Spring Cloud Security

概述

Spring Cloud Security基于Spring框架，提供了一套Web应用安全性的完整解决方案

一般来说，Web应用的安全性包括 用户认证（Authentication） 和 用户授权（Authorization）两个部分，这两天也是Spring Cloud Security重要核心功能

用户认证指的是：验证某个用户是否为系统中的合法主体，也就是说用户能否访问该系统。用户认证一般要求用户提供用户名和密码。系统通过校验用户名和密码来完成认证过程。通俗点说就是系统认为用户是否能登录。

用户授权指的是验证某个用户是否有权限执行某个操作。在一个系统中，不同的用户所具有的权限是不同的。比如对一个文件来说，有的用户只能是进行读取，而有的用户可以进行修改，一般来说，系统会为不同的用户分配不同的角色，儿每个角色则对应一系列的权限。通俗点讲就是系统判断用户是否有权限去做某些事情。 比如说商品的价格商家可以更改但是用户不能更改的权限

特点

Spring Security

- 和Spring无缝整合
- 全面的权限控制（包括根据路径,根据方法等进行精细的权限控制)
- 专门为Web开发而设计

旧版本不能脱离Web环境使用

新版本对整个框架进行了分层抽取，分成了核心模块和Web模块，单独引入核心模块就可以脱离Web环境

- 重量级（需要引入很多其他的组件和依赖）

Shiro

- Apache旗下的轻量级权限控制框架
- 轻量级：Shiro主张的理念是把复杂的事情变简单，针对性能有更高要求的互联网应用有更好的表现
- 通用性：
  - 好处：不局限于Web环境，可以脱离Web环境使用
  - 缺陷：再Web环境下一些特定的需求需要手动编写代码定制

Spring Security的模块划分

Spring Security的模块划分

module	jar	use
Core	spring-security-core.jar	核心模块
Remoting	spring-security-remoting .jar	
Web	spring-security-web.jar	Spring Security和web环境做了分离 所以需要引入web支持
Config	spring- security-config.jar	配置
LDAP	spring-security-ldap.jar	
OAuth2.0 Core	spring-security-oauth2-core.jar	基于OAuth2.0做整合
OAuth2.0 Client	spring-security-oauth2-client.jar	
OAuth2.0 JOSE	spring-security-oauth2-jose.jar	
OAuth2.0 Resource Server	spring-security-oauth2-resource-server.jar	
ACL	spring-security-acl.jar	
CAS	spring-security-cas.jar	单点登录
OpenID	spring-security-openid.jar	
Test	spring-security-test.jar	

快速开始

添加依赖—Spring Boot2.4.2

```
<dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
        </dependency>
<!-- Maven依赖中显示是2.4.2 版本  -->
```

在启动类添加注解 — `@EnableWebSecurity(debug = true)` 在debug模式中生效(实测不加也生效，暂未知道是什么情况下需要添加)

此时访问自己定义的一个controller，访问 http://localhost:39003/test/helloSpring Cloud Security就会生效拦截，转到登录页面

```
package com.hande.phospherus.security.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

```
/**
 * @ClassName SecurityController.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 16:05
 */
@RestController
@RequestMapping("/test")
public class SecurityController {

    @GetMapping("hello")
    public String hello() {
        return "hello security";
    }
}
```

在项目启动的时候控制台会打印默认用户 user

```
2021-01-31 16:23:26.212  INFO 18696 --- [  restartedMain] .s.s.UserDetailsServiceA utoConfiguration :

Using generated security password: a5ee4107-a5b4-461f-866c-e4a74fd7bbdf
```

此时security已经接管了认证授权，但是并未配置权限，所以暂时看不到权限效果

Spring Cloud Security基本原理

过滤器源码分析

Spring Cloud Security本质是一个过滤器链

Spring Cloud Security中有很多的过滤器，在项目启动的时候就会进行加载，多个过滤器联结称为过滤链，每个过滤器都进行放行操作才可以进行后续访问

```
    org.springframework.security.web.context.request.async.WebAsyncManagerIntegrationFilter
    org.springframework.security.web.context.SecurityContextPersistenceFilter
    org.springframework.security.web.header.HeaderWriterFilter
    org.springframework.security.web.csrf.CsrfFilter
    org.springframework.security.web.authentication.logout.LogoutFilter
    org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter
    org.springframework.security.web.authentication.ui.DefaultLoginPageGeneratingFilter
    org.springframework.security.web.authentication.ui.DefaultLogoutPageGeneratingFilter
    org.springframework.security.web.authentication.AnonymousAuthenticationFilter
    org.springframework.security.web.savedrequest.RequestCacheAwareFilter
    org.springframework.security.web.servletapi.SecurityContextHolderAwareRequestFilter
    org.springframework.security.web.session.SessionManagementFilter
    org.springframework.security.web.access.ExceptionTranslationFilter
    org.springframework.security.web.access.intercept.FilterSecurityInterceptor
```

FilterSecurityInterceptor：是一个方法级的权限过滤器，基本位于过滤器的最底部

- 放行操作

```
@Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
      throws IOException, ServletException {
    invoke(new FilterInvocation(request, response, chain));
  }
```

- 放行之前有别的过滤器需要放行，之后再进行判断

```
public void invoke(FilterInvocation filterInvocation) throws IOException, ServletException {
    if (isApplied(filterInvocation) && this.observeOncePerRequest) {
      // filter already applied to this request and user wants us to observe
      // once-per-request handling, so don't re-do security checking
      filterInvocation.getChain().doFilter(filterInvocation.getRequest(), filterInvocation.getResponse());
      return;
    }
    // first time this request being called, so perform security checking
    if (filterInvocation.getRequest() != null && this.observeOncePerRequest) {
      filterInvocation.getRequest().setAttribute(FILTER_APPLIED, Boolean.TRUE);
    }
    InterceptorStatusToken token = super.beforeInvocation(filterInvocation);
    try {
      filterInvocation.getChain().doFilter(filterInvocation.getRequest(), filterInvocation.getResponse());
    }
    finally {
      super.finallyInvocation(token);
    }
    super.afterInvocation(token, null);
  }
```

ExceptionTranslationFilter：是个异常过滤器，用来处理在认证授权过程中抛出的异常

- 判断各种异常，每个异常做不同的处理

