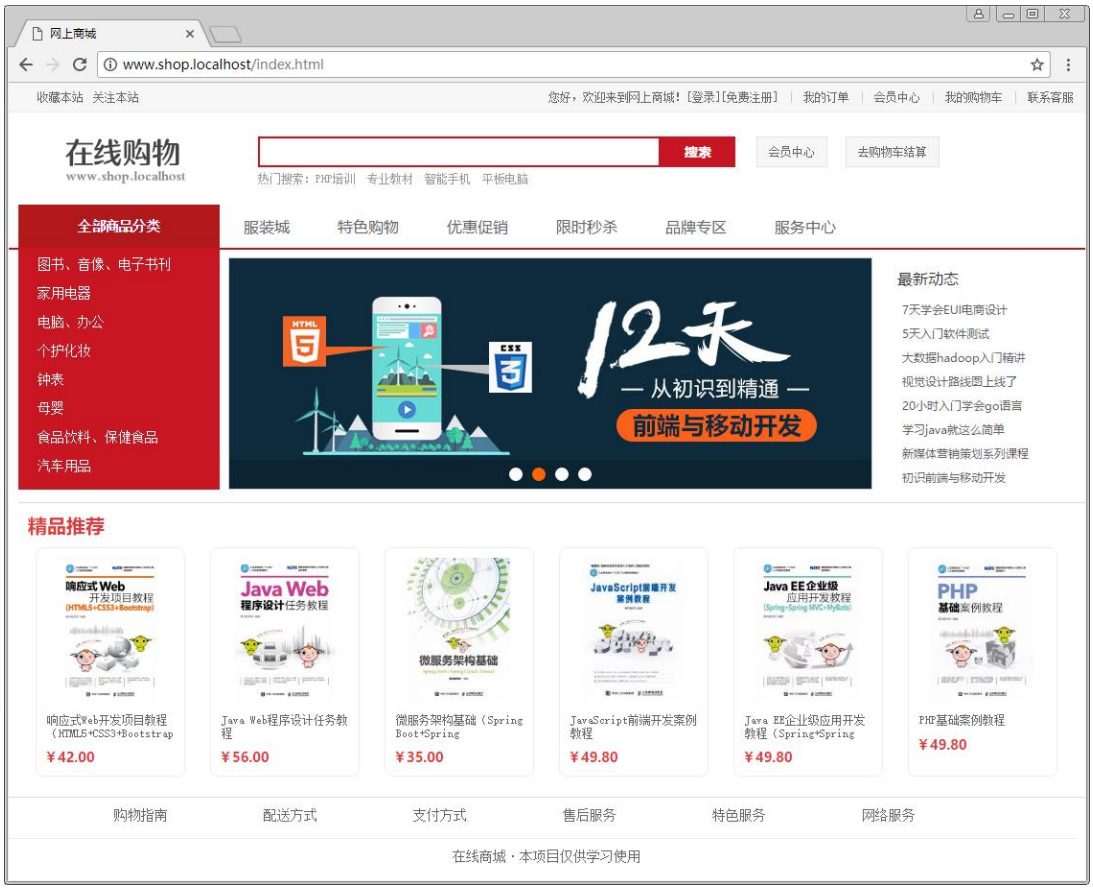

图 9-2　前台首页

（3）项目后台首页效果如图 9-3 所示。

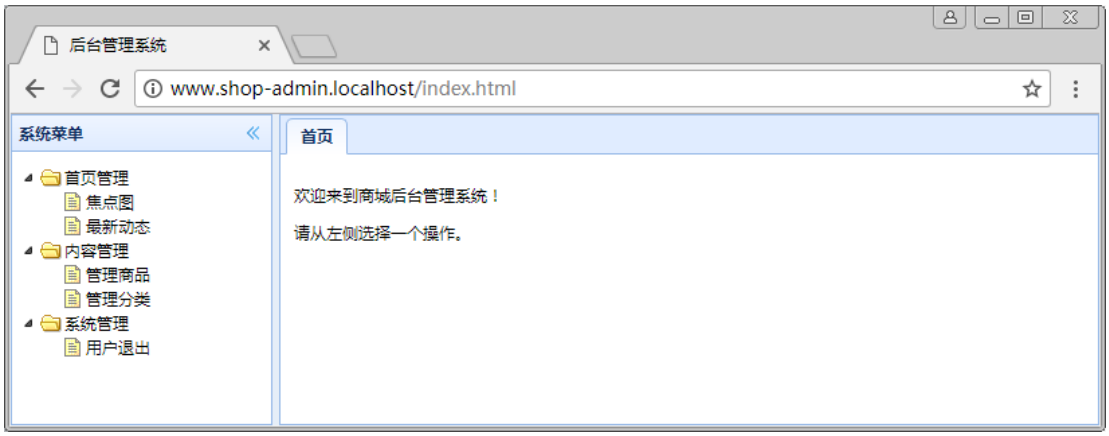

图 9-3　后台首页

（4）前后端交互过程如图 9-4 所示。

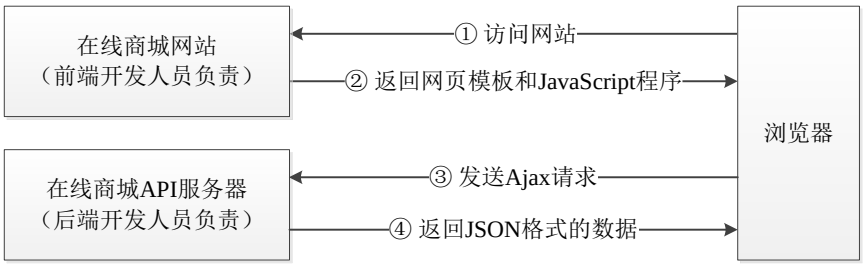

图 9-4　交互过程

【准备工作】搭建开发环境

在前面的章节中，已经讲解了如何安装 WampServer，在本项目的开发中会继续使用该环境。接下来，本节将针对如何在 WampServer 中搭建项目开发环境进行详细讲解。

配置虚拟主机

本项目分为前台和后台，且都需要后端 API，因此总共需要搭建 4 个站点。为了便于学习和测试，下面为这 4 个站点准备不同的域名，从而通过不同的 URL 地址来访问。下面是这 4 个站点的 URL 地址。

- 前台网站：http://www.shop.localhost/
- 前台 API：http://api.shop.localhost/
- 后台网站：http://www.shop-admin.localhost/
- 后台 API：http://api.shop-admin.localhost/

以上 URL 地址中的域名使用了 localhost 后缀，这是因为该后缀在 Chrome 浏览器中会自动解析到本地，从而方便开发测试使用。

若直接在浏览器中访问上述 URL 地址，访问到的是同一个目录，即 C:\wamp\www。为了让这 4 个站点对应的目录分开，就需要借助 Apache 的虚拟主机功能。

Apache 虚拟主机的具体配置步骤如下。

（1）使用鼠标左键单击 WampServer 的通知区域图标，在弹出的功能菜单中依次选择【Apache】→【httpd-vhost.conf】，如图 9-5 所示。

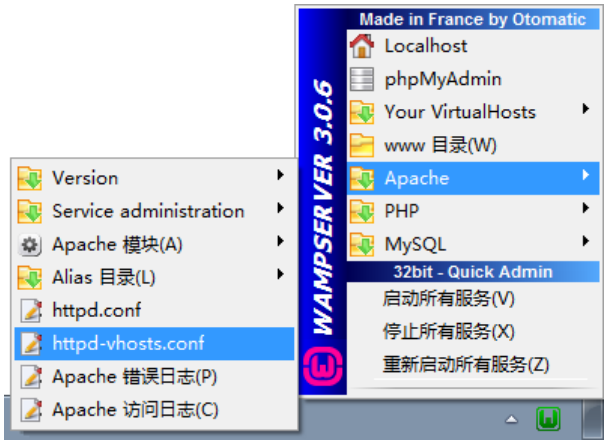

图 9-5　WampServer 功能菜单

执行上述操作后，就会弹出一个编辑器（默认情况下是记事本）自动打开 Apache 的虚拟主机配置文件 httpd-vhost.conf。

（2）在 httpd-vhost.conf 中新增如下代码，配置 4 个虚拟主机。

```
<VirtualHost *:80>
    DocumentRoot "C:/jQuery/chapter09/www.shop.localhost/"
    ServerName www.shop.localhost
</VirtualHost>
<VirtualHost *:80>
    DocumentRoot "C:/jQuery/chapter09/api.shop.localhost/public/"
    ServerName api.shop.localhost
</VirtualHost>
<VirtualHost *:80>
    DocumentRoot "C:/jQuery/chapter09/www.shop-admin.localhost/"
    ServerName www.shop-admin.localhost
</VirtualHost>
<VirtualHost *:80>
    DocumentRoot "C:/jQuery/chapter09/api.shop-admin.localhost/public/"
    ServerName api.shop-admin.localhost
</VirtualHost>
<Directory "C:/jQuery/chapter09/">
    AllowOverride All
    Require local
</Directory>
```

在上述配置中，每一个“<VirtualHost……</VirtualHost>”用于配置一个虚拟主机，DocumentRoot 表示站点目录，ServerName 表示域名。“<Directory……</Directory>”用于配置目录，AllowOverride All 表示允许“.htaccess”分布式配置文件，Require local 表示允许在本地访问。

（3）保存 httpd-vhost.conf 文件后，在 WampServer 中执行“重新启动所有服务”命令重启服务，使配置生效。需要注意的是，若 httpd-vhost.conf 文件有误，服务会启动失败，WampServer 图标变成红色，此时需要检查 httpd-vhost.conf 是否存在错误。

（4）手动创建以下 4 个站点目录，通过对应的 URL 地址访问测试。

```
C:\jQuery\chapter09\www.shop.localhost
C:\jQuery\chapter09\api.shop.localhost
C:\jQuery\chapter09\www.shop-admin.localhost
C:\jQuery\chapter09\api.shop-admin.localhost
```

在测试时，可以在站点目录下编写一个 test.html 文件，然后通过浏览器访问“http://域名/test.html”，观察网页是否显示。

项目部署

本书在配套源代码中提供了数据库、API 服务器和一些素材文件，如表 9-1 所示。

表9-1 文件说明

目录	作用
chapter09\data	数据库文件
chapter09\www.shop.localhost	前台网站相关素材
chapter09\api.shop.localhost	前台 API 服务器文件
chapter09\www.shop-admin.localhost	后台网站相关素材
chapter09\api.shop-admin.localhost	后台 API 服务器文件

了解这些文件目录的作用后，下面进行项目部署，具体步骤如下。

（1）将 chapter09\data 目录中的文件复制到 C:\wamp\bin\mysql\mysql5.7.14\data 目录下，部署项目的数据库。

（2）将表 9-1 中列举的第 2~5 行的目录复制到“C:\jQuery\chapter09”对应的站点目录内，部署这些站点的文件。

> ✪ **注意：**
>
> api.shop.localhost 和 api.shop-admin.localhost 站点下的 PHP 脚本需要访问数据库。在默认情况下，WampServer 中的 MySQL 服务器的用户名为 root，密码为空，此时 PHP 脚本可以正确连接数据库。若读者为 root 用户设置了密码，则 PHP 脚本会连接数据库失败。为此，需要编辑这两个站点下的 application\database.php 文件，填写当前环境下的密码。

接口测试

完成项目部署后，就已经搭建好了 api.shop.localhost 和 api.shop-admin.localhost 这两个 API 服务器。为了测试 API 服务器是否工作正常，可以在前台或后台的站点目录下，创建 test.html 文件进行测试，代码如下。

```
1  <script src="js/jquery-1.12.4.js"></script>
2  <script>
3      $.get('http://api.shop.localhost', function(data) {
4          console.log(data);
5      });
6      $.get('http://api.shop-admin.localhost', function(data) {
```

```
7          console.log(data);
8      });
9  </script>
```

执行上述代码后，若看到如图 9-6 所示的运行结果，说明接口测试成功。

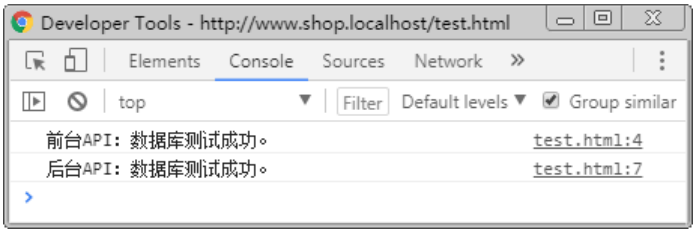

图 9-6　接口测试

目录结构

一个合理的目录结构有利于管理项目中的文件。本项目后台和前台的目录结构分别如表 9-2 和表 9-3 所示。

表9-2　www.shop-admin.localhost 目录结构

类型	文件名称	作用
文件	login.html	用户登录页面
	index.html	后台布局页面（不含内容区域）
目录	content	保存内容区域的页面文件
文件	content\category-list.html	商品分类列表页面
	content\category-add.html	商品分类添加页面
	content\item-list.html	商品列表页面
	content\item-edit.html	商品添加和修改页面
	content\home-slide.html	商城首页的焦点图管理页面
	content\home-news.html	商城首页的最新动态管理页面
目录	js	保存 js 文件
目录	js\jquery-easyui-1.5.4.2	jQuery EasyUI 用户界面库
	js\webuploader-0.1.5	WebUploader 文件上传组件
	js\ueditor1.4.3.3	UEditor 在线编辑器
文件	js\jquery-1.12.4.js	jQuery 库
	js\config.js	后台配置文件
	js\auth.js	后台判断用户是否登录的文件

表9-3　www.shop.localhost 目录结构

类型	文件名称	作用
文件	.htaccess	Apache 分布式配置文件（用于 URL 重写）
	index.html	前台布局页面（不含内容区域）
目录	content	保存内容区域的页面文件

文件	content\index.html	商城首页
	content\find.html	商品列表页
	content\item.html	商品查看页
	content\cart.html	购物车
目录	upload	保存上传文件
	css	保存 css 文件
	img	保存图片文件
	js	保存 js 文件和前端库
目录	js\art-template-4.12.1	art-template 模板引擎
文件	js\jquery-1.12.4.js	jQuery 库
	js\config.js	前台配置文件
	js\common.js	前台公共 js 文件
	js\jquery.slide.js	jQuery 焦点图插件
	js\jquery.page.js	jQuery 分页导航插件
	js\jquery.album.js	jQuery 商品相册插件

需要注意的是，表 9-2 和表 9-3 列举的是项目开发完成后的目录结构，其中有一些文件是在后面的开发任务中才创建出来的。这里列出完整的目录结构，可以方便读者进行参考。

由于前后台的后端 API 不是本课程要讲解的内容，因此这里没有列出具体的目录结构。如果读者已经具备了 PHP 语言基础，则可以阅读这些代码来了解 API 的工作细节。

【任务 1】管理员登录

任务描述

管理员登录功能，就是用户需要输入正确的用户名和密码，才能够登录网站后台进行相关操作。通过这种机制，可以防止无关人员登录到网站后台进行操作，确保系统安全。

本任务需要完成管理员登录功能的开发，页面效果如图 9-7 所示。

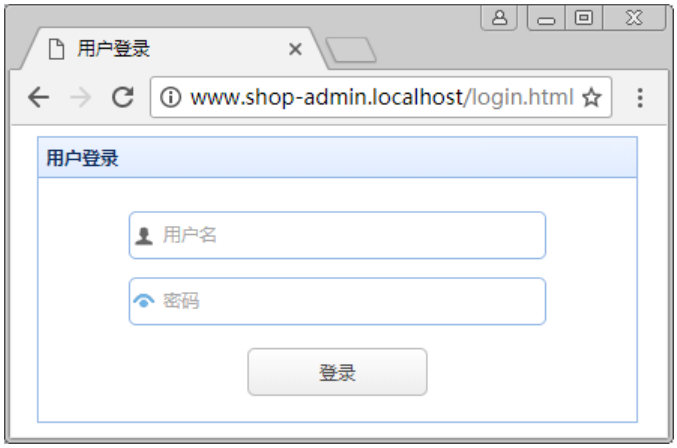

图 9-7　用户登录

在图 9-7 中，用户可以输入后台的用户名（admin）和密码（123456）单击“登录”按钮即可登录。如果输入正确，就会跳转到后台首页。如果输入错误，则提示“登录失败”和具体的错误原因。

接口分析

开发管理员登录功能，需要与后台 API 中的 auth 用户认证接口进行交互。关于该接口的使用说明如表 9-4 所示。

表9-4 用户认证接口

URL	http://api.shop-admin.localhost/v1/auth	
基本信息	请求方式	内容类型
	POST	application/x-www-form-urlencoded
主体	参数名	说明
	username	用户名
	password	密码

在表 9-4 中，URL 是接口的地址，其域名为 api.shop-admin.localhost，域名后面的 v1 是 API 版本号，用于对 API 进行版本控制。版本号后面的部分是资源名称，auth 表示用户认证资源。在基本信息中，请求方式为 POST 表示该接口需要用 POST 方式进行请求；内容类型是指请求头中的 Content-Type 字段的值，在默认情况下，提交 form 表单或通过 jQuery 发送 Ajax 请求时，内容类型为 application/x-www-form-urlencoded。

“主体”是需要发送给 API 的数据，此处的 username 和 password 是用于认证的用户名和密码。在开发时，使用表单收集用户输入的用户名和密码，然后将输入的值发送给 API 进行认证即可。

当 API 认证结束后，会返回 JSON 格式的响应结果。如果响应的状态码为 200，说明认证成功，如果响应的状态码为 403，说明认证失败。

为了使读者更好的理解 API 的使用，下面通过代码进行演示，如下所示。

```
1  var url = 'http://api.shop-admin.localhost/v1/auth';
2  var data = {username: 'admin', password: '123456'};
3  $.post(url, data, function(data) {
4      console.log(data);
5  }).fail(function(xhr) {
6      console.log(xhr.responseText);
7  });
```

在上述代码中，第 1 行的 url 表示 API 的 URL，第 2 行的 data 表示要发送的数据，第 3 行用于发送 POST 方式的 Ajax 请求，第 4 行和第 6 行用于显示 API 返回的结果。

当认证成功时，控制台中输出的结果如图 9-8 所示。

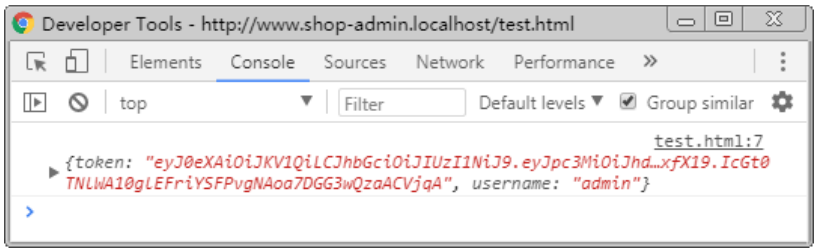

图 9-8　认证成功

当认证失败时，控制台中输出的结果如图 9-9 所示。

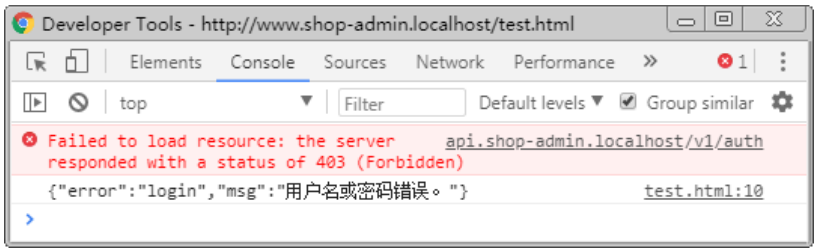

图 9-9　认证失败

从输出结果可以看出，认证成功时，返回的 JSON 包含两个键值对，键名分别为 token 和 username，表示令牌和用户名。令牌是认证成功后的登录凭据，当登录成功后再进行其他操作时，携带这个令牌，就可以通过认证，若没有携带令牌，则说明此时用户没有登录。关于这个令牌的具体使用，将会在下一个任务中进行讲解，此处使用 localStorage（本地存储）将令牌保存起来即可。

若认证失败，则返回的 JSON 包含两个键值对，键名分别为 error 和 msg。error 表示错误类型，如果这个错误是在 auth 资源中发生的，则 error 的值为 auth。msg 表示错误信息，通过错误信息可以了解错误的具体信息。

另外，若 API 响应了其他状态码，如 500、404，则有可能是 API 服务器发生了故障，或没有严格按照 API 的使用说明进行操作，此时应检查 API 服务器是否正常工作，请求 API 的代码是否正确。

代码实现

1. 准备页面

在 www.shop-admin.localhost 虚拟主机目录下创建 login.html 文件，用于显示用户登录页面。在该页面中编写如下 HTML 代码。

【代码 9-1】 login.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>用户登录</title>
    <link rel="stylesheet" href="js/jquery-easyui-1.5.4.2/themes/
    default/easyui.css">
    <link rel="stylesheet" href="js/jquery-easyui-1.5.4.2/themes/
    icon.css">
    <script src="js/jquery-1.12.4.js"></script>
    <script src="js/jquery-easyui-1.5.4.2/jquery.easyui.min.js"></script>
    <script src="js/jquery-easyui-1.5.4.2/locale/easyui-lang-zh_CN.js">
    </script>
  </head>
  <body>
  </body>
```

```
14 </html>
```

上述代码中，第 6、7、9、10 行用于引入 jQuery EasyUI，第 8 行引入 jQuery 文件。

准备好基本页面之后，在【代码 9-1】第 12 行的下面编写代码，利用 jQuery EasyUI 创建一个用户登录界面，具体代码如下。

【代码 9-2】 login.html

