前置知识

1、掌握Spring框架

2、掌握SpringBoot使用

3、掌握JavaWEB技术

1. SpringSecurity 框架简介
2. SpringSecurity 入门案例
3. SpringSecurity Web 权限方案
4. SpringSecurity微服务权限方案
5. SpringSecurity 原理总结

基本原理

SpringSecurity 本质是一个过滤器链

有很多过滤器

通过查看源码

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

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

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

过滤器如何进行加载的？

1、使用SpringSecurity配置过滤器
* DelegatingFilterProxy

```
public void doFilter(ServletReq
    delegateToUse = this.initDelegate(wac);
```

```
protected Filter initDelegate(WebApplicationContext wac) throws ServletExcept
    String targetBeanName = this.getTargetBeanName();
    Assert.state( expression: targetBeanName != null,  message: "No target bea
    Filter delegate = (Filter)wac.getBean(targetBeanName, Filter.class);
    if (this.isTargetFilterLifecycle()) {
```

```
s FilterChainProxy extend
List<Filter> filters = this.getFilters((H
```

两个重要的接口

UserDetailsService接口 ：查询数据库用户名和密码过程

* 创建类继承UsernamePasswordAuthenticationFilter，重写三个方法
* 创建类实现UserDetailService，编写查询数据过程，返回User对象，这个User对象是安全框架提供对象

PasswordEncoder

数据加密接口，用于返回User对象里面密码加密

Spring Security

入门案例

第一步 创建springboot工程

第二步 引入相关依赖

第三步 编写controller进行测试

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.1.RELEASE</version>
    <relativePath/> <!-- lookup parent from repository -->
</parent>

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

```
@RestController
@RequestMapping("/test")
public class TestController {

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

默认用户名：user

密码在启动控制台找到

```
enerated security password: 99ac380d-ef6d-4007-853c-62e8917c4600
```

web权限方案

(1) 认证
(2) 授权

1、设置登录的用户名和密码

第一种方式：通过配置文件

第二种方式：通过配置类

第三种方式：自定义编写实现类

配置他之后，默认账号密码就没有了

* 通过配置文件设置

application.properties

```
server.port=8111
spring.security.user.name=atguigu
spring.security.user.password=atguigu
```

* 通过配置类

```
@Configuration
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        BCryptPasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        String password = passwordEncoder.encode( rawPassword: "123");
        auth.inMemoryAuthentication().withUser( username: "lucy").password(password).roles("admin");
    }

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

* 自定义实现类设置

第一步 创建配置类，设置使用哪个userDetailsService实现类

第二步 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("userDetailsService")
public class MyUserDetailsService implements UserDetailsService {
    @Override
    public UserDetails loadUserByUsername(String s) throws UsernameNotFoundE
        List<GrantedAuthority> auths =
                AuthorityUtils.commaSeparatedStringToAuthorityList( authority
        return new User( username: "mary",
                new BCryptPasswordEncoder().encode( rawPassword: "123"), auth
    }
}
```

roles

用户名、密码、操作权限

```
@Configuration
public class SecurityConfigTest 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(); }
}
```

使用这个进行密码加解密

使用这个获取账号密码权限

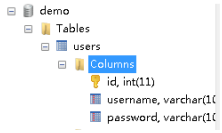
查询数据库完成用户认证

* 整合MyBatisPlus完成数据库操作

第一步 引入相关依赖

```
<!--mybatis-plus-->
<dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.0.5</version>
</dependency>
<!--mysql-->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
```

第二步 创建数据库和数据库表



第三步 创建users表对应实体类

```
@Data
public class Users {
    private Integer id;
    private String username;
    private String password;
}
```

第四步 整合mp，创建接口，继承mp的接口

```
public interface UsersMapper extends BaseMapper<Users> {
}
```

第五步 在MyUserDetailsService调用mapper里面的方法查询数据库进行用户认证

```
//调用usersMapper方法，根据用户名查询数据库
QueryWrapper<Users> wrapper = new QueryWrapper();
// where username=?
wrapper.eq( column: "username",username);
Users users = usersMapper.selectOne(wrapper);
//判断
if(users == null) {//数据库没有用户名，认证失败
    throw  new UsernameNotFoundException("用户名不存在！");
}
List<GrantedAuthority> auths =
        AuthorityUtils.commaSeparatedStringToAuthorityList( authority
//从查询数据库返回users对象，得到用户名和密码，返回
return new User(users.getUsername(),
        new BCryptPasswordEncoder().encode(users.getPassword()), auth
```

new出这个值，说明认证成功了

第六步 在启动类添加注解 MapperScan