```
private void doFilter(HttpServletRequest request, HttpServletResponse response, FilterChain chain)
      throws IOException, ServletException {
    try {
      chain.doFilter(request, response);
    }
    catch (IOException ex) {
      throw ex;
    }
    catch (Exception ex) {
      // Try to extract a SpringSecurityException from the stacktrace
      Throwable[] causeChain = this.throwableAnalyzer.determineCauseChain(ex);
      RuntimeException securityException = (AuthenticationException) this.throwableAnalyzer
          .getFirstThrowableOfType(AuthenticationException.class, causeChain);
      if (securityException == null) {
        securityException = (AccessDeniedException) this.throwableAnalyzer
            .getFirstThrowableOfType(AccessDeniedException.class, causeChain);
      }
      if (securityException == null) {
        rethrow(ex);
      }
      if (response.isCommitted()) {
        throw new ServletException("Unable to handle the Spring Security Exception "
            + "because the response is already committed.", ex);
      }
      handleSpringSecurityException(request, response, chain, securityException);
    }
  }
```

UsernamePasswordAuthenticationFilter：对 `/login` 的POST请求做拦截，校验表单中用户名，密码

- 登录的成功与否的返回都在他的父类中进行了定义
- 在测试的时候使用的Spring Cloud Security的默认的用户名密码，实际的是去查数据库

```
@Override
  public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)
      throws AuthenticationException {
    if (this.postOnly && !request.getMethod().equals("POST")) {
      throw new AuthenticationServiceException("Authentication method not supported: " + request.getMethod());
    }
    String username = obtainUsername(request);
    username = (username != null) ? username : "";
    username = username.trim();
    String password = obtainPassword(request);
    password = (password != null) ? password : "";
    UsernamePasswordAuthenticationToken authRequest = new UsernamePasswordAuthenticationToken(username, password);
    // Allow subclasses to set the "details" property
    setDetails(request, authRequest);
    return this.getAuthenticationManager().authenticate(authRequest);
  }
```

过滤器如何进行加载

如果说需要使用Spring Cloud Security，需要配置过滤器，我们没配置的原因是Spring Boot帮我们做到了，但是本质上他也需要配置的步骤 `DelegatingFilterProxy` ，执行步骤如下

▼ doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain) throws ServletException, IOException

```
@Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain filterChain)
      throws ServletException, IOException {

    // Lazily initialize the delegate if necessary.
    Filter delegateToUse = this.delegate;
    if (delegateToUse == null) {
      synchronized (this.delegateMonitor) {
        delegateToUse = this.delegate;
        if (delegateToUse == null) {
          WebApplicationContext wac = findWebApplicationContext();
          if (wac == null) {
            throw new IllegalStateException("No WebApplicationContext found: " +
                "no ContextLoaderListener or DispatcherServlet registered?");
          }
          //获取到Bean（FilterChainProxy）
          delegateToUse = initDelegate(wac);
        }
        this.delegate = delegateToUse;
      }
    }

    // Let the delegate perform the actual doFilter operation.
    invokeDelegate(delegateToUse, request, response, filterChain);
  }
```

▼ Filter initDelegate(WebApplicationContext wac) throws ServletException — 获取过滤器Bean

```
protected Filter initDelegate(WebApplicationContext wac) throws ServletException {
    String targetBeanName = getTargetBeanName();
    Assert.state(targetBeanName != null, "No target bean name set");
    Filter delegate = wac.getBean(targetBeanName, Filter.class);
    if (isTargetFilterLifecycle()) {
      delegate.init(getFilterConfig());
    }
    return delegate;
  }
```

▼ FilterChainProxy.class doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException

```
@Override
  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
      throws IOException, ServletException {
    boolean clearContext = request.getAttribute(FILTER_APPLIED) == null;
    if (!clearContext) {
      doFilterInternal(request, response, chain);
      return;
    }
    try {
      request.setAttribute(FILTER_APPLIED, Boolean.TRUE);
      doFilterInternal(request, response, chain);
    }
    catch (RequestRejectedException ex) {
      this.requestRejectedHandler.handle((HttpServletRequest) request, (HttpServletResponse) response, ex);
    }
    finally {
      SecurityContextHolder.clearContext();
      request.removeAttribute(FILTER_APPLIED);
    }
  }
```

▼ doFilterInternal(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException

```
private void doFilterInternal(ServletRequest request, ServletResponse response, FilterChain chain)
      throws IOException, ServletException {
    FirewalledRequest firewallRequest = this.firewall.getFirewalledRequest((HttpServletRequest) request);
    HttpServletResponse firewallResponse = this.firewall.getFirewalledResponse((HttpServletResponse) response);
    List<Filter> filters = getFilters(firewallRequest);  //过滤链中的过滤器集合
    if (filters == null || filters.size() == 0) {
      if (logger.isTraceEnabled()) {
        logger.trace(LogMessage.of(() -> "No security for " + requestLine(firewallRequest)));
      }
      firewallRequest.reset();
      chain.doFilter(firewallRequest, firewallResponse);
      return;
    }
    if (logger.isDebugEnabled()) {
      logger.debug(LogMessage.of(() -> "Securing " + requestLine(firewallRequest)));
    }
    VirtualFilterChain virtualFilterChain = new VirtualFilterChain(firewallRequest, chain, filters);
    virtualFilterChain.doFilter(firewallRequest, firewallResponse);
  }
```

▼ List<Filter> getFilters(HttpServletRequest request) — 获取过滤器链

```
private List<Filter> getFilters(HttpServletRequest request) {
    int count = 0;
    for (SecurityFilterChain chain : this.filterChains) {
      if (logger.isTraceEnabled()) {
        logger.trace(LogMessage.format("Trying to match request against %s (%d/%d)", chain, ++count,
            this.filterChains.size()));
      }
      if (chain.matches(request)) {
        return chain.getFilters();
      }
    }
    return null;
  }
```

▼ SecurityFilterChain.class — 加载过滤器

```
public interface SecurityFilterChain {

   boolean matches(HttpServletRequest request);

   List<Filter> getFilters();

}
```

开发过程中的两个重要接口 - UserDetailsService

- 当什么都没有配置的时候，账号和密码是由Spring Cloud Security定义生成的，实际开发中用的密码需要查数据库并且加密，所以需要自定义逻辑进行控制认证 — 需要用到两个接口

`UserDetailsService` — 用于编写用户账户的数据库查询过程 返回值 `UserDetails`

▼ 如果需要自定义逻辑，只需要实现UserDetailsService即可，接口定义如下