```
1  <div class="easyui-panel" data-options="width:'400px'" title="用户登录">
2    <form id="login_form">
3      <p><input class="easyui-textbox" data-options="prompt:'用户名',
4      iconCls:'icon-man',iconAlign:'left',width:'100%',height:'32px'"
5      name="username"></p>
6      <p><input class="easyui-passwordbox" data-options="prompt:'密码',
7      iconAlign:'left',width:'100%',height:'32px'" name="password"></p>
8    </form>
9    <a id="login_btn" href="#" class="easyui-linkbutton"
10   data-options="width:'45%',height:'32px'">登录</a>
11 </div>
```

上述代码创建了一个面板，面板中有一个 id 为 login_form 的表单，在表单中提供了文本框和密码框，分别用于输入用户名和密码，其 name 属性分别为 username 和 password。当提交表单时，将会向后端 API 发送 username 和 password 这两个参数。

2. 提交表单

为了使用户单击“登录”按钮后通过 Ajax 提交表单，需要为 id 为 login_btn 的登录按钮添加单击事件，在单击事件中通过 POST 方式提交给后端 API。接下来在【代码 9-2】第 11 行的下面编写代码，具体代码如下。

【代码 9-3】 login.html

```
1  <script>
2    $('#login_btn').click(function() {
3      $.post(Config.api + 'auth', $('#login_form').serialize())
4      .done(function(data) {
5        localStorage.setItem(Config.tokenName, 'Bearer ' + data.token);
6        location.href = 'index.html';
7      })
8      .fail(function(xhr) {
9        $.messager.alert('登录失败', JSON.parse(xhr.responseText).msg);
10     });
11   });
12 </script>
```

上述代码中，第 3 行和第 5 行用到了 Config 对象，该对象表示项目的配置。为了使代码易于维护，推荐使用独立的配置文件来保存项目配置。因此，下面在</head>前面添加如下代码，引入配置文件。

```
<script src="js/config.js"></script>
```

添加引入代码后，创建对应的 js\config.js 文件，具体代码如下。

【代码 9-4】 js\config.js

```
1  var Config = {
```

```
2    api: 'http://api.shop-admin.localhost/v1/',
3    tokenName: 'shop_admin_auth_token',
4  };
```

在上述代码中，api 表示后台 API 的基础 URL，只包含了版本号，后面的资源则在使用时通过拼接字符串来使用。例如，“Config.api + 'auth'”表示请求的是 auth 资源。

在登录按钮的单击事件中，通过 POST 方式向 auth 发送了 Ajax 请求，发送的数据为表单序列化的结果。假设用户输入了用户名“admin”和密码“123456”，则向 API 提交的数据为“username=admin&password=123456”

当登录成功时，就会进入到【代码 9-3】第 4~7 行的 done()回调函数中，其参数 data 是服务器返回的 JSON 转换成的对象。第 5 行将服务器返回的令牌（即 data.token）保存到本地存储（localStorage）中，保存的名称为 Config.tokenName。在保存令牌时，前面拼接了字符串“Bearer”，这是后端 API 对令牌的格式要求。保存令牌后，通过第 6 行代码进行页面跳转，跳转到了 index.html 后台首页中。关于令牌的使用，将会在【任务 2】中讲解。

当登录失败时，就会进入到【代码 9-3】第 8~10 行的 fail()回调函数中，其参数 xhr 表示 Ajax 对象 XHRHttpRequest。第 9 行用于弹出一个警告框，提示用户登录失败，并显示服务器返回的错误信息。

【任务 2】后台管理界面

任务描述

当用户在网站后台登录成功后，就会进入到后台首页。本项目后台使用了 jQuery EasyUI，在编写后台页面时，需要先搭建后台基本的管理界面。

后台管理界面主要包括“首页管理”“内容管理”和“系统管理”模块。由于篇幅有限，本书重点讲解“内容管理”和“系统管理”模块，而“首页管理”模块不进行讲解。这些模块的基本开发思路都是类似的，相信读者在掌握了“内容管理”模块后，能够自行完成“首页管理”模块的开发。另外，在本书的配套源代码中已经提供了完整的模块，读者可以通过阅读配套源代码进行学习。

接口分析

在本任务中，需要与后台 API 中的测试接口进行交互，具体信息如表 9-5 所示。

表9-5　测试接口

URL	http://api.shop-admin.localhost/v1/test	
基本信息	请求方式	-
	GET	-

测试接口会检测当前用户是否已经登录，如果登录，则响应结果为“ok”，如果未登录，则会返回 JSON 格式的错误信息，可以通过键名 error 获取错误类型，通过键名 msg 获取错误信息文本。

API 判断用户已经登录的依据为，在当前的 HTTP 请求消息中收到了名称为 Authorization 的请求头字段，且该字段中保存了正确的令牌。这个令牌就是上一个任务中，用户认证接口在认证成功时返回的令牌。

如果 API 认证失败，则返回的错误信息有如下几种。

① 未收到 Authorization 请求头；

② Authorization 请求头格式有误；

③ token 认证失败。

下面通过代码演示测试结果接口的使用，如下所示。

```
1  var token = localStorage.getItem('shop_admin_auth_token');
2  $.ajax({
3      type: 'GET', headers: {Authorization: token},
4      url: 'http://api.shop-admin.localhost/v1/test',
5      error: function(xhr) {
6          console.log(xhr.responseText);
7      }
8  });
```

在上述代码中，第 1 行用于取出在用户登录成功时保存的令牌，第 3 行在发送 Ajax 请求时将令牌放在请求头中的 Authorization 字段中发送，API 收到后就会检测令牌是否正确。

需要注意的是，在整个后台的 v1 版本的接口中，除了 auth 资源不需要认证以外，其他资源都需要认证，从而避免未经授权的用户访问这些资源。

代码实现

1. 准备页面

在 www.shop-admin.localhost 虚拟主机的目录下创建 index.html 文件，用于显示后台首页。在该页面中编写如下 HTML 代码。

【代码 9-5】 index.html

```
1  <!DOCTYPE html>
2  <html>
3    <head>
4      <meta charset="utf-8">
5      <title>后台管理系统</title>
6      <!-- 需要引入 jQuery 和 jQuery EasyUI（代码略） -->
7      <script src="js/config.js"></script>
8    </head>
9    <body></body>
10 </html>
```

上述代码中，第 6 行省略了引入 jQuery 和 jQuery EasyUI 的代码，读者可参阅【代码 9-1】的第 6~10 行，与此处代码相同。第 7 行引入了配置文件。

2. 判断用户是否已经登录

由于后台的功能需要用户登录才能使用，因此需要在后台首页中判断当前用户是否已经登录。下面在【代码 9-5】第 7 行的下面添加如下代码，通过引入一个单独的 js 文件来对用户是否登录进行判断。

```
<script src="js/auth.js"></script>
```

添加引入代码后，创建对应的 js/auth.js 文件，具体代码如下。

【代码 9-6】 js\auth.js

```
1  (function() {
2    // 从本地存储中获取令牌
3    var token = localStorage.getItem(Config.tokenName);
4    Config.token = token;
5    // 发送 Ajax 请求时，在请求头中携带令牌用于认证
6    $.ajaxSetup({headers: {Authorization: token}});
7    // 测试令牌是否正确
8    $.get(Config.api).fail(function(xhr) {
9      var data = JSON.parse(xhr.responseText);
10     if (data.error === 'auth') {
11       alert(data.msg + '。\n\n 请重新登录！\n');
12       location.href = 'login.html';
13     }
14   });
15 })();
```

在上述代码中，第 4 行将从 localStorage 中获取的令牌保存到 Config.token 中，方便以后使用。第 6 行配置了 Ajax 的全局默认选项，用于将令牌放入请求头中发送，在请求头中的字段为 Authorization。当后端 API 收到 Authorization 后，就会验证令牌是否正确。

第 8~14 行用于测试令牌是否正确，如果正确则无需处理，而如果失败就进行拦截，提示 API 返回的错误信息，然后跳转到登录页面要求用户重新登录。如果在请求 index.html 时用户还没有登录，或令牌已经过期，就会在此处被拦截。

3. 搭建后台布局

修改【代码 9-5】第 9 行代码，利用 jQuery EasyUI 搭建后台布局，具体代码如下。

【代码 9-7】 index.html

```
1  <body class="easyui-layout">
2    <div title="系统菜单" data-options="region:'west',split:true,
3    width:'200px'">
4      <!-- 在此处编写左侧菜单 -->
5    </div>
6    <div data-options="region:'center'">
7      <div id="tabs" class="easyui-tabs" data-options="fit:true">
8        <div title="首页" class="content">
9          <p>欢迎来到商城后台管理系统！</p>
10         <p>请从左侧选择一个操作。</p>
11       </div>
12     </div>
13   </div>
```

```
14 </body>
```

在上述代码中，第 2~5 行是左侧菜单，第 6~13 行是中心区域。其中，第 7~12 行创建了标签页容器，第 8~11 行创建了后台首页标签页。

4. 左侧菜单

在【代码 9-7】第 4 行的下面编写代码，创建左侧菜单。具体代码如下。

【代码 9-8】 index.html

```
<ul id="menu" class="easyui-tree menutree">
 <li><span>首页管理</span><ul>
   <li data-options="attributes:{'url':'home-slide.html'}">焦点图</li>
   <li data-options="attributes:{'url':'home-news.html'}">最新动态</li>
 </ul></li>
 <li><span>内容管理</span><ul>
  <li data-options="attributes:{'url':'item-list.html'}">管理商品</li>
  <li data-options="attributes:{'url':'category-list.html'}">管理分类</li>
 </ul></li>
 <li><span>系统管理</span>
   <ul><li>用户退出</li></ul>
 </li>
</ul>
```

上述代码中，data-options 保存的 attributes 是自定义属性，该属性中的 url 表示菜单项对应的网页文件。

接下来在【代码 9-7】第 14 行的上面编写代码，实现当菜单项被单击时，在中心区域内添加一个标签页，并根据菜单项的自定义属性中保存的 url 加载文件。具体代码如下。

【代码 9-9】 index.html

```
<script>
  $('#menu').tree({
    onClick: function(node) {
      // 仅当叶子节点的菜单项被单击时，执行操作
      if ($(this).tree('isLeaf', node.target)) {
        /* 此处编写用户退出功能的代码（在下一步中讲解）*/
        // 若标签页已经打开，则切换到该标签页，否则新增标签页
        var tabs = $('#tabs');
        if (tabs.tabs('getTab', node.text)) {
          tabs.tabs('select', node.text);
        } else {
          var href = 'content/' + node.attributes.url;
          tabs.tabs('add', {
            title: node.text,     // 标签页的标题
            href: href,           // 加载的 HTML 文件地址
            closable: true,       // 可以被关闭
            bodyCls: 'content'    // 为标签页的 body 添加 class
          });
        }
```

```
20      }
21    }
22  });
23 </script>
```

上述代码中，第 3 行的 onClick 是 jQuery EasyUI 中的 tree 组件提供的单击事件，其参数 node 表示被单击的菜单项对象。第 5 行的 isLeaf 用于判断 node 节点是否为叶子节点，如果不是叶子节点，被单击后会展开或折叠子节点，而如果是叶子节点，被单击后则会执行第 6~19 行代码。

为了避免同一个菜单项对应的标签页被重复添加，这里将菜单项的文本值（如“管理商品”）与标签页的标题设计成了相同的文本。在第 9 行通过 getTab 根据当前单击的菜单项文本获取对应的标签页对象，如果可以获取到，说明标签页已经被添加，此时执行第 10 行代码，通过 select 选中该标签页；如果没有获取到，说明对应的标签页未被添加，执行第 12~18 行代码添加标签页。

在添加标签页时，第 12 行代码通过 node.attributes.url 获取要加载的网页，并在前面拼接字符串“content/”，表示这些网页文件保存在 content 目录中。第 13~18 行实现了标签页的添加。

接下来对上述代码进行测试。首先创建 content 目录，并在目录中创建 item-list.html 文件，在文件中随意编写代码，单击“管理商品”菜单项，观察对应的标签页是否被添加，是否加载了 content\item-list.html 文件。然后切换到“首页”标签页，再次单击“管理商品”菜单项，观察程序能否自动切换到已经打开的“管理商品”标签页中。

5. 用户退出功能

在后台的左侧菜单中，有一个“用户退出”菜单项，与其他菜单项不同的是，该菜单项不需要创建对应的标签页，而是在单击后提示用户是否确认退出，如果确认则执行用户退出操作。为了实现这个功能，在【代码 9-9】第 6 行的下面编写如下代码。

【代码 9-10】 index.html

```
1  if ($(this).tree('isLeaf', node.target)) {
2    if (node.text === '用户退出') {
3      return logout();
4    }
5  }
```

上述代码通过判断 node.text 的值是否为“用户退出”，来确定当前是否单击了“用户退出”菜单项。如果单击了，则调用 logout()函数实现用户退出功能。

在代码【9-9】第 1 行的下面编写 logout()函数，具体代码如下。

【代码 9-11】 index.html

```
1  function logout() {
2    $.messager.confirm('确认退出', '您确定要退出？', function(r) {
3      if (r) {
4        localStorage.removeItem(Config.tokenName);
5        location.href = 'login.html';
6      }
7    });
8  }
```

在上述代码中，第 2 行用于弹出一个确认框，confirm()方法的第 1 个参数表示确认框的标题，第 2 个参数表提示信息，第 3 个参数是回调函数。该确认框提供了“确定”和“取消”两个按钮，单击按钮后会执行回调函数。回调函数的参数 r 是布尔值，用户单击“确定”时，值为 true，用户单击“取消”时，值为 false。

当用户确定要退出时，就会执行第 4~5 行代码，删除本地存储中保存的令牌，然后跳转到登录页面，表示用户已经退出。

【任务 3】商品分类管理

任务描述

商品分类管理包括“分类列表”“新增分类”“修改分类”和“删除分类”这 4 个主要功能。当用户在后台左侧菜单中单击“管理分类”时，就会打开“管理分类”标签页，该标签页的完成效果如图 9-10 所示。

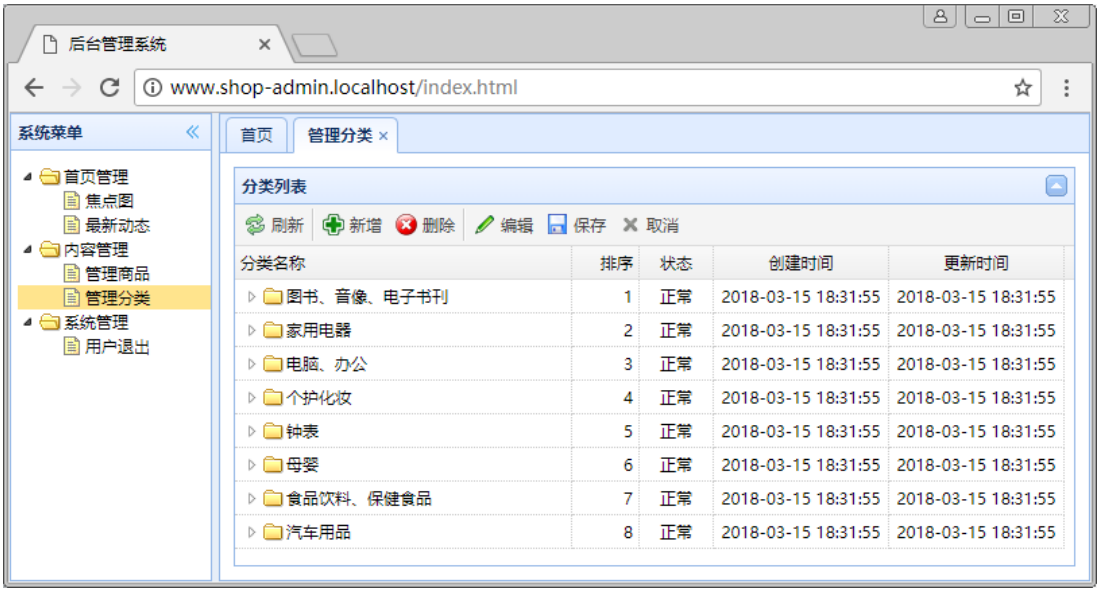

图 9-10　分类管理

从图 9-10 中可以看出，该页面使用了 jQuery EasyUI 的 treegrid 组件，并添加了工具按钮。下面分别介绍商品分类管理页面所提供的各个功能。

1. 分类列表

在“管理分类”标签页打开后，会自动请求后端 API 获取分类列表数据。分类列表具有分类名称、排序、状态、创建时间、更新时间这些列，并支持多级嵌套。通过分类名称左边的小三角可以展开和折叠，最多可以展开到第 3 级，如图 9-11 所示。

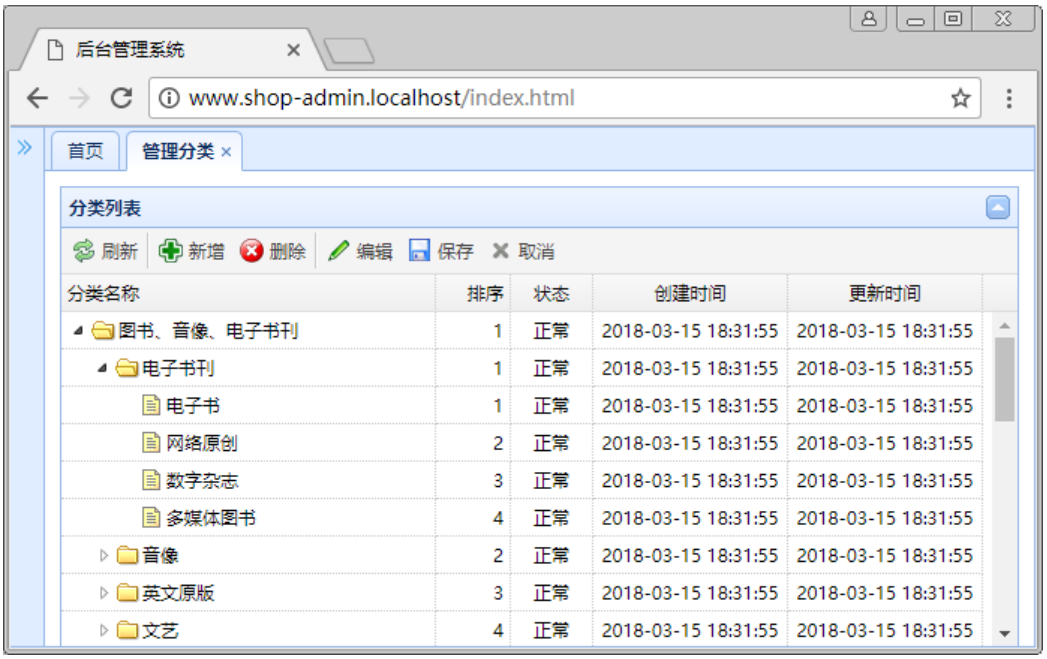

图 9-11　分类列表

在列表中，使用鼠标单击其中一行，就会选中一个分类，然后可以通过工具栏上的按钮，对所选分类进行新增（添加子分类）、编辑和删除操作。

2.　新增分类

选中图 9-11 中的“电子书刊”分类，然后单击“新增”按钮，就会弹出一个标题为“新增分类”的子窗口，如图 9-12 所示。

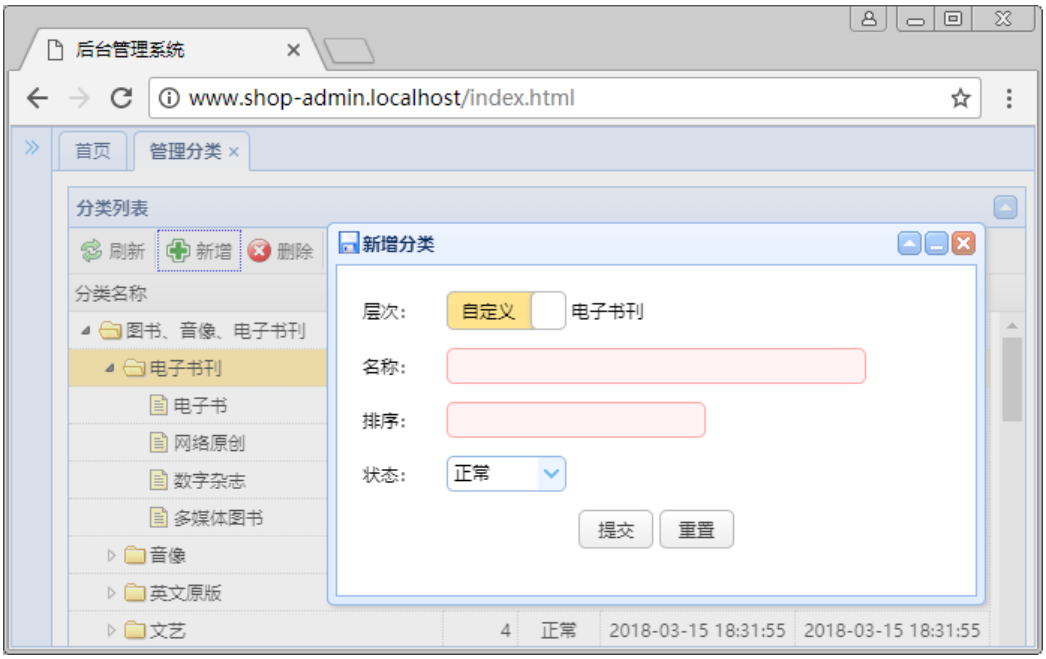

图 9-12　新增分类

在图 9-12 中，提供了层次、名称、排序和状态 4 个表单控件。其中，层次是一个切换按钮（switchbutton），名称是文本框（textbox），排序是数字框（numberbox），状态是组合框（combobox）。

“层次”可以在“自定义”和“最顶层”之间切换，如果选择“自定义”，就表示为当前选中的“电子书刊”添加子分类，如果选择“最顶层”，则表示添加最顶层的分类。另外，若用户在列表中未选中任何一个分类，直接单击“新增”按钮，则层次只能选择“最顶层”。

“名称”和“排序”都是必填项，其中“名称”表示分类的名称，“排序”表示该分类在同级分类中显示的顺序，数值越大则顺序越靠前，且只能输入整数，不允许输入小数。

“状态”下拉菜单提供了“正常”和“下架”两个菜单项，如果选择“正常”，则该分类（包括子分类）会在前台显示，如果选择“下架”，则该分类（包括子分类）不会在前台显示。

在添加子分类时，会判断当前选中的分类是否达到了第 3 级，如果是，则不允许添加子分类。例如，选中“电子书”单击“新增”按钮，会看到如图 9-13 所示的提示信息。

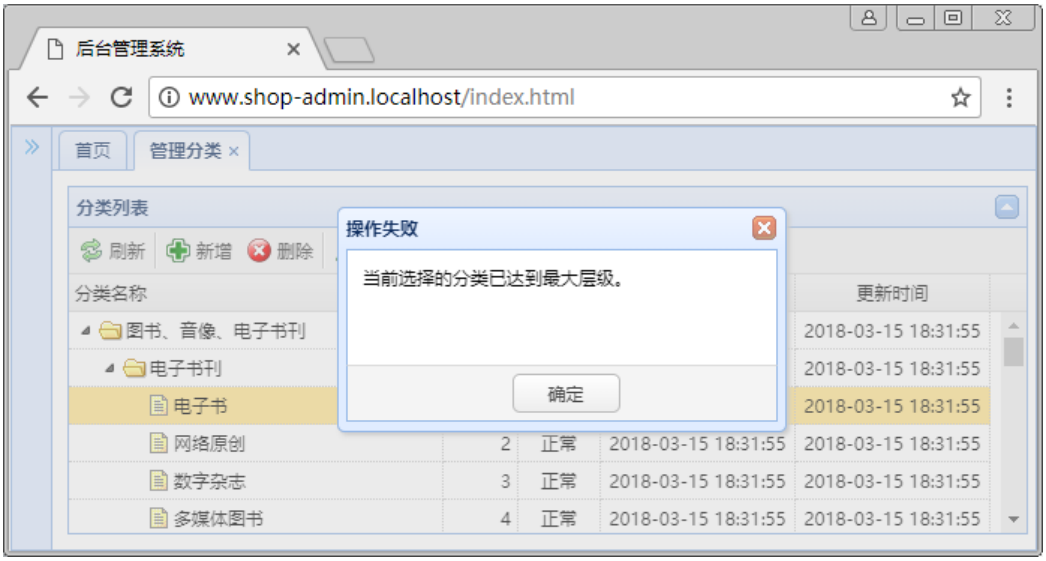

图 9-13　提示信息

3. 修改分类

在分类列表中选中一项分类后，单击“编辑”按钮，就会进入到分类编辑状态，如图 9-14 所示。

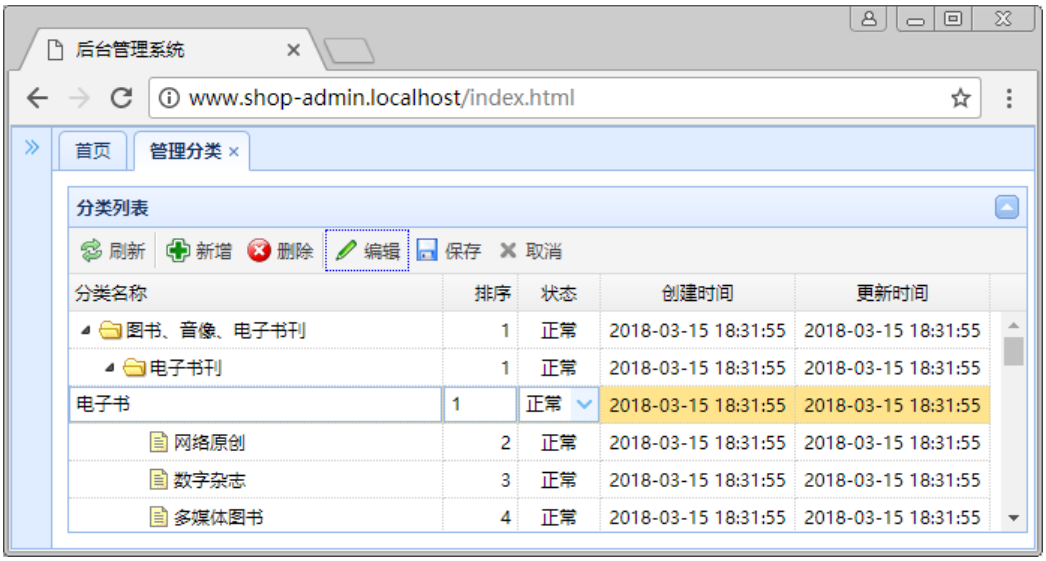

图 9-14　修改分类

在分类编辑模式下，可以修改分类名称、排序和状态的值。当编辑完成后，单击工具栏中的“保存”按钮，就会将用户编辑的结果发送到后端 API 中进行处理。然后程序会自动刷新整个列表，从而看到修改结果。若要取消编辑，可以单击“取消”按钮。

4. 删除分类

在分类列表中选中一项分类后，单击“删除”按钮，可以对分类进行删除操作。需要注意的是，如果选中的分类存在子分类，则该分类不允许被删除，只有先删除了所有的子分类，才可以删除父分类。

在执行删除操作前，程序会弹出“您确定要删除此项”的确认框，只有用户单击“确认”后才会执行删除操作，避免用户误操作导致数据丢失，提升用户体验。

接口分析

开发商品分类管理功能，需要与后台 API 中的分类接口进行交互，该接口提供了查询分类列表、新增分类、修改分类和删除分类的功能。下面分别进行讲解。

1. 查询分类列表

向 categories 资源发送 GET 请求即可查询分类列表，具体说明如表 9-6 所示。

表9-6 查询分类列表

URL	http://api.shop-admin.localhost/v1/categories	
基本信息	请求方式	-
	GET	-
URL 参数	参数名	说明
	view	前端组件名（可选值：treegrid、tree）
	id	上级分类 id（默认值为 0，表示顶级分类）

在表 9-6 中，view 表示 jQuery EasyUI 的组件名，其值可以是 treegrid 或 tree，当通过 treegrid 组件请求 API 时，就选择 treegrid，此时 API 就会按照 treegrid 组件的格式要求来返回数据。id 是分类的唯一标识，它是一个整数，每个分类都有一个 id。当对某个分类进行操作时，就需要将 id 传递给 API。由于分类是树形结构的数据，每个分类都有一个上级 id，因此通过 id 这个参数，就可以查询某一分类下的子分类。

当 view 的值为 treegrid 时，返回的示例结果如下所示。