```
@SpringBootApplication
@MapperScan("com.atguigu.securitydemo1.mapper")
public class SecuritydemolApplication {

    public static void main(String[] args) { SpringApp
```

第七步 配置数据库信息

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/demo?serverTimezone=GMT%2B8
spring.datasource.username=root
spring.datasource.password=root
```

springBoot2.2开始默认支持mysql5.8

自定义设置登录页面
不需要认证可以访问

1、在配置类实现相关的配置

```
protected void configure(HttpSecurity http) throws Exception {
    http.formLogin()    //自定义自己编写的登录页面
        .loginPage("/login.html")  //登录页面设置
        .loginProcessingUrl("/user/login")     //登录访问路径
        .defaultSuccessUrl("/test/index").permitAll()  //登录成功之后，跳转路径
        .and().authorizeRequests()
            .antMatchers( ...antPatterns: "/","/test/hello","/user/login").permitAll()
        .anyRequest().authenticated()
        .and().csrf().disable();  //关闭csrf防护
}
```

//点击登入时，提交数据的url，具体controller不用我们写

//允许访问

//设置哪些路径可以直接访问，不需要验证

2、创建相关页面，controller

```
<input type="text" name="username"/>
<input type="text" name="password"/>
```

哪些需要被保护，哪些不需要被保护

1.1 基于角色或权限进行访问控制

第一个方法：hasAuthority 方法

如果当前的主体具有指定的权限，则返回 true,否则返回false

没有访问权限 403

(type=Forbidden, status=403).

1、在配置类设置当前访问地址有哪些权限

```
//当前登录用户，只有具有admins权限才可以访问这个路径
.antMatchers( ...antPatterns: "/test/index").hasAuthority("admins")
```

2、在UserDetailsService，把返回User对象设置权限

```
antedAuthority> auths =
 AuthorityUtils.commaSeparatedStringToAuthorityList( authorityString: "admins");
```

第二个方法：hasAnyAuthority方法

如果当前的主体有任何提供的角色（给定的作为一个逗号分隔的字符串列表）的话，返回true.

```
//hasAnyAuthority
.antMatchers( ...antPatterns: "/test/index").hasAnyAuthority( ...authorities: "admins,manager")

 rantedAuthority> auths =
   AuthorityUtils.commaSeparatedStringToAuthorityList( authorityString: "admins");
```

第三个方法：hasRole 方法

如果当前主体具有指定的角色，则返回true

```
return "hasRole('ROLE_" + role + "')";
```

```
//3 hasRole方法   ROLE_sale
.antMatchers( ...antPatterns: "/test/index").hasRole("sale")

AuthorityList( authorityString: "admins,ROLE_sale");
', 返回
```

第四个方法：hasAnyRole

表示用户具备任何一个条件都可以访问

```
tyList( authorityString: "admin,role,ROLE_admin,ROLE_role"));
```

自定义403没有权限访问页面

在配置类进行配置就可以了

```
@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");
```

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Sep 27 16:32:26 CST 2020
There was an unexpected error (type=Forbidden, status=403).
Forbidden

没有访问权限！

尚硅谷

认证授权注解使用

第一 @Secured

* 用户具有某个角色，可以访问方法

1、启动类（配置类）开启注解

```
@EnableGlobalMethodSecurity(securedEnabled=true)
public class SecuritydemolApplication {
```

2、在controller的方法上面使用注解，设置角色

```
@GetMapping("update")
@Secured({"ROLE_sale","ROLE_manager"})
public String update() {
```

3、userDetailsService设置用户角色

```
...orityList( authorityString: "admins,ROLE_sale");
```

第二 @PreAuthorize

方法之前进行校验

1、启动类开启注解

```
@EnableGlobalMethodSecurity(securedEnabled=true, prePostEnabled = true)
public class SecuritydemolApplication {
```

2、在controller的方法上面添加注解

```
@GetMapping("update")
//@Secured({"ROLE_sale","ROLE_manager"})
@PreAuthorize("hasAnyAuthority('admins')")
public String update() {
```

