

Web Programming

Web开发技术基础

第10章 Web认证与授权框架

 计算机学院

 授课人：王尊亮

第10章 Web认证与授权框架

10.1 Web认证与授权概述

10.2 Spring Security框架实践

- 10.2.1 HelloWorld
- 10.2.2 用户及密码存储
- 10.2.3 授权

10.3 OAuth2.0认证



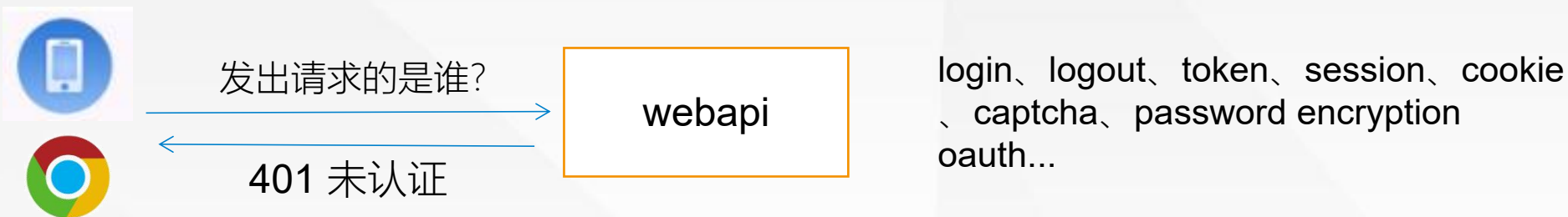
SPRING SECURITY

Protects your application with comprehensive and extensible authentication and authorization support.

