Spring Security

第一章 了解 spring security

spring security 是基于 spring 的安全框架。它提供全面的安全性解决方案，同时在 Web 请求级和方法调用级处理身份确认和授权。在 Spring Framework 基础上，spring security 充分利用了依赖注入（DI）和面向切面编程（AOP）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。是一个轻量级的安全框架。它与 Spring MVC 有很好地集成.

1.1 spring security 核心功能

（1）认证（你是谁，用户/设备/系统）

（2）验证（你能干什么，也叫权限控制/授权，允许执行的操作）

1.2 spring security 原理

基于 Filter , Servlet, AOP 实现身份认证和权限验证

第二章 实例驱动学习

使用的框架和技术

spring boot 2.0.6 版本

spring security 5.0.9 版本

maven 3 以上

jdk8 以上

idea 2019

第一个例子：初探

1.创建 maven 项目

2.加入依赖：spring boot 依赖， spring security 依赖

```
<!--加入 spring boot -->
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-parent</artifactId>
    <version>2.0.6.RELEASE</version>
</parent>
<!--web 开发相关依赖-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!--spring security-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

3.创建应用启动类

```
@SpringBootApplication
public class FirstApplication {
    public static void main(String[] args) {
        SpringApplication.run(FirstApplication.class,args);
    }
```

```
}
```

4.创建 Controller，接收请求

```
@RestController
@RequestMapping("/hello")
public class HelloSecurityController {
    @RequestMapping("/world")
     public String sayHello(){
          return "Hello Spring Secuirty  安全管理框架";
     }
}
```

5.框架生成的用户

用户名： user

密码： 在启动项目时，生成的临时密码。uuid

日志中生成的密码：

generated security password: 9717464c-fafd-47b3-9995-2c18b24f7336

6.自定义用户名和密码

需要在 springboot 配置文件中设置登录的用户名和密码

在 resource 目录下面创建 spring boot 配置文件

application.yml(application.properties)

```
spring:
  security:
```

```
    user:
      name: wkcto
      password: wkcto
```

name:自定义用户名称

password：自定义密码

7.关闭验证

//排除 Secuirty 的配置，让他不启用

```
@SpringBootApplication(exclude = {SecurityAutoConfiguration.class})
public class FirstApplication { }
```

第二个例子：使用内存中的用户信息

1）使用：WebSecurityConfigurerAdapter 控制安全管理的内容。

需要做的使用：继承 WebSecurityConfigurerAdapter，重写方法。实现自定义的认证信息。重写下面的方法。

```
protected void configure(AuthenticationManagerBuilder auth)
```

2）spring security 5 版本要求密码比较加密，否则报错

```
java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"
```

实现密码加密：

1）创建用来加密的实现类（选择一种加密的算法）

```
@Bean
public PasswordEncoder passwordEncoder(){
    //创建 PasawordEncoder 的实现类， 实现类是加密算法
    return new BCryptPasswordEncoder();
}
```

2）给每个密码加密

```
PasswordEncoder pe = passwordEncoder();
pe.encode("123456")
```

注解：

1. @Configuration ：表示当前类是一个配置类(相当于是 spring 的 xml 配置文件），在这个类方法的返回值是 java 对象，这些对象放入到 spring 容器中。

2. @EnableWebSecurity：表示启用 spring security 安全框架的功能

3. @Bean：把方法返回值的对象，放入到 spring 容器中。

第三个例子：基于角色 Role 的身份认证， 同一个用户可以有不同的角色。同时可以开启对方法级别的认证。

基于角色的实现步骤：

1.设置用户的角色

继承 WebSecurityConfigurerAdapter

重写 configure 方法。指定用户的 roles

```
auth.inMemoryAuthentication()
        .withUser("admin")
        .password(pe.encode("admin"))
        .roles("admin","normal");
```

2.在类的上面加入启用方法级别的注解

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
```

3.在处理器方法的上面加入角色的信息，

指定方法可以访问的角色列表

```
//指定 normal 和 admin 角色都可以访问的方法
@RequestMapping("/helloUser")
@PreAuthorize(value = "hasAnyRole('admin','normal')")
public String helloCommonUser(){
    return "Hello 拥有 normal, admin 角色的用户";
}
```

使用@PreAuthorize 指定在方法之前进行角色的认证。

```
hasAnyRole('角色名称 1','角色名称 N')
```

第四个例子，基于 jdbc 的用户认证。

从数据库 mysql 中获取用户的身份信息（用户名称，密码，角色）

1）在 spring security 框架对象用户信息的表示类是 UserDetails.