第三 @PostAuthorize

方法执行之后校验

```
@EnableGlobalMethodSecurity(securedEnabled=true, prePostEnabled = true)
public class SecuritydemolApplication {

@PostAuthorize("hasAnyAuthority('admins')")
public String update() {
```

最后两个注解

@PostFilter 方法返回数据进行过滤 → 返回值过滤，符合条件的才返回

@PreFilter 传入方法数据进行过滤 → 入参必须是list、map、数组这些，符合条件的才传入方法体中

尚硅谷

用户注销

1、在配置类添加退出的配置

```
//退出
http.logout().logoutUrl("/logout").
        logoutSuccessUrl("/test/hello").permitAll();
```

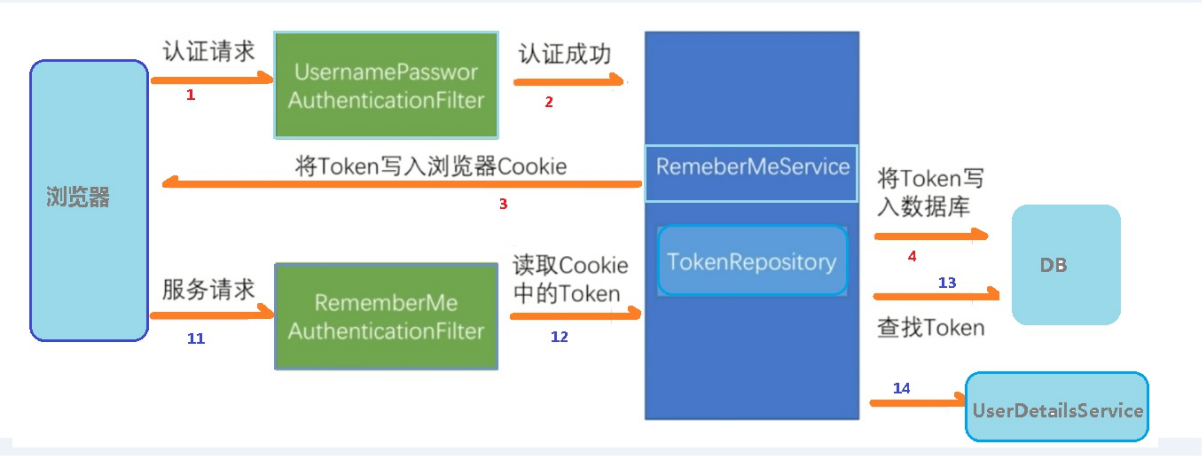
测试

1、修改配置类，登录成功之后跳转到成功页面

2、在成功页面添加超链接，写设置退出路径

```
<a href="/logout">退出</a>
```

3、登录成功之后，在成功页面点击退出
再去访问其他controller不能进行访问的



自动登录

1、cookie技术

2、安全框架机制实现自动登录

一、实现原理

第一步：浏览器 —cookie→ 认证成功 —加密串→ 数据库

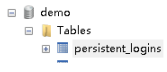
加密串 用户信息字符串

第二步：再次访问

获取cookie信息，拿着cookie信息到数据库进行比对，如果查询到对应信息，认证成功，可以登录

二、具体实现

第一步 创建数据库表



第二步 配置类，注入数据源，配置操作数据库对象

```
@Autowired
private DataSource dataSource; //注入数据源
//配置对象
@Bean
public PersistentTokenRepository persistentTokenRepository() {
    JdbcTokenRepositoryImpl jdbcTokenRepository = new JdbcTokenRepositoryImpl();
    jdbcTokenRepository.setDataSource(dataSource);
    //jdbcTokenRepository.setCreateTableOnStartup(true);
    return jdbcTokenRepository;
}
```

第三步 配置类配置自动登录

```
.and().rememberMe().tokenRepository(persistentTokenRepository())
.tokenValiditySeconds(60)//设置有效时长，单位秒
.userDetailsService(userDetailsService)
```

第五步 在登录页面添加复选框

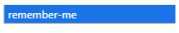
```
<br/>
<input type="checkbox" name="remember-me"/>自动登录
```