```
[
  {
    "id": 1,                                  // 分类 id
    "name": "图书、音像、电子书刊",              // 分类名称
    "status": 1,                              // 上架状态，1 表上架，2 表示下架
    "sort": 1,                                // 排序数值（按数字大小升序排列）
    "created": "2018-03-15 18:31:55",         // 该条记录的创建时间
    "updated": "2018-03-15 18:31:55",         // 该条记录的更新时间
    "state": "closed"                         // 节点状态，值为 closed 或 open
  },
  ……
]
```

在上述示例中，最外层是一个数组，数组中的每一个元素是一个对象，这些对象保存的是从数据库中查询到的一行记录。由于对象的属性（如 id、name）与数据库中的字段对应，因此也可以将这些属性称为字段（如 id 字段、name 字段）。

当 view 的值为 tree 时，返回结果的整体结构与 treegrid 相同，但每个对象中只有 id、text 和 state 字段，其中 text 字段与 treegrid 中的 name 字段的值相同。

2. 新增分类

向 categories 资源发送 POST 请求表示新增分类，具体说明如表 9-7 所示。

表9-7 新增分类

URL	http://api.shop-admin.localhost/v1/categories	
基本信息	请求方式	内容类型
	POST	application/x-www-form-urlencoded
主体	参数名	说明
	parent_id	上级分类 id（默认为 0）
	name	分类名称
	sort	排序数值
	status	上架状态（1 表示上架，2 表示下架）

按照表 9-7 中的说明发送请求后，返回的示例结果为“{"id":"10"}”，其中 10 是数据库为新分类分配的 id，每次新增分类时该值都会增长。

3. 修改分类

向 categories 资源发送 PUT 请求表示修改分类，具体说明如表 9-8 所示。

表9-8 修改分类

URL	http://api.shop-admin.localhost/v1/categories/:id	
基本信息	请求方式	内容类型
	PUT	application/x-www-form-urlencoded
主体	参数名	说明
	name	分类名称
	sort	排序数值
	status	上架状态（1 表示上架，2 表示下架）

在表 9-8 中，URL 最后的“:id”是分类 id，表示要修改数据库中哪一条 id 的记录。例如，修改 id 为 1 的分类，就将“:id”改成 1 即可。若发送请求后响应的状态码为 200，表示修改成功。

4. 删除分类

向 categories 资源发送 DELETE 请求表示删除分类，具体说明如表 9-9 所示。

表9-9 删除分类

URL	http://api.shop-admin.localhost/v1/categories/:id	
基本信息	请求方式	-
	DELETE	-

在表 9-9 中，URL 最后的“:id”表示要删除的分类 id。

需要注意的是，如果要删除的分类不是叶子节点，则不允许被删除。另外，当一个分类被删除后，该分类下的所有商品的所属分类 id，将被自动修改为 0。

代码实现

1. 分类列表

在 www.shop-admin.localhost 虚拟主机的 content 目录下创建 category-list.html 文件，然后在该文件中编写如下代码。

【代码 9-12】 content\category-list.html

```
<table id="category_list" title="分类列表"></table>
<div id="add_window" title="新增分类"
data-options="closed:true,width:'440px',height:'256px'"></div>
```

在上述代码中，第 1 行用于显示分类列表，第 2~3 行用于显示新增分类子窗口。

接下来，创建 jQueryEasyUI 的 treegrid 组件，具体代码如下。

【代码 9-13】 content\category-list.html

```
<script>
  (function() {
    var list = $('#category_list');
    list.treegrid({
      url: Config.api + 'categories?view=list',
      method: 'GET', idField: 'id', treeField: 'name',
      collapsible: true, fit: true,
      columns: [[]], toolbar: [],
    });
  })();
</script>
```

在上述代码中，第 5 行传入了后端 API 的 URL，第 6 行中的 method 表示指定请求方式，idField 表示唯一标识每个分类的字段，treeField 表示树节点字段，这两个字段的名称取决于后端 API 返回的字段名称。将后端 API 返回的 id 字段作为唯一标识后，对某个分类进行修改、删除、查看子分类等其他操作时，就可以将这个 id 发送给后端 API，表示对这个 id 对应的分类进行操作；将后端 API 返回的 name 字段作为树节点后，列表中该字段对应的列就会出现展开或折叠的小三角图标。

第 7 行的 collapsible 用于在组件的右上角提供一个可折叠或展开的按钮；第 8 行的 fit 用于根据网页的宽度来自动调整组件的宽度。

第 8 行的 columns 用于设置列表中每一列的标题和字段名称等参数，toolbar 用于添加工具栏。下面开始编写 columns，将原来的“columns: [[]],”改为如下代码。

【代码 9-14】 content\category-list.html

```
columns: [[
  {title: '分类名称', field: 'name', editor: 'textbox', width: 250},
  {title:'排序',field:'sort',editor:'numberbox',align:'right',width:50},
  {title: '状态', field: 'status', formatter: formatStatus, editor:
                          statusEditor, width: 55,  align: 'center'},
  {title: '创建时间', field: 'created', width: 130, align: 'center'},
```

```
7    {title: '更新时间', field: 'updated', width: 130, align: 'center'}
8  ]],
```

在上述代码中，title 表示列的标题；field 表示字段名，对应后端 API 返回的字段名；editor 表示在编辑分类时，使用哪种表单控件来对该列进行编辑；width 表示列的宽度，数字越大则宽度越宽；align 表示文本的对齐方式。

在“状态”列中，formatter 表示使用一个函数来对值进行格式化，由于后端 API 返回的 status 字段值为 1 和 2（分别表示正常和下架），如果直接将数字显示在列表中，用户无法知道这些数字的含义，因此编写一个 formatStatus()函数来将数字转换成具体的文字。editor 是一个组合框，由于其代码较多，所以用 statusEditor 对象来保存。

接下来在【代码 9-13】第 3 行的上面添加一些代码，完成 formatStatus()函数和 statusEditor 对象。具体代码如下。

【代码 9-15】 content\category-list.html

```
1  var statusEditor = {type: 'combobox', options: {
2    data: [{value: '1', text: '正常'}, {value: '2', text: '下架'}],
3    editable: false, required: true
4  }};
5  function formatStatus(val) {
6    return {1: '正常', 2: '下架'}[val];
7  }
```

在上述代码中，statusEditor 对象保存了组合框的一些基本配置，其中 data 表示显示在组合框中的选项，这里提供了“正常”和“下架”两个选项，每一项的 value 表示具体的值，text 表示显示的文本。editable 表示这个组合框是否可以像文本框一样自由编辑，设为 false 表示不可随意编辑，只能在列表中选择一项。

formatStatus()函数的参数 val 是由 jQuery EasyUI 在调用时自动传入的，其值为该字段的值。函数的返回值则用来替换在列表中显示的文本。

完成 columns 之后，下面编写 toolbar，将原来的“toolbar: [],”改为如下代码。

【代码 9-16】 content\category-list.html

```
1  toolbar: [
2    {text: '刷新', iconCls: 'icon-reload', handler: reload}, '-',
3    {text: '新增', iconCls: 'icon-add', handler: add},
4    {text: '删除', iconCls: 'icon-no', handler: del}, '-',
5    {text: '编辑', iconCls: 'icon-edit', handler: edit},
6    {text: '保存', iconCls: 'icon-save', handler: save},
7    {text: '取消', iconCls: 'icon-clear', handler: cancel}
8  ],
```

上述代码创建了 6 个工具栏按钮，text 表示按钮标题，iconCls 表示按钮图标，handler 表示按钮按下后调用的处理函数。这些函数的具体代码将在后面的步骤中讲解，为了避免此时程序因找不到函数而出错，在【代码 9-13】第 9 行的下面编写如下代码。

```
function reload() {}      function add() {}
function del() {}         function edit() {}
function save() {}        function cancel() {}
```

2. 分类修改

当用户单击工具栏中的“编辑”按钮时，就会调用 edit()函数，执行编辑操作。下面开

始编写分类编辑功能的代码，具体代码如下。

【代码 9-17】 content\category-list.html

```
1  var editingId;
2  function edit() {
3    if (editingId !== undefined) {
4      list.treegrid('select', editingId);
5      return;
6    }
7    var node = list.treegrid('getSelected');
8    if (node) {
9      editingId = node.id;
10     list.treegrid('beginEdit', editingId);
11   }
12 }
```

在上述代码中，第 7 行用于获取用户当前选中的分类 id，然后在第 10 行将这个分类 id 对应的分类切换为编辑模式。第 9 行通过变量 editingId 保存了当前编辑的分类 id，当用户在编辑状态下再次单击编辑按钮时，第 3~6 行的判断就会成立，第 4 行代码用于选中当前编辑的分类 id，第 5 行使用 return 使函数停止向下执行。

接下来编写取消编辑的函数 cancel()，具体代码如下。

【代码 9-18】 content\category-list.html

```
1  function cancel() {
2    if (editingId !== undefined) {
3      list.treegrid('cancelEdit', editingId);
4      editingId = undefined;
5    }
6  }
```

在上述代码中，第 2 行用于判断当前是否处于编辑模式，如果是，则执行第 3 行代码取消编辑，并通过第 4 行代码将当前编辑的分类 id 设为 undefined。

值得一提的是，在用户实际使用的过程中，如果在分类编辑模式下，将这个被编辑的分类折叠起来，然后再编辑其他分类时，edit()函数会检测到当前正处于编辑模式，将无法对其他分类进行编辑，因此推荐在分类折叠时，自动取消编辑。

在【代码 9-13】第 9 行的上面，利用 treegrid 的 onCollapse（折叠）事件来实现折叠时自动取消编辑的功能，具体代码如下。

```
onCollapse: cancel
```

添加上述代码后，就会在分类折叠时自动调用 cancel()函数取消编辑。

以上步骤已经完成了“编辑”和“取消”按钮的功能开发，若用户编辑分类后需要保存编辑结果，则会调用 save()函数进行保存。接下来编写 save()函数，具体代码如下。

【代码 9-19】 content\category-list.html

```
1  function save() {
2    if (editingId === undefined) {
3      return;
4    }
5    var id = editingId;
```

```
6    editingId = undefined;
7    list.treegrid('endEdit', id);
8    var node = list.treegrid('find', id);
9    $.ajax({
10     type: 'PUT',
11     url: Config.api + 'categories/' + id,
12     data: {name: node.name, sort: node.sort, status: node.status}
13   });
14 }
```

在上述代码中，第 2~4 行用于限制仅在编辑模式下才可以进行保存，第 5~6 行用于将 editingId 取出来后设为 undefined，第 7 行用于结束编辑，第 8 行获得了被编辑分类的节点对象，通过第 9~13 行向服务器发送 Ajax 请求，第 10 行指定了请求方式为 PUT，第 11 行指定了请求的 URL 地址，第 12 行表示提交的数据。

3. 列表刷新

编写工具栏中的“刷新”按钮的处理函数 reload()，具体代码如下。

【代码 9-20】 content\category-list.html

```
1  function reload() {
2    cancel();                          // 取消编辑
3    list.treegrid('unselectAll');      // 取消所有选中状态
4    list.treegrid('reload');           // 重新加载 treegrid
5  }
```

从上述代码可以看出，当单击刷新时，会执行取消编辑、取消选中状态和重新加载 treegrid 的操作。treegrid 会重新从后端 API 获取分类数据。

4. 分类删除

编写工具栏中的“删除”按钮的处理函数 del()，具体代码如下。

【代码 9-21】 content\category-list.html

```
1  function del() {
2    if (editingId !== undefined) {
3      list.treegrid('select', editingId);
4      $.messager.alert('操作失败', '请先取消编辑模式。');
5      return;
6    }
7    var node = list.treegrid('getSelected');
8    if (node && !list.treegrid('isLeaf', node.id)) {
9      $.messager.alert('操作失败', '只能删除最底层的分类。');
10     return;
11   }
12   $.messager.confirm('确认删除', '您确定要删除此项？', function(r) {
13     if (r) {
14       list.treegrid('remove', node.id);
15       $.ajax({
16         type: 'DELETE',
17         url: Config.api + 'categories/' + node.id
```

```
18      });
19    }
20  });
21 }
```

在上述代码中，第 2~6 行用于在编辑模式下阻止删除操作；第 7~11 行代码用于限制只允许删除最底层的分类；第 12 行用于在执行删除操作前，提示用户是否确认删除，防止误操作；第 14 行代码用于在列表中删除当前选中的分类；第 15~18 行代码用于向后端 API 发送 Ajax 请求，从而将分类从数据库中删除。

5. 分类添加

编写工具栏中的“新增”按钮的处理函数 add()，具体代码如下。

【代码 9-22】 content\category-list.html

```
1  function add() {
2    if (editingId !== undefined) {
3      list.treegrid('select', editingId);
4      $.messager.alert('操作失败', '请先取消编辑模式。');
5      return;
6    }
7    var node = list.treegrid('getSelected');
8    if (node && list.treegrid('getLevel', node.id) >= 3) {
9      $.messager.alert('操作失败', '当前选择的分类已达到最大层级。');
10     return;
11   }
12   $('#add_window').window({
13     modal: true,            // 子窗口弹出时，不允许操作父窗口
14     maximizable: false,     // 窗口不显示最大化按钮
15     iconCls: 'icon-save',   // 窗口图标
16     href: 'content/category-add.html'   // 从外部文件加载窗口内容
17   }).window('open');
18 }
```

在上述代码中，第 2~6 行用于在编辑模式下阻止进行添加操作，第 7~11 行用于获取当前选中的分类，并判断所选分类是否已经达到最大层级，如果是，则不允许添加子分类。第 12~17 行代码用于弹出新增分类的子窗口，该窗口的内容来自 content/category-add.html 文件，该文件保存了新增分类的表单。

接下来在 content 目录下创建 category-add.html 文件，具体代码如下。

【代码 9-23】 content\category-add.html

```
1  <form id="category_add_form">
2    <table>
3      <tr><td>层次: </td><td>
4        <input id="category_swt" class="easyui-switchbutton"
5        name="parent_id">
6        <span id="category_name"></span>
7      </td></tr>
8      <tr><td>名称: </td><td>
```

```
9       <input class="easyui-textbox" type="text" name="name"
10      data-options="required:true,width:'280px'">
11    </td></tr>
12    <tr><td>排序：</td><td>
13      <input class="easyui-numberbox" type="text" name="sort"
14      data-options="precision:0,required:true">
15    </td></tr>
16    <tr><td>状态：</td><td>
17      <select class="easyui-combobox" name="status"
18      data-options="width:'80px',editable:false">
19        <option value="1">正常</option>
20        <option value="2">下架</option>
21      </select>
22    </td></tr>
23  </table>
24 </form>
25 <a id="category_add" class="easyui-linkbutton" href="#">提交</a>
26 <a id="category_reset" class="easyui-linkbutton" href="#">重置</a>
```

上述代码创建了新增分类的表单，其页面显示效果已经在任务描述中演示过，如图 9-12 所示。其中，第 5 行指定“层次”切换按钮的 name 属性为 parent_id，第 9 行指定“名称”文本框的 name 属性为 name，第 13 行指定“排序”数字框的 name 属性为 sort，第 17 行指定“状态”组合框的 name 属性为 status。这些 name 属性将用来在提交表单时发送。

在【代码 9-23】第 26 行的下面继续编写代码，通过 jQuery 自动填写当前选中的分类，来作为新添加分类的上级分类。具体代码如下。

【代码 9-24】 content\category-add.html