10.1 Web认证与授权概述

Web系统需要的常见安全措施有：身份验证、授权以及常见安全漏洞防护。

身份验证：authentication，确认用户的身份，Verifies you are who you say you are。



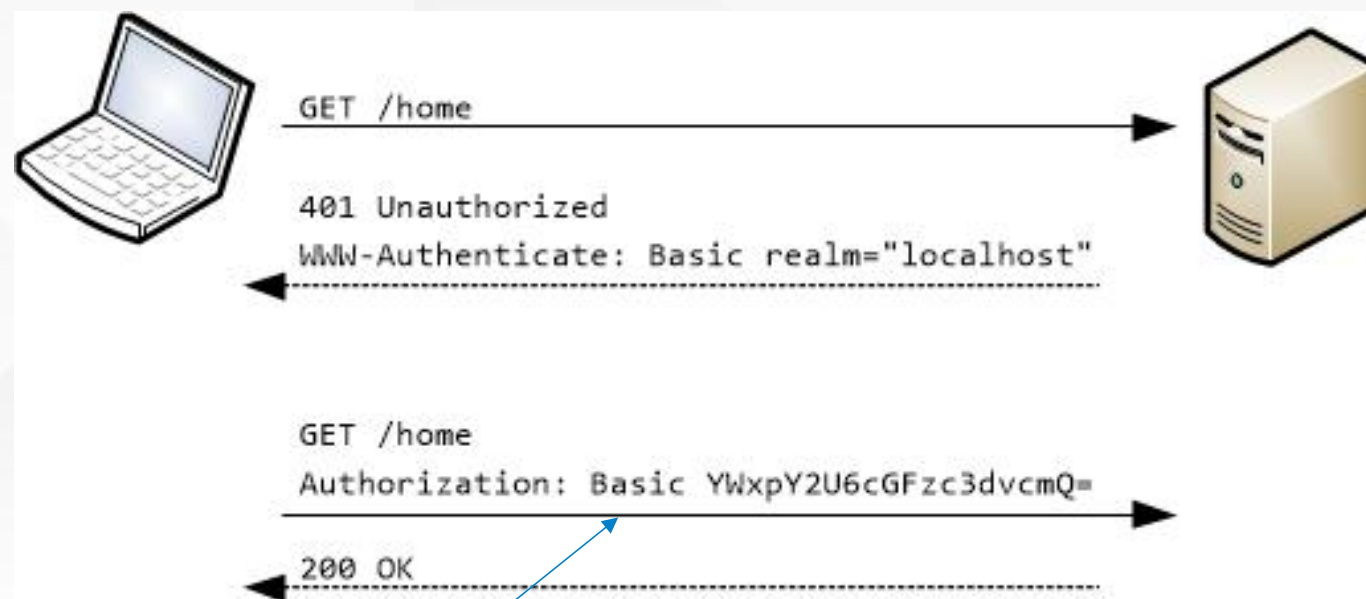
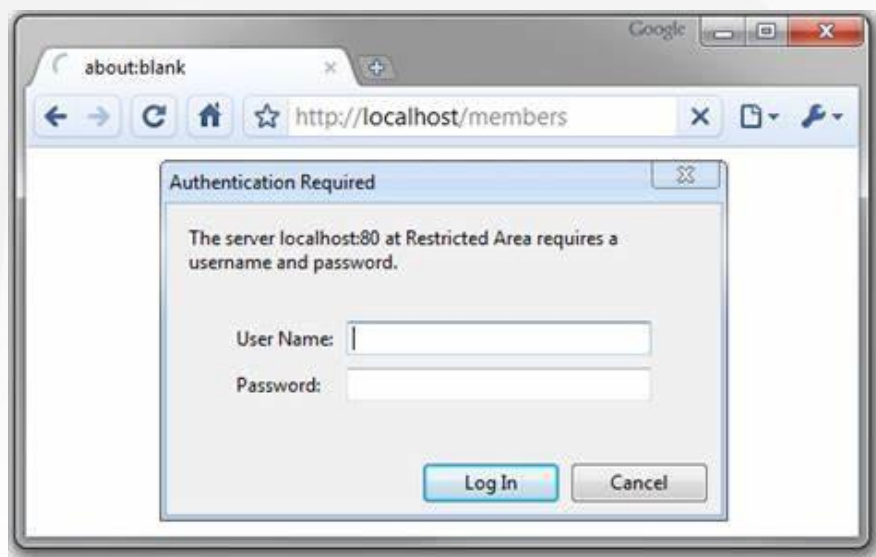
授权：authorization，身份验证后决定了用户访问系统的能力以及达到的程度。Decides if you have permission to access a resource



10.1 Web认证与授权概述

Web系统常用的前后端鉴权机制有以下几种：

- HTTP Basic Authentication: HTTP协议中定义的基本认证方式，简单但不安全。



Basic后的内容为用户名:密码的base64编码

10.1 Web认证与授权概述

Web系统常用的前后端鉴权机制有以下几种：

➤ session-cookie

1. 服务器端创建session，将session保存在内存、DB或redis中，然后将sessionid返回
 2. 登录成功后将用户信息存储在当前session中
10. 后续请求根据session确定是哪个用户