```
public interface UserDetailsService {

  /**
   * Locates the user based on the username. In the actual implementation, the search
   * may possibly be case sensitive, or case insensitive depending on how the
   * implementation instance is configured. In this case, the <code>UserDetails</code>
   * object that comes back may have a username that is of a different case than what
   * was actually requested..
   * @param username the username identifying the user whose data is required.
   * @return a fully populated user record (never <code>null</code>)
   * @throws UsernameNotFoundException if the user could not be found or the user has no
   * GrantedAuthority
   */
  UserDetails loadUserByUsername(String username) throws UsernameNotFoundException;

}
```

▼ 返回值 `UserDetails` 这个类是系统默认的用户主体 — 以后我们只需要使用 `User` class User implements UserDetails, CredentialsContainer 这个实体类即可

```
public interface UserDetails extends Serializable {

  //表示获取登陆用户所有权限
  Collection<? extends GrantedAuthority> getAuthorities();

  //表示获取密码
  String getPassword();

//表示获取用户名
  String getUsername();

  //表示判断账户是否过期
  boolean isAccountNonExpired();

  //表示判断账户是否被锁定
  boolean isAccountNonLocked();

  //表示凭证（密码）是否过期
  boolean isCredentialsNonExpired();

  //表示当前用户是否可用
  boolean isEnabled();

}
```

▼ 自定义用户验证实现步骤

- 创建类并且继承 `UsernamePasswordAuthenticationFilter` ，重写三个方法：accept / successful / unsuccessful
- 创建类实现 `UserDetailsService` ，便携查询数据过程，返回 User 对象，这个 User 对象是安全框架提供的对象

PasswordEncoder接口 — 提供密码加密功能， Spring Cloud Security只认这种方式，别的不认

- 源码

```
public interface PasswordEncoder {

  //表示把参数按照特定的解析规则进行解析
  String encode(CharSequence rawPassword);

  //表示验证从存储中国区的编码密码与编码后提交的原始密码是否匹配。如果密码匹配，则返回true；
  //如果不匹配，则返回false。第一个参数表示需要被解析的密码，第二个参数表示存储的密码
  boolean matches(CharSequence rawPassword, String encodedPassword);

  //如果解析的密码能够再次进行解析且达到更安全的结果返回true，否则返回false，默认返回false
  default boolean upgradeEncoding(String encodedPassword) {
    return false;
  }

}
```

Spring Cloud Security的使用

认证

> 这三种方式不能混用，因为是顺序查找，三选一的方式

- 设置登录的用户名和密码

1. 配置文件配置

```
spring:
  application:
    name: phospherus-security
  security:
    user:
      name: fangyang
      password: 123456
```

2. 通过配置类实现

```
package com.hande.phospherus.security.configuration;

import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * @ClassName SecurityConfiguration.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:34
 */
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {


    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        //加密,这里需要进行PasswordEncoder的创建
        BCryptPasswordEncoder bCryptPasswordEncoder = new BCryptPasswordEncoder();
        String encode = bCryptPasswordEncoder.encode("67890");
        //使用auth设置用户名和密码
        auth.inMemoryAuthentication().withUser("fangyang").password(encode).roles("admin");
    }

    @Bean
    PasswordEncoder password() {
        return new BCryptPasswordEncoder();
    }
}
```

3. 自定义编写实现类 实现 `UserDetails` 接口 — 基本演示

▼ 创建配置类，设置使用哪个 `userDeatilsService` 实现类

```
package com.hande.phospherus.security.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * @ClassName SecurityConfiguration.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:34
 */
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService userDetailsService;
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(password());
    }

    @Bean
    PasswordEncoder password() {
        return new BCryptPasswordEncoder();
    }
}
```

- 编写实现类，返回 `User` 对象，User对象有用户名密码和操作权限 — `@Service` 注解的值必须和 `@Autowired` 的名字一致

```
package com.hande.phospherus.security.service;

import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.List;

/**
 * @ClassName UserDetailServiceImpl.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:11
 */
@Service("userDetailsService")
public class MyUserDetailServiceImpl implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        //权限集合
        List<GrantedAuthority> authorities = AuthorityUtils.commaSeparatedStringToAuthorityList("role");
        //返回用户
        return new User("fang", new BCryptPasswordEncoder().encode("111111"), authorities);
    }
}
```

结合查询数据库完成用户认证 — Mybatis Plus

- 导入依赖

```
<!-- Mybatis Plus Mysql druid-->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.22</version>
            <scope>runtime</scope>
        </dependency>
        <dependency>
            <groupId>com.baomidou</groupId>
            <artifactId>mybatis-plus-boot-starter</artifactId>
            <version>3.4.2</version>
        </dependency>
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid-spring-boot-starter</artifactId>
            <version>1.1.22</version>
        </dependency>
        <!-- Mybatis Plus Mysql druid-->
```

- 建表建实体类
- 整合Mybatis Plus 创建接口， 继承 `BaseMapper`
- 在MyUserDetailServiceImpl调用mapper里面的方法查询数据库进行用户认证

```
package com.hande.phospherus.security.service;

import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.hande.phospherus.security.entity.Users;
import com.hande.phospherus.security.mapper.UserMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.List;

/**
 * @ClassName UserDetailServiceImpl.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:11
 */
@Service("userDetailsService")
public class MyUserDetailServiceImpl implements UserDetailsService {

    @Autowired
    private UserMapper userMapper;

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        //调用UserMapper中的方法查询数据库
        QueryWrapper<Users> wrapper = new QueryWrapper<>();
        wrapper.eq("name", username);
        Users user = userMapper.selectOne(wrapper);
        //数据库没有用户
        if (user == null) {
            throw new UsernameNotFoundException("用户不存在！");
        }
        //权限集合
        List<GrantedAuthority> authorities = AuthorityUtils.commaSeparatedStringToAuthorityList("role");
        //从数据库查询对象中得到用户名和密码，返回
        return new User(user.getName(), new BCryptPasswordEncoder().encode(user.getPass()), authorities);
    }
}
```

- 启动类添加注解 `@MapperScan("com.hande.phospherus.security.mapper")`

自定义登录页面不需要认证也可以访问

- 配置类中实现相关配置

```
package com.hande.phospherus.security.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * @ClassName SecurityConfiguration.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:34
 */
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(password());
    }