必须叫remember-me

登入成功后会返回前端cookie

name:remember-me

value:xxxxxx

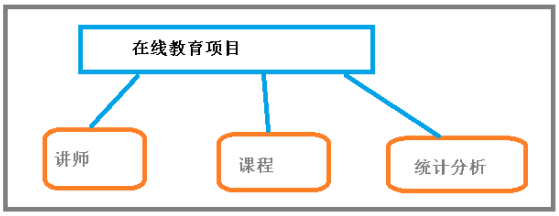


SpringSecurity 微服务权限方案

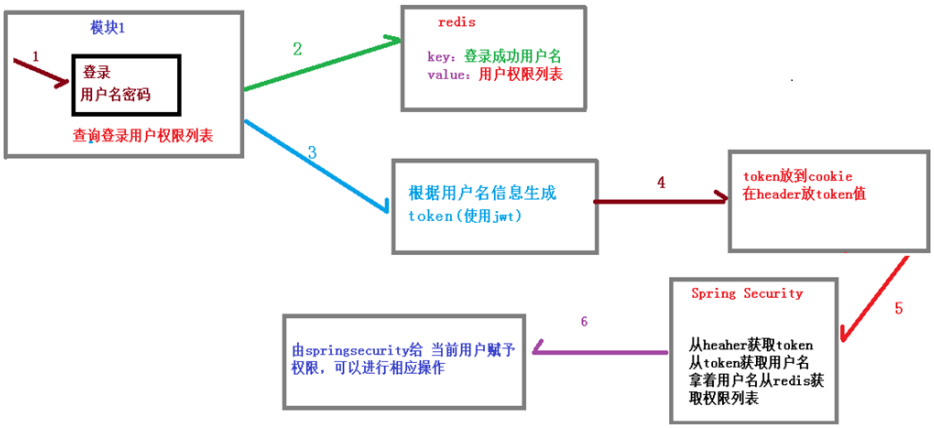
1、什么微服务

2、微服务认证和授权实现过程

3、完成基于SpringSecurity认证授权案例



单点登录　　授权



微服务权限管理案例主要功能：

1、登录（认证）

2、添加角色

3、为角色分配菜单

4、添加用户

5、为用户分配角色

尚硅谷

1、权限管理数据模型

2、案例涉及技术说明

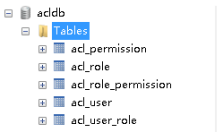
* 添加角色
* 为角色分配菜单
* 添加用户
* 为用户分配角色

角色菜单关系表

cid	rid
1	11
2	11

用户角色关系表

uid	rid
21	11
21	12



1、Maven　　创建父工程：管理项目依赖版本

创建子模块：使用具体依赖

2、SpringBoot　　本质是就是Spring

3、MyBatisPlus　　操作数据库框架

4、SpringCloud

（1）GateWay网关

（2）注册中心　　Nacos

其他技术：

Redis　　Jwt　　Swagger

前端技术

1、搭建项目工程

1、创建父工程 acl_parent　：　管理依赖版本

2、在父工程创建子模块

（1）common

* service_base ： 工具类
* spring_security ： 权限配置

（2）infrastructure

* api_gateway ： 网关

（3）service

* service_acl ：　权限管理微服务模块

在父工程pom文件，定义依赖版本

2、引入依赖

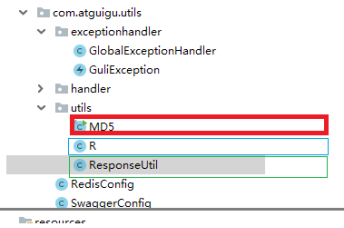
编写使用工具类，比如MD5加密等等

SpringSecurity相关配置 核心

配置gateway网关

实现权限管理功能代码

1、编写common里面需要工具类



字符串进行加密

定义所有controller返回统一结果数据返回

2、编写SpringSecurity认证授权工具类和处理器

2.1 密码处理工具类

```
@Component
public class DefaultPasswordEncoder implements PasswordEncoder {
    //进行MD5加密
    @Override
    public String encode(CharSequence charSequence) {
        return MD5.encrypt(charSequence.toString());
    }
    //进行密码比对
    @Override
    public boolean matches(CharSequence charSequence, String
    encodedPassword) {
        return encodedPassword.equals(MD5.encrypt(charSequence.toString
    ()));
    }
```

2.2 token操作工具类

使用jwt生成token

```
//1 使用jwt根据用户名生成token
public String createToken(String username) {
    String token = Jwts.builder().setSubject(username)
            .setExpiration(new Date(System.currentTimeMillis()+token
            .signWith(SignatureAlgorithm.HS512, tokenSignKey).compre
    return token;
}
//2 根据token字符串得到用户信息
public String getUserInfoFromToken(String token) {
    String userinfo = Jwts.parser().setSigningKey(tokenSignKey).pars
    return userinfo;
```

