链滴

Spring Security 说明和使用

作者：yechuan

原文链接：https://ld246.com/article/1590658568235

来源网站：链滴

springSecurity说明

- springSecurity主要两大核心概念，认证、授权
- 故名思意，认证就是校验用户是否有该路由的权限，授权是指给指定用户授予指定权限

springSecurity使用

- 首先我们得导入依赖

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
 <dependency>
    <groupId>com.alibaba</groupId>
    <artifactId>druid-spring-boot-starter</artifactId>
    <version>1.1.10</version>
  </dependency>
  <dependency>
    <groupId>com.baomidou</groupId>
    <artifactId>mybatis-plus-boot-starter</artifactId>
    <version>3.3.1.tmp</version>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
  </dependency>
```

- 创建数据库

```
create database `security`;
create table `user_table`
(
    `user_id`       int unsigned not null auto_increment comment '用户id，主键',
    `user_name`     varchar(25)  not null comment '用户名',
    `login_name`    varchar(25)  not null comment '用户登录名',
    `login_password` varchar(255) not null comment '用户登录密码',
    `creator_id`    int unsigned not null default '1' comment '创建者id',
    `create_time`   timestamp    not null default current_timestamp comment '创建时间戳',
    `role_id`       int unsigned not null comment '用户权限id',
    primary key (user_id),
    unique index (create_time)
) engine = innodb
  default char set utf8mb4 comment '用户表';
 create table `role_table`
(
    `role_id`    int unsigned not null auto_increment comment '角色id',
    `role_name`  varchar(25)  not null comment '角色名称',
    `role_access` varchar(25)  not null comment '角色权限',
    primary key (role_id)
) engine = innodb
  default char set utf8mb4 comment '角色权限表';
```

- 添加数据

```
    use upload_model;
insert into role_table(role_name, role_access) value
    ('管理员', 'ADMIN'),
    ('普通用户', 'COMMON'),
    ('游客', 'GUEST');
insert into user_table
    (user_name, login_name, login_password, creator_id, create_time,  role_id)
VALUES ('admin','admin','$2a$10$Q96zmpvV4Udk81xT9ZfdbeF0zRezkHWiHt/jBIbG5391PObg
j35i',0,now(),1);
```

- 创建实体类与dao层接口

```
@Data
@TableName("user_table")
public class UserTable implements Serializable {


    private static final long serialVersionUID = -5157610940656242 76L;
    /**
     * 用户id，主键
     */
    @TableId
    private Integer userId;

    /**
     * 用户名
     */
    private String userName;

    /**
     * 用户登录名
     */
    private String loginName;

    /**
     * 用户登录密码
     */
    private String loginPassword;

    /**
     * 创建者id
     */
    private Integer creatorId;

    /**
     * 创建时间戳
     */
    private Date createTime;

    /**
     * 用户权限id
     */
```

```
    private Integer roleId;

}
@Data
@TableName("role_table")
public class RoleTable implements Serializable {
    private static final long serialVersionUID = -2322014478068634121L;
    /**
     * 角色id
     */
    @TableId
    private Integer roleId;

    /**
     * 角色名称
     */
    private String roleName;

    /**
     * 角色权限
     */
    private String roleAccess;

}
@Mapper
@Repository
public interface RoleTableDao extends BaseMapper<RoleTable> {
}
@Mapper
@Repository
public interface UserTableDao extends BaseMapper<UserTable> {
}
```

- 创建 **WebSecurityConfigurerAdapter**的子类，并添加到容器

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled = true)// 可使用注解
public class CustomSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        //  /user/register 路由不进行拦截
        http.authorizeRequests()
            .antMatchers("/user/register").permitAll();
        // /user/loginTest 路由需要 ADMIN 权限方可访问
        http.authorizeRequests()
                .antMatchers("/user/loginTest").hasAuthority("ADMIN");
        // 开启登录页面，
        http.formLogin();
    }
}
```

- 创建 **UserDetailsService**的子类，并添加到容器

```
@Slf4j
@Component
public class CustomUserDetailsServiceImpl implements UserDetailsService {


    @Autowired
    UserTableDao userDao;

    @Autowired
    RoleTableDao roleDao;

    @Override
    public UserDetails loadUserByUsername(String loginName) throws UsernameNotFoundExc
ption {
        log.info("登录用户为：{}", loginName);
        UserTable user = userDao.selectOne(new QueryWrapper<UserTable>().eq("login_name",
oginName));
        RoleTable role = roleDao.selectById(user.getRoleId());
        log.info("该用户权限为：{}", role.getRoleAccess());
        return new User(loginName, user.getLoginPassword(), AuthorityUtils.commaSeparatedStr
ngToAuthorityList(role.getRoleAccess()));
    }

    /**
    * 密码加密解密器
    */
    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

- 控制器

```
@RestController
@RequestMapping("user")
public class UserController {

    @Autowired
    UserTableDao userTableDao;



    @PostMapping("register")
    public String doRegister(@RequestParam("loginname") String loginName, @RequestParam
"password") String passWord) {
        return "register Ok";
    }

    /**
    * 本路由需要admin权限方可进入
    */
    @PreAuthorize("hasAuthority('ADMIN')")
    @GetMapping("noLoginTest")
```

```
    public String noLoginTest() {
        return "noLoginTest";
    }


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