第7章 登录与认证

学习目标

- 能够使用BCrypt进行密码加密
- 完成 Spring Security入门案例
- 完成青橙登录认证
- 完成青橙菜单展示
- 完成用户登录日志

1. BCrypt密码加密

1.1 BCrypt简介

在用户模块，对于用户密码的保护，通常都会进行加密。我们通常对密码进行加密，然后存放在数据库中，在用户进行登录的时候，将其输入的密码进行加密然后与数据库中存放的密文进行比较，以验证用户密码是否正确。 目前，MD5和BCrypt比较流行。相对来说，BCrypt比MD5更安全。

BCrypt 官网http://www.mindrot.org/projects/jBCrypt/

1.2 快速入门

（1）我们从官网下载源码

（2）新建工程，将源码类BCrypt拷贝到工程

（3）新建测试类，main方法中编写代码，实现对密码的加密

```
String gensalt = BCrypt.gensalt();//这个是盐  29个字符，随机生成
System.out.println(gensalt);
String password = BCrypt.hashpw("123456", gensalt);  //根据盐对密码进行加密
System.out.println(password);//加密后的字符串前29位就是盐
```

（4）新建测试类，main方法中编写代码，实现对密码的校验。BCrypt不支持反运算，只支持密码校验。

```
boolean checkpw = BCrypt.checkpw("123456",
"$2a$10$61ogZY7EXsMDWeVGQpDq3OBF1.phaUu7.xrwLyWFTOu8woE08zMIW");
System.out.println(checkpw);
```

2. 安全框架Spring Security

2.1 Spring Security简介

2.1.1 安全框架概述

什么是安全框架？ 解决系统安全问题的框架。如果没有安全框架，我们需要手动处理每个资源的访问控制，非常麻烦。使用安全框架，我们可以通过配置的方式实现对资源的访问限制。

2.1.2 常用安全框架

Spring Security：spring家族一员。是一个能够为基于Spring的企业应用系统提供声明式的安全访问控制解决方案的安全框架。它提供了一组可以在Spring应用上下文中配置的Bean，充分利用了Spring IoC，DI（控制反转Inversion of Control ,DI:Dependency Injection 依赖注入）和AOP（面向切面编程）功能，为应用系统提供声明式的安全访问控制功能，减少了为企业系统安全控制编写大量重复代码的工作。

Apache Shiro 是一个功能强大且易于使用的Java安全框架,提供了认证,授权,加密,和会话管理。

我们课程中采用Spring Security。

2.1.3 认证与授权

认证：限制用户只能登陆才可以访问资源。

授权：限制用户必须有某资源的访问权限才可以访问。

2.2 快速入门

2.2.1 最简单案例

需求：实现简单的登陆，当用户没有登陆访问主页执行拦截跳转到登陆，登陆后跳转到主页。实现退出登陆的功能，退出后再次访问主页仍然拦截。用户名和密码不连接数据库，直接在配置文件中配置。

（1）新建war工程，pom文件引入依赖

```
    <packaging>war</packaging>
    <properties>
        <spring.version>5.0.5.RELEASE</spring.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-web</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework.security</groupId>
            <artifactId>spring-security-config</artifactId>
            <version>${spring.version}</version>
        </dependency>
        <dependency>
            <groupId>javax.servlet</groupId>
            <artifactId>servlet-api</artifactId>
            <version>2.5</version>
            <scope>provided</scope>
        </dependency>
    </dependencies>
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.tomcat.maven</groupId>
                <artifactId>tomcat7-maven-plugin</artifactId>
                <configuration>
                    <!-- 指定端口 -->
                    <port>9090</port>
                    <!-- 请求路径 -->
                    <path>/</path>
                </configuration>
            </plugin>
        </plugins>
    </build>
```

（2）创建webapp/WEB-INF/web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
    version="2.5">
     <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-security.xml</param-value>
     </context-param>
     <listener>
        <listener-class>
            org.springframework.web.context.ContextLoaderListener
        </listener-class>
     </listener>
     <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
     </filter>
     <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
     </filter-mapping>
</web-app>
```

（3）resources下创建spring-security.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans:beans xmlns="http://www.springframework.org/schema/security"
    xmlns:beans="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd
                        http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd">
    <!-- 页面拦截规则 -->
    <http>
        <intercept-url pattern="/**" access="hasRole('ROLE_ADMIN')" />
        <form-login/>
        <logout/>
    </http>
    <!-- 认证管理器 -->
    <authentication-manager>
        <authentication-provider>
            <user-service>
                <user name="admin" password="{noop}123456"
authorities="ROLE_ADMIN"/>
            </user-service>
        </authentication-provider>
    </authentication-manager>
</beans:beans>
```