```
1  <script>
2    var list = $('#category_list');
3    var cname = $('#category_name');
4    var cid = 0;
5    var node = list.treegrid('getSelected');
6    if (node) {
7      cid = node.id;
8      cname.text(node.name);
9    }
10 </script>
```

上述代码通过获取列表中选中的分类 node，使用 node.name 获取到了分类的名称，然后将 id 为 category_name 的元素的内容设为分类名称。第 7 行使用变量 cid 保存了当前选中的分类 id，该变量将在后面的步骤中使用。

当用户单击“层次”开关按钮时，会自动在“自定义”和“最顶层”之间切换，为了实现这个效果，下面在【代码 9-24】第 9 行的下面编写如下代码。

【代码 9-25】 content\category-add.html

```
1  $('#category_swt').switchbutton({
2    onText: '自定义',          // 按钮 on 状态显示文本
3    offText: '最顶层',         // 按钮 off 状态显示的文本
```

```
4    checked: cid !== 0,        // 若 cid 不为 0，则为 on 状态，否则为 off 状态
5    disabled: cid === 0,       // 若 cid 为 0，则将按钮禁用
6    value: cid,                // 按钮的 value 属性值为 cid
7    onChange: function(checked) {
8      cname.toggle(checked);   // 按钮改变事件，为 on 时显示 cname，为 off 则隐藏
9    }
10 });
```

通过上述代码可以看出，使用 onText 和 offText 即可设置开关状态显示的文本。如果用户未选择任何分类，直接单击了“新增”按钮，则此时变量 cid 的值为 0，表示没有上级分类，此时添加的分类就属于最顶层分类。

完成以上步骤后，用户就可以在添加分类的表单中进行填写操作了。当用户填写完成后，单击“提交”按钮，就需要将表单数据发送到后端 API 服务器。接下来在【代码 9-25】第 10 行的下面继续编写代码，具体如下所示。

【代码 9-26】 content\category-add.html

```
1  var addForm = $('#category_add_form');
2  $('#category_add').click(function() {
3    if (!addForm.form('validate')) {
4      $.messager.alert('提示', '表单还未填写完成! ');
5      return;
6    }
7    $.post(Config.api + 'categories', addForm.serialize(), function(data) {
8      if (node && list.treegrid('isLeaf', cid)) {
9        node.state = 'closed';
10       list.treegrid('refresh', node.id);
11     }
12     list.treegrid('reload', cid);
13     $('#add_window').window('close');
14   });
15 });
```

在上述代码中，第 2 行为“提交”按钮添加了单击事件，第 3~6 行进行表单验证，如果验证未通过则提示信息并阻止继续执行。第 7~14 行用于将表单序列化后提交给后端 API，其中第 8~11 行判断当前选中的分类节点 node 是否为叶子节点，如果是，则将 node.state 设为 closed，然后刷新 node 节点，此时 node 将不再是叶子节点，并显示为已折叠的状态。

第 12 行用于重新加载当前选中节点的子节点，重新加载后，当前选中节点将会展开子节点，这样就可以看到新添加的子节点了。

最后，为表单的重置按钮添加单击事件，实现表单的重置效果。在【代码 9-26】第 15 行的下面继续编写代码，具体如下所示。

【代码 9-27】 content\category-add.html

```
1  $('#category_reset').click(function() {
2    addForm.form('reset');
3  });
```

编写上述代码后，用户在表单中随意填写或更改内容，然后单击重置按钮，表单都可以恢复成最初的状态。

【任务 4】商品管理

任务描述

商品管理包括“商品列表”“新增商品”“修改商品”“删除商品”和“商品上下架”这 5 个主要功能。当用户在后台左侧菜单中单击“管理商品”时，就会打开“管理商品”标签页，该标签页的完成效果如图 9-15 所示。

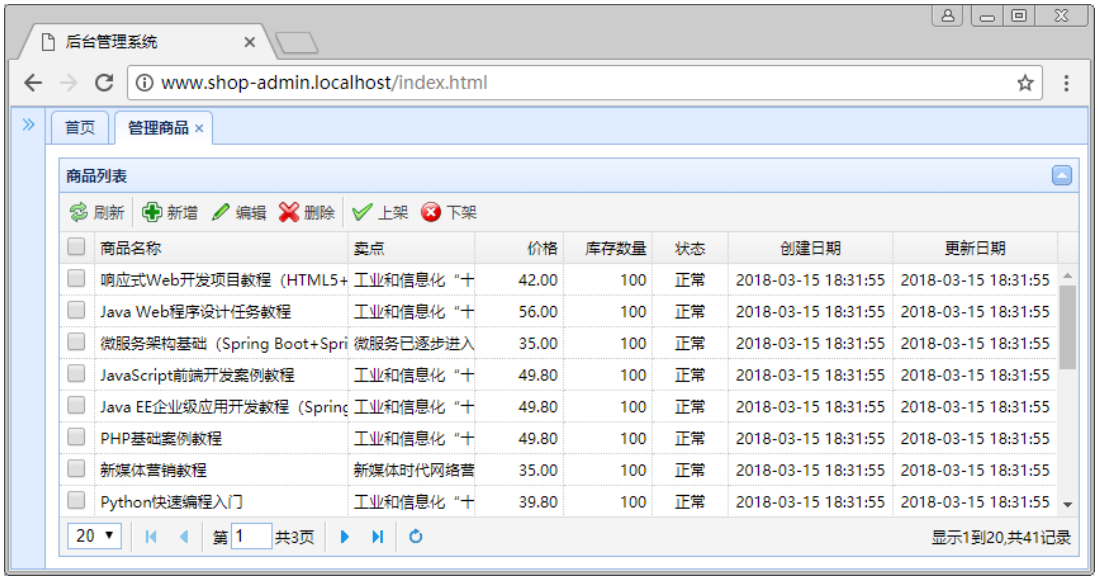

图 9-15　商品管理

从图 9-15 中可以看出，该页面使用了 jQuery EasyUI 的 datagrid 组件，并添加了工具按钮。接下来分别介绍商品管理页面所提供的各个功能。

1. 商品列表

在“管理商品”标签页打开后，会自动到后端 API 请求商品列表数据。商品列表提供了商品名称、卖点、价格、库存数量、状态、创建日期、更新时间这些列。在列表的底部，还提供了分页功能，用户可以分页浏览商品列表。

在列表中，用户使用鼠标单击其中一行，就会选中一件商品。若继续单击其它行，则可以选中多件商品。在每件商品名称的左边，都有一个复选框，表示该商品是否被选中。

2. 新增商品

在工具栏中单击“新增”按钮，就会打开一个“新增商品”的标签页，该标签页的完成效果如图 9-16 所示。

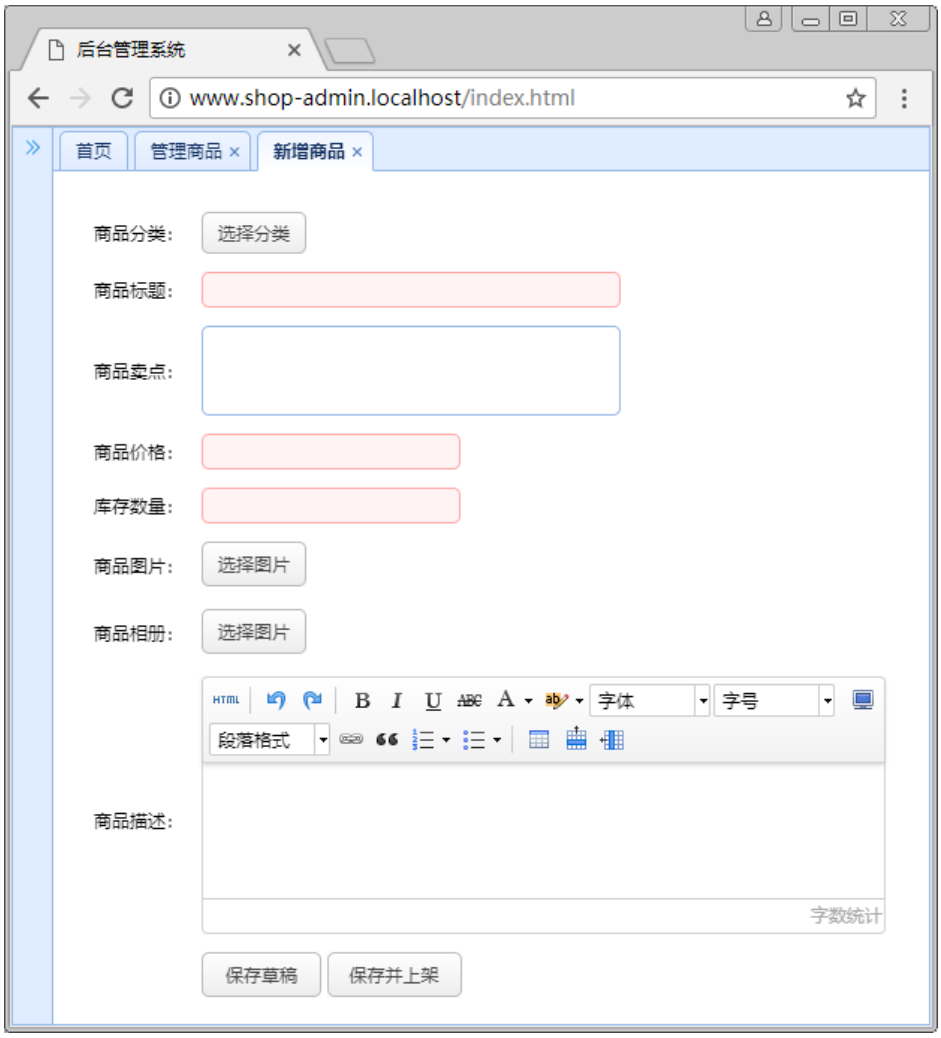

图 9-16　商品管理

在图 9-16 中，单击“选择分类”按钮，会弹出如图 9-17 所示的“选择分类”子窗口。

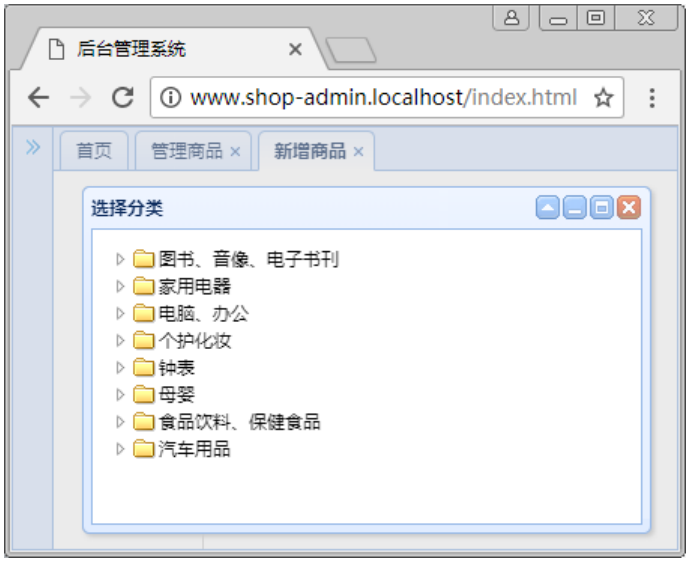

图 9-17　选择分类

在“选择分类”子窗口中，可以选择新添加商品的所属分类，并且只能选择叶子节点的分类，非叶子节点则可以展开或折叠。

在添加商品时，还可以上传商品图片和商品相册。但由于此功能涉及的代码非常多，本任务没有对其进行讲解，在下一个任务中会单独进行讲解。

商品描述使用了 UEditor 编辑器，用户可以利用编辑器方便地进行文字格式处理，从而使商品描述的呈现效果更加美观。

在表单的底部，提供了“保存草稿”和“保存并上架”两个提交按钮，区别在于前者的商品状态为“下架”，后者的商品状态为“正常”，即上架状态，该商品会在前台中显示。

3. 修改商品

在商品列表中选中一件商品后，单击“编辑”按钮，就会打开“编辑商品”标签页。“编辑商品”标签页的显示效果与“新增商品”标签页类似，其区别在于，“编辑商品”标签页会将被编辑商品的原有数据取出来，填入到表单中，从而使用户在原有数据的基础上进行修改。“编辑商品”标签页同样提供了“保存草稿”和“保存并上架”两个提交按钮，单击前者，该商品状态保存为“下架”，单击后者，该商品状态保存为“正常”。

4. 删除商品

在商品列表中选中一件或多件商品后，单击“删除”按钮，可以对商品进行删除操作。为了避免用户误操作，在执行删除操作前，会弹出确认框，只有用户单击“确认”后才会执行删除操作。

5. 商品上下架

在商品列表中选中一件或多件商品后，单击“上架”或“下架”按钮，可以对商品的上下架状态进行操作。为了避免用户误操作，在执行操作前，会弹出确认框，防止误操作。

接口分析

开发商品管理功能，需要与后台 API 中的商品接口进行交互，该接口提供了查询商品列表、查询商品信息、新增商品、修改商品、商品上下架切换和删除商品功能。下面分别进行讲解。

1. 查询商品列表

向 items 资源发送 GET 请求即可查询商品列表，具体说明如表 9-10 所示。

表9-10 查询商品列表

URL	http://api.shop-admin.localhost/v1/items	
基本信息	请求方式	-
	GET	-
URL 参数	参数名	说明
	page	查询的页码（默认为 1）
	rows	每页的记录数（默认为 20，最多 100）

在表 9-10 中，page 和 rows 用于分页查询。发送请求后，示例结果如下。

```
{
  "total": 26,                          // 总记录数
  "rows": [                             // 当前查询的页码下的记录
    {
      "id": 1, "cid": 3,                // 商品 id、所属分类 id
```

```
        "title": "", "sell_point": "",     // 商品名称、卖点
        "price": "100.00", "num": 11,      // 价格、库存数
        "pic": "",  "status": 1,           // 商品图片、上架状态（1 上架，2 下架）
        "content": "", "album": "",        // 商品描述、商品相册
        "created": "", "updated": ""       // 创建时间、更新时间
      },
      ……
    ]
}
```

在上述字段中，pic 和 album 保存的是图片路径。其中，album 可以保存多张图片的路径，每个路径使用“|”分隔。

2. 查询商品信息

在通过 GET 方式请求 items 资源时，若在 URL 中加入商品 id，就表示查询指定商品的信息，具体如表 9-11 所示。

表9-11 查询商品信息

URL	http://api.shop-admin.localhost/v1/items/:id	
基本信息	请求方式	-
	GET	-

以查询 id 为 1 的商品为例，示例结果如下。

```
{
  "id": 1, "cid": 3, "title": "", "sell_point": "", "price": "100.00",
  "num": 11, "pic": "", "status": 1, "content": "", "album": "",
   "created": "", "updated": "", "category_name": "电子书"
},
```

从示例结果可以看出，这个结果相当于商品列表数据中 rows 数组内的某一个对象，不同之处在于多了 category_name 字段，表示所属分类名称。

3. 新增商品

向 items 资源发送 POST 请求表示新增商品，具体说明如表 9-12 所示。

表9-12 新增商品

URL	http://api.shop-admin.localhost/v1/items	
基本信息	请求方式	内容类型
	POST	application/x-www-form-urlencoded
主体	参数名	说明
	cid	所属分类 id
	title	商品名称
	sell_point	卖点
	price	价格
	num	库存数
	content	商品描述
	status	上架状态（1 表示上架，2 表示下架）

按照表 9-12 中的说明发送请求后，返回的示例结果为“{"id":"10"}”，其中 10 是数据库为新商品分配的 id，每次新增商品时该值都会增长。

4. 修改商品

向 items 资源发送 PUT 请求表示修改商品，具体说明如表 9-13 所示。

表9-13 修改商品

URL	http://api.shop-admin.localhost/v1/items/:id	
基本信息	请求方式	内容类型
	PUT	application/x-www-form-urlencoded
主体	参数名	说明
	cid	所属分类 id
	title	商品名称
	sell_point	卖点
	price	价格
	num	库存数
	content	商品描述
	status	上架状态（1 表示上架，2 表示下架）

在表 9-13 中，URL 最后的“:id”是商品 id，表示要修改数据库中哪一条 id 的记录。

5. 商品上下架切换

向 items 资源发送 PATCH 请求表示修改某些字段，目前只允许修改 status 字段，用来对商品进行上下架切换。具体说明如表 9-14 所示。

表9-14 商品上下架切换

URL	http://api.shop-admin.localhost/v1/items	
基本信息	请求方式	内容类型
	PATCH	application/x-www-form-urlencoded
URL 参数	参数名	说明
	ids	商品 id 字符串（多个用“,”分隔）
主体	参数名	说明
	status	上架状态（1 表示上架，2 表示下架）

商品上下架切换功能支持对多件商品同时执行相同操作，在 ids 参数中传入多个商品 id 即可。例如“1,2,3”表示对 id 为 1、2 和 3 的商品进行操作。

6. 删除商品

向 items 资源发送 DELETE 请求表示删除商品，具体说明如表 9-15 所示。

表9-15 删除商品

URL	http://api.shop-admin.localhost/v1/items	
基本信息	请求方式	-
	DELETE	-
URL 参数	参数名	说明
	ids	商品 id 字符串（多个用“,”分隔）

在删除商品时，服务器会自动将用户上传的商品图片、相册图片一并删除。

代码实现

1. 商品列表页面

在 www.shop-admin.localhost 虚拟主机的 content 目录下创建 item-list.html 文件，然后在该文件中编写如下代码。

【代码 9-28】 content\item-list.html

```
<table id="item_list" title="商品列表"></table>
<script>
  (function() {
    var list = $('#item_list');
    list.datagrid({
      collapsible: true, fit: true, pagination: true, pageSize: 20,
      url: Config.api + 'items', method: 'GET',
      columns: [[]], toolbar: []
    });
  })();
</script>
```

上述代码创建了 jQuery EasyUI 的 datagrid 组件，第 6~8 行配置了组件的选项。其中，pagination 用于在列表底部显示分页功能；pageSize 表示每页显示的数量。

下面开始编写 columns，将原来第 8 行的“columns: [[]],” 改为如下代码。

【代码 9-29】 content\item-list.html

```
columns: [[
  {field: 'checkbox', checkbox: true},
  {title: '商品名称', field: 'title', width: 200},
  {title: '卖点', field: 'sell_point', width: 100},
  {title: '价格', field: 'price', width: 70, align: 'right'},
  {title: '库存数量', field: 'num', width: 70, align: 'right'},
  {title: '状态', field: 'status', width: 60, align: 'center',
                                    formatter: formatStatus},
  {title: '创建日期', field: 'created', width: 130, align: 'center'},
  {title: '更新日期', field: 'updated', width: 130, align: 'center'}
]],
```

上述代码配置了每一列的标题、字段等信息。其中第 8 行用到了 formatStatus()函数，在【代码 9-28】第 9 行的下面编写该函数，具体代码如下。

【代码 9-30】 content\item-list.html

```
function formatStatus(val) {
  return {1: '正常', 2: '下架'}[val];
}
```

完成 columns 之后，还需要编写 toolbar，将原来的“toolbar: []”改为如下代码。

【代码 9-31】 content\item-list.html

```
toolbar: [
```

```
2    {text: '刷新', iconCls: 'icon-reload', handler: reload}, '-',
3    {text: '新增', iconCls: 'icon-add', handler: add},
4    {text: '编辑', iconCls: 'icon-edit', handler: edit},
5    {text: '删除', iconCls: 'icon-cancel', handler: del}, '-',
6    {text: '上架', iconCls: 'icon-ok', handler: function() {status(1);}},
7    {text: '下架', iconCls: 'icon-no', handler: function() {status(2);}}
8  ]
```

上述代码用到了 reload()、add()、edit()、del()和 status()函数，这些函数将在后面的步骤中实现。

2. 列表刷新

编写工具栏“刷新”按钮的处理函数 reload()，具体代码如下。

【代码 9-32】 content\item-list.html

```
1  function reload() {
2    list.datagrid('reload');
3  }
```

上述第 2 行代码执行后，datagrid 会重新从后端 API 获取商品列表数据。

3. 商品删除

编写工具栏中的“删除”按钮的处理函数 del()，具体代码如下。

【代码 9-33】 content\item-list.html

```
1  function del() {
2    var ids = getSelectionsIds();
3    if (ids.length === 0) {
4      $.messager.alert('操作失败', '未选中商品!');
5      return;
6    }
7    $.messager.confirm('确认', '确定删除选中的 ' + ids.length + ' 件商品吗? ',
8    function(r) {
9      if (r) {
10       var url = Config.api + 'items?ids=' + ids.join(',');
11       $.ajax({type: 'DELETE', url: url, success: function() {
12         $.messager.alert('提示', '删除商品成功! ', undefined, reload);
13       }});
14     }
15   });
16 }
```

在上述代码中，第 2 行调用了 getSelectionsIds()函数，该函数用于获取列表中当前选中的商品。在【代码 9-33】第 16 行的下面编写该函数，具体代码如下。

【代码 9-34】 content\item-list.html

