SpringSecurity实现多表账户登录

需求：针对公司员工，普通用户等各类型用户，将其分别存储在不同的用户表中，基于SpeingSecurity实现用户认证，也就是登陆功能

流程

- 首先做数据库设计
- 基于SpringBoot创建一个项目
- 项目中做相关的实现
- 通过apifox接口测试工具进行测试
- 分别测试不同用户的登陆方法，是否调用了对应的登录逻辑【登陆也称为认证】

> 注意：权限这块并没有涉及，仅仅是用户数据这块

数据表设计

本文先不涉及权限，表设计就是两张用户表

员工表

```
CREATE TABLE `ums_sys_user` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '用户ID',
  `username` varchar(30) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT
NULL COMMENT '用户账号',
  `nickname` varchar(30) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT
NULL COMMENT '用户昵称',
  `email` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci DEFAULT
'' COMMENT '用户邮箱',
  `mobile` varchar(11) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci DEFAULT
'' COMMENT '手机号码',
  `sex` int DEFAULT '0' COMMENT '用户性别（0男 1女 2未知）',
  `avatar` varchar(100) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci DEFAULT
'' COMMENT '头像地址',
  `password` varchar(100) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci
DEFAULT '' COMMENT '密码',
  `status` int DEFAULT '0' COMMENT '帐号状态（0正常 1停用）',
  `creator` bigint DEFAULT '1' COMMENT '创建者',
  `create_time` datetime DEFAULT NULL COMMENT '创建时间',
  `updater` bigint DEFAULT '1' COMMENT '更新者',
  `update_time` datetime DEFAULT NULL COMMENT '更新时间',
  `remark` varchar(500) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci DEFAULT
NULL COMMENT '备注',
  `deleted` tinyint DEFAULT '0',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8mb4
COLLATE=utf8mb4_0900_ai_ci ROW_FORMAT=DYNAMIC COMMENT='后台用户表';
```

客户表

```
CREATE TABLE `ums_site_user` (
  `id` bigint NOT NULL AUTO_INCREMENT COMMENT '用户ID',
  `username` varchar(30) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT
NULL COMMENT '用户账号',
  `nickname` varchar(30) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT
NULL COMMENT '用户昵称',
  `openid` varchar(30) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci NOT NULL
COMMENT '微信openid',
  `email` varchar(50) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci DEFAULT
'' COMMENT '用户邮箱',
  `mobile` varchar(11) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci DEFAULT
'' COMMENT '手机号码',
  `sex` int DEFAULT '0' COMMENT '用户性别（0男 1女 2未知）',
  `avatar` varchar(100) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci DEFAULT
'' COMMENT '头像地址',
  `password` varchar(100) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci
DEFAULT '' COMMENT '密码',
  `status` int DEFAULT '0' COMMENT '帐号状态（0正常 1停用）',
  `create_time` datetime DEFAULT NULL COMMENT '创建时间',
  `updater` bigint DEFAULT '1' COMMENT '更新者',
  `update_time` datetime DEFAULT NULL COMMENT '更新时间',
  `remark` varchar(500) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci DEFAULT
NULL COMMENT '备注',
  `deleted` tinyint DEFAULT '0',
  PRIMARY KEY (`id`) USING BTREE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
ROW_FORMAT=DYNAMIC COMMENT='外部用户表';
```

创建项目

项目结构

登陆功能，使用非常简单的三层架构，技术选型有：

- SpringBoot 3.1.X
- SpringSecurity 6.1.X
- Mybatis Plus
- lombok【简化实体类】，可以通过注解生成getter、setter方法，构造方法，toString方法等
- maven

pom文件

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-security</artifactId>
    </dependency>
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
```

```
        <version>8.0.33</version>
    </dependency>
    <dependency>
        <groupId>com.baomidou</groupId>
        <artifactId>mybatis-plus-boot-starter</artifactId>
        <version>3.5.3.2</version>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
    </dependency>
</dependencies>
```

application.yml文件配置

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/spring-security?
useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useSSL
=true&serverTimezone=GMT%2B8
    username: root
    password: stt123456
```

创建三层架构

Controller

```
@RestController
@RequestMapping("/auth")
public class AuthController {

    private final ISysUserService sysUserService;
    private final ISiteUserService siteUserService;

    public AuthController(ISysUserService sysUserService, ISiteUserService
siteUserService) {
        this.sysUserService = sysUserService;
        this.siteUserService = siteUserService;
    }

    /**
     * 后端管理系统登陆
     * 返回值：token
     */
    @PostMapping("sys_login")
    public String sysLogin(@RequestBody LoginParam loginParam) {

        return "后台用户登陆======》" +sysUserService.sysLogin(loginParam);
    }

    @PostMapping("site_login")
    public String siteLogin(@RequestBody LoginParam loginParam) {
```

```
        return "APP用户登陆======》" + siteUserService.siteLogin(loginParam);
    }

}
```