UserDetails 是一个接口，高度抽象的用户信息类（相当于项目中的 User 类）

User 类：是 UserDetails 接口的实现类， 构造方法有三个参数：

username，password, authorities

需要向 spring security 提供 User 对象， 这个对象的数据来自数据库的查询。

2）实现 UserDetailsService 接口，

重写方法 UserDetails loadUserByUsername(String var1)

在方法中获取数据库中的用户信息， 也就是执行数据库的查询，条件是用户名称。

细说 Spring Security 安全框架（二）

第三章 基于角色的权限

3.1 认证和授权

authentication：认证， 认证访问者是谁。 一个用户或者一个其他系统是不是当前要访问的系统中的有效用户。

authorization：授权， 访问者能做什么

比如说张三用户要访问一个公司 oa 系统。 首先系统要判断张三是不是公司中的有效用户。

认证张三是不是有效的用户，是不是公司的职员

授权： 判断张三能否做某些操作， 如果张三是个领导可以批准下级的请假， 其他的操作。

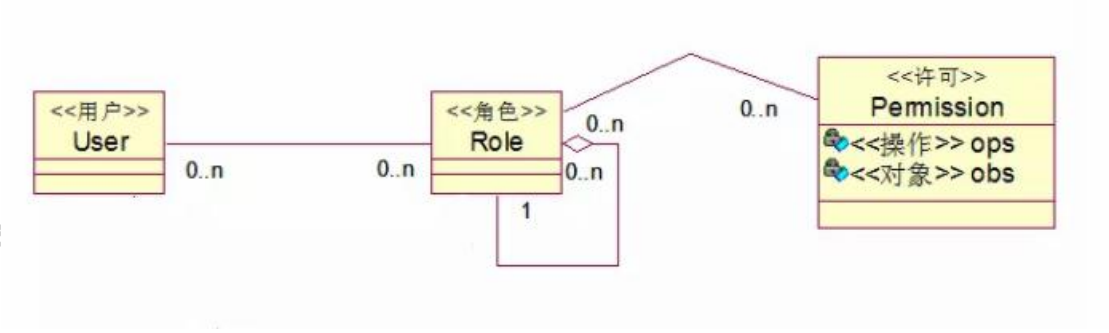
如果张三只是一个普通用户，只能看自己的相关数据， 只能提交请假申请等等。

3.2 RBAC 是什么？

RBAC 是基于角色的访问控制（Role-Based Access Control ）

在 RBAC 中，权限与角色相关联，用户通过成为适当角色的成员而得到这些角色的权限。这就极大地简化了权限的管理。这样管理都是层级相互依赖的，权限赋予给角色，而把角色又赋予用户，这样的权限设计很清楚，管理起来很方便。

其基本思想是，对系统操作的各种权限不是直接授予具体的用户，而是在用户集合与权限集合之间建立一个角色集合。每一种角色对应一组相应的权限。一旦用户被分配了适当的角色后，该用户就拥有此角色的所有操作权限。这样做的好处是，不必在每次创建用户时都进行分配权限的操作，只要分配用户相应的角色即可，而且角色的权限变更比用户的权限变更要少得多，这样将简化用户的权限管理，减少系统的开销。



RBAC： 用户是属于角色的， 角色拥有权限的集合。 用户属于某个角色， 他就具有角色对应的权限。

系统中有张三，李四，他们是普通员工，只能查看数据。

系统中经理，副经理他们能修改数据。

设计有权限的集合，角色: 经理角色，具有修改数据的权限，删除，查看等等。

普通用户角色： 只读角色，只能看数据 ，不能修改，删除。

让张三，李四是只读的，普通用户角色。 让经理，副经理他们都是经理角色。

公司以后增加新的普通员工，加入到“普通用户角色”就可以了，不需要在增加新的角色。

公司增加经理了， 只要加入到“经理角色”就可以了。

权限：能对资源的操作， 比如增加，修改，删除，查看等等。

角色：自定义的， 表示权限的集合。一个角色可以有多个权限。

RBAC 设计中的表：

1. 用户表： 用户认证（登录用到的表）

   用户名，密码，是否启用，是否锁定等信息。

2. 角色表：定义角色信息

角色名称， 角色的描述。

3.用户和角色的关系表： 用户和角色是多对多的关系。

一个用户可以有多个角色， 一个角色可以有多个用户。

4.权限表， 角色和权限的关系表

角色可以有哪些权限。

3.3 spring specurity 中认证的接口和类

1） UserDetails：接口，表示用户信息的。

boolean isAccountNonExpired(); 账号是否过期

boolean isAccountNonLocked();账号是否锁定

boolean isCredentialsNonExpired();证书是否过期

boolean isEnabled();账号是否启用

Collection<? extends GrantedAuthority> getAuthorities(); 权限集合

User 实现类

org.springframework.security.core.userdetails.User

可以：自定义类实现 UserDetails 接口，作为你的系统中的用户类。这个类可以交给 spring security 使用。

2） UserDetailsService 接口：

主要作用：获取用户信息，得到是 UserDetails 对象。一般项目中都需要自定义类实现这个接口，从数据库中获取数据。

一个方法需要实现：

UserDetails loadUserByUsername(String var1) ：根据用户名称，获取用户信息（用户名称，密码，角色结合，是否可用，是否锁定等信息）