```
1  function getSelectionsIds() {
2    var sels = list.datagrid('getSelections');
3    var ids = [];
4    for (var i in sels) {
5      ids.push(sels[i].id);
```

```
6    }
7    return ids;
8  }
```

从上述代码可以看出，该函数的返回值是一个数组，数组中保存了用户选中的每一件商品的 id。

获取到用户选中的商品 id 后，【代码 9-33】第 3 行判断选中商品的数量，如果未选中商品则弹出提示并阻止继续执行；第 7 行弹出了确认框，防止用户误操作；第 10~13 行用于向后端 API 发送 Ajax 请求，请求方式为 DELETE，URL 参数 ids 表示待删除的商品 id，按照后端 API 的格式要求，使用“ids.join(',') ”将 ids 数组转换为以“,”分隔的字符串；第 12 行用于在删除成功后提示用户，第 4 个参数传入了 reload()函数，表示在用户单击确定后，重新加载商品列表数据。重新加载后，列表中被删除的商品就会消失。

4. 商品上下架

商品上下架的开发思路与商品删除类似，下面编写 status()函数，具体代码如下。

【代码 9-35】 content\item-list.html

```
1  function status(val) {
2    var ids = getSelectionsIds();
3    if (ids.length === 0) {
4      $.messager.alert('操作失败', '未选中商品！');
5      return;
6    }
7    var str = {1: '上架', 2: '下架'}[val];
8    $.messager.confirm('确认操作', '确定' + str + '选中的 ' + ids.length +
9    ' 件商品吗？', function(r) {
10     if (r) {
11       $.ajax({
12         type: 'PATCH',
13         url: Config.api + 'items?ids=' + ids.join(','),
14         data: {status: val},
15         success: function() {
16           $.messager.alert('提示', str + '商品成功！', undefined, reload);
17         }
18       });
19     }
20   });
21 }
```

在上述代码中，第 11~18 行代码用于向后端 API 发送 Ajax 请求。其中，第 12 行将请求方式设为 PATCH；第 13 行的 URL 参数 ids 表示待操作的商品 id；第 14 行表示将商品的 status 字段修改为 val，val 是 status()函数被调用时传入的参数，当用户单击“上架”按钮时，val 的值为 1，单击“下架”按钮时，val 的值为 2。第 7 行用于将 val 的值转换成对应的文字，保存为变量 str，从而在显示提示信息时使用。

5. 商品添加和修改页面

商品的添加与修改的标签页几乎相同，区别在于，修改商品时，会将商品原有数据填入到表单中，而添加商品时，表单中没有数据。为了避免重复编写代码，这里将添加与修

改的标签页共用同一个 HTML 文件，即 content\item-edit.html，通过判断是否存在待修改的商品 id 来区分当前是添加还是修改操作。

在商品列表工具栏中单击“编辑”按钮，就会调用 edit()函数。因此，接下来编写 edit()函数，弹出一个“编辑商品”的标签页，具体代码如下。

【代码 9-36】 content\item-list.html

```
1  function edit() {
2    var node = list.datagrid('getSelected');
3    if (!node) {
4      $.messager.alert('操作失败', '请先在列表中选择一项。');
5      return;
6    }
7    if (list.datagrid('getSelections').length > 1) {
8      $.messager.alert('操作失败', '只能选择一项进行编辑。');
9      return;
10   }
11   if ($('.item-edit-' + node.id).length > 0) {
12     $.messager.alert('操作失败', '该商品正在编辑中。');
13     return;
14   } else {
15     $('#tabs').tabs('add', {
16       title: '编辑商品', href: 'content/item-edit.html', closable: true,
17       bodyCls: 'content item-edit-' + node.id,
18       onLoad: function() {
19         itemEdit($(this), node.id);
20       }
21     });
22   }
23 }
```

在上述代码中，第 2 行用于获取当前选中的商品，第 3~6 行用于检测至少选择一件商品，第 7~10 行用于检测至多选择一件商品。第 15~21 行用于弹出“编辑商品”标签页，在弹出之前，为了避免同一件商品的标签页被重复打开，在第 17 行将当前编辑的商品 id 保存在标签页的 class 属性中，class 的名称为“item-edit-商品 id”。在弹出标签页之前，第 11 行代码判断了带有“item-edit-商品 id”这个 class 属性的标签页是否存在，如果已经存在，则执行第 12~13 行代码，提示用户该商品正在编辑中。

第 19 行在新标签页的加载事件中调用 itemEdit()函数，用于对标签页中的元素进行操作，将在后面的步骤中实现。其中，itemEdit()函数的第 1 个参数是标签页对象，第 2 个参数是商品 id。

完成 edit()函数后，再来编写“新增”按钮的 add()函数，具体代码如下。

【代码 9-37】 content\item-list.html

```
1  function add() {
2    if ($('.item-edit-0').length > 0) {
3      $.messager.alert('操作失败', '新增商品标签页已经打开。');
4    } else {
5      $('#tabs').tabs('add', {
```

```
      title: '新增商品', href: 'content/item-edit.html',
      closable: true, bodyCls: 'content item-edit-0',
      onLoad: function() {
        itemEdit($(this), 0);
      }
    });
  }
}
```

从上述代码可以看出，add()函数的开发思路与 edit()函数类似。由于新增商品时并不存在商品 id，因此这里将商品 id 使用 0 来表示。

接下来编写商品编辑页面 content\item-edit.html，具体代码如下。

【代码 9-38】 content\item-edit.html

```
<div id="item_edit">
  <form class="item-edit-form" method="post">
    <table>
      <tr><td>商品分类：</td><td>
        <a class="select_category easyui-linkbutton" href="#"
        data-options="width:'70px',height:'28px'">选择分类</a>
        <span class="select_category_name"></span>
        <input type="hidden" name="cid">
      </td></tr>
      <tr><td>商品标题：</td><td>
        <input name="title" data-options="required:true,
        validType:'length[0,80]',width:'280px'">
      </td></tr>
      <tr><td>商品卖点：</td><td>
        <input name="sell_point" data-options="multiline:true,
        validType:'length[0,150]',height:'60px',width:'280px'">
      </td></tr>
      <tr><td>商品价格：</td><td>
        <input name="price" data-options="min:0,max:99999999,
        precision:2,required:true">
      </td></tr>
      <tr><td>库存数量：</td><td>
        <input name="num" data-options="min:0,max:99999999,
        precision:0,required:true">
      </td></tr>
      <tr><td>商品图片：</td><td>
        <div class="upload_pic">
          <div class="webuploader-list"></div>
          <div class="webuploader-file-picker">选择图片</div>
        </div>
      </td></tr>
      <tr><td>商品相册：</td><td>
```

```
33        <div class="upload_album">
34          <div class="webuploader-list"></div>
35          <div class="webuploader-file-picker">选择图片</div>
36        </div>
37      </td></tr>
38      <tr><td>商品描述：</td><td>
39        <textarea name="content" style="width:700px;height:300px;">
40        </textarea>
41      </td></tr>
42      <tr><td></td><td>
43        <input type="hidden" name="status" value="1">
44        <a class="easyui-linkbutton item-edit-save">保存草稿</a>
45        <a class="easyui-linkbutton item-edit-submit">保存并上架</a>
46      </td></tr>
47    </table>
48  </form>
49  <div class="tree_window" title="选择分类" data-options="modal:true,
50  width:'500px',height:'450px'">
51    <ul></ul>
52  </div>
53 </div>
```

上述代码利用 jQuery EasyUI 在网页中创建了“商品标题”“商品卖点”等表单控件，并对每个表单控件设置了验证规则。

回到商品列表页面 content\item-list.html，在【代码 9-37】第 13 行的下面编写 itemEdit()函数，对标签页内的表单进行处理，具体代码如下。

【代码 9-39】 content\item-list.html

```
1  function itemEdit(obj, id) {
2    createForm();
3    createCategory();
4    function createForm() {}          // 处理表单控件
5    function createCategory() {}      // 处理分类选择控件
6    function createEditor() {}        // 处理编辑器
7    function createImage(image) {}    // 处理图片组件
8  }
```

从上述代码可以看出，itemEdit()函数内部又将功能分为 4 个函数。这 4 个函数的代码将在后面的步骤中实现。

6. 分类选择组件

在表单中单击“选择分类”按钮后，需要弹出一个标题为“选择分类”的子窗口，在子窗口中可以选择一个叶子节点的分类作为当前商品的所属分类。为了实现这个效果，下面开始编写 createCategory()函数，创建分类选择组件。具体代码如下。

【代码 9-40】 content\item-list.html

```
1  function createCategory() {
2    var selbtn = obj.find('.select_category');
```

```
3   var selname = obj.find('.select_category_name');
4   var treewin = obj.find('.tree_window');
5   selbtn.click(function() {
6     treewin.window({
7       onOpen: function() {
8         $(this).find('ul').tree({
9           url: Config.api + 'categories?view=tree',
10          method: 'GET', animate: true,
11          onClick: function(node) {
12            if ($(this).tree('isLeaf', node.target)) {
13              selbtn.parent().find('[name=cid]').val(node.id);
14              selname.text(node.text);
15              treewin.window('close');
16            }
17          }
18        });
19      }
20    });
21  });
22 }
```

上述代码为“选择分类”按钮添加了单击事件，单击后会弹出子窗口。第 7 行的 onOpen 事件用于在子窗口打开后执行一些操作；第 8~18 行用于在子窗口内创建一个 tree 组件；第 11 行用于为 tree 中的每一个节点添加单击事件；第 12 行用于判断单击的节点是否为叶子节点，如果是叶子节点，则将 name 为 cid 的元素的值设为节点的 id，从而将表单中的 cid 字段设为所选分类的 id；第 14 行用于将所选的分类名称显示出来；第 15 行用于在选择分类后关闭子窗口。

7. 将商品原有数据填写到表单

当进行商品编辑时，需要将商品原有的信息查询出来，填写到表单中。下面开始编写 createForm()函数，具体代码如下。

【代码 9-41】 content\item-list.html

```
1  function createForm() {
2    var title = obj.find('[name=title]').textbox();
3    var sell_point = obj.find('[name=sell_point]').textbox();
4    var price = obj.find('[name=price]').numberbox();
5    var num = obj.find('[name=num]').numberbox();
6    var editor = createEditor();
7    if (id === 0) {
8      createImage({pic: '', album: ''});
9    } else {
10     $.get(Config.api + 'items/' + id, function(data) {
11       obj.find('.select_category_name').text(data.category_name);
12       obj.find('[name=cid]').val(data.cid);
13       title.textbox('setValue', data.title);
14       sell_point.textbox('setValue', data.sell_point);
```

```
15      price.textbox('setValue', data.price);
16      num.textbox('setValue', data.num);
17      createImage({pic: data.pic, album: data.album});
18      editor.ready(function() {
19        this.setContent(data.content);
20      });
21    });
22  }
23 }
```

在上述代码中，第 2~5 行用于获取商品标题、商品卖点、商品价格和库存数量这些表单控件对象；第 10 行用于请求后端 API，根据 id 获取商品信息，然后通过第 11~16 行代码将获取到的商品信息填入到表单中。

第 6 行和第 18~20 行是与编辑器相关的代码。其中，第 6 行调用的 createEditor()函数将在下一步中实现；第 19 行用于将商品原有的描述信息填写到编辑器中。

第 8 行和第 17 行调用了 createImage()函数，用于处理图片组件。该函数将在下一个任务中讲解。在 createImage()函数的参数中，pic 和 album 用于在修改商品时显示已经存在的商品图片和商品相册，由于在添加商品时并不存在商品图片和商品相册，因此这里的 pic 和 album 的值都是空字符串。

8. 编辑器

在表单中，“商品描述”这个表单控件将会用到编辑器。这里使用 UEditor 编辑器，在 index.html 文件的</head>前添加 UEditor 引入代码，如下所示。

【代码 9-42】 index.html

```
1 <script src="js/ueditor1.4.3.3/ueditor.config.js"></script>
2 <script src="js/ueditor1.4.3.3/ueditor.all.min.js"></script>
```

引入编辑器后，开始编写 createEditor()函数，具体代码如下。

【代码 9-43】 content\item-list.html

```
1  function createEditor() {
2    UEDITOR_CONFIG['UEDITOR_HOME_URL'] = '/js/ueditor1.4.3.3/';
3    UEDITOR_CONFIG['serverUrl'] = '';
4    UEDITOR_CONFIG['textarea'] = 'Content';
5    UEDITOR_CONFIG['elementPathEnabled'] = false;
6    UEDITOR_CONFIG['toolbars'] = [['fullscreen', 'source', '|', 'undo',
7     'redo', '|', 'bold', 'italic', 'underline', 'strikethrough',
8     'forecolor', 'backcolor', 'fontfamily', 'fontsize'], ['paragraph',
9     'link', 'blockquote', 'insertorderedlist', 'insertunorderedlist',
10    '|', 'inserttable', 'insertrow', 'insertcol']];
11   var ue_id = 'item_edit_content_' + id;
12   obj.find('[name=content]').attr('id', ue_id);
13   UE.delEditor(ue_id);
14   return UE.getEditor(ue_id);
15 }
```

上述代码中，第 2~10 行用于对编辑器进行基本配置，第 12 行用于从容器 obj 中查找 name 属性为 content 的元素，即“商品描述”这个表单控件，将其 id 属性设为 ue_id。ue_id

是基于当前商品 id 生成的不重复 id，从而区分不同标签页中的编辑器。

由于 UE.getEditor()遇到相同 id 时，不会重复创建编辑器，而同一件商品的标签页，用户可以关闭后再次打开，此时就会遇到相同 id 的情况。为了解决这个问题，第 13 行代码在创建编辑器前先删除了相同 id 的编辑器，从而保证每次都能够正确创建编辑器。

9. 提交表单

用户在新增商品或编辑商品的表单中填写完成后，就需要提交表单。表单提交按钮一共有两个，分别是“保存草稿”和“保存并上架”。两者的区别是，前者提交的 status 字段值为 2，后者提交的 status 字段值为 1。

在【代码 9-41】第 22 行的下面编写代码，完成两个按钮的功能，具体代码如下。

【代码 9-44】 content\item-list.html

```
1  // 保存草稿（下架）
2  obj.find('.item-edit-save').click(function() {
3    submit(2);
4  });
5  // 保存并上架
6  obj.find('.item-edit-submit').click(function() {
7    submit(1);
8  });
9  function submit(status) {
10   var form = obj.find('.item-edit-form');
11   if (!form.form('validate')) {
12     $.messager.alert('操作失败', '表单还未填写完成！');
13     return;
14   }
15   editor.sync();
16   obj.find('[name=status]').val(status);
17   if (id === 0) {
18     $.post(Config.api + 'items', form.serialize(), function(data) {
19       id = data.id;
20       $.messager.alert('提示', '保存成功。');
21     });
22   } else {
23     $.ajax({type: 'PUT', url: Config.api + 'items/' + id, data:
24     form.serialize(), success: function() {
25       $.messager.alert('提示', '保存成功。');
26     }});
27   }
28 }
```

从上述代码可以看出，“保存草稿”和“保存并上架”这两个按钮在单击时都会调用 submit()函数，区别在于实参不同。submit()函数的参数 status 表示表单中 status 字段的值，1 表示正常，2 表示下架。第 11~14 行用于表单验证；第 15 行用于将编辑器的内容回写到原来的 name 为 content 的控件里；第 17 行用于判断当前是添加商品还是修改商品，如果是添加，则执行第 18~21 行代码，如果是修改，则执行第 23~26 行代码。添加与修改操作的 Ajax

代码都是按照后端 API 的要求来编写的。在操作完成后，第 20 行和第 25 行显示了处理结果。

【任务 5】商品图片管理

任务描述

在新增商品或修改商品时，都需要用到图片上传功能。该功能完成后的页面效果如图 9-18 所示。

图 9-18　商品图片管理

从图 9-18 中可以看出，页面中有“商品图片”和“商品相册”两个表单项，对应有两个“选择图片”按钮。需要注意的是，“商品图片”只能上传一张，而“商品相册”可以上传多张，最多允许上传 10 张图片。

当用户选择图片后，图片就会自动上传，并将预览图显示在“选择图片”按钮上方的图片列表中。若当前页面是修改商品，则商品原有的图片会显示在图片列表中。图片列表中的每一张图片都在右上角提供了“删除”功能，用于删除图片。

接口分析

上传商品图片和商品相册图片时，需要与后台 API 中的文件上传接口进行交互，该接口提供了上传文件、删除文件功能。下面分别进行讲解。

1. 上传文件

向 files 资源发送 POST 请求表示上传文件，具体说明如表 9-16 所示。

表9-16　上传文件

URL	http://api.shop-admin.localhost/v1/files	
基本信息	请求方式	内容类型
	POST	multipart/form-data

URL 参数	参数名	说明
	type	文件类型，目前只支持图片（image）
	relation	关联的表名（如 item 表示商品表）
	relation_id	关联的 id（如 item 表中的商品 id）
	name	关联的字段名（pic 或 album）
主体	参数名	说明
	name	与 URL 参数中的 name 一致
	……	（其他参数由程序自动发送）

在表 9-16 中，若 type 的值为 image， relation 的值为 item，relation_id 的值为 0，name 的值为 pic，则表示在数据库中的商品表中新增一条记录，将图片路径保存在新增记录的 pic 字段中。若将其中的 relation_id 改为已经存在的商品 id，则会更新该商品的 pic 字段。

上传文件成功后，返回的示例结果如下所示。

```
{
  "path": "image……90b124.jpg",     // 图片保存路径
  "relation_id": 1                  // 关联的 id（若新增了记录，则为新增的 id）
}
```

在上述结果中，path 是图片的保存路径。若要查看这张图片，需要在 path 前面添加如下 URL 地址来访问。

```
http://www.shop.localhost/upload/
```

需要注意的是，当 name 的值为 pic 时，上传的图片用于在前台商品列表中显示，服务器会自动生成 220*220 像素的缩略图，但不会保留用户上传的原图。

当 name 的值为 album 时，服务器会自动生成 3 种尺寸的缩略图，分别为 100*100、350*350、800*800，用于在商品相册组件中显示。由于同一张图片有 3 种尺寸，服务器返回的 path 是一个带有“[prefix]”的路径，示例结果如下。

```
image/2018-03/15/[prefix]71e1d4a63cdb33acb7b257dc8b422b66.jpg
```

在查看图片时，需要将路径中的“[prefix]”替换成图片尺寸前缀，3 种图片尺寸从小到大的前缀依次为“album_small_”“album_mid_”和“album_big_”。

2. 删除文件

向 files 资源发送 DELETE 请求表示删除文件，具体说明如表 9-17 所示。

表9-17 删除文件

URL	http://api.shop-admin.localhost/v1/files	
基本信息	请求方式	-
	DELETE	-
URL 参数	参数名	说明
	relation	关联的表名（如 item 表示商品表）
	relation_id	关联的 id（如 item 表中的商品 id）
	name	关联的字段名（pic 或 album）
	path	图片路径

从表 9-17 中可以看出，若要删除一张图片，不仅需要传递图片的路径，还需要传递与图片相关联的数据表名和该表中的记录 id。

代码实现

1. 创建图片上传组件

本项目使用 WebUploader 组件进行图片上传，在 index.html 文件的</head>前添加该组件的引入代码，如下所示。

【代码 9-45】 index.html

```
<link rel="stylesheet" href="js/webuploader-0.1.5/webuploader.css">
<script src="js/webuploader-0.1.5/webuploader.html5only.min.js"></script>
```

引入 WebUploader 后，在 content\item-list.html 文件中编写 createImage()函数，该函数位于【代码 9-39】第 7 行。具体代码如下。

【代码 9-46】 content\item-list.html

```
function createImage(image) {
  var pic = imagesUploader(obj.find('.upload_pic'), 'pic', 1);
  var album = imagesUploader(obj.find('.upload_album'), 'album', 10);
  function imagesUploader(obj, name, max) {}
}
```

在上述代码中，第 2 行用于创建“商品图片”的图片上传组件，第 3 行用于创建“商品相册”的图片上传组件。第 4 行的 imagesUploader()函数将在下一步中实现，该函数的第 1 个参数表示页面中对应的元素，第 2 个参数表示发送给后端 API 的 name 属性名，第 3 个参数表示文件数量最大限制。

接下来开始编写 imagesUploader()函数，具体代码如下。

【代码 9-47】 content\item-list.html

```
function imagesUploader(obj, name, max) {
  var picker_id = name + '_picker_' + id;
  obj.find('.webuploader-file-picker').attr('id', picker_id);
  var url = Config.api + 'file?type=image&name=' + name +
            '&relation=item&relation_id=' + id;
  var uploader = WebUploader.create({
    auto: true,                // 选完文件后，是否自动上传
    server: url,               // 文件接收服务端
    pick: '#' + picker_id,     // 选择文件的按钮 id
    fileVal: name,             // 上传文件的 name
    duplicate: true,           // 允许重复上传相同的文件
    accept: {                  // 只允许上传图片类型的文件
      title: 'Images',
      extensions: 'gif,jpg,jpeg,bmp,png',
      mimeTypes: 'image/*'
    }
  });
  // 其他操作……
  return uploader;
}
```

在上述代码中，第 2~3 行用于为“选择图片”按钮添加 id 属性，第 4~5 行用于根据后端 API 的要求将参数放入 URL 中，第 6~17 行用于创建 WebUploader 组件，第 19 行将创建后的组件对象返回。

2. 扩展上传组件的功能

在【代码 9-47】第 18 行的下面继续编写代码，利用 WebUploader 提供的事件对上传功能进行扩展，主要包括添加认证令牌、判断文件数量、显示预览图、显示上传进度、显示提示等功能。具体代码如下。

【代码 9-48】 content\item-list.html