SysUserServiceImpl

```
@Service
@Slf4j
public class SysUserServiceImpl extends ServiceImpl<SysUserMapper, SysUser>
implements ISysUserService {

    @Autowired
    @Qualifier("sysUserAuthenticationManager")
    private AuthenticationManager authenticationManager;

    /**
     * 登陆是SpringSecurity实现的，我们就是去告诉SpringSecurity现在要登陆
     * SpringSecirity登陆是通过 AuthticationManager 实现的
     * 将AuthticationManager引入到service中，调用他的认证方法就可以了
     * @param loginParam
     * @return
     */
    @Override
    public String sysLogin(LoginParam loginParam) {
        // 通过authenticationManager 的认证方法实现登录，该方法需要传入 Authentication
对象 就是一个认证对象
        // Authentication里边存储的就是用户的认证信息，权限，用户名，密码的等信息，其实就是
loadUserByUsername方法返回的UserDetails
        UsernamePasswordAuthenticationToken authenticationToken =
                new
UsernamePasswordAuthenticationToken(loginParam.getUsername(),
loginParam.getPassword());

        Authentication authenticate =
authenticationManager.authenticate(authenticationToken);
        // 获取用户信息
        SysUser sysUser = (SysUser) authenticate.getPrincipal();
        log.info("sysUser==========》{}",sysUser);
        // 返回的是token
        return sysUser.getUsername();
    }
}
```

SiteUserServiceImpl

```
@Service
@Slf4j
public class SiteUserServiceImpl extends ServiceImpl<SiteUserMapper, SiteUser>
implements ISiteUserService {

    /**
     * 将AuthenticationManager注入
     */
```

```
    @Autowired
    @Qualifier("siteUserAuthenticationManager")
    private AuthenticationManager authenticationManager;

    @Override
    public String siteLogin(LoginParam loginParam) {
        UsernamePasswordAuthenticationToken authenticationToken =
                new UsernamePasswordAuthenticationToken(loginParam.getMobile(),
loginParam.getPassword());
        Authentication authenticate =
authenticationManager.authenticate(authenticationToken);
        // 强转为用户类型
        SiteUser siteUser = (SiteUser) authenticate.getPrincipal();
        log.info("siteUser==========>{}",siteUser);
        return siteUser.getUsername();
    }
}
```

实现login功能

项目中引入SpringSecurity，SpringSecurity在实现用户登录【认证】时需要使用到两个接口

- UserDetailsService：是一个接口，提供了一个方法loadUserByUsername();
- UserDetails：是一个接口，用来存储用户权限，状态【是否禁用，超时等】

通过UserDetailsService查询用户，将用户信息放到UserDetails中，剩下的就交给SpringSecurity的AuthenticationManager做判断，判断用户是否允许登录

分两步走

创建UserDetailsService接口实现类

查询用户，分别为客户和用后台系统用户创建对应的查询用户的实现类

```
// 系统用户的DetailsService
@Service
public class SysUserDetailsService implements UserDetailsService {

    private final SysUserMapper sysUserMapper;

    public SysUserDetailsService(SysUserMapper sysUserMapper) {
        this.sysUserMapper = sysUserMapper;
    }

    /**
     * 此方法从数据库中查询用户
     * 返回一个 UserDetails
     */
    @Override
    public UserDetails loadUserByUsername(String username) throws
UsernameNotFoundException {
        log.info("后台系统用户登录==============》");
```

```
        // 根据用户名查询用户
        SysUser sysUser = sysUserMapper.selectOne(new
LambdaQueryWrapper<SysUser>().eq(SysUser::getUsername, username));
        // 有权限的话，需要查询该用户对应的权限
        if(sysUser == null) {
            throw new UsernameNotFoundException("用户或密码不正确");
        }
        return sysUser;
    }
}
// APP用户的DetailsService
@Slf4j
public class SiteUserDetailsService implements UserDetailsService {

    private final SiteUserMapper siteUserMapper;

    public SiteUserDetailsService(SiteUserMapper siteUserMapper) {
        this.siteUserMapper = siteUserMapper;
    }

    @Override
    public UserDetails loadUserByUsername(String mobile) throws
UsernameNotFoundException {
        log.info("APP用户登录==================》");
        SiteUser siteUser = siteUserMapper.selectOne(new
LambdaQueryWrapper<SiteUser>().eq(SiteUser::getMobile, mobile));
        if(siteUser == null) {
            throw new UsernameNotFoundException("用户名或密码错误！");
        }
        return siteUser;
    }
}
```

创建UserDetails接口实现类

存储用户信息，同样的创建两个实现类，存储不同的用户信息，再实体类上直接修改

```
// 后台管理系统用户类
@TableName("ums_sys_user")
@Data
public class SysUser implements Serializable, UserDetails {

    private Long id;
    private String username;
    private String nickname;
    private String email;
    private String mobile;
    private Integer sex;
    private String avatar;
    @JsonIgnore
    private String password;
    private Integer status;
    private Long creator;
    private Long updater;
    private String remark;
```

```
    @TableLogic
    private Integer deleted;

    private LocalDateTime createTime;
    private LocalDateTime updateTime;

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return null;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
// APP用户实体类
@Data
@TableName("ums_site_user")
public class SiteUser implements Serializable, UserDetails {

    private Long id;
    private String username;
    private String nickname;
    private String openid;
    private String email;
    private String mobile;
    private Integer sex;
    private String avatar;
    @JsonIgnore
    private String password;
    private Integer status;
    private Long updater;
    private String remark;
    @TableLogic
    private Integer deleted;

    private LocalDateTime createTime;
    private LocalDateTime updateTime;
```

```
    /**
     * 权限。现在并没有查询权限
     * @return
     */
    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return null;
    }

    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @Override
    public boolean isEnabled() {
        return true;
    }
}
```