    @Bean
    PasswordEncoder password() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //自定义自己的登录页面
        http.formLogin()
                //登录页面设置
                .loginPage("/login.html")
                //登录访问路径，登录的时候跳到哪一个controller中，这个过程Spring Cloud Security已经帮我们做到了
                .loginProcessingUrl("/user/login")
                //登录成功之后跳转的路径
                .defaultSuccessUrl("/test/index").permitAll()
                //定义哪些方法需要保护，哪些不需要保护
                .and().authorizeRequests()
                //设置哪些路径可以直接访问，不需要认证
                .antMatchers("/", "/test/hello", "/user/login").permitAll()
                //所有请求都可以访问
                .anyRequest().authenticated()
                //关闭csrf的防护
                .and().csrf().disable();
    }
}
```

▼ 编写相关页面 - login.html — 页面中的用户名和密码标签必须叫 `username` 和 `password` ，因为Spring Cloud Security源码中指定了

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Login</title>
    </head>
    <body>
        <form action="/user/login" method="post">
            用户名：<input type="text" name="username">
            密码：<input type="text" name="password">
            <br>
            <input type="submit" value="login">
        </form>
    </body>
</html>
```

▼ 编写Controller

```
package com.hande.phospherus.security.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

/**
 * @ClassName SecurityController.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 16:05
 */
@RestController
@RequestMapping("/test")
public class SecurityController {

    @GetMapping("hello")
    public String hello() {
        return "hello security";
    }

    @GetMapping("index")
    public String index() {
        return "hello index";
    }

}
```

授权

基于角色和权限进行访问控制

▼ hasAuthority() — 如果当前的主体具有指定的权限，则返回true，否则返回false。一般只针对某一个权限进行操作，如果有多个就做不到了，比如只有admin一个可以访问某个路径，如果admin和role都可以访问，就做不到

- 在配置类设置当前访问地址有哪些权限

```
package com.hande.phospherus.security.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * @ClassName SecurityConfiguration.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:34
 */
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(password());
    }

    @Bean
    PasswordEncoder password() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //自定义自己的登录页面
        http.formLogin()
                //登录页面设置
                .loginPage("/login.html")
                //登录访问路径，登录的时候跳到哪一个controller中，这个过程Spring Cloud Security已经帮我们做到了
                .loginProcessingUrl("/user/login")
                //登录成功之后跳转的路径
                .defaultSuccessUrl("/test/index").permitAll()
                //定义哪些方法需要保护，哪些不需要保护
                .and().authorizeRequests()
                //设置哪些路径可以直接访问，不需要认证
                .antMatchers("/", "/test/hello", "/user/login").permitAll()
                //当前的登录用户只有具有admin权限才能够访问
                .antMatchers("/test/index").hasAuthority("admins")
                //所有请求都可以访问
                .anyRequest().authenticated()
                //关闭csrf的防护
                .and().csrf().disable();
    }
}
```

- 修改service

```
package com.hande.phospherus.security.service;
```

```
import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.hande.phospherus.security.entity.Users;
import com.hande.phospherus.security.mapper.UserMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

/**
 * @ClassName UserDetailServiceImpl.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:11
 */
@Service("userDetailsService")
public class MyUserDetailServiceImpl implements UserDetailsService {

    @Autowired
    private UserMapper userMapper;

  /*  @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        //权限集合
        List<GrantedAuthority> authorities = AuthorityUtils.commaSeparatedStringToAuthorityList("role");
        //返回用户
        return new User("fang", new BCryptPasswordEncoder().encode("111111"), authorities);
    }*/

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        //调用UserMapper中的方法查询数据库
        QueryWrapper<Users> wrapper = new QueryWrapper<>();
        wrapper.eq("username", username);
        Users user = userMapper.selectOne(wrapper);
        //数据库没有用户
        if (user == null) {
            throw new UsernameNotFoundException("用户不存在！");
        }
        List<GrantedAuthority> authorities = new ArrayList<>();
        if (user.getUsername().equals("fang")) {
            //权限集合
            authorities = AuthorityUtils.commaSeparatedStringToAuthorityList("admins");
        } else {
            authorities = AuthorityUtils.commaSeparatedStringToAuthorityList("role");
        }
        //从数据库查询对象中得到用户名和密码，返回
        return new User(user.getUsername(), new BCryptPasswordEncoder().encode(user.getPassword()), authorities);
    }
}
```

▼ hasAnyAuthority() — 可以针对多个权限返回

```
package com.hande.phospherus.security.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * @ClassName SecurityConfiguration.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:34
 */
```

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(password());
    }

    @Bean
    PasswordEncoder password() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //自定义自己的登录页面
        http.formLogin()
                //登录页面设置
                .loginPage("/login.html")
                //登录访问路径，登录的时候跳到哪一个controller中，这个过程Spring Cloud Security已经帮我们做到了
                .loginProcessingUrl("/user/login")
                //登录成功之后跳转的路径
                .defaultSuccessUrl("/test/index").permitAll()
                //定义哪些方法需要保护，哪些不需要保护
                .and().authorizeRequests()
                //设置哪些路径可以直接访问，不需要认证
                .antMatchers("/", "/test/hello", "/user/login").permitAll()
                //当前的登录用户只有具有admin权限才能够访问
                .antMatchers("/test/index").hasAuthority("admins")
                .antMatchers("/test/any").hasAnyAuthority("admins", "role")
                //所有请求都可以访问
                .anyRequest().authenticated()
                //关闭csrf的防护
                .and().csrf().disable();
    }
}
```

▼ hasRole() — 如果用户具备给定角色就允许访问，否则出现403， 如果当前的主体具有指定的角色，则返回true，否则返回false

- 源码中会将你的角色加上前缀 `ROLE_xxx` 的形式，所以我们在配置类中写的 `sale` ，但是在设置用户的时候需要加上前缀

```
private static String hasRole(String role) {
    Assert.notNull(role, "role cannot be null");
    Assert.isTrue(!role.startsWith("ROLE_"),
        () -> "role should not start with 'ROLE_' since it is automatically inserted. Got '" + role + "'");
    return "hasRole('ROLE_" + role + "')";
  }
```

- service

```
package com.hande.phospherus.security.service;

import com.baomidou.mybatisplus.core.conditions.query.QueryWrapper;
import com.hande.phospherus.security.entity.Users;
import com.hande.phospherus.security.mapper.UserMapper;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.authority.AuthorityUtils;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

/**
 * @ClassName UserDetailServiceImpl.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:11
 */
@Service("userDetailsService")
public class MyUserDetailServiceImpl implements UserDetailsService {

    @Autowired
    private UserMapper userMapper;

