

参考：

前言

前面的文章也对Spring Security有了基本的了解，然而处理login请求是完全通过Spring Security默认实现的，这无疑不会符合有特殊需求的情况，比如我要添加一个验证码验证。那不用慌，Spring Security是支持我们自定义表单登录的。

Demo

既然拿验证码举例，我们先把验证码这块加入成功，验不验证待会再说。

先贴出验证码的处理类。

```
public class VerifyServlet extends HttpServlet {

    private static final long serialVersionUID = -5051097528828603895L;

    /**
     * 验证码图片的宽度。
     */
    private int width = 100;

    /**
     * 验证码图片的高度。
     */
    private int height = 30;

    /**
     * 验证码字符个数
     */
    private int codeCount = 4;

    /**
     * 字体高度
     */
    private int fontHeight;
```

```

/**
 * 干扰线数量
 */
private int interLine = 16;

/**
 * 第一个字符的x轴值，因为后面的字符坐标依次递增，所以它们的x轴值是codeX的倍数
 */
private int codeX;

/**
 * codeY,验证字符的y轴值，因为并行所以值一样
 */
private int codeY;

/**
 * codeSequence 表示字符允许出现的序列值
 */
char[] codeSequence = { 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q',
'R',
        'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9' };

/**
 * 初始化验证图片属性
 */
@Override
public void init() throws ServletException {
    // 从web.xml中获取初始信息
    // 宽度
    String strWidth = this.getInitParameter("width");
    // 高度
    String strHeight = this.getInitParameter("height");
    // 字符个数
    String strCodeCount = this.getInitParameter("codeCount");
    // 将配置的信息转换成数值
    try {
        if (strWidth != null && strWidth.length() != 0) {
            width = Integer.parseInt(strWidth);
        }
        if (strHeight != null && strHeight.length() != 0) {
            height = Integer.parseInt(strHeight);
        }
        if (strCodeCount != null && strCodeCount.length() != 0) {
            codeCount = Integer.parseInt(strCodeCount);
        }
    } catch (NumberFormatException e) {
        e.printStackTrace();
    }
    // width-4 除去左右多余的位置，使验证码更加集中显示，减得越多越集中。
    // codeCount+1 //等比分配显示的宽度，包括左右两边的空格
    codeX = (width - 4) / (codeCount + 1);
    // height - 10 集中显示验证码

```

```

        fontHeight = height - 10;
        codeY = height - 7;
    }

    /**
     * @param request
     * @param response
     * @throws ServletException
     * @throws java.io.IOException
     */
    @Override
    protected void service(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, java.io.IOException {
        // 定义图像buffer
        BufferedImage buffImg = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        Graphics2D gd = buffImg.createGraphics();
        // 创建一个随机数生成器类
        Random random = new Random();
        // 将图像填充为白色
        gd.setColor(Color.LIGHT_GRAY);
        gd.fillRect(0, 0, width, height);
        // 创建字体，字体的大小应该根据图片的高度来定。
        Font font = new Font("Times New Roman", Font.PLAIN, fontHeight);
        // 设置字体。
        gd.setFont(font);
        // 画边框。
        gd.setColor(Color.BLACK);
        gd.drawRect(0, 0, width - 1, height - 1);
        // 随机产生16条干扰线，使图象中的认证码不易被其它程序探测到。
        gd.setColor(Color.gray);
        for (int i = 0; i < interLine; i++) {
            int x = random.nextInt(width);
            int y = random.nextInt(height);
            int xl = random.nextInt(12);
            int yl = random.nextInt(12);
            gd.drawLine(x, y, x + xl, y + yl);
        }
        // randomCode用于保存随机产生的验证码，以便用户登录后进行验证。
        StringBuffer randomCode = new StringBuffer();
        int red = 0, green = 0, blue = 0;
        // 随机产生codeCount数字的验证码。
        for (int i = 0; i < codeCount; i++) {
            // 得到随机产生的验证码数字。
            String strRand = String.valueOf(codeSequence[random.nextInt(36)]);
            // 产生随机的颜色分量来构造颜色值，这样输出的每位数字的颜色值都将不同。
            red = random.nextInt(255);
            green = random.nextInt(255);
            blue = random.nextInt(255);
            // 用随机产生的颜色将验证码绘制到图像中。
            gd.setColor(new Color(red, green, blue));
            gd.drawString(strRand, (i + 1) * codeX, codeY);
        }
    }