集群扩展时需要共享session



10.1 Web认证与授权概述

Web系统常用的前后端鉴权机制有以下几种：

➤ Token 验证

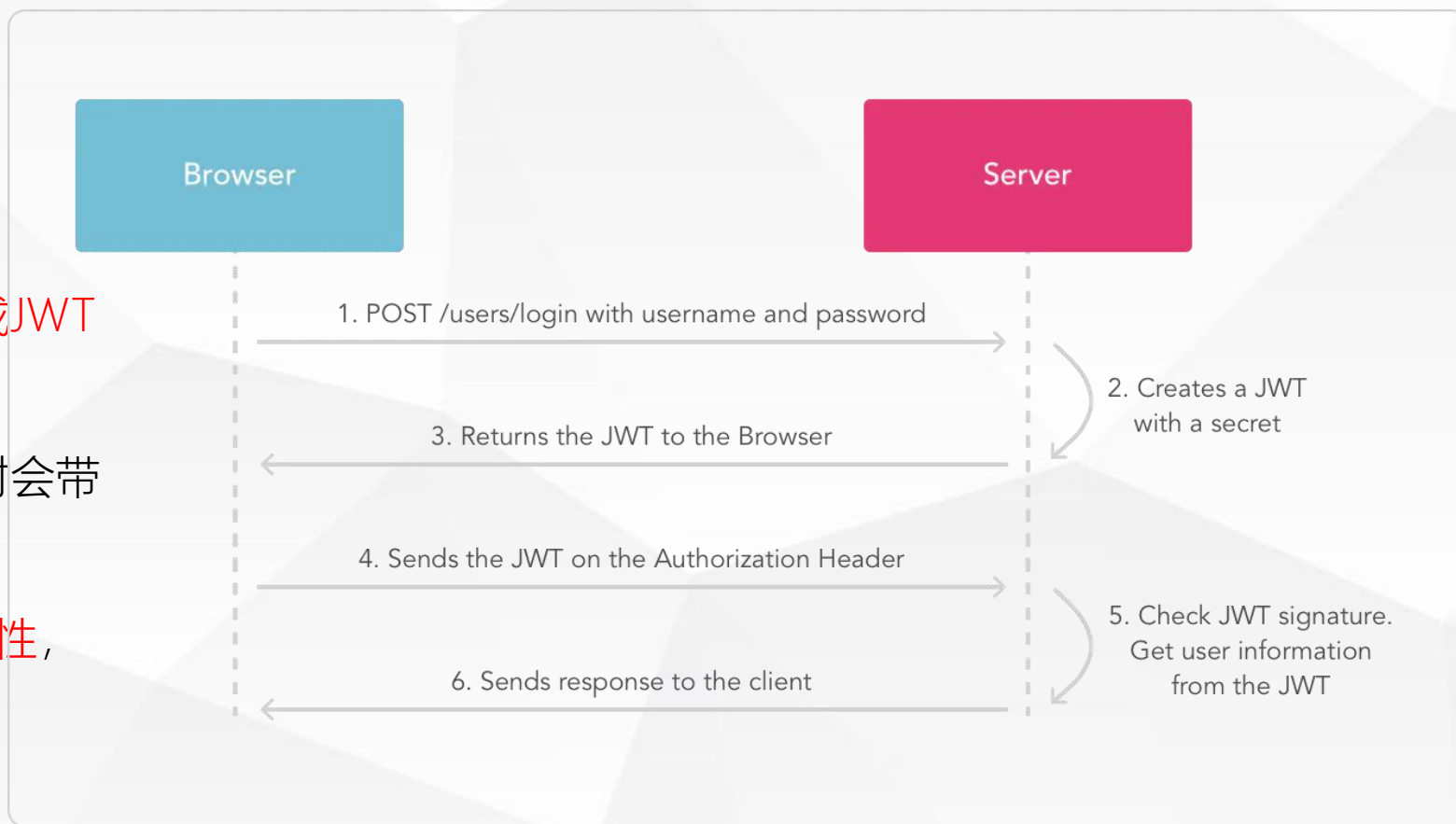
1. 客户端使用用户名跟密码请求登录
2. 服务端收到请求，去验证用户名与密码
3. 验证成功后，服务端签发一个Token
4. 客户端收到 Token 以后存储在 Cookie 里或者 Local Storage 里
5. 客户端每次向服务端请求资源的时候需要带着服务端签发的Token
6. 服务端收到请求，然后去验证客户端请求里面带着的Token，如果验证成功，就向客户端返回请求的数据

10.1 Web认证与授权概述

Web系统常用的前后端鉴权机制有以下几种：

- JWT(Json Web Token)是实现token技术的一个开放标准协议
- 适用于分布式场景、跨域认证

1. 客户端使用账号和密码请求登录接口
2. 登录成功后服务器使用签名密钥生成JWT
3. 服务器将JWT返回给客户端。
4. 客户端再次向服务端请求其他接口时会带上JWT。
5. 服务器接收到JWT后验证签名的有效性，对客户端做出相应的响应。



10.1 Web认证与授权概述

常见的Java 安全框架：Spring Security、Apache Shiro等

➤ Apache Shiro: <http://shiro.apache.org/>, 提供了认证、授权、加密和会话管理等功能。

Authentication: 身份认证/登录;