  /*  @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        //权限集合
        List<GrantedAuthority> authorities = AuthorityUtils.commaSeparatedStringToAuthorityList("role");
        //返回用户
        return new User("fang", new BCryptPasswordEncoder().encode("111111"), authorities);
    }*/

    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        //调用UserMapper中的方法查询数据库
        QueryWrapper<Users> wrapper = new QueryWrapper<>();
        wrapper.eq("username", username);
        Users user = userMapper.selectOne(wrapper);
        //数据库没有用户
        if (user == null) {
            throw new UsernameNotFoundException("用户不存在！");
        }
        List<GrantedAuthority> authorities = new ArrayList<>();
        if (user.getUsername().equals("fang")) {
            //权限集合
            authorities = AuthorityUtils.commaSeparatedStringToAuthorityList("admins,ROLE_sale");
        } else {
            authorities = AuthorityUtils.commaSeparatedStringToAuthorityList("role");
        }
        //从数据库查询对象中得到用户名和密码，返回
        return new User(user.getUsername(), new BCryptPasswordEncoder().encode(user.getPassword()), authorities);
    }
}
```

▼ hasAnyRole() — 如果当前的主体具有指定角色中的一个，则返回true，否则返回false

```
package com.hande.phospherus.security.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * @ClassName SecurityConfiguration.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:34
 */
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(password());
    }

    @Bean
    PasswordEncoder password() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //自定义自己的登录页面
        http.formLogin()
                //登录页面设置
                .loginPage("/login.html")
                //登录访问路径，登录的时候跳到哪一个controller中，这个过程Spring Cloud Security已经帮我们做到了
                .loginProcessingUrl("/user/login")
                //登录成功之后跳转的路径
                .defaultSuccessUrl("/test/index").permitAll()
                //定义哪些方法需要保护，哪些不需要保护
                .and().authorizeRequests()
                //设置哪些路径可以直接访问，不需要认证
                .antMatchers("/", "/test/hello", "/user/login").permitAll()
                //满足多个角色中的一个允许访问
                .antMatchers("/test/index").hasAnyRole("sale","product")
                //所有请求都可以访问
                .anyRequest().authenticated()
                //关闭csrf的防护
                .and().csrf().disable();
    }
}
```

配置没有权限访问跳转自定义页面

▼ 配置类配置

```
package com.hande.phospherus.security.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * @ClassName SecurityConfiguration.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:34
 */
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(password());
    }

    @Bean
    PasswordEncoder password() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //配置没有权限访问跳转自定义页面
        http.exceptionHandling().accessDeniedPage("/unauth.html");
        //自定义自己的登录页面
        http.formLogin()
                //登录页面设置
                .loginPage("/login.html")
                //登录访问路径，登录的时候跳到哪一个controller中，这个过程Spring Cloud Security已经帮我们做到了
                .loginProcessingUrl("/user/login")
                //登录成功之后跳转的路径
                .defaultSuccessUrl("/test/index").permitAll()
                //定义哪些方法需要保护，哪些不需要保护
                .and().authorizeRequests()
                //设置哪些路径可以直接访问，不需要认证
                .antMatchers("/", "/test/hello", "/user/login").permitAll()
                ////当前的登录用户只有具有admin权限才能够访问
                //.antMatchers("/test/index").hasAuthority("admins")
                //.antMatchers("/test/any").hasAnyAuthority("admins", "role")
                //满足一个角色允许访问
                .antMatchers("/test/index").hasRole("sale")
                //满足多个角色中的一个允许访问
                .antMatchers("/test/index").hasAnyRole("sale", "product")
                //所有请求都可以访问
                .anyRequest().authenticated()
                //关闭csrf的防护
                .and().csrf().disable();
    }
}
```

注解使用 — 使用注解之前先要在启动类/配置类开启注解功能 `@EnableGlobalMethodSecurity()`

- `@Secured` — 判断是否具有角色，另外需要注意的是这里匹配的字符串需要添加前缀 `“ROLE_”` ，开启注解 `@EnableGlobalMethodSecurity(securedEnabled = true)`

  在具体方法上使用注解

```
@Secured({"ROLE_sec", "ROLE_manager"})
    @GetMapping("secured")
    public String secured() {
        return "hello secured";
    }
```

```
authorities = AuthorityUtils.commaSeparatedStringToAuthorityList("admins,ROLE_sec,ROLE_pre");
```

- `@PreAuthorize` 适合进入方法前的权限验证 可以将登录用户的 `roles/permissions` 参数传到方法中 ，开启注解 `@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true)`

  他的写法是结合四种授权方法写的，如下

```
@PreAuthorize("hasRole('ROLE_pre')")
    @GetMapping("pre")
    public String pre() {
        return "hello secured";
    }
```

```
authorities = AuthorityUtils.commaSeparatedStringToAuthorityList("admins,ROLE_sec,ROLE_pre");
```

- `@PostAuthorize()` 在方法执行之后进行权限验证，适合验证带有返回值的权限。开启注解 `@EnableGlobalMethodSecurity(securedEnabled = true, prePostEnabled = true)` 和上面同样的注解开启方式

```
@PostAuthorize("hasRole('ROLE_pos')")
    @GetMapping("pos")
    public String pos() {
        System.out.println("PostAuthorize 已经执行了方法，会在方法执行之后进行校验");
        return "hello PostAuthorize";
    }
```

```
authorities = AuthorityUtils.commaSeparatedStringToAuthorityList("admins,ROLE_sec,ROLE_pre,ROLE_pos");
```

- `@PostFilter` — 权限验证之后对返回的数据进行过滤，留下限定满足条件的数据

```
//返回用户名为admin1的数据 其他的不返回
    @PostFilter(value = "filterObject.username == 'admin1'")
    @GetMapping("postFilter")
    public List<Users> postFilter() {
        return new ArrayList<>(Arrays.asList(
                new Users(1L, "admin1", "dadafd"),
                new Users(2L, "admin2", "jjgjgg")
        ));
    }
```

▼ `@PreFilter` — 对传入的参数做过滤 仅仅可以对集合类型的参数或返回值进行过滤，filterObject是使用@PreFilter和@PostFilter时的一个内置表达式，表示集合中的当前对象，@PreFilter标注的方法拥有多个集合类型的参数时，需要通过@PreFilter的filterTarget属性指定当前@PreFilter是针对哪个参数进行过滤的

```
//如果id能对2进行整除，进行后续操作，否则不能进行
    @PreFilter(value = "filterObject.id%2 == 0")
    @GetMapping("preFilter")
    public List<Users> preFilter(@RequestBody List<Users> list) {
        list.forEach(System.out::println);
        return list;
    }
```

注销

▼ 在配置类中添加退出映射地址