UserDetailsService 接口的实现类：

1. InMemoryUserDetailsManager：在内存中维护用户信息。

   优点：使用方便。

   缺点：数据不是持久的。系统重启后数据恢复原样。

2.JdbcUserDetailsManager ：用户信息存放在数据库中，底层使用 jdbcTemplate 操作数据库。 可以 JdbcUserDetailsManager 中的方法完成用户的管理

createUser ： 创建用户

updateUser：更新用户

deleteUser：删除用户

userExists：判断用户是否存在

数据库文件：

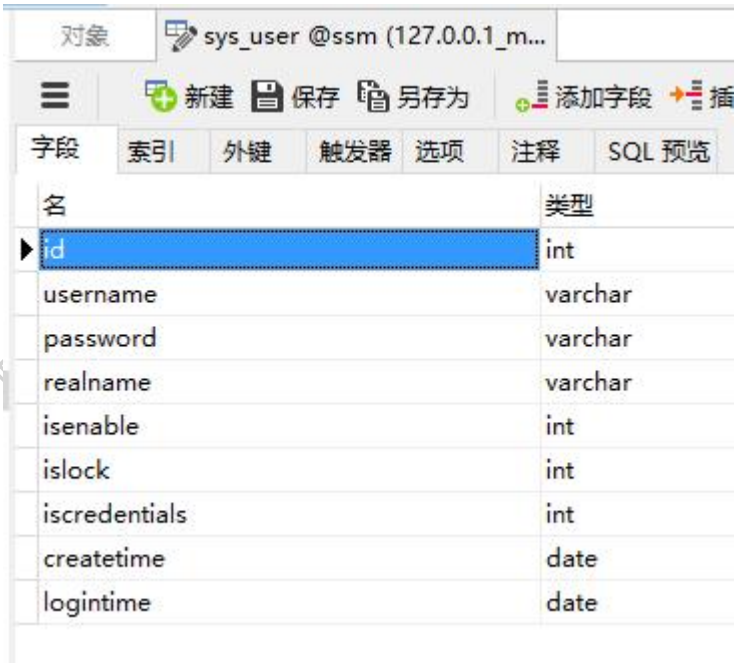
org.springframework.security.core.userdetails.jdbc

users.ddl 文件

```
create table users(username varchar_ignorecase(50) not null primary key,password varchar_ignorecase(500) not null,enabled boolean not null);
create table authorities (username varchar_ignorecase(50) not null,authority varchar_ignorecase(50) not null,constraint fk_authorities_users foreign key(username) references users(username));
create unique index ix_auth_username on authorities (username,authority);
```

3.4 定义用户，角色，角色关系表

用户信息表 sys_user：



sys_role：角色表

字段 | 索引 | 外键 | 触发器 | 选项 | 注释 | SQL 预览

名	类型
id	int
rolename	varchar
rolememo	varchar

sys_user_role: 用户-角色关系表

名	类型
userid	int
roleid	int

蛙课网

该水印由迅读PDF生成，
如果想去掉该水印，请访问并下载：
http://www.pdfxd.com

细说 Spring Security 安全框架（三）

蛙课网　蛙课网　蛙课网

对象　sys_user @ssm (127.0.0.1_m...

新建　保存　另存为　添加字段　插

字段　索引　外键　触发器　选项　注释　SQL 预览

名	类型
id	int
username	varchar
password	varchar
realname	varchar
isenable	int
islock	int
iscredentials	int
createtime	date
logintime	date

蛙课网

字段　索引　外键　触发器　选项　注释　SQL 预览

名	类型
id	int
rolename	varchar
rolememo	varchar

迅读PDF大师　蛙课网

名	类型
userid	int
roleid	int

3.5 pom 依赖

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter</artifactId>
  <version>2.0.6.RELEASE</version>
</parent>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.11</version>
    <scope>test</scope>
  </dependency>
  <!--security-->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
  </dependency>
  <!--web-->
  <dependency>
```

```
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<!--mysql 驱动-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.9</version>
</dependency>
    <!--mybatis-->
<dependency>
    <groupId>org.mybatis.spring.boot</groupId>
    <artifactId>mybatis-spring-boot-starter</artifactId>
    <version>2.0.1</version>
</dependency>
</dependencies>
```

3.6 创建实体类

定义用户信息类，实现 spring security 框架中的 UserDetails.

