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。

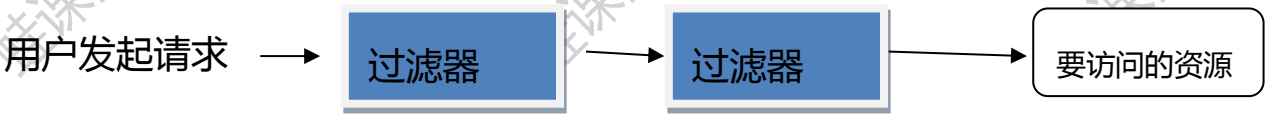
③增加验证码参数

```
$("#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);
```