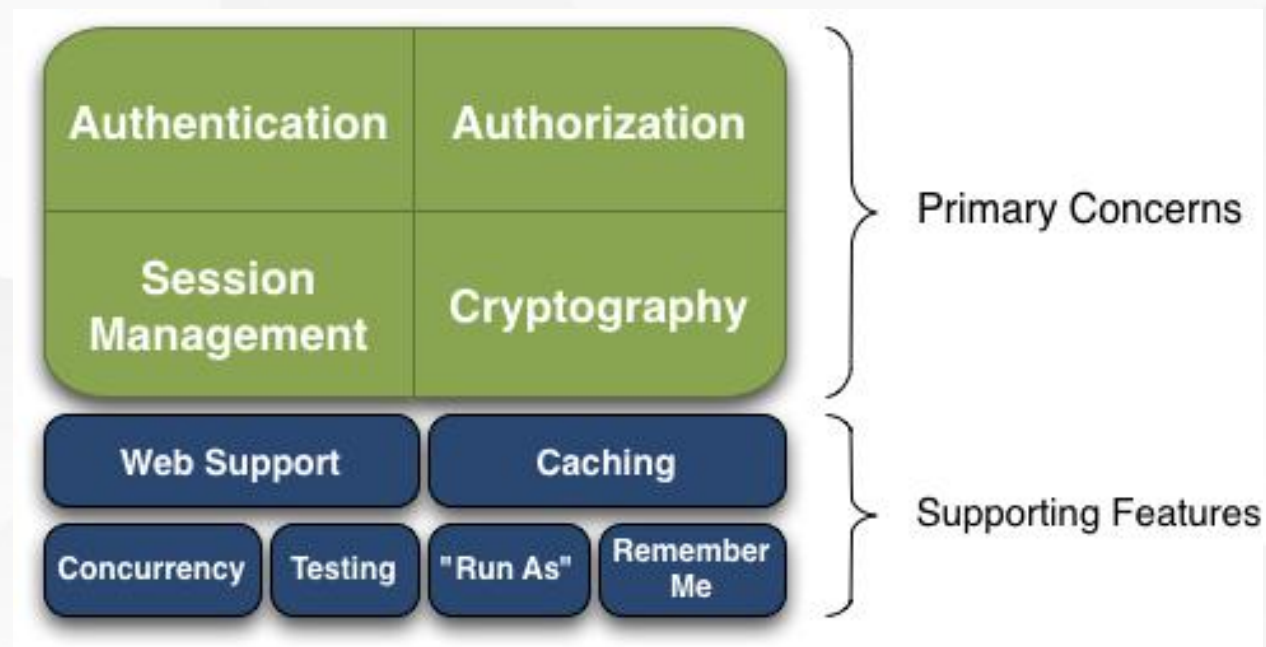
Authorization: 授权, 即权限验证

Session Manager: 会话管理,

Cryptography: 加密, 保护数据的安全性

Web Support: 非常容易集成到Web环境;

Caching: 用户登录后, 其用户信息、拥有的角色/权限可以缓存提高效率;



Apache Shiro特性

10.1 Web认证与授权概述

Spring Security简介, [文档地址](#)

Spring security 是一个强大的和高度可定制的身份验证和访问控制框架, 支持身份验证、授权、以及多项安全漏洞防护。

身份验证方式包括: Username and Password、OAuth 2.0 Login、Remember-Me型用户身份验证、CAS用户身份验证、X.509用户身份验证等

授权方式包括: 角色权限、通过表达式控制URL权限以及方法权限等、基于ACL的领域对象安全等特性

安全漏洞防护包括: CSRF (跨域请求伪造) 防护、xss攻击防护以及其它一些HTTP响应报头安全措施。

10.1 Web认证与授权概述

Spring Security简介, [文档地址](#)

