

# SpringSecurity

JSD PROJECT DAY01



## 目录 | CONTENTS

01 简介

02 认证

03 授权

# 01

## 简介

# 初识SpringSecurity框架

---

# 什么是SpringSecurity

- Spring Security是一个基于Spring框架的安全性认证和访问控制框架，主要用于保护Web应用程序。它提供了一组与安全相关的服务和类，使得开发者可以方便地为Web应用程序添加认证和访问控制功能。
- 主要作用:帮助我们进行登录的认证,和项目中涉及到权限时进行权限管理操作

# 02

## 登录认证

# 认证流程

---

# 1. 引入SpringSecurity框架

在Spring Boot项目中使用Spring Security时需要添加依赖项：

```
<!-- Spring Boot支持Spring Security的依赖项，用于处理认证与授权 -->  
<dependency>  
    <groupId>org.springframework.boot</groupId>  
    <artifactId>spring-boot-starter-security</artifactId>  
</dependency>
```



## 2. 对项目的影晌

- 所有的请求（包括根本不存在的）都必须登录，否则自动跳转到登录页面
- 默认的用户名是user，密码是启用项目时在控制台提示的一串UUID值
- 登录时，如果在打开登录页面后重启过服务器端，应该刷新登录页面，否则，第1次输入并提交是无效的
- 当登录成功后，会自动跳转到此前尝试访问的URL
- 当登录成功后，可通过 /logout 退出登录
- 默认不接受普通的POST请求，如果提交POST请求，会响应403（Forbidden）
  - ✓ 具体原因参见后续的CSRF相关内容

### 3. 关于Spring Security的配置

```
@Slf4j
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // super.configure(http); // 不要保留调用父类同名方法的代码，不要保留！不要保留！不要保留！
    }

}
```

## 4. 默认的登录表单

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    // 如果调用以下方法，当需要访问通过认证的资源，但是未通过认证时，将自动跳转到登录页面
    // 如果未调用以下方法，将响应403
    http.formLogin();

    // super.configure(http); // 不要保留调用父类同名方法的代码，不要保留！不要保留！不要保留！
}
```

## 5. 设置白名单

```
@Override
```

```
protected void configure(HttpSecurity http) throws Exception {  
    String[] urls = {"/reg.html", "/login.html", "/reg", "/login", "/current"};  
    http.authorizeRequests() // 对请求进行授权  
        .mvcMatchers(urls) // 匹配某些路径  
        .permitAll() // 直接许可，即不需要认证即可访问  
        .anyRequest() // 任意请求  
        .authenticated(); // 要求通过认证的  
}
```

## 6. 模拟登录

- 尝试登录用户名为root密码为123456
- 需要创建UserDetailsService接口的实现类，并实现loadUserByUsername方法
- 此时进行登录时会自动调用此方法
- 设置默认密码编码为无编码

```
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {
    @Bean
    public PasswordEncoder passwordEncoder() { return NoOpPasswordEncoder.getInstance(); }
```

- 登录成功时需要响应一个UserDetail对象，如果响应的是一个null则会认为是用户名不存在

```
UserDetails userDetails = User.builder()
    .username("root").password("123456")
    .disabled(false) // 账号禁用
    .accountLocked(false) // 账号锁定
    .accountExpired(false) // 账号过期
    .credentialsExpired(false) // 凭证过期
    .authorities("这是一个临时使用的山寨权限") // 权限
    .build();
return userDetails;
```

## 7. 使用自己的页面进行登录

- 需要在配置类中添加`http.formLogin().loginPage("/login.html");`
- 在SpringSecurity框架的配置类中添加认证管理器
- 添加后在自己的Controller里面添加以下代码才会触发UserDetailServiceImpl实现类

*// 创建认证对象*

```
Authentication authentication = new UsernamePasswordAuthenticationToken(  
    userLoginDTO.getUsername(), userLoginDTO.getPassword());
```

*// 通过认证管理器 执行认证, 并获取结果*

```
Authentication authenticateResult = authenticationManager.authenticate(authentication);
```

*// 将认证结果存入到SecurityContext*

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

- CSRF指的是Cross-Site Request Forgery，中文翻译为“跨站请求伪造”，也称跨域攻击。它是一种网络攻击方式，攻击者利用用户在其他网站上登录过的身份信息，在用户不知情的情况下发送恶意请求来执行非法操作。
- CSRF的攻击流程如下：
  - ✓ 用户在网站A上登录并获取Cookie
  - ✓ 用户在不注销网站A的情况下访问了网站B
  - ✓ 网站B通过HTML代码里插入恶意链接的形式，向网站A发出请求
  - ✓ 因为用户在网站A已经有了Cookie，所以请求被成功执行
  - ✓ 一旦攻击者成功发起CSRF攻击，就能够执行一些非法或损害用户隐私的操作，比如修改用户密码、盗取用户资料、转账等等。为了防止CSRF攻击，通常需要采取一些措施，如增加验证码、使用CSRF Token、验证Referer等。

## 9. 关闭跨域攻击防御机制

- SpringSecurity框架默认开启了跨域攻击的防护, 通过携带CSRF token对请求进行判断.
- 因为现在都是前后端分离架构, 客户端发出的请求都是异步请求, 不存在form表单, 所以不存在跨域攻击的问题, 通过以下代码关闭跨域攻击, 否则请求受限

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    // 禁用“防止伪造的跨域攻击”的防御机制
    http.csrf().disable();

    // 暂不关心其它代码
}
```