```
public class SysUser implements UserDetails {

```

```
private Integer id;

private String username;

private String password;

private String realname;


private boolean isExpired;

private boolean isLocked;

private boolean isCredentials;

private boolean isEnabled;



private Date createTime;

private Date loginTime;


private List<GrantedAuthority> authorities;


public SysUser() {

}


public SysUser(String username, String password, String realname,

               boolean isExpired, boolean isLocked,
```

```
                    boolean isCredentials, boolean isEnabled,
                    Date createTime, Date
loginTime,List<GrantedAuthority> authorities) {
        this.username = username;
        this.password = password;
        this.realname = realname;
        this.isExpired = isExpired;
        this.isLocked = isLocked;
        this.isCredentials = isCredentials;
        this.isEnabled = isEnabled;
        this.createTime = createTime;
        this.loginTime = loginTime;
        this.authorities = authorities;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }

    @Override
```

```
public String getPassword() {
    return password;
}

@Override
public String getUsername() {
    return username;
}

@Override
public boolean isAccountNonExpired() {
    return isExpired;
}

@Override
public boolean isAccountNonLocked() {
    return isLocked;
}

@Override
public boolean isCredentialsNonExpired() {
```

```
        return isCredentials;
    }

    @Override
    public boolean isEnabled() {
        return isEnabled;
    }

    public Integer getId() {
        return id;
    }

    public Date getCreateTime() {
        return createTime;
    }

    public Date getLoginTime() {
        return loginTime;
    }

    public String getRealname() {
```

```
        return realname;
    }



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

    public void setUsername(String username) {
        this.username = username;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public void setRealname(String realname) {
        this.realname = realname;
    }

    public void setExpired(boolean expired) {
```

```
        isExpired = expired;
    }

    public void setLocked(boolean locked) {
        isLocked = locked;
    }

    public void setCredentials(boolean credentials) {
        isCredentials = credentials;
    }

    public void setEnabled(boolean enabled) {
        isEnabled = enabled;
    }

    public void setCreateTime(Date createTime) {
        this.createTime = createTime;
    }

    public void setLoginTime(Date loginTime) {
        this.loginTime = loginTime;
```

```
}

public void setAuthorities(List<GrantedAuthority> authorities) {
    this.authorities = authorities;
}

@Override
public String toString() {
    return "SysUser{" +
            "id=" + id +
            ", username='" + username + '\'' +
            ", password='" + password + '\'' +
            ", realname='" + realname + '\'' +
            ", isExpired=" + isExpired +
            ", isLocked=" + isLocked +
            ", isCredentials=" + isCredentials +
            ", isEnabled=" + isEnabled +
            ", createTime=" + createTime +
            ", loginTime=" + loginTime +
            ", authorities=" + authorities +
            '}';
```

```
    }
}
```

SysRole 类：

```
public class SysRole {
    private Integer id;
    private String name;
    private String memo;

    public Integer getId() {
        return id;
    }

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

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public String getMemo() {
        return memo;
    }

    public void setMemo(String memo) {
        this.memo = memo;
    }

    @Override
    public String toString() {
        return "SysRole{" +
                "id=" + id +
                ", name='" + name + '\'' +
                ", memo='" + memo + '\'' +
                '}';
    }
}
```

3.7 Mapper 类

```
@Repository //创建 dao 对象
public interface SysUserMapper {

    int insertSysUser(SysUser user);

    //根据账号名称，获取用户信息
    SysUser selectSysUser(String username);
}
```

```
public interface SysRoleMapper {

   List<SysRole> selectRoleByUser(Integer userId);
}
```

3.8 创建 UserDetatilsService 接口的实现类

根据 username 从数据查询账号信息 SysUser 对象。

在根据 user 的 id， 去查 Sys_Role 表获取 Role 信息

```
@Service
public class JdbcUserDetatilsService implements UserDetailsService {

    @Autowired
    private SysUserMapper userMapper;

    @Autowired
    private SysRoleMapper roleMapper;

    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        //1. 根据 username 获取 SysUser
        SysUser user  = userMapper.selectSysUser(username);
        System.out.println("loadUserByUsername user:"+user);
        if( user != null){
            //2. 根据 userid 的，获取 role
            List<SysRole> roleList =
roleMapper.selectRoleByUser(user.getId());
            System.out.println("roleList:"+ roleList);
```

```
            List<GrantedAuthority> authorities = new ArrayList<>();
            String roleName = "";
            for (SysRole role : roleList) {
                roleName   = role.getName();
                GrantedAuthority authority = new
SimpleGrantedAuthority("ROLE_"+roleName);
                authorities.add(authority);
            }
            user.setAuthorities(authorities);
            return user;
        }
        return user;
    }
}
```

3.9 创建 Controller

1.IndexController ： 转发到首页 index.html