2.3 退出处理器

```
//退出处理器
public class TokenLogoutHandler implements LogoutHan
```

```
public void logout(HttpServletRequest request, HttpServletResponse respor
    //1 从header里面获取token
    //2 token不为空，移除token，从redis删除token
    String token = request.getHeader( s: "token");
    if(token != null) {
        //移除
        tokenManager.removeToken(token);
        //从token获取用户名
        String username = tokenManager.getUserInfoFromToken(token);
        redisTemplate.delete(username);
    }
    ResponseUtil.out(response, R.ok());
```

2.4 未授权统一处理类

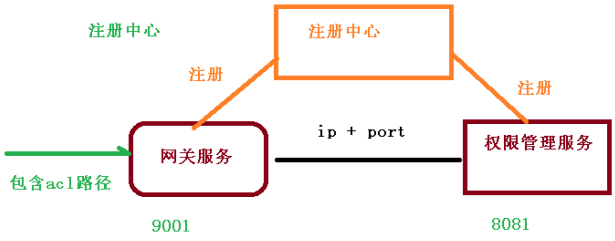
```
public class UnauthEntryPoint implements AuthenticationEntryPoint {
    @Override
    public void commence(HttpServletRequest httpServletRequest,
HttpServletResponse httpServletResponse, AuthenticationException e) throws
IOException, ServletException {
        ResponseUtil.out(httpServletResponse, R.error());
    }
}
```

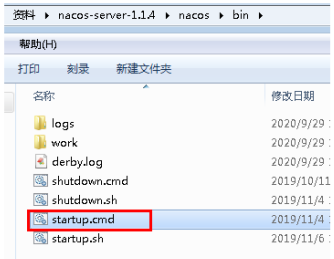
启动Redis和Nacos

1、启动Redis

```
[root@online bin]# ./redis-server /etc/redis.conf
[root@online bin]# ps -ef | grep redis
root      8758     1  0 16:32 ?        00:00:00 ./redis-server *:6379
root      8770  7751  0 16:33 pts/1    00:00:00 grep --color=auto redis
[root@online bin]# ./redis-cli
127.0.0.1:6379> keys *
1) "susan"
2) "lucytest"
127.0.0.1:6379>
```

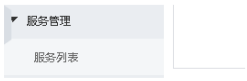
2、启动Nacos





访问地址： http://localhost:8848/nacos/

默认用户名密码：nacos



filter

TokenAuthenticationFilter 授权过滤

TokenLoginFilter 认证过滤器

编写自定义认证 和 授权的过滤器

1、认证的过滤器

```
class TokenLoginFilter extends UsernamePasswordAuthenticationFilter {
    //1 获取表单提交用户名和密码
    @Override
    public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)
            throws AuthenticationException {
        //获取表单提交数据
        try {
            User user = new ObjectMapper().readValue(request.getInputStream(), User.class);
            return authenticationManager.authenticate(new UsernamePasswordAuthenticationToken
(user.getUsername(),user.getPassword(),
                    new ArrayList<>()));
        } catch (IOException e) {
            e.printStackTrace();
            throw new RuntimeException();
        }
    }
```

```
    //2 认证成功调用的方法
    @Override
    protected void successfulAuthentication(HttpServletRequest request,
                                            HttpServletResponse response, FilterChain chain,
Authentication authResult)
            throws IOException, ServletException {
        //认证成功，得到认证成功之后用户信息
        SecurityUser user = (SecurityUser)authResult.getPrincipal();
        //根据用户名生成token
        String token = tokenManager.createToken(user.getCurrentUserInfo
().getUsername());
        //把用户名称和用户权限列表放到redis
        redisTemplate.opsForValue().set(user.getCurrentUserInfo().getUsername
(),user.getPermissionValueList());
        //返回token
        ResponseUtil.out(response, R.ok().data("token",token));
    }
```

```
    //3 认证失败调用的方法
    protected void unsuccessfulAuthentication(HttpServletRequest request, HttpServletResponse response,
AuthenticationException failed)
            throws IOException, ServletException {
        ResponseUtil.out(response, R.error());
    }
```