## 10. 通过真实数据进行登录

- 通过之前所学的Mybatis框架进行登录的查询

```
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {  
    UserVo user = userMapper.selectByUserName(username);  
    if (user == null) {  
        return null; // 会默认抛出AuthenticationException 用户名找不到的异常  
    }  
    // 准备授权列表  
    List<GrantedAuthority> authorities =  
        AuthorityUtils.createAuthorityList(user.getIsAdmin() == 1 ? "ADMIN" : "USER");  
    AuthenticationUser userDetails = new AuthenticationUser(user.getId(), user.getNickname(),  
        username, user.getPassword(),  
        enabled: true, accountNonExpired: true,  
        credentialsNonExpired: true,  
        accountNonLocked: true, authorities);  
    return userDetails;  
}
```

# 11. 统一异常处理

- 涉及两个异常一个是用户名或密码错误的异常, 另一个是账号禁用异常

```
@ExceptionHandler({InternalAuthenticationServiceException.class, BadCredentialsException.class})
public JsonResult handleAuthenticationException(AuthenticationException e) {
    log.warn("异常信息: {}", e.getMessage());
    if (e instanceof InternalAuthenticationServiceException) {
        log.warn("用户名不存在");
        return JsonResult.fail(StatusCode.USER_NOT_EXISTS, message: "用户名不存在");
    }
    // 就是密码错误
    log.warn("密码错误");
    return JsonResult.fail(StatusCode.USER_PASSWORD_ERROR, message: "密码错误");
}
```

```
@ExceptionHandler
public JsonResult handleDisabledException(DisabledException e) {
    log.warn("异常信息: {}", e.getMessage());
    return JsonResult.fail(StatusCode.ERROR_UNAUTHORIZED_DISABLED, message: "账号被禁用");
}
```

## 12. 自定义UserDetails

- 默认的UserDetails里面只有用户的用户名，没有其它的信息如果需要用到类似id和nickname等信息的话需要自定义UserDetails

```
@Getter
@ToString
public class AuthenticationUser extends User {
    private Integer id;
    private String nickname;
    public AuthenticationUser(Integer id, String nickname, String username, String password, boolean enabled, boolean accountNonExpired,
        super(username, password, enabled, accountNonExpired, credentialsNonExpired, accountNonLocked, authorities);
        this.id = id;
        this.nickname=nickname;
    }
}
```

- 为了便于使用和维护，JDK类库按照包结构划分，不同功能的类划分在不同的包中；
- 经常使用的包如下表所示：

包	功能
java.lang	Java程序的基础类，如字符串、多线程等，该包中的类使用的频率非常高，不需要import，可以直接使用
java.util	常用工具类，如集合、随机数产生器、日历、时钟等
java.io	文件操作、输入/输出操作
java.net	网络操作
java.math	数学运算相关操作
java.security	安全相关操作
java.sql	数据库访问
java.text	处理文字、日期、数字、信息的格式

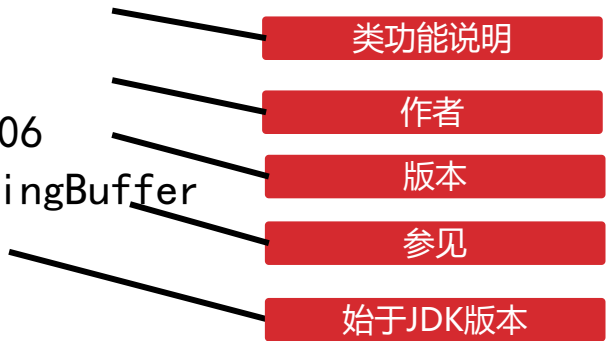
# 文档注释规范

---

- 以 `/**` 开始，以 `*/` 结束；
- 加在类和方法的开头，用于说明作者，时间，版本，要实现功能的详细描述等信息；
- 通过javadoc工具，可以轻松的将此注释转换为HTML文档说明；学习者和程序员主要通过文档了解API的功能；
- 文档注释不同于普通的注释（`//...` 或 `/*...*/`），普通注释写在程序之中，用于程序员进行代码维护和交流，无法通过工具生成文档；而文档注释（`/**...*/`）写在类和方法的开头，专门用于生成供API使用者进行参考的文档资料。