```
@Controller
public class IndexController {
```

```
    @GetMapping("/index")
    public String toIndexHtml(){
        return "forward:/index.html";
    }
}
```

2.MyController

```
@RestController
public class MyController {

    @GetMapping(value = "/access/user",produces =
"text/html;charset=utf-8")
    public String sayUser(){
        return "zs 是 user 角色";
    }

    @GetMapping(value = "/access/read",produces =
"text/html;charset=utf-8")
    public String sayRead(){
        return "lisi 是 read 角色";
    }
```

```
    @GetMapping(value = "/access/admin",produces =
"text/html;charset=utf-8")
    public String sayAdmin(){
        return "admin 是 user ， admin 角色";
    }
}
```

3.10 继承 WebSecurityConfigurerAdapter

注入自定义的 UserDetatilsService

```
@Configuration
@EnableWebSecurity
public class CustomSecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService;
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
```

```
        //super.configure(auth);
        auth.userDetailsService(userDetailsService).passwordEncoder(new
BCryptPasswordEncoder());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        System.out.println("=======configure HttpSecurity========== ");
        http.authorizeRequests()
                .antMatchers("/index").permitAll()
                .antMatchers("/access/user/**").hasRole("USER")
                .antMatchers("/access/read/**").hasRole("READ")
                .antMatchers("/access/admin/**").hasRole("AMDIN")
                .anyRequest().authenticated()
                .and()
                .formLogin();
    }


}
```

3.11 主类

```
@MapperScan(value = "com.wkcto.mapper")
@SpringBootApplication
public class UserRoleApplication {

    @Autowired
    SysUserMapper userMapper;

    public static void main(String[] args) {
        SpringApplication.run(UserRoleApplication.class,args);
    }

    //@PostConstruct
    public void jdbcInit(){

        Date curDate = new Date();
        PasswordEncoder encoder = new BCryptPasswordEncoder();



        List<GrantedAuthority> list = new ArrayList<>();
```

```
        //参数角色名称，需要以"ROLE_"开头， 后面加上自定义的角色名称
        GrantedAuthority authority = new
SimpleGrantedAuthority("ROLE_"+"READ");
        list.add(authority);



        SysUser user = new SysUser(
                "lisi",encoder.encode("456"),"李四
",true,true,true,true,curDate, curDate, list
        );
        userMapper.insertSysUser(user);

        List<GrantedAuthority> list2 = new ArrayList<>();
        GrantedAuthority authority2 = new
SimpleGrantedAuthority("ROLE_"+"AMDIN");
        GrantedAuthority authority3 = new
SimpleGrantedAuthority("ROLE_"+"USER");
        list.add(authority2);
        list.add(authority3);

        SysUser user2 = new SysUser(
```

```
                "admin",encoder.encode("admin"),"管理员
",true,true,true,true,curDate, curDate, list2
        );
        userMapper.insertSysUser(user2);

    }
}
```

3.12 mapper 文件

在 resources 目录创建了 mapper，存放 mybatis 的 xml 文件

1. SysUser 表的操作 mapper 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.wkcto.mapper.SysUserMapper">

    <!--定义 列和 属性的对应关系-->
    <resultMap id="userMapper" type="com.wkcto.entity.SysUser">
        <id column="id" property="id"/>
```

```
        <result column="username" property="username"/>
        <result column="password" property="password" />
        <result column="realname" property="realname" />
        <result column="isenable" property="isEnabled" />
        <result column="islock" property="isLocked" />
        <result column="iscredentials" property="isCredentials" />
        <result column="createtime" property="createTime" />
        <result column="logintime" property="loginTime" />
        <result column="isexpire" property="isExpired" />
    </resultMap>

    <insert id="insertSysUser">
        insert into sys_user(username,password,realname,
            isenable,islock,iscredentials,createtime,logintime)

         values(#{username},#{password},#{realname},#{isEnabled},
            #{isLocked},#{isCredentials},#{createTime},#{loginTime})
    </insert>

    <select id="selectSysUser" resultMap="userMapper">
```

```
        select id, username,password,realname,isexpire,
            isenable,islock,iscredentials,createtime,logintime
            from sys_user where username=#{username}
    </select>
</mapper>
```

2.SysRoleMapper 文件

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper
        PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
        "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="com.wkcto.mapper.SysRoleMapper">

    <!--定义 列和 属性的对应关系-->
    <resultMap id="roleMapper" type="com.wkcto.entity.SysRole">
        <id column="id" property="id"/>
        <result column="rolename" property="name"/>
        <result column="rolememo" property="memo" />
    </resultMap>
```