2、授权过滤器

```
blic class TokenAuthFilter extends BasicAuthenticationFilter {
    private UsernamePasswordAuthenticationToken getAuthentication(HttpServletRequest request) {
        //从header获取token
        String token = request.getHeader("token");
        if(token != null) {
            //从token获取用户名
            String username = tokenManager.getUserInfoFromToken(token);
            //从redis获取对应权限列表
            List<String> permissionValueList = (List<String>)redisTemplate.opsForValue().get(username);
            Collection<GrantedAuthority> authority = new ArrayList<>();
            for(String permissionValue : permissionValueList) {
                SimpleGrantedAuthority auth = new SimpleGrantedAuthority(permissionValue);
                authority.add(auth);
            }
            return new UsernamePasswordAuthenticationToken(username,token,authority);
        }
        return null;
    }
```

核心配置类

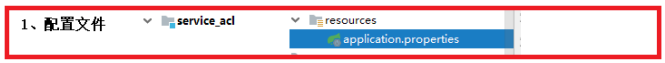
```
//设置退出的地址和token，redis操作地址
@Override
protected void configure(HttpSecurity http) throws Exception {
    http.exceptionHandling() ExceptionHandlingConfigurer<HttpSecurity>
            .authenticationEntryPoint(new UnauthEntryPoint())//没有权限访问
            .and().csrf().disable() HttpSecurity
            .authorizeRequests() ExpressionInterceptUrlRegistry
            .anyRequest().authenticated() ExpressionUrlAuthorizationConfigurer<HttpSecurity>.ExpressionIntercept
            .and().logout().logoutUrl("/admin/acl/index/logout")//退出路径
            .addLogoutHandler(new TokenLogoutHandler(tokenManager, redisTemplate)).and() HttpSecurity
            .addFilter(new TokenLoginFilter(authenticationManager(), tokenManager, redisTemplate)) HttpSecuri
            .addFilter(new TokenAuthFilter(authenticationManager(), tokenManager, redisTemplate)).httpBasic(
}
```

```
//调用userDetailsService和密码处理
@Override
public void configure(AuthenticationManagerBuilder auth) throws Exception {
    auth.userDetailsService(userDetailsService).passwordEncoder(defaultPasswordEncoder);
}
//不进行认证的路径，可以直接访问
@Override
public void configure(WebSecurity web) throws Exception {
    web.ignoring().antMatchers( ...antPatterns: "/api/**");
}
```

UserDetailsService

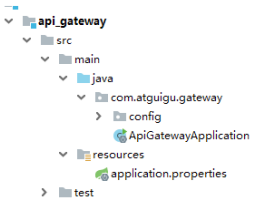
```java
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    //根据用户名查询数据
    User user = userService.selectByUsername(username);
    //判断
    if(user == null) {
        throw new UsernameNotFoundException("用户不存在");
    }
    com.atguigu.security.entity.User curUser = new com.atguigu.security.entity.User();
    BeanUtils.copyProperties(user,curUser);

    //根据用户查询用户权限列表
    List<String> permissionValueList = permissionService.selectPermissionValueByUserId(user.getId());
    SecurityUser securityUser = new SecurityUser();
    securityUser.setPermissionValueList(permissionValueList);
    return securityUser;
}
```


1、配置文件

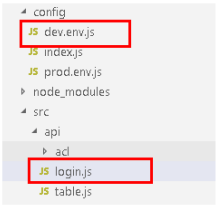


2、权限模块

3、整合网关



4、前端整合



1、认证流程

UsernamePasswordAuthenticationFilter ：

(1)查看过滤器的父类 AbstractAuthenticationProcessingFilter

第一步 过滤的方法，判断提交方式是否post提交

```
public void doFilter(ServletRequest req, ServletResponse res, FilterCha
    HttpServletRequest request = (HttpServletRequest)req;
    HttpServletResponse response = (HttpServletResponse)res;
    if (!this.requiresAuthentication(request, response)) {
        chain.doFilter(request, response);
    } else {
```

第二步 调用子类的方法进行身份认证，认证成功之后，把认证信息封装到对象里面

```
Authentication authResult;
try {
    authResult = this.attemptAuthentication(request, response)
    if (authResult == null) {
        return;
    }
```

第三步 session策略处理

```
this.sessionStrategy.onAuthentication(authResult, request, response);
```

第四步 1 认证失败抛出异常，执行认证失败的方法

```
} catch (InternalAuthenticationServiceException var8) {
    this.logger.error( o: "An internal error occurred while trying
    this.unsuccessfulAuthentication(request, response, var8);
    return;
} catch (AuthenticationException var9) {
```