{noop}是制定密码加密策略为不加密 。noop的意思是明文保存的密码 (noop: No Operation)

（4）webapp下创建index.html，内容随意。

（5）启动工程，打开浏览器输入地址 http://localhost:9090 ，浏览器显示

这个登陆页面时SpringSecurity帮我们自动生成的。

输入正确的密码，进入首页，输入错误的密码显示如下信息

配置说明：

intercept-url 表示拦截页面

/* 表示的是该目录下的资源，只包括本级目录不包括下级目录

/** 表示的是该目录以及该目录下所有级别子目录的资源

form-login 为开启表单登陆

2.2.2 密码加密策略

修改配置文件中的password为bcrypt加密后的密码，并制定加密策略为bcrypt

```
<user-service>
    <user name="admin"
       password="
{bcrypt}$2a$10$61ogZY7EXsMDWeVGQpDq3OBF1.phaUu7.xrwLyWFTOu8woE08zMIW"
authorities="ROLE_ADMIN"/>
</user-service>
```

spring security官方推荐使用更加安全的bcrypt加密方式。spring security 5支持的加密方式有bcrypt、ldap、MD4、MD5、noop、pbkdf2、scrypt、SHA-1、SHA-256、sha256。

我们还有另外一种配置方式，来制定加密策略

```
    <!--认证管理器-->
    <authentication-manager>
         <authentication-provider>
             <user-service>
                 <user name="admin"
password="$2a$10$EPtdfwSJ0ABj5JsCyLqhFe1g503DgA4lQvOxyZF/3usoyje5/q/Dy"
authorities="ROLE_ADMIN"></user>
             </user-service>
             <password-encoder ref="bcryptEncoder"></password-encoder>
         </authentication-provider>
    </authentication-manager>
<beans:bean id="bcryptEncoder"
class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"
/>
```

2.2.3 自定义登录页

（1）创建页面login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>登录</title>
</head>
<body>
<form action="/login" method="post">
    <table>
        <tr>
            <td>用户名</td>
            <td><input name="username"> </td>
        </tr>
        <tr>
            <td>密码</td>
            <td><input type="password" name="password"> </td>
        </tr>
    </table>
    <button>登录</button>
</form>
</body>
</html>
```

（2）创建login_error.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>登录错误</title>
</head>
<body>
  用户名和密码错误!
</body>
</html>
```

（3）修改spring-security.xml拦截规则部分

```
    <!-- 以下页面不被拦截 -->
    <http pattern="/login.html" security="none"></http>
    <http pattern="/login_error.html" security="none"></http>
    <!-- 页面拦截规则 -->
    <http>
        <intercept-url pattern="/**" access="hasRole('ROLE_ADMIN')" />
        <form-login login-page="/login.html" default-target-
url="/index.html" authentication-failure-url="/login_error.html"/>
        <csrf disabled="true"/>
    </http>
```

（4）测试，浏览器显示了我们自定义的登录页面

`<csrf disabled="true"/>` 为关闭跨域请求伪造控制。因为静态页无法动态生成token，所以将此功能关闭。一般静态页采用图形验证码的方式实现防止跨域请求伪造的功能。

2.2.4 UserDetailsService

我们刚才的例子是将用户名和密码配置在配置文件中，实际的企业级开发更多的是从数据库中提取用户名和密码信息，如何做到呢？我们这里学习UserDetailsService的使用。

（1）创建UserDetailsServiceImpl

```
public class UserDetailsServiceImpl implements UserDetailsService {
    public UserDetails loadUserByUsername(String s) throws
UsernameNotFoundException {
        System.out.println("经过UserDetailsServiceImpl");
        //构建角色集合 ，项目中此处应该是根据用户名查询用户的角色列表
        List<GrantedAuthority> grantedAuths = new
ArrayList<GrantedAuthority>();
        grantedAuths.add(new SimpleGrantedAuthority("ROLE_ADMIN"));
        return new
User(s,"$2a$10$61ogZY7EXsMDWeVGQpDq3OBF1.phaUu7.xrwLyWFTOu8woE08zMIW",
grantedAuths);
    }
}
```