```
    <select id="selectRoleByUser" resultMap="roleMapper">
        select r.id, r.rolename,r.rolememo from sys_user_role ur , sys_role r
        where ur.roleid = r.id and ur.userid=#{userid}
    </select>
</mapper>
```

3.13 index.html

在 resources/static 目录中，创建 index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>
    <p>验证访问</p>
    <a href="/access/user">验证 zs</a> <br/>
    <a href="/access/read">验证 lisi</a><br/>
    <a href="/access/admin">验证 admin</a><br/>
```

```
</body>
</html>
```

3.14. application.properties

```
#数据源配置
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/ssm
spring.datasource.username=root
spring.datasource.password=123456

#mybatis 需要的 mapper 文件位置
mybatis.mapper-locations=classpath:/mapper/*Mapper.xml
# 别名
mybatis.type-aliases-package=com.wkcto.entity
```

3.14 默认的登录页面

1.访问地址 /login

2.请求方式 post

3.请求参数

用户名 username

密码 password

Spring Security

第四章 自定义登录和验证码的使用

第一个部分，完善自定义登录页面。

第二个部分，使用验证码功能（生成验证码；检查提交的验证码）

4.1 完善自定义登录页面

1. 创建页面

登录页面 resources/static/mylogin.html

action: /login 可以自定义

method：post 这个是一定的。

参数： username ，password 可以自定义

```
可以使用 http 对象.usernameParameter("myname")
        http 对象.passwordParameter("mypwd")
```

错误提示页面 resources/static/error.html

```
error.html 登录错误，请检查用户名和密码
```

2.设置自定义登录参数

```
重写 protected void configure(HttpSecurity http) 方法
```

①：设置访问的白名单，无需登录验证就可以访问的地址

```
.antMatchers("/index","/mylogin.html","/login","/error.html")
.permitAll()
```

②：指定登录页面，登录的 uri 地址

```
.loginPage("/mylogin.html")  //登录的自定义视图页面
.loginProcessingUrl("/login") //form 中登录访问 uri 地址
```

③：指定登录错误的提示页面

```
.failureUrl("/error.html") //登录验证错误的提示页面
```

3.关闭跨域访问的安全设置

```
//关于跨域访问的安全设置，先禁用
.csrf().disable();
```

4.2 ajax 登录方式

上面的登录方式是 基于表单 form 的。 对于现在的前后端分类的方式不适合。 如果要使用前后端分离， 一般使用 json 作为数据的交互格式。 需要使用另一种方式才可以。

ajax 方式，用户端发起请求， springsecurity 接收请求验证用户的用

户名和密码，把验证结果返回给请求方（json 数据）

api 说明：

①AuthenticationSuccessHandler：

当 spring security 框架验证用户信息成功后执行的接口，执行的是 onAuthenticationSuccess（）方法

②：AuthenticationFailureHandler：

当 spring security 框架验证用户信息失败后执行的接口，接口中方法 onAuthenticationFailure()方法

实现步骤：

1.加入 jquery.js 文件

在 static 目录中，创建 js 目录，拷贝 jquery-3.4.1.js 文件

2.修改 mylogin.html 为 myajax.html

3.在 myajax.html 文件中加入 jquery

```
<script type="text/javascript"
src="/js/jquery-3.4.1.js"></script>
```

4.在 myajax.html 文件中，加入 ajax 请求处理

```
<script type="text/javascript">
    $(function(){
```

```
        //jquery 的入口函数
        $("#btnLogin").click(function(){

            var uname = $("#username").val();
            var pwd = $("#password").val();
            $.ajax({
                url:"/login",
                type:"POST",
                data:{
                    "username":uname,
                    "password":pwd
                },
                dataType:"json",
                success:function(resp){
                    alert("代码：" + resp.code+"  提示："+
resp.msg)
                }
            })
        })
    })
</script>
```

5.myajax.html 定义的页面 dom 对象

```
<div >
    用户名：<input type="text" id="username" value=""> <br/>
    密码：<input type="text" id="password" value=""> <br/>
    <button id="btnLogin">使用 ajax 登录</button>
</div>
```