关联

将SpringSecurity的AuthenticationManager 【认证管理器，管登陆的组件】，与我们写的登陆逻辑关联起来【loadUserByUsername方法】，实现方式就是在SpringSecurity的配置类中实现

```
/**
 * 现在使用的是SpringSecurity 6.1.5版本，开启SpringSecurity的自定义配置，
 * 需要使用 @EnableWebSecurity注解，而不再是继承Adpater
 */
@Configuration
@EnableWebSecurity
public class SecurityConfig {

    @Autowired
    private SysUserDetailsService sysUserDetailsService;

    @Autowired
    private SiteUserDetailsService siteUserDetailsService;

    // 配置SpringSecurity的过滤器链
    @Bean
    public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
        // 设置登陆接口放行
```

```
        http.authorizeHttpRequests(auth ->
auth.requestMatchers("/auth/sys_login","/auth/site_login").permitAll().anyReques
t().authenticated());
        // 关闭csrf
        http.csrf(csrf -> csrf.disable());
        return http.build();
    }


    // 配置AuthenticationManager，配置两个。一个管理后台用户
    @Primary
    @Bean("sysUserAuthenticationManager")
    public AuthenticationManager sysUserAuthenticationManager(PasswordEncoder
passwordEncoder) {
        DaoAuthenticationProvider authenticationProvider = new
DaoAuthenticationProvider();
        // 关联UserDetailsService
        authenticationProvider.setUserDetailsService(sysUserDetailsService);
        // 关联密码管理器
        authenticationProvider.setPasswordEncoder(passwordEncoder);
        return new ProviderManager(authenticationProvider);
    }


    // 配置AuthenticationManager，管理APP用户
    @Bean("siteUserAuthenticationManager")
    public AuthenticationManager siteUserAuthenticationManager(PasswordEncoder
passwordEncoder) {
        DaoAuthenticationProvider authenticationProvider = new
DaoAuthenticationProvider();
        // 关联UserDetailsService
        authenticationProvider.setUserDetailsService(siteUserDetailsService);
        // 关联密码管理器
        authenticationProvider.setPasswordEncoder(passwordEncoder);
        return new ProviderManager(authenticationProvider);
    }

    /**
     * 密码管理器，会将明文密码转换成密文，加密，而且不能解码
     * @return
     */
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

密码

需要对数据库中存储的密码进行编码，因为SpringSecurity进行密码匹配时，会对用户输入的密码先编码，再验证，先通过PasswordEncoder生成加密后的密码

```
@SpringBootTest
```

```java
public class MyTestApplication {

    @Autowired
    private PasswordEncoder passwordEncoder;

    @Test
    public void test() {
        // 密码加密
        String encode = passwordEncoder.encode("123456");
        System.out.println(encode);

    }
}
```

问题

```
with message: Found 2 beans for type interface org.springframework.security.authentication.AuthenticationManager, but none marked as primary
```

SpringSecurity配置

创建两个AuthenticationManager

```
@Primary
@Bean("sysAuthenticationManager")
public AuthenticationManager sysAuthenticationManager(PasswordEncoder
passwordEncoder) {
    DaoAuthenticationProvider authenticationProvider = new
DaoAuthenticationProvider();
    authenticationProvider.setUserDetailsService(sysUserDetailsService);
    authenticationProvider.setPasswordEncoder(passwordEncoder);
    ProviderManager providerManager = new
ProviderManager(authenticationProvider);
    providerManager.setEraseCredentialsAfterAuthentication(false);
    return providerManager;
}

/**
 * 外部用户验证管理器
 * @param passwordEncoder
 * @return
 */
@Bean("siteAuthenticationManager")
public AuthenticationManager siteAuthenticationManager(PasswordEncoder
passwordEncoder) {
    DaoAuthenticationProvider authenticationProvider = new
DaoAuthenticationProvider();
    authenticationProvider.setUserDetailsService(siteUserDetailsService);
    authenticationProvider.setPasswordEncoder(passwordEncoder);
    ProviderManager providerManager = new
ProviderManager(authenticationProvider);
    providerManager.setEraseCredentialsAfterAuthentication(false);
    return providerManager;
}
```

数据库配置

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://localhost:3306/springsecurity?
useUnicode=true&characterEncoding=utf8&zeroDateTimeBehavior=convertToNull&useSSL
=true&serverTimezone=GMT%2B8
    username: root
    password: stt123456
```

AuthenticationManager

AuthenticationManager用于定义SpringSecurity如何进行身份认证，之后将认证信息封装在Authentication对象上，设置到SecurityContextHolder上，AuthenticationManager常用的实现是ProviderManager，你也可以对其做自定义实现。