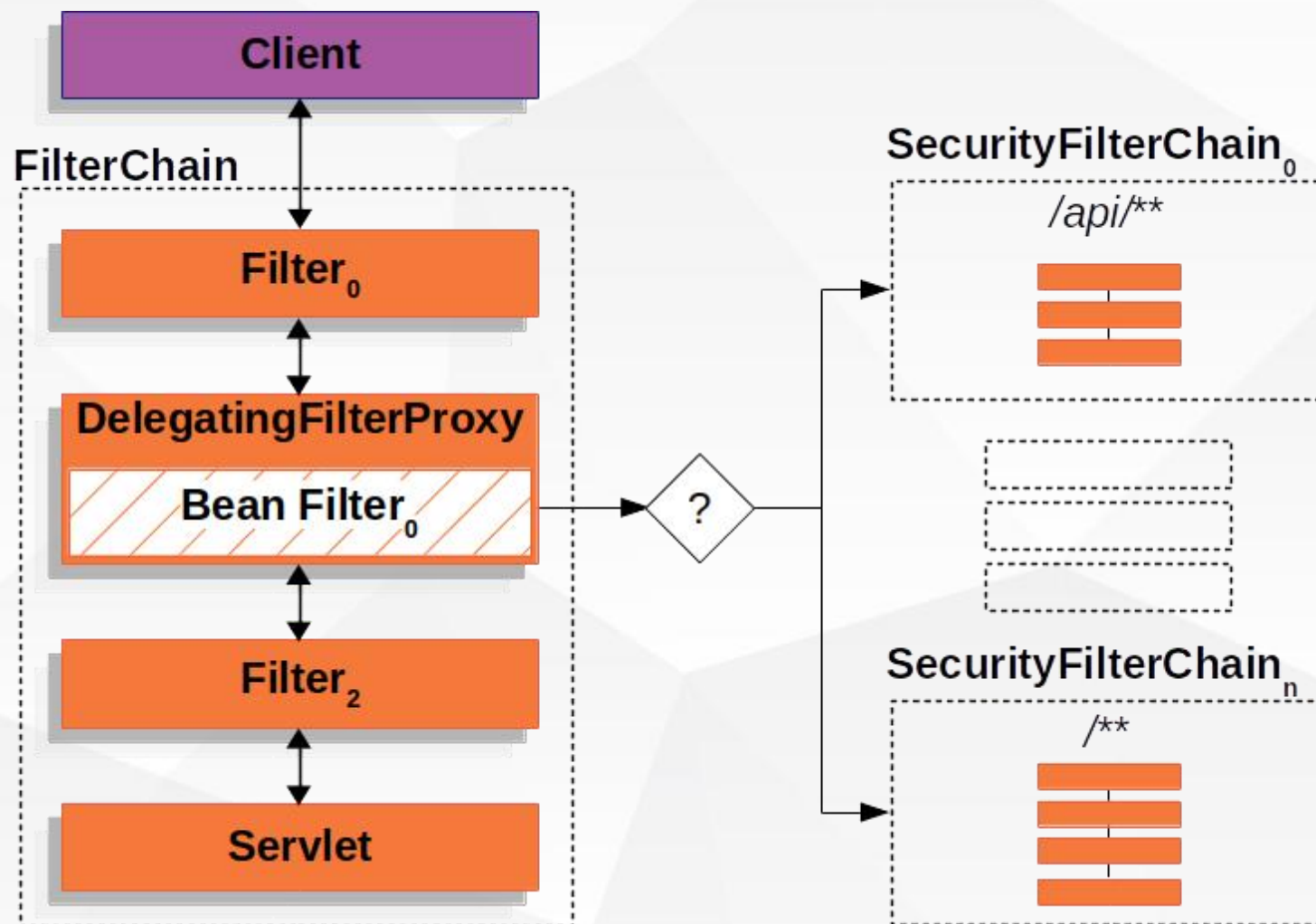
Spring security 基于表达式的权限控制示例:

```
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .authorizeRequests()  
            .antMatchers("/resources/**", "/signup", "/about").permitAll()  
            .antMatchers("/admin/**").hasRole("ADMIN")  
            .antMatchers("/db/**").access("hasRole('ADMIN') and hasRole('DBA')")  
            .anyRequest().authenticated()  
        .and()  
        // ...  
        .formLogin();  
}
```