6.创建 handler 实现两个不同接口

```
@Component
public class MySuccessHandler implements
AuthenticationSuccessHandler {
    /*
        参数：
            request ： 请求对象
            response: 应答对象
            authentication: spring security 框架验证用户信息成功后
的封装类。
     */
    @Override
public void onAuthenticationSuccess(HttpServletRequest
request,
```

```
                    HttpServletResponse response,
                    Authentication authentication) throws
IOException, ServletException {
//登录的用户信息验证成功后执行的方法
response.setContentType("text/json;charset=utf-8");

        Result result = new Result();
        result.setCode(0);
        result.setError(1000);
        result.setMsg("登录成功");

        OutputStream out = response.getOutputStream();
        ObjectMapper om = new ObjectMapper();
        om.writeValue(out,result);
        out.flush();
        out.close();
    }
}
```

```
@Component
public class MyFailureHandler implements
```

```
AuthenticationFailureHandler {
    /*
        参数:
          request : 请求对象
          response: 应答对象
          authentication: spring security框架验证用户信息成功后
的封装类。
     */
    @Override
    public void onAuthenticationFailure(HttpServletRequest
request,
                                        HttpServletResponse
response,
AuthenticationException e) throws IOException,
ServletException {
        //当框架验证用户信息失败时执行的方法
        response.setContentType("text/json;charset=utf-8");
        Result  result  = new Result();
        result.setCode(1);
        result.setError(1001);
        result.setMsg("登录失败");
```

```
        OutputStream out = response.getOutputStream();
        ObjectMapper om = new ObjectMapper();
        om.writeValue(out,result );
        out.flush();
        out.close();
    }
}
```

7.在 pom.xml 文件加入 jackson 依赖

```
<!--jackson-->
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-core</artifactId>
  <version>2.9.8</version>
</dependency>
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
  <version>2.9.8</version>
</dependency>
```

8. 创建作为结果的对象 Result

```
public class Result {
    // code=0 成功;  code =1 失败
    private int code;
    //表示错误码
    private int error;
    //消息文本
    private String msg;
    //set | get 方法
}
```

9.配置 handler

```
.formLogin()
.successHandler(successHandler)
.failureHandler(failureHandler)
```

4.3 验证码

验证码：使用的字母和数字的组合，使用 6 为验证码。

介绍：验证码给用户的显示是通过图片完成的。在 html 页面中使用 <img> 指定一个图片。图片内容是验证码。

生成验证码：自定义实现；实现开源的库。

实现验证码：使用 servlet， 也可以使用 controller

实现验证功能：

1. 创建 Controller 类：CaptchaController

①创建图像类：BufferedImage

```
BufferedImage image = new BufferedImage(width,height,
BufferedImage.TYPE_INT_RGB);
```

②获取图像上的画笔

```
Graphics g  = image.getGraphics();
```

使用 Graphices 在 image 上画内容，可以是文字，线条，图形等

③给图形设置背景色

```
g.setColor(Color.white);
```

④创建 Font

```
Font font  = new Font("宋体",Font.BOLD,16);
g.setFont(font);
```

⑤绘制文字

```
for(int i=0;i<charCount;i++){
    ran  = new Random().nextInt(len);
    buffer.append(chars[ran]);
    g.setColor(makeColor());
```

```
    g.drawString(chars[ran],(i+1)*space,drawY);
}
```

g.drawString(要绘制的文字，x，y)

生成颜色的方法：

```
private Color makeColor(){
    Random random = new Random();
    int r = random.nextInt(255);
    int g = random.nextInt(255);
    int b = random.nextInt(255);
    return new Color(r,g,b);
}
```

⑥设置干扰线

```
for(int m=0;m<4;m++){
    g.setColor(makeColor());
    int dot [] = makeLineDot();
    g.drawLine(dot[0],dot[1],dot[2],dot[3]);
}
```

画线的方法 drawLine(x1,y1,x2,y2)

x1,y1 是线的起点坐标

x2,y2 是线的终点坐标

生成线端点的方法

```
private int [] makeLineDot(){
    Random random = new Random();
    int x1 = random.nextInt(width/2);
    int y1 = random.nextInt(height);
    int x2 = random.nextInt(width);
    int y2 = random.nextInt(height);
    return new int[]{x1,y1,x2,y2};
}
```

⑦ 设置缓存，不要缓存

```
response.setHeader("Pragma","no-cache");
response.setHeader("Cache-Control","no-cache");
response.setDateHeader("Expires",0);
```

⑧ 设置输出的内容类型

```
response.setContentType("image/png");
```

⑨输出 Image

```
OutputStream out = response.getOutputStream();
/*
```

```
    RenderedImage im, 输出的图像
    String formatName, 图像的格式 jpg, jpeg,  png
    ImageOutputStream output 输出到哪
  */
ImageIO.write(image,"png",out);
out.flush();
out.close();
```

⑩ 把验证码存放到 session

```
request.getSession().setAttribute("code",buffer.toString());
```

4.4 修改 myajax.html 增加验证码

①myajax.html 增加 img

```
<div >
    用户名：<input type="text" id="username" value=""> <br/>
    密  码：<input type="text" id="password"
value=""> <br/>
    验证码：<input type="text" id="code" value=""> <br/>
     <!--图像，显示验证码的值 -->
     <img id="imagecode" src="/captcha/code" />
```

```
        <a href="javascript:void(0)"       onclick="changeCode()">
重新获取</a>
        <br/>
        <br/>
        <button id="btnLogin">使用 ajax 登录</button>
</div>
```