```

```

        // 将产生的四个随机数组合在一起。
        randomCode.append(strRand);
    }
    // 将四位数字的验证码保存到Session中。
    HttpSession session = request.getSession();
    session.setAttribute("validateCode", randomCode.toString());
    // 禁止图像缓存。
    response.setHeader("Pragma", "no-cache");
    response.setHeader("Cache-Control", "no-cache");
    response.setDateHeader("Expires", 0);

    response.setContentType("image/jpeg");
    // 将图像输出到Servlet输出流中。
    ServletOutputStream sos = response.getOutputStream();
    ImageIO.write(buffImg, "jpeg", sos);
    sos.close();
}
}

```

在启动类中进行注入。

```

/**
 * 注入验证码servlet
 */
@Bean
public ServletRegistrationBean indexServletRegistration() {
    ServletRegistrationBean registration = new ServletRegistrationBean(new
VerifyServlet());
    registration.addUrlMappings("/getVerifyCode");
    return registration;
}

```

login.html如下改动。

```

<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:th="http://www.thymeleaf.org"
    xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3">
<head>
<title>Spring Security Example</title>
</head>
<body>
    <div th:if="${param.error}">用户名或密码错</div>
    <div th:if="${param.logout}">您已注销成功</div>
    <form th:action="@{/login}" method="post">
        <div>
            <label> 用户名 : <input type="text" name="username" />
            </label>
        </div>
        <div>
            <label> 密 码 : <input type="password" name="password" />
            </label>
        </div>
    </form>

```

```

<div>
    <input type="text" class="form-control" name="verifyCode"
        required="required" placeholder="验证码" /> 
</div>
<div>
    <label> <input type="checkbox" name="remember-me" /> 自动登录 </label>
    <input type="submit" value="登录" />
</div>
</form>
<script>
    function refresh(obj) {
        obj.src = "getVerifyCode?" + Math.random();
    }

    function mouseover(obj) {
        obj.style.cursor = "pointer";
    }
</script>
</body>
</html>

```

WebSecurityConfig类中对该验证码的请求路径放行。

```

authorizeRequests()
    .antMatchers("/", "/home", "/getVerifyCode").permitAll() //定义不需要拦截的路径

```

接下来我们启动看效果。



当然现在这个验证码还是没有效果的，如何验证有三种方式Ajax验证，过滤器验证，Spring Security验证，我们这边肯定要学会用第三种咯。

前文我们基于Spring Security虽说用的很舒坦，但完全封闭的自动实现，又有点无奈的感觉，我们其实可以通过自己实现几个类然后注册我们自己的类就行了，WebAuthenticationDetails（获取登录时的额外

信息），AuthenticationDetailsSource（填充我们自己重写的详细信息），AuthenticationProvider（重写验证）。

我们先看看如何把我们的额外信息放入到Spring Security。

```
public class CustomWebAuthenticationDetails extends WebAuthenticationDetails {
    private static final long serialVersionUID = 6975601077710753878L;
    private final String verifyCode;

    public CustomWebAuthenticationDetails(HttpServletRequest request) {
        super(request);
        // verifyCode为页面中验证码的name
        verifyCode = request.getParameter("verifyCode");
    }

    public String getVerifyCode() {
        return this.verifyCode;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(super.toString()).append("; VerifyCode: ").append(this.getVerifyCode());
        return sb.toString();
    }
}

public class CustomWebAuthenticationDetails extends WebAuthenticationDetails {
    private static final long serialVersionUID = 6975601077710753878L;
    private final String verifyCode;

    public CustomWebAuthenticationDetails(HttpServletRequest request) {
        super(request);
        // verifyCode为页面中验证码的name
        verifyCode = request.getParameter("verifyCode");
    }

    public String getVerifyCode() {
        return this.verifyCode;
    }

    @Override
    public String toString() {
        StringBuilder sb = new StringBuilder();
        sb.append(super.toString()).append("; VerifyCode: ").append(this.getVerifyCode());
        return sb.toString();
    }
}
```

```
/**
 * 该接口用于在Spring Security登录过程中对用户的登录信息的详细信息进行填充
 */
```

```

*/
@Component
public class CustomAuthenticationDetailsSource
    implements AuthenticationDetailsSource<HttpServletRequest,
WebAuthenticationDetails> {
    @Override
    public WebAuthenticationDetails buildDetails(HttpServletRequest request) {
        return new CustomWebAuthenticationDetails(request);
    }
}