10.1 Web认证与授权概述

Spring Security简介, [文档地址](#)

Spring security 框架原理图



10.1 Web认证与授权概述

Spring Security简介, [文档地址](#)

Spring security Authentication 框架



SecurityContext: 包含当前认证主体的上下文

Authentication: 认证主体

Principal: 主体信息

Credentials: 主体凭据

Authorities: 主体的权限列表

```
SecurityContext context = SecurityContextHolder.getContext();  
Authentication authentication = context.getAuthentication();  
String username = authentication.getName();  
Object principal = authentication.getPrincipal();  
Collection<? extends GrantedAuthority> authorities = authentication.getAuthorities();
```

获取当前登录的用户及权限

10.2 Spring Security框架实践

Spring security 的引入方式，在创建spring boot项目时勾选security中的Spring Security或者直接在POM.xml中添加下面的依赖

Dependencies:

Security

☒ Spring Security

☐ OAuth2 Client

☐ OAuth2 Resource Server

☐ Spring LDAP

☐ Okta

Spring Security

Highly customizable authentication and access-control framework for Spring applications.

[Securing a Web Application](#) ➤ [Spring Boot and OAuth2](#) ➤
[Authenticating a User with LDAP](#) ➤

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

10.2 Spring Security框架实践

10.2.1 HelloWorld

示例代码: <https://gitee.com/buptnetwork/spring-security-demo>

创建spring boot项目spring-security-demo,勾选如下图有所示的依赖:

Server URL: start.spring.io ⚙

Name:

Location:

Language: ☒ Java ☐ Kotlin ☐ Groovy

Type: ☒ Maven ☐ Gradle

Group:

Artifact:

Package name:

Project SDK:

Java:

Packaging: ☒ Jar ☐ War

Added dependencies:

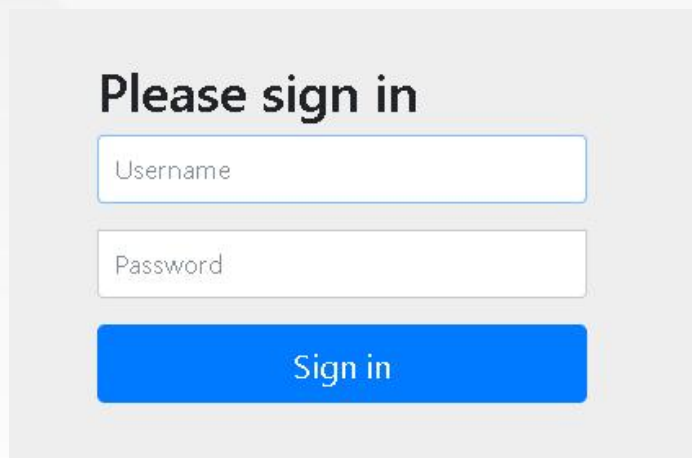
- × Spring Security
- × Spring Web

10.2 Spring Security框架实践

10.2.1 Spring Security HelloWorld

示例代码: <https://gitee.com/buptnetwork/spring-security-demo>

运行项目观察控制台输出, 浏览器访问系统



A login form titled "Please sign in". It contains two input fields: "Username" and "Password". Below the fields is a blue button labeled "Sign in".

输入用户名`user`, 密码为控制台随机密码后可以继续访问

控制台输出随机密码

```
Using generated security password: e22ec506-c693-45a2-9f05-4ee03fcc14f4
```

```
This generated password is for development use only. Your security configuration
```