（2）修改配置文件 spring-security.xml认证管理器部分

```
    <!-- 认证管理器 -->
    <authentication-manager>
        <authentication-provider user-service-ref="userDetailsService">
            <password-encoder ref="bcryptEncoder"></password-encoder>
        </authentication-provider>
    </authentication-manager>
    <beans:bean id="userDetailsService"
class="com.itheima.demo.UserDetailsServiceImpl"></beans:bean>
    <beans:bean id="bcryptEncoder"
class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"
/>
```

3. 青橙登录认证

3.1 青橙登录页

3.1.1 需求分析

实现管理后台的登录功能。登录页采用前端提供的页面。

3.1.2 代码实现

qingcheng_common_web已经添加了springsecurity依赖

（1）添加登录页面（资源提供）。修改login.html ,指定表单提交地址为/login ,用户名和密码框的name 为username 和password

（2）添加图标文件到webapp 。 资料\静态原型\网站前台\qingcheng\img\favicon.ico

（3）qingcheng_web_manager工程web.xml添加配置

```
  <filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-
class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>

  </filter-mapping>
```

注意：此配置必须加在**springmvc**之前

（4）qingcheng_web_manager工程添加配置文件applicationContext_security.xml，内容参照入门案例spring-security.xml ,添加配置

```
    <http pattern="/css/**" security="none"></http>
    <http pattern="/img/**" security="none"></http>
    <http pattern="/js/**" security="none"></http>
```

修改目标页地址为main.html

```
<form-login login-page="/login.html" default-target-url="/main.html"
authentication-failure-url="/login.html" />
```

（5）qingcheng_web_manager工程添加UserDetailsServiceImpl，同入门案例。

3.2 访问数据库实现用户认证

3.2.1 表结构分析

系统数据库qingcheng_system .tb_admin 表（管理员表）

字段名称	字段含义	字段类型	字段长度	备注
id	id	INT		
login_name	用户名	VARCHAR		
password	密码	VARCHAR		
status	状态	CHAR		

3.2.2 代码实现

修改UserDetailsServiceImpl

```
public class UserDetailsServiceImpl implements UserDetailsService {

    @Reference
    private AdminService adminService;

    public UserDetails loadUserByUsername(String s) throws
UsernameNotFoundException {
        //查询管理员
        Map map= new HashMap();
        map.put("loginName",s);
        map.put("status","1");
        List<Admin> list = adminService.findList(map);
        if(list.size()==0){
            return null;
        }
        //构建角色集合 ，项目中此处应该是根据用户名查询用户的角色列表
        List<GrantedAuthority> grantedAuths = new
ArrayList<GrantedAuthority>();
        grantedAuths.add(new SimpleGrantedAuthority("ROLE_ADMIN"));
        return new User(s,list.get(0).getPassword(), grantedAuths);
    }
}
```

4. 青橙系统菜单展示

4.1 需求分析

从数据库中读取菜单数据并展示。菜单为三级菜单，树形结构如下:

```
{
  "data": [
    {
        "path": "1",//菜单项所对应的路由路径
        "title": "首页", //菜单项名称
        "icon":"iconHome",//是否有子菜单，若没有，则为[]
    },
    {
      "path": "2",
      "title": "商品",//一级菜单
      "icon":"iconCommodity",
      "children": [
        {
            "path": "2-1",
            "title": "商品管理",//二级菜单
            "linkUrl":"",
            "icon":"iconSp",
            "children":[
                {
                    "path": "2-1-1",
                    "title": "商品列表",
                    "linkUrl":"all-medical-list.html",//三级菜单
                },
                {
                    "path": "2-1-2",
                    "title": "添加商品",
                    "linkUrl":"commodity-add.html",
                }
            ]
        },
        {
            "path": "2-2",
            "title": "添加配置",
            "linkUrl":"",
            "icon":"iconSet",
            "children":[
                {
                    "path": "2-2-1",
                    "title": "商品分类",

                    "linkUrl":"all-medical-list.html",
```

```
                },
                {
                    "path": "2-2-2",
                    "title": "规格参数",
                    "linkUrl":"all-medical-list.html",
                }
            ]
        }
      ]
    }
  ]
}
```