```

```

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.authorizeRequests().antMatchers("/", "/home", "/getVerifyCode").permitAll() // 定义不需要拦截的路径
        .anyRequest().authenticated().and().formLogin().loginPage("/login")// 设置登录页
        .permitAll()
        // 指定AuthenticationDetailsSource
        .authenticationDetailsSource(authenticationDetailsSource)
        // 退出
        .and().logout().permitAll()
        // 自动登录
        .and().rememberMe();
}

```

然后把CustomAuthenticationDetailsSource注入进来这样指定即可。

这里我们的验证码已经和用户名和密码一样被Spring Security管理了，但显然Spring Security默认的验证肯定不会去管验证码的，这时候我们就得自己重写验证类。

```

@Component
public class CustomAuthenticationProvider implements AuthenticationProvider {
    @Autowired
    private CustomUserDetailsService customUserDetailsService;

    @Override
    public Authentication authenticate(Authentication authentication) throws
AuthenticationException {
        // 获取用户输入的用户名和密码
        String inputName = authentication.getName();
        String inputPassword = authentication.getCredentials().toString();

        CustomWebAuthenticationDetails details = (CustomWebAuthenticationDetails)
authentication.getDetails();

        String verifyCode = details.getVerifyCode();
        if (!validateVerify(verifyCode)) {
            throw new DisabledException("验证码输入错误");
        }

        // userDetails为数据库中查询到的用户信息

```

```

        UserDetails userDetails =
customUserDetailsService.loadUserByUsername(inputName);

        System.out.println("用户: " + userDetails);

        // 如果是自定义AuthenticationProvider, 需要手动密码校验
        if (!userDetails.getPassword().equals(inputPassword)) {
            throw new BadCredentialsException("密码错误");
        }

        return new UsernamePasswordAuthenticationToken(inputName, inputPassword,
userDetails.getAuthorities());
    }

    private boolean validateVerify(String inputVerify) {
        // 获取当前线程绑定的request对象
        HttpServletRequest request = ((ServletRequestAttributes)
RequestContextHolder.getRequestAttributes())
            .getRequest();
        // 不分区大小写
        // 这个validateCode是在servlet中存入session的名字
        String validateCode = ((String)
request.getSession().getAttribute("validateCode")).toLowerCase();
        inputVerify = inputVerify.toLowerCase();

        System.out.println("验证码: " + validateCode + "用户输入: " + inputVerify);

        return validateCode.equals(inputVerify);
    }

    @Override
    public boolean supports(Class<?> authentication) {
        // 这里不要忘记, 和UsernamePasswordAuthenticationToken比较
        return authentication.equals(UsernamePasswordAuthenticationToken.class);
    }
}

```

```

@Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.authenticationProvider(customAuthenticationProvider);
    auth.userDetailsService(userDetailsService).passwordEncoder(new PasswordEncoder() {
        // 密码加密, 由各自系统决定, 该处明文
        @Override
        public String encode(CharSequence charSequence) {
            return charSequence.toString();
        }

        @Override
        public boolean matches(CharSequence charSequence, String s) {
            return s.equals(charSequence.toString());
        }
    });
}

```


这样注入后使用即可。

localhost:8080/study/login

应用

VP+企业级管理平台

github开源项目

H+ 后台主题UI框

用户名：

admin

密 码：

...

www

☐ 自动登录

登录

localhost:8080/study/login?error

应用

VP+企业级管理平台

github开源项目

H+ 后台主

用户名或密码或验证码错

用户名：

密 码：

验证码

☐ 自动登录

登录

乱输入就会这样，输入正确就能像之前一样正常登录。