```
1  // 执行上传前，将认证令牌放入请求头
2  uploader.on('uploadBeforeSend', function (obj, data, headers) {
3    headers.Authorization = Config.token;
4  });
5  // 当有文件添加时，判断文件数量
6  uploader.on('beforeFileQueued', function() {
7    if (obj.find('.webuploader-url').length >= max) {
8      $.messager.alert('操作失败', '最多只能上传' + max + '张图片！');
9      return false;
10   }
11 });
12 // 当有文件添加后，显示预览图
13 uploader.on('fileQueued', function(file) {
14   var li = fileItem(file.id, file.name);
15   var img = li.find('img');
16   obj.find('.webuploader-list').append(li);
17   uploader.makeThumb(file, function (error, src) {
18     if (error) {
19       img.replaceWith('<span class="webuploader-tip">不能预览</span>');
20       return;
21     }
22     img.attr('src', src);
23   }, 100, 100);
24 });
25 // 上传过程中创建进度条显示进度
26 uploader.on('uploadProgress', function(file, percentage) {
27   var li = $('#' + file.id);
28   var percent = li.find('.webuploader-progress span');
29   if (!percent.length) {
30     percent = $('<p class="webuploader-progress"><span></span></p>')
31               .appendTo(li).find('span');
32   }
33   percent.css('width', percentage * 100 + '%');
34 });
35 // 上传完成，删除进度条
36 uploader.on('uploadComplete', function (file) {
```

```
37   $('#' + file.id).find('.webuploader-progress').remove();
38 });
39 // 上传失败，显示错误提示
40 uploader.on('uploadError', function(file) {
41   var li = $('#' + file.id);
42   var error = li.find('div.webuploader-error');
43   if (!error.length) {
44     error = $('<div class="webuploader-error">上传失败</div>').appendTo(li);
45   }
46 });
```

在上述代码中，第 14 行调用了 fileItem()函数，该函数用于返回一个新创建的图片元素对象，参数 id 表示元素的 id 属性，参数 name 表示文件名。

在【代码 9-46】第 3 行的下面编写 fileItem()函数，具体代码如下。

【代码 9-49】 content\item-list.html

```
1  function fileItem(fid, name, src) {
2    var item = $('<div class="webuploader-file-item webuploader-
     thumbnail"><img></div>');
3    fid && item.attr('id', fid);
4    name && item.append('<div class="webuploader-info">'+ name +'</div>');
5    src && item.find('img').attr('src', src);
6    return item;
7  }
```

通过上述代码第 3~5 行可以看出，fid、name 和 src 都是可选参数，src 表示图片的 URL 地址。对于新上传的图片，在【代码 9-48】第 22 行为图片设置了 src 属性，不需要使用 src 参数。而 fileItem()函数在显示商品原有图片时也会用到，因此提供了 src 参数。

在【代码 9-48】第 46 行的下面继续编写代码，处理上传成功事件。具体代码如下。

【代码 9-50】 content\item-list.html

```
1  uploader.on('uploadSuccess', function(file, response) {
2    var li = $('#' + file.id);
3    if (!li.find('.webuploader-done').length) {
4      id === 0 && updateId(response.relation_id);
5      fileControl(li, response.path);
6    }
7  });
```

在上述代码中，第 4 行调用了 updateId()函数，用于在商品添加时，将后端 API 返回的商品 id 替换当前保存的商品 id。第 5 行调用了 fileControl()函数，用于为图片增加“删除”按钮和保存文件路径的隐藏域。

在【代码 9-46】第 3 行的下面编写 updateId()和 fileControl()函数，具体代码如下。

【代码 9-51】 content\item-list.html

```
1  function updateId(relation_id) {
2    function replace(obj) {
3      obj.server = obj.server.replace('&relation_id=0', '&relation_id=' + id);
4    }
5    id = relation_id;
```

```
6    replace(pic.options);
7    replace(album.options);
8  }
9  function fileControl(obj, path) {
10   obj.append('<div class="webuploader-opt"><span class="webuploader-
     del">[删除]</span></div>');
11   obj.append('<input class="webuploader-url" type="hidden" value="'
12    + path + '">');
13 }
```

在上述代码中，第 6~7 行用于更改 pic 和 album 对象中保存的后端 API 地址，第 5 行用于更改 id 变量保存的商品 id。更改后，当下次进行上传图片时，图片会与已经新增的商品记录进行关联。

3. 删除图片

上传图片成功后，就会在图片右上角显示“[删除]”按钮，通过该按钮可以删除图片。接下来在【代码 9-47】第 17 行的下面编写代码，实现删除图片功能，具体代码如下。

【代码 9-52】 content\item-list.html

```
1  obj.find('.webuploader-list').on('click', '.webuploader-del', function() {
2    var pObj = $(this).parents('.webuploader-file-item');
3    $.messager.confirm('确认删除', '您确定要立即删除此图片？', function(r) {
4      if (r) {
5        $.ajax({
6          type: 'DELETE',
7          url: Config.api + 'file?relation=item&relation_id=' + id,
8          data: {name: name, path: pObj.find('.webuploader-url').val()},
9          success: function() {
10           pObj.remove();
11         }
12       });
13     }
14   });
15 });
```

上述代码按照后端 API 的格式发送了 Ajax 请求，用于删除指定 id 的商品关联的指定路径的图片。

4. 修改商品时显示已有图片

在修改商品时，为 createImage()函数传入了参数，该参数包含了商品原有的图片路径。接下来在【代码 9-46】第 3 行的下面编写代码，读取图片路径，将图片显示在页面中。具体代码如下。

【代码 9-53】 content\item-list.html

```
1  var url = Config.uploadURL;
2  if (image.pic !== '') {
3    var picList = obj.find('.upload_pic .webuploader-list');
4    var item = fileItem(false, false, url + image.pic);
5    picList.append(item);
```

```
6    fileControl(item, image.pic);
7  }
8  if (image.album !== '') {
9    var albumList = obj.find('.upload_album .webuploader-list');
10   var albs = image.album;
11   for (var i in albs) {
12     var item = fileItem(false, false, url + albs[i].replace('[prefix]',
13     'album_small_'));
14     albumList.append(item);
15     fileControl(item, albs[i]);
16   }
17 }
```

在上述代码中，第 2~7 行用于将商品原有的“商品图片”显示在页面中，第 8~17 行用于将商品原有的“商品相册”显示在页面中。其中，第 1 行表示图片上传后的 URL，用于拼接在后端 API 返回的图片路径的前面，从而得到图片完整的 URL 地址。在显示商品相册时，由于后端 API 返回的图片路径不能直接显示，需要将路径中的“[prefix]”替换成图片尺寸前缀，因此第 12~13 行代码进行了替换。

最后，在代码【9-4】第 2 行的下面增加如下代码，配置 URL 地址。

【代码 9-54】 js\config.js

```
uploadURL: 'http://www.shop.localost/upload/',
```

【任务 6】商城首页

任务描述

商城首页主要由顶部导航栏、头部、主导航栏、内容区域、服务链接、页脚组成。主导航栏左侧的第一项是“全部商品分类”，在它下面显示了商品分类菜单，当鼠标滑到菜单上时，会显示该分类下的二级分类和三级分类，如图 9-19 所示。

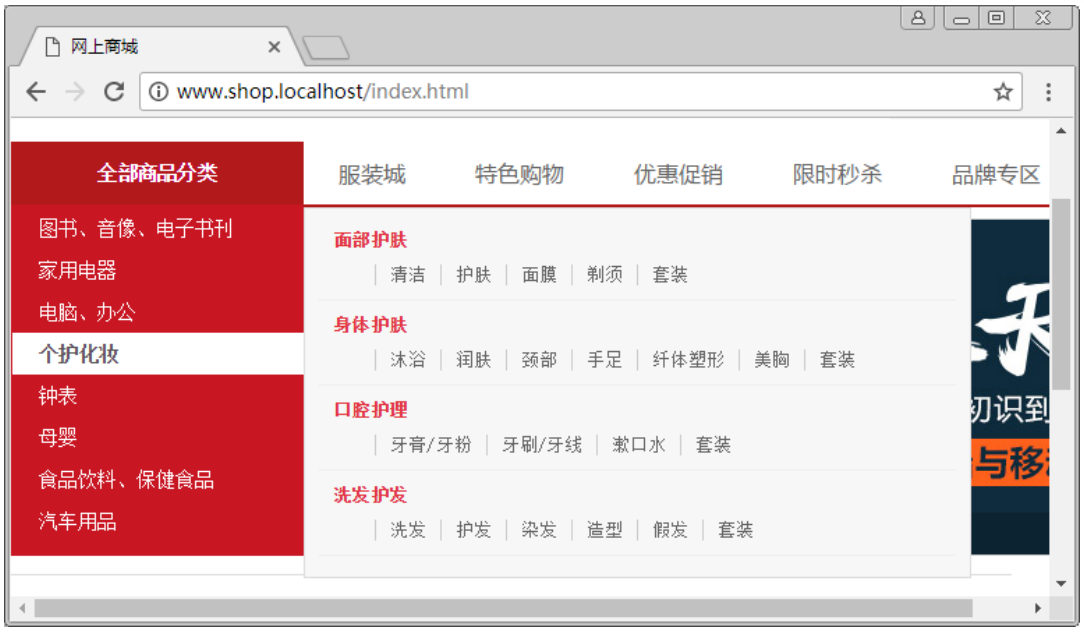

图 9-19　商品分类菜单

在商品分类菜单的左边，是“焦点图”模块，这些图片来自后端 API。在本页面中，“商品分类”“焦点图”“最新动态”和“精品推荐”模块中的数据，都是使用 jQuery 到后端 API 中获取的，获取后使用 art-template 将数据填入到模板中。

接口分析

在开发商城首页功能时，用到的前台 API 有商品分类（categories）、焦点图（slides）、最新动态（news）和精品推荐（goods），下面分别进行讲解。

1. 查询分类

向 categories 资源发送 GET 请求表示查询分类数据，具体说明如表 9-18 所示

表9-18　查询分类

URL	http://api.shop.localhost/v1/categories	
基本信息	请求方式	-
	GET	-
URL 参数	参数名	说明
	view	前端组件名（可选值：nav、crumbs）
	id	分类 id（view=crumbs 时有效）

当 view 为 nav 时，会返回所有 status 为 1 的分类数据，示例结果如下。

```
[
  {"id": 1183, "parent_id": 1, "name": "test"}, ……
]
```

当 view 为 crumbs 时，会返回 id 的分类路径。例如，查询 id 为 3 的分类的路径，返回的示例结果如下所示。

```
 [
  {"id": 1, "parent_id": 0, "name": "图书、音像、电子书刊"},
  {"id": 2, "parent_id": 1, "name": "电子书刊"},
```

```
  {"id": 3, "parent_id": 2, "name": "电子书"}
]
```

从上述结果可以看出，查询结果包含了 id 为 3 的分类和它所有的上级分类，且顺序为上级分类在前面，子级分类在后面。

2. 查询焦点图

向 slides 资源发送 GET 请求表示查询焦点图数据，具体说明如表 9-19 所示。

表9-19 查询焦点图

URL	http://api.shop.localhost/v1/slides	
基本信息	请求方式	-
	GET	-

请求成功后，返回的结果示例如下所示。

```
[
  {"id": 1, "title": "图片 1", "pic": "image……32.jpg", "url": "#"},
]
```

在上述结果中，id 表示焦点图 id，title 表示图片的名称，pic 表示图片路径，url 表示焦点图的链接地址。

3. 查询最新动态

向 news 资源发送 GET 请求表示查询最新动态数据，具体说明如表 9-20 所示。

表9-20 查询最新动态

URL	http://api.shop.localhost/v1/news	
基本信息	请求方式	-
	GET	-

请求成功后，返回的示例结果如下所示。

```
[
  {"id": 1, "title": "标题 1", "color": "", "url": "#"},
]
```

在上述结果中，id 表示动态 id，title 表示动态的标题，color 表示标题的文本颜色，url 表示动态的链接地址。其中，color 的值与 CSS 中的 color 样式的格式要求相符，如果为空则表示使用默认颜色。

4. 查询精品推荐

向 goods 资源发送 GET 请求表示查询精品推荐数据，具体说明如表 9-21 所示。

表9-21 查询精品推荐

URL	http://api.shop.localhost/v1/goods	
基本信息	请求方式	-
	GET	-
URL 参数	参数名	说明
	count	返回的商品数量

请求成功后，返回的示例结果如下所示。

```
[
  {"id": 26, "title": "", "price": "100.00", "pic": "image……60f.jpg"},
```

```
]
```

在上述结果中，id 表示商品 id，title 表示商品标题，price 表示商品价格，pic 表示商品图片。

代码实现

1. 准备页面

从本任务开始，代码在 www.shop.localhost 虚拟主机中进行编写。首先编写前台页面文件，由于该页面的代码过多，下面仅展示页面的基本结构，完整代码读者可通过本书配套源代码获取。

【代码 9-55】 index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>网上商城</title>
    <link rel="stylesheet" href="css/style.css">
    <script src="js/jquery-1.12.4.js"></script>
    <script src="js/art-template-4.12.1/template-web.js"></script>
    <script src="js/config.js"></script>
  </head>
  <body>
    <!-- 顶部导航栏-->
    <div class="top"></div>
    <!-- 头部（从左到右依次是 LOGO、搜索框、快捷按钮） -->
    <div class="header"></div>
    <!-- 主导航栏 -->
    <div class="nav"></div>
    <!-- 内容区域 -->
    <div class="content"></div>
    <!-- 服务链接 -->
    <div class="service"></div>
    <!-- 页脚 -->
    <div class="footer"></div>
    <!-- 引入 JavaScript 公共代码文件 -->
    <script src="js/common.js"></script>
  </body>
</html>
```

在上述代码中，第 6 行引入了样式文件，第 8 行引入了 art-template 模板引擎，第 9 行引入了前台配置文件，第 25 行引入了 JavaScript 公共代码文件。其中，JavaScript 公共代码文件用于保存在各个页面中都可以使用的代码。

接下来创建前台的配置文件，具体代码如下。

【代码 9-56】 js\config.js

```
var Config = {
  api: 'http://api.shop.localhost/v1/',
  uploadURL: 'upload/'
};
```

5. 载入内容区域

在前台的各个页面中，除了内容区域，其他部分是不变的，因此我们可以将内容区域分离出来，保存到单独的文件中，在页面打开后使用 Ajax 进行载入。

在 index.html 的内容区域中编写代码，创建一个 id 为 content 的 元素作为 Ajax 加载内容后的容器。具体代码如下。

【代码 9-57】 index.html

```
<!-- 内容区域 -->
<div class="content">
  <div id="content"></div>
</div>
```

需要注意的是，前台不同的页面，URL 地址是不同的，但它们都共用同一个网页文件，即 index.html。为了实现这个效果，本书在配套源代码中提供了“.htaccess”文件，即 Apache 分布式配置文件。在该文件中，已经编写了配置代码，对前台进行 URL 重写。当用户访问的页面不存在时，会自动重写给 index.html。经过处理后，请求 URI 与实际请求网页、通过 Ajax 载入的内容区域网页的对应关系如表 9-22 所示。

表9-22 前台 URL 重写对照表

页面名称	请求 URI	实际请求网页	内容区域网页
首页	/	index.html	content\index.html
	/index.html	index.html	content\index.html
商品列表页	/find-{分类 id}.html	index.html	content\find.html
商品查看页	/item-{商品 id}.html	index.html	content\item.html
购物车	/cart.html	index.html	content\cart.html

在表 9-22 中，以商品列表页为例，当用户请求“http://www.shop.localhost/find-1.html”时，由于 find-1.html 这个文件不存在，Web 服务器实际返回的是 index.html。在 index.html 中，通过 JavaScript 读取当前 URL 地址，提取出当前的请求 URI“find-1.html”。提取后，将分类 id 分离出来，得到页面文件 find.html 和值为 1 的分类 id。然后通过 Ajax 加载 content 目录下的 find.html，在 find.html 中，使用当前请求的分类 id 到后端 API 中读取数据。

为了实现这个效果，下面在 JavaScript 公共代码文件中编写代码，具体代码如下。

【代码 9-58】 js\common.js

```
var Common = {
  id: 0,                // 当前请求 id
  content: 'index',     // 当前请求的内容区域（对应 content/index.html）
  init: function() {
    this.getParams(location.href);
  },
  getParams: function(url) {
    var url = url.match(/\/(\w+)(?:-(\d+))?\.html/);
    if (url !== null) {
        this.content = url[1];
```

```
11        this.id = url[2] === undefined ? 0 : url[2];
12      }
13    },
14    getContent: function() {
15      var content = this.content;
16      $.ajax({
17        type: 'GET', url: 'content/' + content + '.html', cache: false,
18        success: function (data) {
19          $('#content').html(data);
20      });
21    },
22  };
23  Common.init();
24  Common.getContent();
```

在上述代码中，第 7 行的 getParams()方法用于从参数 url 中获取请求的内容区域和 id，如果内容区域不存在，则默认值为“index”；如果 id 不存在，则默认值为 0。第 14 行的 getContent()方法用于根据当前请求的内容区域，通过 Ajax 载入对应的页面。其中，第 17 行将 cache 设为 false，表示不对 Ajax 进行缓存，从而方便调试程序。

上述代码实现了根据请求 URI 加载内容区域。为了更好的用户体验，当用户单击页面中的链接时，无需重新载入 index.html，只需更新内容区域即可。为了实现这个效果，在【代码 9-58】第 5 行的下面编写如下代码。

【代码 9-59】 js\common.js

```
1  $(document).on('click', '.pjax', function() {
2    var url = $(this).attr('href');
3    Common.getParams('/' + url);
4    Common.getContent();
5    history.pushState(undefined, undefined, url);
6    return false;
7  });
```

上述代码实现了当单击 class 为“pjax”的链接时，更新内容区域的内容和地址栏中显示的地址，并阻止默认触发的页面跳转。

6. 主导航栏的商品分类

在 index.html 中找到主导航栏区域，利用 art-template 输出商品分类，具体代码如下。

【代码 9-60】 index.html

```
1  <div class="nav">
2    <div id="category_show">
3      <div><a class="pjax" href="find-0.html">全部商品分类</a></div>
4      <div id="category_list"></div>
5      <script id="category_list_tpl" type="text/html">
6      </script>
7    </div>
8    <ul><li>服装城</li><li>特色购物</li><li>优惠促销</li><li>限时秒杀</li>
9        <li>品牌专区</li><li>服务中心</li></ul>
10 </div>
```

在上述代码中，第 5~6 行用于编写模板，第 4 行用于填充模板编译结果。

在【代码 9-58】第 20 行的下面编写如下代码，从后端 API 获取数据。

【代码 9-61】 js\common.js

```
1  var categoryList = $('#category_list');
2  $.get(Config.api + 'categories?view=nav', function(data) {
3    var html = template('category_list_tpl', {item: data});
4    categoryList.html(html);
5  });
```

获取数据后，在【代码 9-60】第 6 行的下面编写如下代码，输出数据。

【代码 9-62】 index.html

```
1  {{each item v1}}
2  {{if v1.parent_id === 0}}
3  <div class="category-item">
4    <div class="category-main">
5      <a class="pjax" href="find-{{v1.id}}.html">{{v1.name}}</a>
6    </div>
7    <div class="category-sub" style="display:none;">
8      {{each item v2}}
9      {{if v2.parent_id === v1.id}}
10     <dl>
11       <dt><a class="pjax" href="find-{{v2.id}}.html">{{v2.name}}</a></dt>
12       <dd>
13       {{each item v3}}
14       {{if v3.parent_id === v2.id}}
15         <a class="pjax" href="find-{{v3.id}}.html">{{v3.name}}</a>
16       {{/if}}
17       {{/each}}
18       </dd>
19     </dl>
20     {{/if}}
21     {{/each}}
22   </div>
23 </div>
24 {{/if}}
25 {{/each}}
```

当用户使用鼠标滑到分类菜单中时，会显示所选分类下的二级、三级分类。为了实现这个效果，在【代码 9-58】第 4 行的下面编写如下代码。

【代码 9-63】 js\common.js

```
1  $('.category-item').hover(function() {
2    $(this).find('.category-sub').show();
3    $(this).children('.category-main').children('a').addClass('on');
4  }, function() {
5    $(this).find('.category-sub').hide();
6    $(this).children('.category-main').children('a').removeClass('on');
```

```
});
```

上述代码用于实现当鼠标滑过菜单中的每一行时，显示该行内的“.category-sub”元素，并为该行添加 class 属性“on”，用于改变样式，呈现出被选择的效果。

由于前台各页面共用了 index.html，而分类菜单在首页是默认展开的，其他页面则默认是收起的。为了实现这个效果，在【代码 9-58】第 18 行的下面编写如下代码。

```
var categoryList = $('#category_list');
var categoryShow = $('#category_show');
if (content === 'index') {
  categoryList.show();
  categoryShow.off('mouseenter').off('mouseleave');
} else {
  categoryList.hide();
  categoryShow.hover(function() {
    categoryList.show();
  }, function() {
    categoryList.hide();
  });
}
```

上述代码第 4~5 行用于在首页中展开分类菜单，第 7~12 行用于在其他页面中收起分类菜单。其中，第 8~12 行用于当主导航栏中的“全部商品分类”被鼠标移入时，展开分类菜单，被鼠标移出时，收起分类菜单；第 5 行则取消了鼠标移入移除的效果。