```
package com.hande.phospherus.security.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;

/**
 * @ClassName SecurityConfiguration.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:34
 */
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private UserDetailsService userDetailsService;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(password());
    }

    @Bean
    PasswordEncoder password() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //添加退出映射地址
        http.logout().logoutUrl("/logout").logoutSuccessUrl("/test/hello").permitAll();
        //配置没有权限访问跳转自定义页面
        http.exceptionHandling().accessDeniedPage("/unauth.html");
        //自定义自己的登录页面
        http.formLogin()
                //登录页面设置
                .loginPage("/login.html")
                //登录访问路径，登录的时候跳到哪一个controller中，这个过程Spring Cloud Security已经帮我们做到了
                .loginProcessingUrl("/user/login")
                //登录成功之后跳转的路径
                .defaultSuccessUrl("/success.html").permitAll()
                //定义哪些方法需要保护，哪些不需要保护
                .and().authorizeRequests()
                //设置哪些路径可以直接访问，不需要认证
                .antMatchers("/", "/test/hello", "/user/login", "/test/preFilter", "/test/postFilter").permitAll()
                ////当前的登录用户只有具有admin权限才能够访问
                //.antMatchers("/test/index").hasAuthority("admins")
                //.antMatchers("/test/any").hasAnyAuthority("admins", "role")
                //满足一个角色允许访问
                .antMatchers("/test/index").hasRole("sale")
                //满足多个角色中的一个允许访问
                .antMatchers("/test/index").hasAnyRole("sale", "product")
                //所有请求都可以访问
                .anyRequest().authenticated()
                //关闭csrf的防护
                .and().csrf().disable();
    }
}
```

▼ 编写带有退出接口的页面

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        <h1>登陆成功</h1>
        <a href="/logout">退出</a>
    </body>
</html>
```

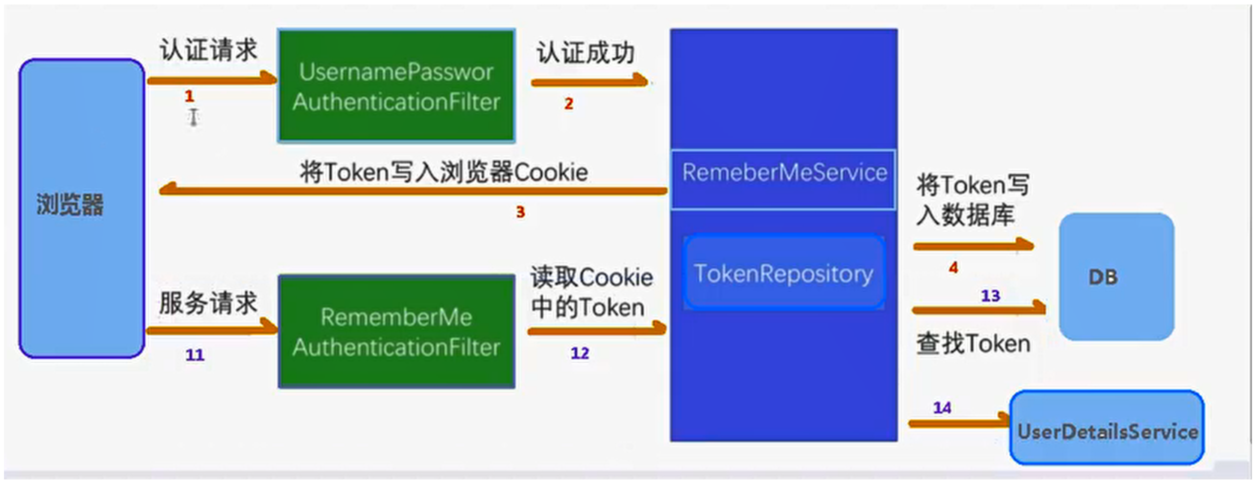
基于数据库实现自动登录

1. cookie 客户端技术-缺点在于内容需要存储到浏览器中
2. 安全框架机制实现自动登录

实现原理

主要通过两部分，一部分是cookie，一部分是数据库

1. 在认证成功之后，向浏览器中存储cookie的加密串，向数据库中存储cookie加密串和用户信息字符串
2. 再次访问时，首先获取cookie信息，到数据库中进行不对，如果能够查询到对应的信息，认证成功，可以登录



源码调用过程

▼ `AbstractAuthenticationProcessingFilter.class` 通过 `private RememberMeServices rememberMeServices = new NullRememberMeServices();` 实例实现

▼ `RememberMeServices.class` `void loginSuccess(HttpServletRequest request, HttpServletResponse response,Authentication successfulAuthentication);` 方法实现

▼ `AbstractRememberMeServices.class` `final void loginSuccess(HttpServletRequest request, HttpServletResponse response, Authentication successfulAuthentication)`

▼ `onLoginSuccess(request, response, successfulAuthentication);` 有两个重写的方法

`PersistentTokenBasedRememberMeServices.class` `TokenRepository` 生成token 同时添加到数据库中,添加数据库需要 `JdbcTokenRepositoryImpl.class → createNewToken/updateToken`

```
@Override
protected void onLoginSuccess(HttpServletRequest request, HttpServletResponse response,
      Authentication successfulAuthentication) {
   String username = successfulAuthentication.getName();
   this.logger.debug(LogMessage.format("Creating new persistent login for user %s", username));
   PersistentRememberMeToken persistentToken = new PersistentRememberMeToken(username, generateSeriesData(),
         generateTokenData(), new Date());
   try {
      this.tokenRepository.createNewToken(persistentToken);
      addCookie(persistentToken, request, response);
   }
   catch (Exception ex) {
      this.logger.error("Failed to save persistent token ", ex);
   }
}
```

`RememberMeAuthenticationFilter.class` `doFilter` `Authentication rememberMeAuth = this.rememberMeServices.autoLogin(request, response);`