第四步 2 认证成功，调用认证成功的方法

```
if (this.continueChainBeforeSuccessfulAuthentication) {
    chain.doFilter(request, response);
}

this.successfulAuthentication(request, response, chain, authResult);
```

(2) 上面第二步 调用子类的方法进行认证过程，查看源码

UsernamePasswordAuthenticationFilter 类

public Authentication attemptAuthentication(HttpServletReque 方法

方法第一步 判断是否post提交

```
if (this.postOnly && !request.getMethod().equals("POST")) {
    throw new AuthenticationServiceException("Authentication method not supported: "
```

方法第二步 获取表单提交数据

```
else {
    String username = this.obtainUsername(request);
    String password = this.obtainPassword(request);
    if (username == null) {
```

方法第三步 使用获取数据，构造成对象，标记未认证
把请求一些属性信息设置到对象里面
调用方法进行身份认证（调用userDetailsService）

```
UsernamePasswordAuthenticationToken authRequest = new UsernamePasswordAuthenticationToken(userna
this.setDetails(request, authRequest);
return this.getAuthenticationManager().authenticate(authRequest);
```

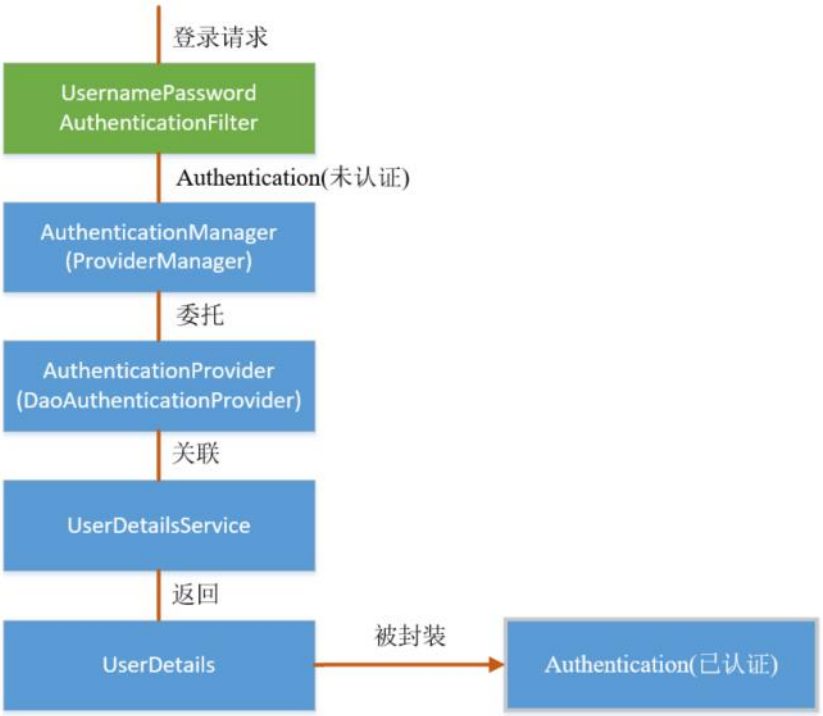
(3) 查看UsernamePasswordAuthenticationToken构建过程

```
blic interface Authentication extends Principal, S
    Collection<? extends GrantedAuthority> getAuthor

    Object getCredentials();

    Object getDetails();

    Object getPrincipal();

    boolean isAuthenticated();
```

```
ublic class UsernamePasswordAuthenticationToken extends Abstra
    private static final long serialVersionUID = 520L;
    private final Object principal;
    private Object credentials;

    未认证的方法
    public UsernamePasswordAuthenticationToken(Object principa
        super((Collection)null);
        this.principal = principal;
        this.credentials = credentials;
        this.setAuthenticated(false);
    }

    已经认证的方法
    public UsernamePasswordAuthenticationToken(Object principa
        super(authorities);
```