4.2 表结构分析

tb_menu （菜单表）

字段名称	字段含义	字段类型	字段长度	备注
id	菜单ID	VARCHAR		
name	菜单名称	VARCHAR		
icon	图标	VARCHAR		
url	URL	VARCHAR		
parent_id	上级菜单ID	VARCHAR		

4.3 代码实现

4.3.1 后端代码

（1）MenuService接口新增方法定义，用于返回全部菜单

```
public List<Map> findAllMenu();
```

（2）MenuServiceImpl实现此方法

```
    /**
     * 查询全部菜单
     * @return
     */
    public List<Map> findAllMenu() {
        List<Menu> menuList = findAll();//查询全部菜单列表
        return findMenuListByParentId(menuList,"0");//一级菜单列表
    }

    /**
     * 查询下级菜单ID
     * @param menuList
     * @param parentId
     * @return
     */
    private List<Map> findMenuListByParentId(List<Menu> menuList,String
parentId){
        List<Map> mapList=new ArrayList<>();
        for(Menu menu:menuList){  //循环一级菜单
            if(menu.getParentId().equals(parentId)){
                Map map=new HashMap();
                map.put("path",menu.getId());
                map.put("title",menu.getName());
                map.put("icon",menu.getIcon());
                map.put("linkUrl",menu.getUrl());

map.put("children",findMenuListByParentId(menuList,menu.getId()));
                mapList.add(map);
            }
        }
        return mapList;
    }
```

（3）qingcheng_web_manager工程MenuController新增方法

```
    @GetMapping("/findMenu")
    public List<Map> findMenu(){
        return menuService.findAllMenu();
    }
```

4.3.2 前端代码

修改页面的JS代码

```
created() {
    //.......

    axios.get("/menu/findMenu.do").then( response=>{
        this.menuList=response.data;
        this.data=response.data[0]
        var data=[]
        for(var i=0;i<response.data[0].children.length;i++){
            data.push(response.data[0].children[i].path)
        }
        this.openeds=data
    })
}
```

4.3.3 同源策略设置

由于我们的main.html是框架页，需要修改同源策略。

```
        <headers>
            <frame-options policy="SAMEORIGIN"></frame-options>
        </headers>
```

response header 可用于指示是否应该允许浏览器呈现在一个页面（同源策略）

同源策略限制了从同一个源加载的文档或脚本如何与来自另一个源的资源进行交互。这是一个用于隔离潜在恶意文件的重要安全机制。

解释一下同源。如果两个url，协议、地址和端口都相同，我们称两个url为同源。

Spring Security下，X-Frame-Options默认为DENY. 如果不修改同源策略，框架页内将无法显示内容。

DENY：浏览器拒绝当前页面加载任何Frame页面
SAMEORIGIN：frame页面的地址只能为同源域名下的页面
ALLOW-FROM：origin为允许frame加载的页面地址。

4.3.4 获取当前登录人

需求：主界面显示当前登陆人

实现思路：后端编写controller输出当前登录人，前端异步调用。

（1）后端代码实现：

```
@RestController
@RequestMapping("/login")
public class LoginController {

    @GetMapping("/name")
    public Map showName(){
        String name =
SecurityContextHolder.getContext().getAuthentication().getName();
        Map map=new HashMap();
        map.put("name",name);
        return map;
    }

}
```

（2）前端代码实现：

在main.html中添加属性loginName，用于存储当前的登陆人

created()方法添加代码

```
//加载并显示当前登录名
axios.get('/login/name.do').then( response=>{
    this.loginName= response.data.name;
})
```

页面显示登陆人

```
{{loginName}}
```

4.3.5 退出登录

当我们在spring security配置文件中配置了`<logout/>`后，框架会为我们自动提供退出功能，地址为/logout，要求以post方式提交。

main.html新增方法

```
exit(){
    axios.post('/logout').then(response=>{
        location.href="login.html";
    })
}
```