`AbstractRememberMeServices.class` `this.userDetailsChecker.check(user);`

```
@Override
  public final Authentication autoLogin(HttpServletRequest request, HttpServletResponse response) {
    String rememberMeCookie = extractRememberMeCookie(request);
    if (rememberMeCookie == null) {
      return null;
    }
    this.logger.debug("Remember-me cookie detected");
    if (rememberMeCookie.length() == 0) {
      this.logger.debug("Cookie was empty");
      cancelCookie(request, response);
      return null;
    }
    try {
      String[] cookieTokens = decodeCookie(rememberMeCookie);
      UserDetails user = processAutoLoginCookie(cookieTokens, request, response);
      this.userDetailsChecker.check(user);
      this.logger.debug("Remember-me cookie accepted");
      return createSuccessfulAuthentication(request, user);
    }
    catch (CookieTheftException ex) {
      cancelCookie(request, response);
      throw ex;
    }
    catch (UsernameNotFoundException ex) {
      this.logger.debug("Remember-me login was valid but corresponding user not found.", ex);
    }
    catch (InvalidCookieException ex) {
      this.logger.debug("Invalid remember-me cookie: " + ex.getMessage());
    }
    catch (AccountStatusException ex) {
      this.logger.debug("Invalid UserDetails: " + ex.getMessage());
    }
    catch (RememberMeAuthenticationException ex) {
      this.logger.debug(ex.getMessage());
    }
    cancelCookie(request, response);
    return null;
  }
```

`AccountStatusUserDetailsChecker.class`

具体实现

▼ 建表，`JdbcTokenRepositoryImpl` 中自带有建表语句

```
CREATE TABLE persistent_logins (username VARCHAR ( 64 ) NOT NULL,series VARCHAR ( 64 ) PRIMARY KEY,token VARCHAR ( 64 ) NOT NULL last_u
```

▼ 修改配置类 — 注入数据源，配置操作数据库对象 — 注入数据源 配置对象 配置自动登录 设定有效时长

```
package com.hande.phospherus.security.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.rememberme.JdbcTokenRepositoryImpl;
import org.springframework.security.web.authentication.rememberme.PersistentTokenRepository;

import javax.sql.DataSource;

/**
 * @ClassName SecurityConfiguration.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:34
 */
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService;

    //注入数据源
    @Autowired
    private DataSource dataSource;

    //配置对象
    @Bean
    public PersistentTokenRepository persistentTokenRepository() {
        JdbcTokenRepositoryImpl jdbcTokenRepository = new JdbcTokenRepositoryImpl();
        jdbcTokenRepository.setDataSource(dataSource);
        //自动创建表
        //jdbcTokenRepository.setCreateTableOnStartup(true);
        return jdbcTokenRepository;
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(password());
    }

    @Bean
    PasswordEncoder password() {
        return new BCryptPasswordEncoder();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //添加退出映射地址
        http.logout().logoutUrl("/logout").logoutSuccessUrl("/test/hello").permitAll();
        //配置没有权限访问跳转自定义页面
        http.exceptionHandling().accessDeniedPage("/unauth.html");
        //自定义自己的登录页面
        http.formLogin()
                //登录页面设置
                .loginPage("/login.html")
                //登录访问路径，登录的时候跳到哪一个controller中，这个过程Spring Cloud Security已经帮我们做到了
                .loginProcessingUrl("/user/login")
                //登录成功之后跳转的路径
                .defaultSuccessUrl("/success.html").permitAll()
                //定义哪些方法需要保护，哪些不需要保护
                .and().authorizeRequests()
                //设置哪些路径可以直接访问，不需要认证
                .antMatchers("/", "/test/hello", "/user/login", "/test/preFilter", "/test/postFilter").permitAll()
                ////当前的登录用户只有具有admin权限才能够访问
                //.antMatchers("/test/index").hasAuthority("admins")
                //.antMatchers("/test/any").hasAnyAuthority("admins", "role")
                //满足一个角色允许访问
                .antMatchers("/test/index").hasRole("sale")
                //满足多个角色中的一个允许访问
                .antMatchers("/test/index").hasAnyRole("sale", "product")
                //所有请求都可以访问
                .anyRequest().authenticated()
                //配置自动登录
                .and().rememberMe().tokenRepository(persistentTokenRepository())
                //设置token有效时常，秒为单位
                .tokenValiditySeconds(600)
                //关闭csrf的防护
                .and().csrf().disable();
    }
}
```

▼ 在登录页面中添加复选 — 必须为 `remember-me`

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>Login</title>
    </head>
    <body>
        <form action="/user/login" method="post">
            用户名：<input type="text" name="username">
            密码：<input type="text" name="password">
            <br>
            <input type="submit" value="login">
            <input type="checkbox" name="remember-me">自动登录
        </form>

    </body>
</html>
```

CSRF理解

跨站请求伪造（Cross-site request forgery），也被称为one-click attack或者session riding，通常缩写为CSRF或者XSRF，是一种挟持用户在当前已登录的Web应用程序上执行非本意的操作的攻击方法，跟跨网站脚本（XSS）相比，XSS利用的是用户对指定网站的信任，CSRF利用的是网站对用户网页浏览器的信任

跨站请求攻击，简单地说，是攻击者通过一些技术手段欺骗用户的浏览器去访问一个自己曾经认证过的网站并运行一些操作(如发邮件，发消息，甚至财产操作如转账和购买商品)。由于浏览器曾经认证过，所以被访问的网站会认为是真正的用户操作而去运行。这利用了web中用户身份验证的一个漏洞：简单的身份验证只能保证请求发自某个用户的浏览器，却不能保证请求本身是用户自愿发出的。

从Spring Security 4.0开始，默认情况下会启用CSRF保护，以防止CSRF攻击应用程序，Spring Security CSRF会针对PATCH，POST， PUT和DELETE方法进行防护。

▼ 用法

在配置类中开启CSRF防护，默认是开启，注释掉

```
package com.hande.phospherus.security.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.rememberme.JdbcTokenRepositoryImpl;
import org.springframework.security.web.authentication.rememberme.PersistentTokenRepository;

import javax.sql.DataSource;

/**
 * @ClassName SecurityConfiguration.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:34
 */
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {


    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //配置url的访问权限
        http.authorizeRequests()
                .antMatchers("/").permitAll()
                .antMatchers("/**update**").permitAll()
                .antMatchers("/login/**").permitAll()
                .anyRequest().authenticated();

        //关闭csrf保护功能
        //http.csrf().disable();

        http.formLogin()
                //登录页面设置
                .loginPage("/userLogin").permitAll()
                .usernameParameter("username").passwordParameter("password")
                //登录成功之后跳转的路径
                .defaultSuccessUrl("/")
                .failureUrl("/userLogin?error");
    }


