# 开始一个Security

## 导入Jar包引入Security管理

<dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-security</artifactId>  
</dependency>

## 基于表单认证

创建一个配置类，继承WebSecurityConfigurerAdapter，重写Config方法。

@Configuration

public class BrowserSecurityConfig extends WebSecurityConfigurerAdapter {

@Override

protected void configure(HttpSecurity http) throws Exception {

http.formLogin() // 表单方式

.and()

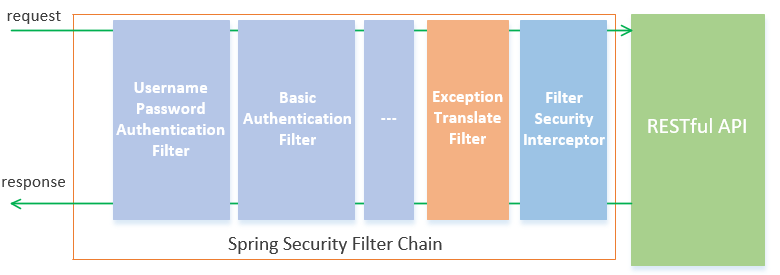
.authorizeRequests() // 授权配置

.anyRequest() // 所有请求

.authenticated(); // 都需要认证

}

}



首先第一个Filter是拦截基于表单方式的登录认证，*BasicAuthentication*

*Filter*用于处理基于HTTP Basic方式的登录验证，后面可能还有其他过滤器，但是主要就是这俩，ExceptionTranslateFilter用于处理抛出的异常，比如用户初始未登录，然后前面两个Filter会抛出异常，然后ExceptionTranslateFilter会拦截然后重定向到登陆页面，FilterSecurityInterceptor用于判断当前身份验证是否成功，是否有相应的权限，当身份认证失败或者权限不足会抛出异常。

## 初始未登录流程：

代码直接跳转到*FilterSecurityInteceptor*往下执行，因为当前请求没有经过身份认证，所以将抛出异常并被*ExceptionTranslateFilter*捕获：捕获异常后重定向到登录表单登录页面，当我们在表单登录页面输入信息点login后，代码跳转到*UsernamePasswordAuthenticationFilter*过滤器的*attemptAuthentication*方法上，判断用户名和密码是否正确之后，代码又跳回*FilterSecurityInterceptor*的*beforeInvocation*方法执行上，当认证通过时，*FilterSecurityInterceptor*代码往下执行*doFilter*，然后代码最终跳转到*/hello*上。

# 自定义认证

实现UserDetailsService，重写方法，



其中PasswordEncoder在BrowserSecurityConfig中注册。

## 自定义登陆页面



## 自定义拦截规则

假如现在有这样一个需求：在未登录的情况下，当用户访问html资源的时候跳转到登录页，否则返回JSON格式数据，状态码为401。

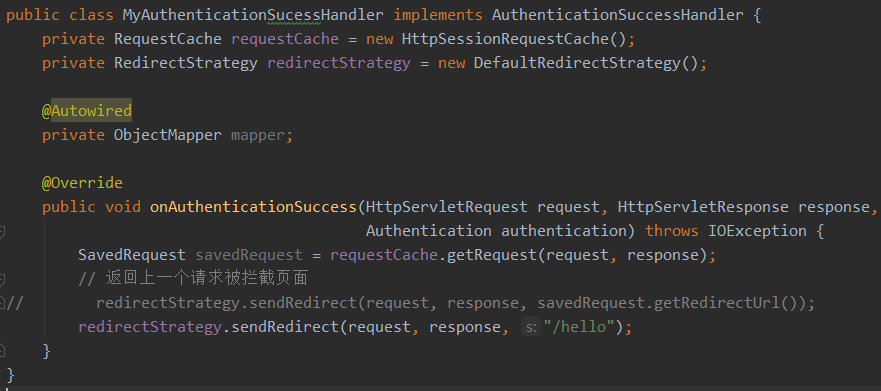
要实现这个功能我们将*loginPage*的URL改为*/authentication/require*，并且在*antMatchers*方法中加入该URL，让其免拦截:

@Override  
protected void configure(HttpSecurity http) throws Exception {  
 http.formLogin() // 表单登录  
 // http.httpBasic() // HTTP Basic  
 .loginPage("/authentication/require") // 登录跳转 URL  
 .loginProcessingUrl("/login") // 处理表单登录 URL  
 .and()  
 .authorizeRequests() // 授权配置  
 .antMatchers("/authentication/require", "/login.html").permitAll() // 登录跳转 URL 无需认证  
 .anyRequest() // 所有请求  
 .authenticated() // 都需要认证  
 .and().csrf().disable();  
}



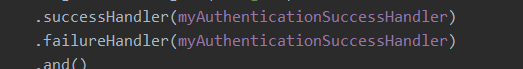
## 登录到指定页面

实现AuthenticationSuccessHandler方法，即可。



## 自定义成功、失败页面

实现implements AuthenticationSuccessHandler , AuthenticationFailureHandler接口，分别重写两个方法即可。然后记得在config里面配置



# 添加图形验证码

## 添加依赖：

|  |  |
| --- | --- |
| 1 2 3 4 | <dependency>  <groupId>org.springframework.social</groupId>  <artifactId>spring-social-config</artifactId> </dependency> |

### 创建验证码类：

public class ImageCode {  
  
 private BufferedImage image;  
  
 private String code;  
  
 private LocalDateTime expireTime;  
  
 public ImageCode(BufferedImage image, String code, int expireIn) {  
 this.image = image;  
 this.code = code;  
 this.expireTime = LocalDateTime.now().plusSeconds(expireIn);  
 }  
  
 public ImageCode(BufferedImage image, String code, LocalDateTime expireTime) {  
 this.image = image;  
 this.code = code;  
 this.expireTime = expireTime;  
 }  
  
 boolean isExpire() {  
 return LocalDateTime.now().isAfter(expireTime);  
 }  
 // get,set 略  
}

### 创建生成验证码Controller



### createImageCode():

public static ImageCode createImageCode() {

int width = 100; // 验证码图片宽度

int height = 36; // 验证码图片长度

int length = 4; // 验证码位数

int expireIn = 60; // 验证码有效时间 60s

BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE\_INT\_RGB);

Graphics g = image.getGraphics();

Random random = new Random();

g.setColor(getRandColor(200, 250));

g.fillRect(0, 0, width, height);

g.setFont(new Font("Times New Roman", Font.ITALIC, 20));

g.setColor(getRandColor(160, 200));

for (int i = 0; i < 155; i++) {

int x = random.nextInt(width);

int y = random.nextInt(height);

int xl = random.nextInt(12);

int yl = random.nextInt(12);

g.drawLine(x, y, x + xl, y + yl);

}

StringBuilder sRand = new StringBuilder();

for (int i = 0; i < length; i++) {

String rand = String.valueOf(random.nextInt(10));

sRand.append(rand);

g.setColor(new Color(20 + random.nextInt(110), 20 + random.nextInt(110), 20 + random.nextInt(110)));

g.drawString(rand, 13 \* i + 6, 16);

}

g.dispose();

return new ImageCode(image, sRand.toString(), expireIn);

}

public static Color getRandColor(int fc, int bc) {

Random random = new Random();

if (fc > 255) {

fc = 255;

}

if (bc > 255) {

bc = 255;

}

int r = fc + random.nextInt(bc - fc);

int g = fc + random.nextInt(bc - fc);

int b = fc + random.nextInt(bc - fc);

return new Color(r, g, b);

}

### 添加Img标签并放行该请求：