(4) 查看ProviderManager源码，认证实现

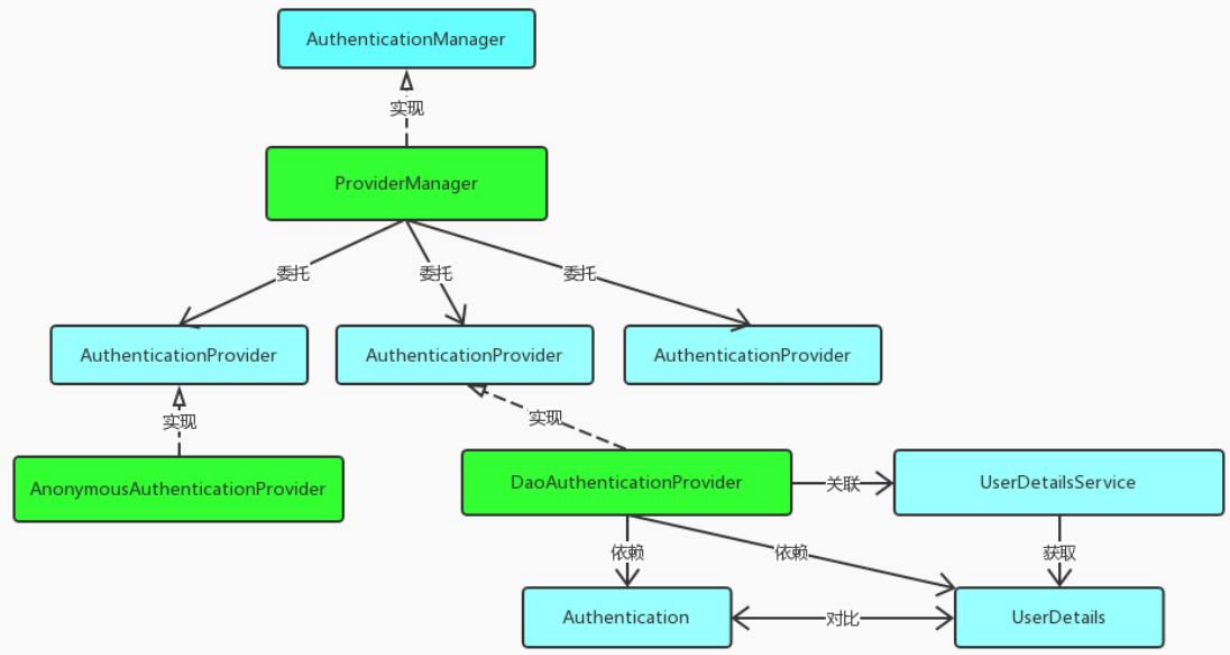
(5) 认证成功 和 认证失败的方法



权限访问流程　　ExceptionTranslationFilter 过滤器和 FilterSecurityInterceptor 过滤器

```
; ExceptionTranslationFilter ex
iblic void doFilter(ServletRequest req, ServletResponse res, FilterChain chain) th
  HttpServletRequest request = (HttpServletRequest)req;
  HttpServletResponse response = (HttpServletResponse)res;

  try {        1 前端请求，直接放行
      chain.doFilter(request, response);
      this.logger.debug( o: "Chain processed normally");
  } catch (IOException var9) {
      throw var9;
  } catch (Exception var10) {  2 如果抛出异常，进行捕获，进行处理
      Throwable[] causeChain = this.throwableAnalyzer.determineCauseChain(var10);
      RuntimeException ase = (AuthenticationException)this.throwableAnalyzer.getF
      if (ase == null) {
          ase = (AccessDeniedException)this.throwableAnalyzer.getFirstThrowableOf
      }
```

```
; FilterSecurityInterceptor ext
                                    fi.getRequest().setAttribute( s: "__spring_security_filterSecurityInte
                                }        判断请求是否有权限

                                InterceptorStatusToken token = super.beforeInvocation(fi);

                                                进行springmvc处理
                                try {
                                    fi.getChain().doFilter(fi.getRequest(), fi.getResponse());
                                } finally {
                                    super.finallyInvocation(token);
                                }
```



请求间认证共享

（1）认证成功的方法，把认证信息对象放到对象

把认证信息对象，封装到SecurityContext里面，存入SecurityContextHolder里面

```
protected void successfulAuthentication(HttpServletRequest request, HttpSe:
    if (this.logger.isDebugEnabled()) {
        this.logger.debug( o: "Authentication success. Updating SecurityCon
    }

    SecurityContextHolder.getContext().setAuthentication(authResult);
```

（2）SecurityContext对象

对象Authentication进行封装

（3）SecurityContextHolder，使用ThreadLocal进行操作

```
s SecurityContextPersistenceFilter exter
```