}
```

在页面中添加隐藏项

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta charset="UTF-8">
        <title>用户修改</title>
    </head>
    <body>
        <div align="center">
            <form action="update_token" method="post">
                <input type="hidden" th:name="${_csrf.parameterName}" th:value="_csrf.token">
                用户名：<input type="text" name="username"><br>
                密码：<input type="password" name="password"><br>
                <button type="submit">修改</button>
            </form>
        </div>
    </body>
</html>
```

▼ Spring Cloud Security实现 CSRF 原理分析

生成 csrfToken 保存到 HttpSession 或者 Cookie 中，每次请求都带着这个值去请求

`org.springframework.security.web.csrf.CsrfFilter doFilterInternal`

```
@Override
  protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
      throws ServletException, IOException {
    request.setAttribute(HttpServletResponse.class.getName(), response);
    CsrfToken csrfToken = this.tokenRepository.loadToken(request); //生成Token存储，到Session或者Cookie中
    boolean missingToken = (csrfToken == null);
    if (missingToken) {
      csrfToken = this.tokenRepository.generateToken(request);
      this.tokenRepository.saveToken(csrfToken, request, response);
    }
    request.setAttribute(CsrfToken.class.getName(), csrfToken);
    request.setAttribute(csrfToken.getParameterName(), csrfToken);//拿到表单传过来的去比对
    if (!this.requireCsrfProtectionMatcher.matches(request)) {
      if (this.logger.isTraceEnabled()) {
        this.logger.trace("Did not protect against CSRF since request did not match "
            + this.requireCsrfProtectionMatcher);
      }
      filterChain.doFilter(request, response);
      return;
    }
    String actualToken = request.getHeader(csrfToken.getHeaderName());
    if (actualToken == null) {
      actualToken = request.getParameter(csrfToken.getParameterName());
    }
    if (!csrfToken.getToken().equals(actualToken)) {
      this.logger.debug(
          LogMessage.of(() -> "Invalid CSRF token found for " + UrlUtils.buildFullRequestUrl(request)));
      AccessDeniedException exception = (!missingToken) ? new InvalidCsrfTokenException(csrfToken, actualToken)
          : new MissingCsrfTokenException(actualToken);
      this.accessDeniedHandler.handle(request, response, exception);
      return;
    }
    filterChain.doFilter(request, response);
  }
```

```
//在请求的时候，以下的不做防护
private static final class DefaultRequiresCsrfMatcher implements RequestMatcher {

    private final HashSet<String> allowedMethods = new HashSet<>(Arrays.asList("GET", "HEAD", "TRACE", "OPTIONS"));

    @Override
    public boolean matches(HttpServletRequest request) {
      return !this.allowedMethods.contains(request.getMethod());
    }

    @Override
    public String toString() {
      return "CsrfNotRequired " + this.allowedMethods;
    }

  }
```

- 测试
  - 配置类

```
package com.hande.phospherus.security.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.authentication.rememberme.JdbcTokenRepositoryImpl;
import org.springframework.security.web.authentication.rememberme.PersistentTokenRepository;

import javax.sql.DataSource;

/**
 * @ClassName SecurityConfiguration.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:34
 */
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired
    private UserDetailsService userDetailsService;

    //注入数据源
    @Autowired
    private DataSource dataSource;

    //配置对象
    @Bean
    public PersistentTokenRepository persistentTokenRepository() {
        JdbcTokenRepositoryImpl jdbcTokenRepository = new JdbcTokenRepositoryImpl();
        jdbcTokenRepository.setDataSource(dataSource);
        //自动创建表
        //jdbcTokenRepository.setCreateTableOnStartup(true);
        return jdbcTokenRepository;
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(userDetailsService).passwordEncoder(password());
    }

    @Bean
    PasswordEncoder password() {
        return new BCryptPasswordEncoder();
    }

  @Override
    protected void configure(HttpSecurity http) throws Exception {
        //配置url的访问权限
        http.authorizeRequests()
                .antMatchers("/").permitAll()
                .antMatchers("/**update**").permitAll()
                .antMatchers("/login/**").permitAll()
                .anyRequest().authenticated();

        //关闭csrf保护功能
        http.csrf().disable();

        http.formLogin()
                //登录页面设置
                .loginPage("/login.html")
                .usernameParameter("username").passwordParameter("password")
                //登录成功之后跳转的路径
                .defaultSuccessUrl("/")
                .failureUrl("/userLogin?error");
    }
```

▼ 两个Controller

```
package com.hande.phospherus.security.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

/**
 * @ClassName CSRFController.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 22:27
 */
@Controller
public class CSRFController {
    @GetMapping("/update")
    public String test(Model model) {
        return "csrf/csrfTest";
    }

    @PostMapping("/update_token")
    public String getToken() {
        return "csrf/csrf_token";
    }

}
```

```
package com.hande.phospherus.security.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PostMapping;

/**
 * @ClassName CSRFController.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 22:27
 */
@Controller
public class LoginController {
    @GetMapping("/userLogin")
    public String login() {
        return "login/login";
    }


}
```

▼ service

```
package com.hande.phospherus.security.service;

import org.springframework.security.core.authority.SimpleGrantedAuthority;
import org.springframework.security.core.userdetails.User;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
import java.util.List;

/**
 * @ClassName UserDetailServiceImpl.java
 * @author yangyangSheep
 * @Description
 * @createTime 2021年01月31日 17:11
 */
@Service
public class UserDetailServiceImpl implements UserDetailsService {


    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

        List<SimpleGrantedAuthority> authorities = new ArrayList<>();
        authorities.add(new SimpleGrantedAuthority("role"));
        UserDetails userDetails = new User("fang", new BCryptPasswordEncoder().encode("123"), authorities);
        //从数据库查询对象中得到用户名和密码，返回
        return userDetails;
    }
}
```

▼ 模板

src\main\resources\templates\csrf\csrf_token.html

csrf_token.html

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta charset="UTF-8">
        <title>Title</title>
    </head>
    <body>
        <span th:text="${_csrf.token}"></span>
    </body>
</html>
```

csrfTest.html

```
<!DOCTYPE html>
<html lang="en" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org">
    <head>
        <meta charset="UTF-8">
        <title>用户修改</title>
    </head>
    <body>
        <div align="center">
            <form action="/update_token" method="post">
                <input type="hidden" th:name="${_csrf.parameterName}" th:value="_csrf.token">
                用户名：<input type="text" name="username">
                密码：<input type="password" name="password">
                <br>
                <button type="submit">修改</button>
            </form>
        </div>
    </body>
</html>
```

微服务权限方案

微服务权限方案-github：security-parent

集成OAuth2.0

-
-