退出菜单调用：

```
<span style="display:block;" @click="exit()">退出</span>
```

5. 管理员登录日志

5.1 需求分析

管理员登录后，记录管理员名称、登录时间、ip、浏览器类型、所在地区等信息 。

5.2 表结构分析

tb_login_log 表

字段名称	字段含义	字段类型	字段长度	备注
id	id	INT		
login_name	登录名	VARCHAR		
ip	ip	VARCHAR		
browser_name	浏览器类型	VARCHAR		
location	地区	VARCHAR		
login_time	登录时间	DATETIME		

5.3 代码实现

5.3.1 登录成功处理器

（1）spring security为我们提供了一个叫“登录成功处理器”的组件，我们可以实现在登录后进行的后续处理逻辑。

```
public class AuthenticationSuccessHandlerImpl implements
AuthenticationSuccessHandler {

    @Override
    public void onAuthenticationSuccess(HttpServletRequest request,
HttpServletResponse response, Authentication authentication) throws
IOException, ServletException {
        System.out.println("登录成功处理器到此一游");
        request.getRequestDispatcher("/main.html").forward(request,
response);
    }
}
```

（2）applicationContext_security.xml新增配置

```
<beans:bean id="loginHandler"
class="com.qingcheng.controller.AuthenticationSuccessHandlerImpl">
</beans:bean>
```

```
<!--设置登陆成功处理器-->
<form-login login-page="/login.html"
                    default-target-url="/main.html"
                    authentication-failure-url="/login.html"
                    authentication-success-handler-ref="loginHandler"/>
```

5.3.2 登录日志处理

修改qingcheng_web_manager的AuthenticationSuccessHandlerImpl

```
public class AuthenticationSuccessHandlerImpl implements
AuthenticationSuccessHandler {

    @Reference
    private LoginLogService loginLogService;

    @Override
    public void onAuthenticationSuccess(HttpServletRequest
httpServletRequest, HttpServletResponse httpServletResponse,
Authentication authentication) throws IOException, ServletException {
        //登录后会调用
        System.out.println("登录成功了，我要在这里记录日志");
        String loginName = authentication.getName();//当前登录用户
        String ip = httpServletRequest.getRemoteAddr();  //获取访问的ip
        LoginLog loginLog=new LoginLog();
        loginLog.setLoginName(loginName);
        loginLog.setLoginTime(new Date()); // 当前登录时间
        loginLog.setIp(ip);
        loginLogService.add(loginLog);
httpServletRequest.getRequestDispatcher("/main.html").forward(httpServlet
Request,httpServletResponse);

    }
}
```

5.3.3 根据IP获取城市信息

（1）将资源中提供的工具类WebUtil复制到qingcheng_common_web工程

（2）WebUtil类的getCityByIP用于根据IP地址获取城市信息，修改代码

```
loginLog.setLocation(WebUtil.getCityByIP(ip));//保存城市信息
```

5.3.4 获取浏览器名称

```
String agent = httpServletRequest.getHeader("user-agent");
```

获取到的信息如下：

> Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/71.0.3578.98 Safari/537.36.....

我们没有必要存这么长的字符串，我们只需要存储一个浏览器名称就可以了，所以我们用到了WebUtil给我们提供的方法 getBrowserName(String agent)

5.3.5 我的登录日志列表

需求：查询我的登录日志列表

实现思路：后端获取当前登录人账号作为查询条件。前端精简生成的增删改查代码。

```
    /**
     * 查询当前登录人的登录日志
     * @param page
     * @param size
     * @return
     */
    @GetMapping("/findPageByLogin")
    public PageResult<LoginLog> findPageByLogin(int page, int size){
        //添加条件
        String loginName =
SecurityContextHolder.getContext().getAuthentication().getName();
        Map map=new HashMap();
        map.put("loginName",loginName);
        return loginLogService.findPage(map,page,size);
    }
```

6. 修改密码（作业）

6.1 需求分析

需求：界面输入原密码，新密码，确认密码。 新密码和确认密码必须一致，在前端校验。提交给后端后，后端判断原密码是否正确，如果正确，修改密码，注意密码要使用BCrypt加密。

6.2 思路提示

这里我们会使用两个知识点：

（1）获取当前登陆人。

（2）BCrypt校验密码与加密。