7. 焦点图

焦点图是仅在首页中显示的模块。编写 content\index.html 文件，具体代码如下。

【代码 9-64】 content\index.html

```
<div class="slide">
  <div class="slide-num"></div>
  <ul></ul>
  <script id="slide_tpl" type="text/html">
    {{each item v}}
    <li><a href="{{v.url}}"><img src="{{v.pic}}"
    alt="{{v.title}}"></a></li>
    {{/each}}
  </script>
  <script src="js/jquery.slide.js"></script>
</div>
<script>
  $.get(Config.api + 'slides', function(data) {
    Common.slidePath(data);
    $('.slide ul').html(template('slide_tpl', {item: data}));
    $('.slide').slide();
  });
</script>
```

上述代码用于从后端 API 获取焦点图数据，然后输出到页面中。第 10 行引入了焦点图插件“js/jquery.slide.js”，读者可通过配套源代码获取该插件。由于后端 API 返回的图片路

径不包含基础 URL，第 14 行通过 Common.slidePath()方法对图片路径进行了处理。在【代码 9-58】第 21 行的下面编写该方法，具体代码如下。

【代码 9-65】 js\common.js

```
1  slidePath: function(data) {
2    for (var i in data) {
3      data[i].pic = Config.uploadURL + data[i].pic;
4    }
5  },
```

上述代码实现了在焦点图的图片路径前拼接了上传文件的 URL。

8. 最新动态

编写“最新动态”模块的相关代码，具体代码如下。

【代码 9-66】 content\index.html

```
1  <div class="news">
2    <div class="news-title">最新动态</div>
3    <ul class="news-content"></ul>
4    <script id="news_tpl" type="text/html">
5      {{each item v}}
6      <li><a style="{{if v.color !== ''}}color:{{v.color}}{{/if}}"
7      href="{{v.url}}" target="_blank">{{v.title}}</a></li>
8      {{/each}}
9    </script>
10 </div>
11 <script>
12   $.get(Config.api + 'news', function(data) {
13     $('.news-content').html(template('news_tpl', {item: data}));
14   });
15 </script>
```

在上述代码中，第 6 行在输出每个动态链接时，判断该动态是否设置了文本颜色，如果已经设置了颜色，则将链接文本设为指定颜色。

9. 精品推荐

编写“精品推荐”模块的相关代码，具体代码如下。

【代码 9-67】 content\index.html

```
1  <div class="best">
2    <div class="best-title">精品推荐</div>
3    <div class="best-content"></div>
4    <script id="best_tpl" type="text/html">
5      {{each item v}}
6      <ul class="item">
7        <li><a href="item-{{v.id}}.html" target="_blank">
8        <img src="{{v.pic}}" alt="{{v.title}}"></a></li>
9        <li class="goods"><a href="item-{{v.id}}.html"
10       target="_blank">{{v.title}}</a></li>
11       <li class="price">￥{{v.price}}</li>
```

```
12     </ul>
13     {{/each}}
14   </script>
15 </div>
16 <script>
17   $.get(Config.api + 'goods?count=6', function(data) {
18     Common.itemPicPath(data);
19     $('.best-content').html(template('best_tpl', {item: data}));
20   });
21 </script>
```

在上述代码中，第 17 行在请求后端 API 时，传递了参数“count=6”，表示最多取出 6 条记录。由于返回的图片路径不包含基础 URL，第 18 行调用了 Common.itemPicPath()方法对图片路径进行处理。在【代码 9-65】第 5 行的下面编写该方法，具体代码如下。

【代码 9-68】 js\common.js

```
1  itemPicPath: function(data) {
2    for (var i in data) {
3      if (data[i].pic === '') {
4        data[i].pic = 'img/preview.jpg';
5      } else {
6        data[i].pic = Config.uploadURL + data[i].pic;
7      }
8    }
9  },
```

在上述代码中，第 3 行判断图片路径是否为空字符串，如果是，说明该商品没有上传图片，此时执行第 4 行代码，将图片路径更改为“img/preview.jpg”，以一个默认的图片来替代。如果不是，则执行第 6 行代码，在图片路径前拼接上传文件的 URL。

【任务 7】商品列表

任务描述

当用户在主导航栏的商品分类菜单中单击某一个分类时，就会进入到商品列表页面。该页面用于浏览某一分类下的商品，查看商品名称、预览图和价格，并提供了分页、筛选和排序等功能。其页面效果如图 9-20 所示。

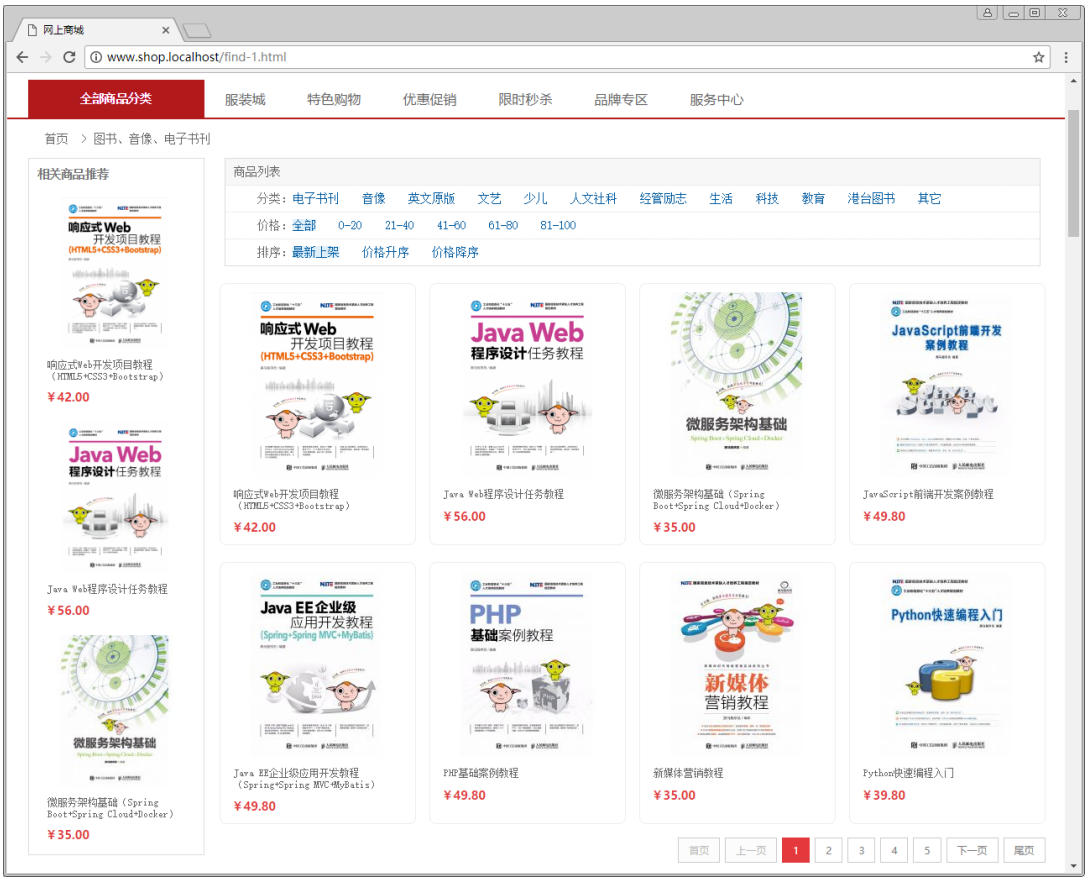

图 9-20　商品列表页

在图 9-20 中，当用户切换分类时，会在商品列表的上方提供一个分类导航，用于显示当前分类的所在路径。在列表的右侧，还提供了“相关商品推荐”，用于向用户推荐一些相关的商品。

接口分析

在开发商品列表页面时，需要与前台 API 中的商品接口进行交互，该接口提供了查询商品列表的功能，具体如表 9-23 所示。

表9-23　查询商品列表

URL	http://api.shop.localhost/v1/items	
基本信息	请求方式	-
	GET	-
URL 参数	参数名	说明
	price_max	价格区间最大值
	price_min	价格区间最小值
	sort	排序方式
	cid	分类 id
	page	页码值（最小值为 1）
	pagesize	每页条数（最大值为 50）

在表 9-23 的 URL 参数中，若 price_max 和 price_min 的值都为 0，则表示不对价格进行筛选；sort 支持的排序方式有 id-desc（id 降序）、price-asc（价格升序）和 price-desc（价格降序）；cid 表示查询指定分类 id 的商品，如果指定的分类中包含子分类，则返回的结果中包含子分类的商品。

请求成功后，返回的示例结果如下所示。

```
{
  "priceMax": 100,                    // 符合查询条件的商品中的最大价格值
  "item": [                           // 当前页码值下的商品
    {"id": 6, "title": "", "price": "100.00", "pic": "image……0f.jpg"},
    ……
  ],
  "total": 1,                         // 符合查询条件的总记录数
  "category": [                       // 分类 id（cid）的子分类
    {"id": 1183, "name": "test"},
    ……
  ],
  "recommend": [                      // 相关商品推荐
   {"id": 8, "title": "", "price": "100.00", "pic": "image……0f.jpg"},
   ……
  ]
}
```

上述结果提供了商品列表页面所需的数据。在开发具体功能时，从以上结果中取出所需要的值即可。

代码实现

1. 分类导航

创建商品列表页 content\find.html 文件，在该文件中编写分类导航，具体代码如下。

【代码 9-69】 content\find.html

```
<div class="crumbs"></div>
<script id="crumbs_tpl" type="text/html">
  <a href="index.html">首页</a>
  {{each item v}}
    <i></i><a class="pjax" href="find-{{v.id}}.html">{{v.name}}</a>
  {{/each}}
</script>
<script>
  (function() {
    var params = {price_max: 0, price_min: 0, sort: 'id-desc',
                  cid: Common.id, page: 1, pagesize: 20};
    var url = Config.api + 'categories?view=crumbs&id=' + params.cid;
    $.get(url, function(data) {
      $('.crumbs').html(template('crumbs_tpl', {item: data}));
```

```
15    });
16  })();
17 </script>
```

上述代码实现了向后端 API 请求数据，然后填入到页面中。第 10 行保存了当前页面用到的请求参数，在发送其他请求时会用到。

2. 商品列表

在【代码 9-69】第 15 行的下面继续编写代码，向后端 API 请求商品列表数据。具体代码如下。

【代码 9-70】 content\find.html

```
1  var isFirst = true;    // 是否为页面打开后首次加载数据
2  loadData();            // 加载数据
3  function loadData() {
4    // 请求商品列表数据
5    $.get(Config.api + 'items', params, function(data) {
6      Common.itemPicPath(data.item);
7      $('.find-item').html(template('find_item_tpl', {item: data.item}));
8      if (isFirst) {
9        isFirst = false;
10       firstLoadData(data);
11     }
12   });
13 }
```

在上述代码中，第 5 行请求了商品列表数据，第 8 行用于判断当前是否为页面加载后首次加载数据，如果是，则调用 firstLoadData()函数执行首次加载时的一些操作，该函数的具体代码将在后面的步骤中编写。

在【代码 9-69】第 7 行的下面编写代码，输出商品列表内容，具体代码如下。

【代码 9-71】 content\find.html

```
1  <div class="find">
2    <div class="find-ls">
3      <div class="find-item"></div>
4      <script id="find_item_tpl" type="text/html">
5        {{each item v}}
6        <ul>
7          <li><a href="item-{{v.id}}.html" target="_blank">
8          <img src="{{v.pic}}" alt="{{v.title}}"></a></li>
9          <li class="find-item-name"><a href="item-{{v.id}}.html"
10         target="_blank">{{v.title}}</a></li>
11         <li class="find-item-price">￥{{v.price}}</li>
12       </ul>
13       {{/each}}
14     </script>
15   </div>
16 </div>
```

3. 相关商品推荐

在【代码 9-70】第 12 行的下面编写代码，获取推荐商品信息。具体代码如下。

【代码 9-72】 content\find.html

```
function firstLoadData(data) {
  // 相关商品推荐
  Common.itemPicPath(data.recommend);
  $('.find-ad-content').html(template('find_ad_tpl', {item:
  data.recommend}));
}
```

在【代码 9-71】第 1 行的下面编写代码，通过模板输出数据，具体代码如下。

【代码 9-73】 content\find.html

```
<div class="find-ad">
  <div class="find-ad-title">相关商品推荐</div>
  <div class="find-ad-content"></div>
  <script id="find_ad_tpl" type="text/html">
    {{each item v}}
    <ul class="find-ad-item">
      <li><a href="item-{{v.id}}.html" target="_blank">
      <img src="{{v.pic}}" alt="{{v.title}}"></a></li>
      <li class="find-ad-name"><a href="item-{{v.id}}.html"
      target="_blank">{{v.title}}</a></li>
      <li class="find-ad-price">￥{{v.price}}</li>
    </ul>
    {{/each}}
  </script>
</div>
```

4. 按分类筛选商品

商品列表支持多种筛选条件，在【代码 9-71】第 2 行的下面编写代码，提前准备这些筛选条件的 HTML 结构。具体代码如下。

【代码 9-74】 content\find.html

```
<ul class="selector">
  <li class="selector-title">商品列表</li>
  <!-- 分类 -->
  <li class="selector-category"></li>
  <script id="selector_category_tpl" type="text/html">
    {{if item.length > 0}}
      <dl>
        <dt>分类：</dt>
        <dd>
          {{each item v}}
          <a class="pjax" href="find-{{v.id}}.html">{{v.name}}</a>
          {{/each}}
        </dd>
```

```
14      </dl>
15    {{/if}}
16  </script>
17  <!-- 价格 -->
18  <!-- 排序 -->
19 </ul>
```

在代码【9-72】第 5 行的下面编写代码，通过模板输出数据，具体代码如下。

【代码 9-75】 content\find.html

```
1  // 分类
2  $('.selector-category').html(template('selector_category_tpl',
3  {id: data.cid ,item: data.category}));
```

5. 按价格筛选商品

在【代码 9-74】第 17 行的下面编写代码，实现商品价格筛选模板，具体代码如下。

【代码 9-76】 content\find.html

```
1  <li>
2    <dl>
3      <dt>价格：</dt>
4      <dd class="selector-price"></dd>
5      <script id="selector_price_tpl" type="text/html">
6        <a href="#" data-param="0-0">全部</a>
7        {{each item v}}
8        <a href="#" data-param="{{v}}">{{v}}</a>
9        {{/each}}
10     </script>
11   </dl>
12 </li>
```

在上述代码中，价格链接有一个“data-param”自定义属性，该属性表示该链接的价格区间，如“0-20”表示筛选价格为 0~20 范围内的商品。如果不需要对价格进行筛选，则使用“0-0”来表示。

在【代码 9-76】第 12 行的下面编写代码，通过模板输出数据，具体代码如下。

【代码 9-77】 content\find.html

```
1  // 生成价格链接
2  $('.selector-price').html(template('selector_price_tpl',
3  {item: priceDist(data.priceMax, 5)})).find('a:first')
4  .addClass('selector-curr');
```

在上述代码中，第 3 行调用了 priceDist()函数，该函数用于输出价格筛选的链接，第 1 个参数表示价格最大值，第 2 个参数表示生成链接的个数。

在【代码 9-72】第 6 行的下面编写 priceDist()函数，具体代码如下。

【代码 9-78】 content\find.html

```
1  function priceDist(max, count) {
2    if (max <= 0) {
3      return '';
4    }
5    var size = Math.ceil(max / Math.max(count, 1));
```

```
6    var end = size, start = 0, rst = [];
7    for (var i = 0; i < count; ++i) {
8      rst.push(start + '-' + end);
9      start = end + 1;
10     end += size;
11   }
12   return rst;
13 }
```

从上述代码可以看出，priceDist()函数用于生成从 1~max 价格范围内的 count 个价格链接。假设 max 为 100，count 为 5，则生成的链接分别为“0-20”“21-40”“41-60”“61-80”“81-100”。

在【代码 9-78】第 13 行的下面继续编写代码，为价格链接添加单击事件。当价格链接被单击时，发送新的价格区间参数，重新请求商品列表数据。具体代码如下。

【代码 9-79】 content\find.html

```
1  $('.selector-price').on('click', 'a', function() {
2    var param = $(this).attr('data-param');
3    params.price_max = param[0];
4    params.price_min = param[1];
5    loadData();
6    $(this).addClass('selector-curr').siblings()
7    .removeClass('selector-curr');
8    return false;
9  });
```

在上述代码中，第 3~4 行更新了请求参数中的价格值，从而使第 5 行调用 loadData()函数请求数据时，按照指定的价格区间进行查询。

6. 商品列表排序

在【代码 9-74】第 18 行的下面编写代码，创建排序链接，具体代码如下。

【代码 9-80】 content\find.html

```
1  <li>
2    <dl>
3      <dt>排序：</dt>
4      <dd class="selector-sort">
5        <a href="#" data-param="id-desc">最新上架</a>
6        <a href="#" data-param="price-asc">价格升序</a>
7        <a href="#" data-param="price-desc">价格降序</a>
8      </dd>
9    </dl>
10 </li>
```

在【代码 9-77】第 4 行的下面编写代码，将排序中的第 1 个链接设为选中效果，具体代码如下。

【代码 9-81】 content\find.html

```
1  // 将默认排序链接设为选中效果
2  $('.selector-sort a:first').addClass('selector-curr');
```

在【代码 9-79】第 9 行的下面编写代码，实现在单击排序链接时，发送新的排序参数，

重新请求商品列表数据。具体代码如下。

【代码 9-82】 content\find.html

```
1  $('.selector-sort a').click(function() {
2    params.sort = $(this).attr('data-param');
3    loadData();
4    $(this).addClass('selector-curr').siblings().
5    removeClass('selector-curr');
6    return false;
7  });
```

7. 商品列表分页

在【代码 9-71】第 14 行的下面编写代码，创建分页链接，具体代码如下。

【代码 9-83】 content\find.html

```
1  <div class="pagelist">
2    <span class="pagelist-first pagelist-btn">首页</span>
3    <span class="pagelist-prev pagelist-btn">上一页</span>
4    <div class="pagelist-nav"></div>
5    <span class="pagelist-next pagelist-btn">下一页</span>
6    <span class="pagelist-last pagelist-btn">尾页</span>
7  </div>
8  <script src="js/jquery.page.js"></script>
```

在上述代码中，第 8 行引入了分页插件，读者可通过配套源代码获取该插件。接下来在【代码 9-72】的基础上按照如下代码进行修改。

【代码 9-84】 content\find.html

```
1  var pagelist = $('.pagelist').page({
2    total: data.total,
3    size: params.pagesize
4  });
5  pagelist.click(function(page) {
6    params.page = page;
7    loadData();
8  });
```

在上述代码中，第 1~4 行调用了分页插件的 page()方法，并传入了总记录数和每页显示的记录数。第 5~8 行用于当分页链接被单击时，将 params.page 更新为被单击的链接所对应的页码值，然后重新请求商品列表数据。

【任务 8】商品详情

任务描述

当用户在商品列表中单击一件商品后，就会打开商品详情页，显示商品的详细信息。商品详情页开发完成后的效果如图 9-21 所示。

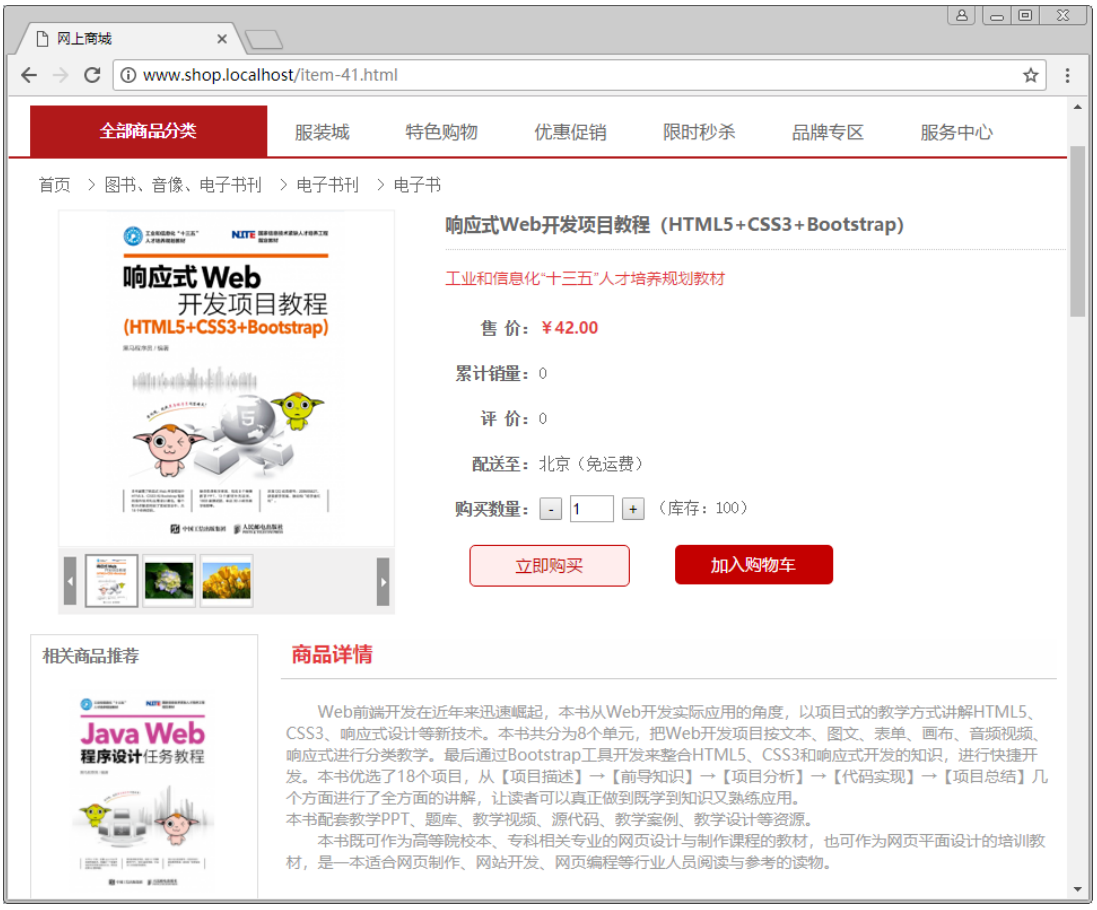

图 9-21　商品详情页

在商品详情页中，主导航栏下方是一个分类导航，用于显示当前商品所属的分类。在树状分类结构中的路径，其显示效果与商品列表中的分类导航相同。

在分类导航下方，左侧是商品相册，右侧是商品信息。其中，商品相册中的图片来自后台上传的图片，浏览时，可以在多张图片之间切换，并且支持鼠标滑过小图查看大图。商品信息中包含了商品的名称、卖点、价格、库存等信息。由于本项目并未实现下订单、在线支付和评价功能，因此累计销量与评价皆显示为 0。