# 什么是JDK API

```
/**
 * The String class represents character strings...
 * ... ..
 * @author Lee Boynton
 * @version 1.204, 06/09/06
 * @see java.lang.StringBuffer
 * @since JDK1.0
 */
public final class String
    implements java.io.Serializable, Comparable<String>,
               CharSequence {
    ... ..
}
```



The diagram consists of five red rectangular boxes with white text, each connected by a black line to a specific annotation in the code above. The boxes and their corresponding annotations are: '类功能说明' (Class Function Description) connected to '\* The <code>String</code> class represents character strings...', '作者' (Author) connected to '\* @author Lee Boynton', '版本' (Version) connected to '\* @version 1.204, 06/09/06', '参见' (See) connected to '\* @see java.lang.StringBuffer', and '始于JDK版本' (Started with JDK Version) connected to '\* @since JDK1.0'.

- 类功能说明
- 作者
- 版本
- 参见
- 始于JDK版本

# 什么是JDK API

```
/**
 * ...
 *
 * @param charsetName
 *         The name of a supported
 *
 * @return The resultant byte array
 *
 * @throws UnsupportedOperationException
 *         If the named charset is not supported
 */
public byte[] getBytes(String charsetName) {
    ... ..
}
```

方法功能说明

参数说明

返回值说明

异常抛出说明



# 02

## 字符串基本操作

# String及常用API

---

# String是不可变对象

- java.lang.String使用了final修饰，不能 被继承；
- 字符串底层封装了字符数组及针对字符数组的操作算法；
- 字符串一旦创建，对象永远无法改变，但字符串引用可以重新赋值；
- Java字符串在内存中采用Unicode编码方式，任何一个字符对应两个字节的定长编码。

- Java为了提高性能，静态字符串（字面量/常量/常量连接的结果）在常量池中创建，并尽量使用同一个对象，重用静态字符串；
- 对于重复出现的字符串直接量，JVM会首先在常量池中查找，如果存在即返回该对象。

- String在内存中采用Unicode编码，每个字符占用两个字节； 任何一个字符（无论中文还是英文）都算1个字符长度，占用两个字节。

# 使用 indexOf 实现检索

- indexOf方法用于实现在字符串中检索另外一个字符串
- String提供几个重载的indexOf方法

<code>int indexOf (String str )</code>	在字符串中检索str，返回其第一次出现的位置，如果找不到则返回-1
--	-----------------------------------

---

<code>int indexOf ( String str, int fromIndex)</code>	从字符串的 fromIndex 位置开始检索
---	------------------------

- String还定义有 lastIndexOf 方法：

<code>int lastIndexOf (String str, int from )</code>	str在字符串中多次出现时，将返回最后一个出现的位置
--	----------------------------

---

# 使用 substring 获取子串

- substring 方法用于返回一个字符串的子字符串。
- substring 常用重载方法定义如下：

String substring ( int beginIndex , int endIndex )	返回字符串中从下标 beginIndex（包括）开始到 endIndex（不包括）结束的子字符串
--	--

---

String substring ( int beginIndex )	返回字符串中从下标 beginIndex（包括）开始到字符串结尾的子字符串
--	---------------------------------------

# trim去除两侧空白

- trim方法用于去掉一个字符串的前导和后继空字符



- **String中定义有charAt () 方法:**

char charAt  
(int index)

方法charAt () 用于返回字符串指定位置的字符。参数index表示指定的位置

---

# startsWith和endsWith

- 用于检测一个字符串是否以指定字符串开头或结尾

- 用于转换字符串中英文字母的大小写形式

- String提供了若干的静态的重载方法valueOf
- 该方法用于将其他类型转换为字符串类型

# 03

## StringBuilder及其API

- StringBuilder封装可变的字符串，对象创建后可以通过调用方法改变其封装的字符序列。
- StringBuilder有如下常用构造方法：

```
public StringBuilder ( )
```

```
public StringBuilder ( String str )
```

# StringBuilder常用方法

StringBuilder类的常用方法	功能描述
StringBuilder append(String str)	追加字符串
StringBuilder insert (int dstOffset, String s)	插入字符串
StringBuilder delete(int start, int end)	删除字符串
StringBuilder replace(int start, int end, String str)	替换字符串
StringBuilder reverse()	字符串反转

# StringBuilder总结

- StringBuilder是可变字符串。字符串的内容计算，建议采用StringBuilder实现，这样性能会好一些；
- java的字符串连接的过程是利用StringBuilder实现的

```
String s = "AB"; String s1 = s + "DE"+1;
String s1 = new StringBuilder(s).append("DE")
    .append(1).toString();
```
- StringBuffer 和StringBuilder
  - StringBuffer是线程安全的，同步处理的，性能稍慢
  - StringBuilder是非线程安全的，并发处理的，性能稍快





## 总结 | SUMMARY

- JDK API
- String API
- StringBuilder API