②增加 changeCode 函数

```
function changeCode() {
    //new Date 目的是浏览器不使用缓存，每次获取新的内容
    var url="/captcha/code?t="+new Date();
    $("#imagecode").attr("src",url);
}
```

使用 jquery 的 attr 函数，设置 img 标签的 src 属性。

③增加验证码参数

```
$("#btnLogin").click(function(){
    var uname = $("#username").val();
    var pwd = $("#password").val();
    var txtcode = $("#txtcode").val();
    $.ajax({
        url:"/login",
```

```
        type:"POST",
        data:{
            "username":uname,
            "password":pwd,
            "code":txtcode
        },
        dataType:"json",
        success:function(resp){
            alert("代码：" + resp.code+"  提示："+ resp.msg)
        }
    })
})
```

4.5 进行验证 code

使用的是过滤器，整个 spring security 框架都是过滤器实现的。

目前使用表单登录，验证用户名和密码使用的过滤器是

UsernamePasswordAuthenticationFilter

在验证 username ,password 的值之前 ,就应该先验证 code 是否正确。按照这个思路，在过滤器链条中，在 UsernamePasswordAuthenticationFilter 之前增加一个自定义的过滤器 ,让这个新加的过滤器验证 session 中的 code 和请求中的 code 是否一样。 如果验证失败抛出异常。 spring security 框架根据异常决定身份认证是否正确。

实现自定义的过滤器方式：

1.直接实现 Filter 接口

2.继承 OncePerRequestFilter：只执行一次的过滤器

实现步骤：

1.创建异常类，继承 AuthenticationException

```
public class VerificationException extends
AuthenticationException {
    public VerificationException(String msg, Throwable t) {
        super(msg, t);
    }
    public VerificationException(String msg) {
        super(msg);
    }
```

```
    public VerificationException() {
        super("验证错误，请重新输入");
    }
}
```

2.创建过滤器类，继承 OncePerRequestFilter

```
public class VerificationCodeFilter extends
OncePerRequestFilter {

    private MyFailureHandler failureHandler = new
MyFailureHandler();
    @Override
    protected void doFilterInternal(HttpServletRequest
request,
                                    HttpServletResponse
response,
                                    FilterChain filterChain)
throws ServletException, IOException {

        System.out.println("VerificationCodeFilter
doFilterInternal ");
```

```
//只有是 login 操作，才需要这个过滤器参与验证码的使用
String uri = request.getRequestURI();
if( !"/login".equals(uri)){
    //过滤器正常执行，不参与验证码操作
    filterChain.doFilter(request,response);
} else {
    //登录操作，需要验证 code
    try{
        //验证：code 是否正确
        verifcatioinCode(request);
        //如果验证通过，过滤器正常执行
        filterChain.doFilter(request,response);


    }catch (VerificationException e){
        Result result  = new Result();
        result.setCode(1);
        result.setError(1002);
        result.setMsg("验证码错误！！！");
        failureHandler.setResult(result);
```

```
failureHandler.onAuthenticationFailure(request,response,e);
                }
            }
        }

    private void verifcatioinCode(HttpServletRequest
request){
        HttpSession session = request.getSession();
        //获取请求中的 code
        String requestCode = request.getParameter("code");
        //获取 session 中的 code
        String sessionCode = "";

        Object attr = session.getAttribute("code");
        if(attr !=null ){
            sessionCode = (String)attr;
        }
        System.out.println("VerificationCodeFilter
doFilterInternal
requestCode:"+requestCode+"|sessionCode:"+sessionCode);
        //处理逻辑
```

```
        if(!StringUtils.isEmpty(sessionCode)){
            //在 session 中的 code， 用户看到这个 code 了。
            //如果能到这段代码，说明用户已经发起了登录请求的。
            //session 中的现在的这个 code 就应该无用
            session.removeAttribute("code");
        }

        //判断 code 是否正确。
        if( StringUtils.isEmpty(requestCode) ||
                StringUtils.isEmpty(sessionCode) ||
                !requestCode.equals(sessionCode) ){
            //失败
            throw new VerificationException();
        }
    }
}
```

3.把自定义的过滤器添加到过滤器链中

```
//在框架的过滤器链条中，增加一个自定义过滤器
http.addFilterBefore(new VerificationCodeFilter(),
UsernamePasswordAuthenticationFilter.class);
```

蛙课网 蛙课网 蛙课网
蛙课网 蛙课网 蛙课网
蛙课网 蛙课网 蛙课网