在商品相册的下方，是相关商品推荐，该功能与商品列表页中的相关商品推荐相同。在其右边，是商品详情，用于显示后台中通过在线编辑器录入的商品详情数据。

接口分析

开发商品详情页面时，需要与前台 API 中的商品接口进行交互，该接口提供了查询指定 id 的商品信息的功能，具体如表 9-24 所示。

表9-24　查询商品信息

URL	http://api.shop.localhost/v1/items/:id	
基本信息	请求方式	-
	GET	-

请求成功后，返回的示例结果如下所示。

```
{
  "id": 6, "cid": 3, "title": "", "sell_point": "",
```

```
  "price": "100.00", "num": 100, "album": [], "content": "",
  "recommend": [
    {"id": 26, "title": "", "price": "100.00", "pic": "image……0f.jpg"},
    ……
  ]
}
```

在上述结果中，id、cid、title、sell_point、price、num、album、content 都是数据库中商品表的字段，在前面已经讲过。其中，album 字段已经按照“|”分割成了数组。recommend 中保存的是自动推荐的相关商品。

代码实现

1. 查询商品数据

创建商品详情页对应的 content\item.html 文件，在该文件中编写代码，实现根据 id 查询商品详情，具体代码如下。

【代码 9-85】 content\item.html

```
<script>
  (function() {
    var id = Common.id;
    $.get(Config.api + 'items/' + id, function(data) {
      // 查询成功后，将数据填入到模板中
    }).fail(function() {
      alert('商品数据不存在！');
      location.href = 'index.html';
    });
  });
</script>
```

2. 分类导航

查询到商品所属分类后，即可通过 data.cid 查询分类导航。在【代码 9-85】第 1 行的上面编写代码，在页面中编写分类导航的模板，具体代码如下。

【代码 9-86】 content\item.html

```
<div class="crumbs"></div>
<script id="crumbs_tpl" type="text/html">
  <a class="pjax" href="index.html">首页</a>
  {{each item v}}
    <i></i><a class="pjax" href="find-{{v.id}}.html">{{v.name}}</a>
  {{/each}}
</script>
```

完成模板后，在【代码 9-85】第 5 行的下面编写如下代码，将数据填入模板。

【代码 9-87】 content\item.html

```
$.get(Config.api + 'categories?view=crumbs&id=' + data.cid,
function(data) {
```

```
3    $('.crumbs').html(template('crumbs_tpl', {item: data}));
4  });
```

3. 商品信息和商品详情

在【代码 9-86】第 7 行的下面编写代码，准备页面结构，具体代码如下。

【代码 9-88】 content\item.html

```
1  <div class="item">
2    <!-- 商品相册 -->
3    <div class="album"></div>
4    <!-- 商品信息 -->
5    <div class="item-info"></div>
6    <script id="item_info_tpl" type="text/html">
7    </script>
8    <!-- 相关商品推荐 -->
9    <div class="item-ads"></div>
10   <!-- 商品详情 -->
11   <div class="item-desc">
12     <div class="item-desc-title">商品详情</div>
13     <div class="item-desc-content"></div>
14   </div>
15 </div>
```

在【代码 9-87】第 3 行的下面编写如下代码，将商品信息和商品详情数据填入模板。

【代码 9-89】 content\item.html

```
1  $('.item-info').html(template('item_info_tpl', {item: data}));
2  $('.item-desc-content').html(data.content);
3  loadSuccess();
```

在上述代码中，第 3 行调用了 loadSuccess()函数，表示在加载数据成功后执行的一些操作，该函数将在后面的步骤中实现。

在【代码 9-88】第 6 行的下面编写代码，创建商品信息模板，具体代码如下。

【代码 9-90】 content\item.html

```
1  <h1>{{item.title}}</h1>
2  <div class="item-info-sell">{{item.sell_point}}</div>
3  <table>
4    <tr><th>售 价：</th><td><span class="item-info-price">
5    ￥{{item.price}}</span></td></tr>
6    <tr><th>累计销量：</th><td>0</td></tr>
7    <tr><th>评 价：</th><td>0</td></tr>
8    <tr><th>配送至：</th><td>北京（免运费）</td></tr>
9    <tr>
10     <th>购买数量：</th>
11     <td>
12       <input type="button" value="-" class="item-num-sub">
13       <input type="text" value="1" id="item_num" class="item-num">
14       <input type="button" value="+" class="item-num-add">
15       （库存：<span class="item-num-stock">{{item.num}}</span>）
```

```
    </td>
  </tr>
  <tr>
    <td colspan="2" class="item-info-button">
      <a href="#">立即购买</a>
      <a href="#" id="cart_add">加入购物车</a>
    </td>
  </tr>
</table>
```

在上述代码中，第 12~14 行用于控制商品购买数量。其中，第 13 行是一个文本框，表示购买数量，第 12 行是一个“-”按钮，表示减少数量，第 14 行是一个“+”按钮，表示增加数量。

4. 控制商品数量

在【代码 9-85】第 9 行的下面编写 loadSuccess()函数，通过该函数对商品的购买数量进行控制，具体代码如下。

【代码 9-91】 content\item.html

```
function loadSuccess() {
  var num = $('#item_num');
  var stock = parseInt($('.item-num-stock').text());
  // 减少购买数量
  $('.item-num-sub').click(function() {
    var n = parseInt(num.val());
    if (n < 1) {
      return false;
    }
    num.val(n - 1);
  });
  // 增加购买数量
  $('.item-num-add').click(function() {
    var n = parseInt(num.val());
    if (n > stock) {
      return false;
    }
    num.val(n + 1);
  });
  // 自动纠正购买数量
  num.keyup(function() {
    var n = parseInt($(this).val());
    if (n < 1) {
      $(this).val(1);
    } else if (n > stock) {
      $(this).val(stock);
    }
  });
```

```
29 }
```

在上述代码中，第 4~11 行用于在单击“-”按钮时，将购买数量文本框中的数字减 1，并限制最小值为 1；第 12~19 行用于在单击“+”按钮时，将购买数量文本框中的数字加 1，并限制最大值为库存数；第 20~28 行用于在购买数量文本框中直接输入数字时，将数字限制在“1~库存数”这个范围内。

5. 商品相册

在【代码 9-89】第 2 行的下面编写如下代码，对商品相册数据进行处理。

【代码 9-92】 content\item.html

```
1 Common.itemAlbumPath(data.album);
2 $('.album').html(template('album_tpl', {item: data.album}));
```

上述代码中，第 1 行调用了 Common.itemAlbumPath()方法，用于对后端 API 返回的图片路径进行处理。在【代码 9-68】第 9 行的下面编写代码该方法，具体代码如下。

【代码 9-93】 js\common.js

```
1 itemAlbumPath: function(data) {
2   for (var i in data) {
3     data[i] = Config.uploadURL + data[i].replace('[prefix]',
4     'album_small_');
5   }
6 },
```

在上述代码中，第 3~4 行用于将图片路径中的“[prefix]”替换成“album_small_”，表示获取相册中的小尺寸的图片。

在【代码 9-88】第 3 行的下面编写代码，创建商品相册的模板，具体代码如下。

【代码 9-94】 content\item.html

```
1 <script id="album_tpl" type="text/html">
2   <div class="album-mid">
3     <img class="album-mid-img" alt="预览图">
4     <div class="album-mask"></div>
5   </div>
6   <div class="album-small">
7     <div class="album-btn-left"></div>
8     <div class="album-list">
9       <ul>
10      {{each item v}}
11        <li><img src="{{v}}" alt="预览图"></li>
12      {{/each}}
13      </ul>
14    </div>
15    <div class="album-btn-right"></div>
16  </div>
17  <div class="album-big"><img class="album-big-img" alt="预览图"></div>
18 </script>
19 <script src="js/jquery.album.js"></script>
```

在上述代码中，第 10~12 行用于输出商品相册中底部的小图列表。当鼠标单击小图时，会使用第 3 行中的<img>元素来显示对应的中图。当鼠标滑过中图时，通过第 17 行代码中的<img>元素来显示对应的大图。

第 19 行引入了 js/jquery.album.js 插件，为了使该插件生效，在【代码 9-92】第 2 行的下面编写如下代码。

【代码 9-95】 content\item.html

```
1  // 调用 album()方法使插件生效
2  $('.album').album();
```

上述代码执行后，即可实现商品相册的交互效果。在该插件中，会自动根据小图路径替换成中图或大图路径。

6. 相关商品推荐

在【代码 9-92】第 2 行代码的下面编写代码，将数据传递给模板，具体代码如下。

【代码 9-96】 content\item.html

```
1  Common.itemPicPath(data.recommend);
2  $('.item-ads-content').html(template('item_ads_tpl',{item:
3  data.recommend}));
```

将【代码 9-88】第 9 行代码修改成如下代码，创建相关商品推荐模板。

【代码 9-97】 content\item.html

```
1  <div class="item-ads">
2    <div class="item-ads-title">相关商品推荐</div>
3    <div class="item-ads-content"></div>
4    <script id="item_ads_tpl" type="text/html">
5      {{each item v}}
6      <ul class="item-ads-item">
7        <li>
8          <a href="item-{{v.id}}.html" target="_blank">
9            <img src="{{v.pic}}" alt="{{v.title}}">
10         </a>
11       </li>
12       <li class="item-ads-name"><a href="{{v.pic}}" target="_blank">
13       {{v.title}}</a></li>
14       <li class="item-ads-price">￥{{v.price}}</li>
15     </ul>
16     {{/each}}
17   </script>
18 </div>
```

完成上述代码后，即可在商品详情页中显示“相关商品推荐”。

【任务 9】购物车

任务描述

当用户在商品详情页单击“加入购物车”按钮后，就会将当前商品加入到购物车，然后跳转到购物车页面，显示购物车中的商品。完成后的购物车页面效果如图 9-22 所示。

图 9-22　购物车

从图 9-22 中可以看出，购物车页面是一个表格布局。在表格中，第 1 列是一个复选框，表示提交订单时是否提交该商品，提供了全选功能；第 2 列是商品名称，通过单击商品名称可以跳转到商品详情页；第 3 列是商品的单价，第 4 列是购买数量，第 5 列用于从购物车中删除商品。在表格的右下方，显示了当前选择的商品数量和总价格。

接口分析

在前台 API 中，并没有为购物车专门提供一个接口，而是通过商品接口进行交互。这是因为本项目将用户添加到购物车的商品通过 localStorage 保存在了本地。考虑到用户将商品添加到购物车之后，网站后台可能会修改商品信息，因此只在 localStorage 中保存了商品 id 和购买数量，没有保存商品名称、价格等其他信息。在购物车页面中，需要将用户添加到购物车中的商品信息显示出来，因此需要同商品接口来查询商品信息。

下面通过表 9-25 介绍如何通过商品接口查询购物车页面所需的信息。

表9-25　查询购物车

URL	http://api.shop.localhost/v1/items	
基本信息	请求方式	-
	GET	-
URL 参数	参数名	说明
	view	前端组件名（值为 shopcart 表示购物车）
	ids	商品 id 字符串（多个用“,”分隔）

从表 9-25 中可以看出，查询商品列表与查询购物车的区别在于 view 参数。当 view 参数被省略，或值为 list 时，表示查询商品列表；当值为 shopcart 时，表示查询购物车中的商品信息。由于购物车中的商品有多个，因此通过 ids 参数来传入商品 id。

代码实现

1. 购物车数据管理

本项目使用 localStorage 对添加到购物车中的商品 id 和购买数量进行保存。在【代码 9-93】第 6 行的下面编写代码，具体代码如下。

【代码 9-98】 js\common.js

```
1  // 将商品添加到购物车（id 表示商品 id，num 表示购买数量）
2  addCart: function(id, num) {
3    id = parseInt(id);
4    num = parseInt(num);
5    var data = this.getCart();
6    var found = false;
7    for (var i in data) {
8      if (data[i].id === id) {
9        found = i;
10       break;
11     }
12   }
13   if (found === false) {
14     data.unshift({id: id, num: num});
15   } else {
16     data[found].num += num;
17   }
18   localStorage.setItem('shop_front_cart', JSON.stringify(data));
19 },
20 // 获取购物车中的商品
21 getCart: function() {
22   var data = JSON.parse(localStorage.getItem('shop_front_cart'));
23   return data === null ? [] : data;
24 },
```

在上述代码中，第 18 行用于将需要保存到购物车的数据通过 JSON.stringify()方法转换成字符串后保存到本地存储中，第 22 行用于从本地存储中取出字符串，然后通过 JSON.parse()方法将字符串恢复成原来的数据。

在保存到购物车前，第 7~12 行用于判断当前需保存的商品 id 在 data 中是否已经存在，如果存在，则执行第 16 行代码，增加商品数量，如果不存在，则执行第 14 行代码，将商品 id 和购买数量添加到 data 中。

2. 添加商品到购物车

在【代码 9-91】第 28 行的下面编写如下代码，为“加入购物车”按钮添加单击事件。

【代码 9-99】 content\item.html

```
$('#cart_add').click(function() {
  // 将 商品 id 和 购买数量 添加到本地存储（如果存在，则增加数量）
  Common.addCart(id, num.val());
  alert('商品已成功加入购物车！');
  location.href = 'cart.html';
  return false;
});
```

3. 查看购物车中的商品

创建购物车页面 content\cart.html 文件，先将购物车中的商品 id 取出，然后向后端 API 请求商品名称、价格、库存等数据，具体代码如下。

【代码 9-100】 content\cart.html

```
<script>
  (function() {
    var cart = Common.getCart();
    var ids = [];
    for (var i in cart) {
      ids.push(cart[i].id);
    }
    $.get(Config.api + 'carts', {ids: ids.join(',')}, function(data) {
      $('.shopcart').html(template('shopcart_tpl',
      {item: data, cart: cart}));
      loadSuccess();
    });
  })();
</script>
```

在上述代码中，第 3~7 行用于将购物车中已保存的数据取出，然后将其中的商品 id 保存为 ids 数组。第 8 行将 ids 数组转换成用“,”分隔的字符串，发送给后端 API。第 11 行调用了 loadSuccess()函数，该函数将在后面的步骤中实现。

在【代码 9-100】第 1 行的上面编写代码，实现购物车页面的模板，具体代码如下。

【代码 9-101】 content\cart.html

```
<div class="title">我的购物车</div>
<table class="shopcart"></table>
<script id="shopcart_tpl" type="text/html">
  <tr>
    <th><a href="#" class="shopcart-checkall">全选</a></th>
    <th>商品名称</th><th>单价(元)</th><th>数量</th><th>操作</th>
  </tr>
  {{each cart v1}}
  {{each item v}}
  {{if v1.id === v2.id}}
  <tr class="shopcart-item">
    <td><input type="checkbox" class="shopcart-check"></td>
```

```
13    <td><a href="item-{{v2.id}}.html" target="_blank" class="bold">
14    {{v2.title}}</a></td>
15    <td><span class="shopcart-price">{{v2.price}}</span></td>
16    <td>
17      <button class="shopcart-num-sub">-</button>
18      <input class="shopcart-num" type="text" value="{{v1.num}}">
19      <button class="shopcart-num-add">+</button>
20      <input class="shopcart-id" type="hidden" value="{{v2.id}}">
21      <input class="shopcart-stock" type="hidden" value="{{v2.num}}">
22    </td>
23    <td><a href="#" class="cart-del">删除</a></td>
24  </tr>
25  {{/if}}
26  {{/each}}
27  {{/each}}
28  <tr>
29    <td><a href="#" class="cart-checkall">全选</a></td>
30    <td colspan="4">
31      共<span class="shopcart-sum"></span>件商品
32      总计：<span class="shopcart-total">￥<span></span></span>
33      <input type="submit" value="提交订单">
34    </td>
35  </tr>
36 </script>
```

在上述代码中，第 10 行用于对 id 进行判断，这是因为后端 API 返回的商品顺序与购物车中保存的商品的顺序不一定相同，且有些商品可能已经下架，通过判断可以使其符合购物车中的顺序，并将已经下架的商品剔除。第 20 行和第 21 行将商品 id 和库存数量放在隐藏域中，用于在控制购买数量时读取这些信息。

4. 全选和计算总价

在【代码 9-100】第 12 行的下面编写 loadSuccess()函数，具体代码如下。

【代码 9-102】 content\cart.html

```
1  function loadSuccess() {
2    var check = $('.shopcart-check');
3    var checked = false;
4    check.click(total);                          // 复选框被单击时更新统计
5    $('.shopcart-checkall').click(checkAll); // 全选 或 全不选
6    checkAll();                                  // 默认为全选状态
7    // 根据 checked 的值全选或全不选
8    function checkAll() {
9      checked = !checked;
10     check.prop('checked', checked);
11     check.attr('checked', checked);
12     total();
13   }
```

```
14  // 统计商品数量和总价格
15  function total() {
16    var sum = 0;      // 总数量
17    var total = 0;    // 总价格
18    $('.shopcart-check:checked').each(function() {
19      var item = $(this).parents('.shopcart-item');
20      var price = parseFloat(item.find('.shopcart-price').text());
21      var num = parseInt(item.find('.shopcart-num').val());
22      sum += num;
23      total += price * num;
24    });
25    $('.shopcart-total span').text(total.toFixed(2));
26    $('.shopcart-sum').text(sum);
27  }
28 }
```

在上述代码中，checkAll()函数用于执行全选或全不选操作，total()函数用于统计商品数量和总价格。第 18 行在进行统计时，遍历了购物车中已选中的商品，未选中的商品表示不需要购买，因此不进行统计。第 4 行和第 12 行用于当商品复选框的勾选状态发生变化时，调用 total()函数重新进行统计。

5. 调整购买数量

为了实现调整购买数量的功能，在【代码 9-102】第 27 行下面编写代码，为控制购买数量的按钮和文本框添加事件。具体代码如下。

【代码 9-103】 content\cart.html

```
1  // 减少购买数量
2  $('.shopcart-num-sub').click(function() {
3    var id =  $(this).parent().find('.shopcart-id');
4    var num = $(this).parent().find('.shopcart-num');
5    var n = parseInt(num.val());
6    if (n <= 1) {
7      return false;
8    }
9    num.val(n - 1);
10   Common.editCart(id.val(), n - 1);
11   total();
12 });
13 // 增加购买数量
14 $('.shopcart-num-add').click(function() {
15   var id =  $(this).parent().find('.shopcart-id');
16   var num = $(this).parent().find('.shopcart-num');
17   var stock = $(this).parent().find('.shopcart-stock');
18   var n = parseInt(num.val());
19   if (n >= parseInt(stock.val())) {
20     return false;
21   }
```

```
22   num.val(n + 1);
23   Common.editCart(id.val(), n + 1);
24   total();
25 });
26 // 自动纠正购买数量
27 $('.shopcart-num').keyup(function() {
28   var stock = $(this).parent().find('.shopcart-stock').val();
29   var n = parseInt($(this).val());
30   if (n < 1 || n !== n) {
31     $(this).val(1);
32   } else if (n > stock) {
33     $(this).val(stock);
34   }
35   total();
36 });
37 // 保存数量变更结果
38 $('.shopcart-num').blur(function() {
39   var id =  $(this).parent().find('.shopcart-id').val();
40   var n = $(this).val();
41   Common.editCart(id, n);
42   total();
43 });
```

上述代码与商品详情页中的控制购买数量非常相似，不同之处在于，这里需要将购买数量的更改结果保存到 localStorage 中，因此调用了 Common.editCart()方法。接下来需要在 Common 对象中增加该方法，在【代码 9-98】第 24 行的下面编写如下代码实现。

【代码 9-104】 js\common.js

```
1  editCart: function(id, num) {
2    id = parseInt(id);
3    num = parseInt(num);
4    var data = this.getCart();
5    for (var i in data) {
6      if (data[i].id === id) {
7        data[i].num = num;
8        if (num <= 0) {
9          data.splice(i, 1);
10       }
11       break;
12     }
13   }
14   localStorage.setItem('shop_front_cart', JSON.stringify(data));
15 }
```

在上述代码中，第 7 行用于更新商品数量，如果商品数量为 0，则执行第 9 行代码，从购物车中删除商品。

6. 从购物车中删除商品

在【代码 9-103】第 42 行的下面编写如下代码，为“删除”链接添加单击事件。

【代码 9-105】 content\cart.html

```
1  $('.shopcart-del').click(function() {
2    var id = $(this).parents().find('.shopcart-id').val();
3    $(this).parents('.shopcart-item').remove();
4    Common.editCart(id, 0);
5    total();
6  });
```

在上述代码中，第 3 行用于从购物车页面中删除商品，第 4 行用于从 localStorage 中删除保存的购物车商品。至此，购物车功能全部开发完成。