10.2 Spring Security框架实践

10.2.1 Spring Security HelloWorld

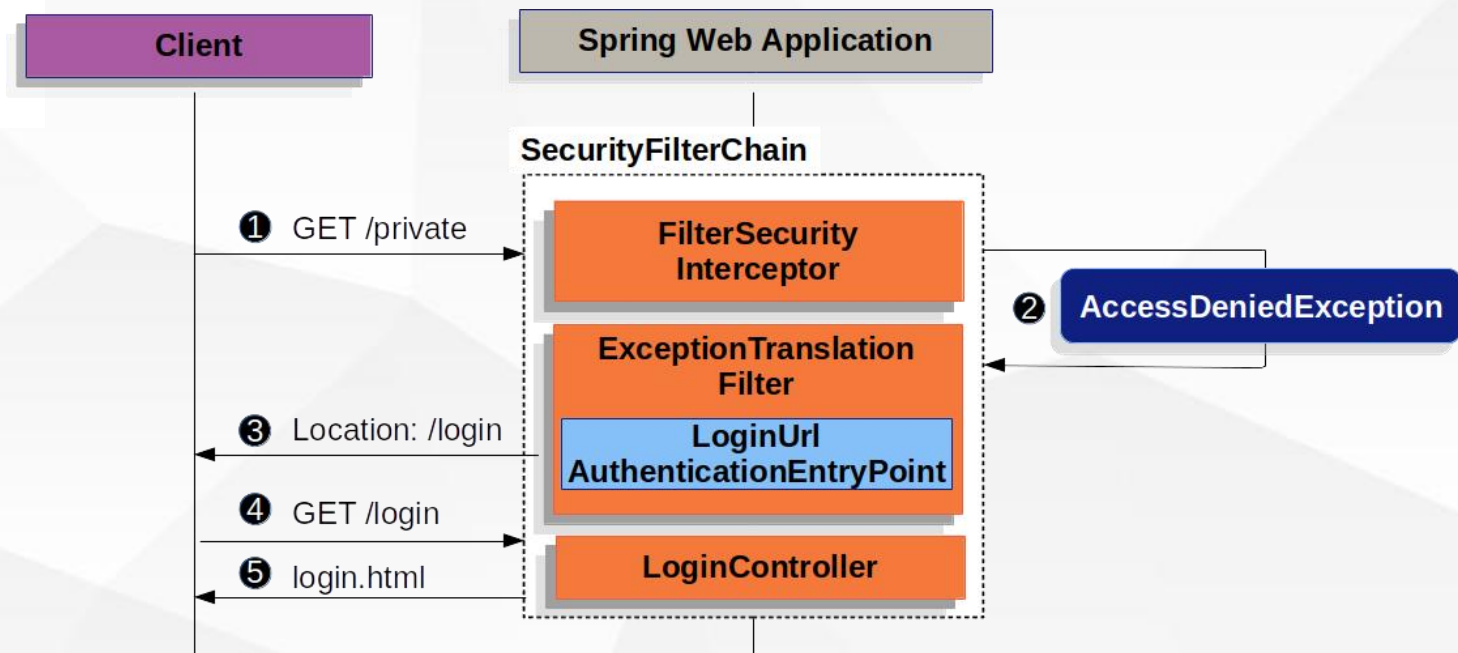
示例代码: <https://gitee.com/buptnetwork/spring-security-demo>

可以在application.properties文件中设置进入系统的用户名及密码

```
spring.security.user.name=admin  
spring.security.user.password=123456
```

Please sign in

Sign in



10.2 Spring Security框架实践

10.2.1 Spring Security HelloWorld

示例代码: <https://gitee.com/buptnetwork/spring-security-demo>

默认的退出登录地址: <http://127.0.0.1:8080/logout>

Are you sure you
want to log out?

Log Out

Please sign in

You have been signed out

Username

Password

Sign in

请求网址: <http://127.0.0.1:8080/logout>

请求方法: POST

状态代码: 302

远程地址: 127.0.0.1:8080

引荐来源网址政策: strict-origin-when-cross-origin

响应标头

查看源代码

Cache-Control: no-cache, no-store, max-age=0, must-revalidate

Connection: keep-alive

Content-Length: 0

Date: Mon, 28 Mar 2022 17:12:53 GMT

Expires: 0

Keep-Alive: timeout=60

Location: <http://127.0.0.1:8080/login?logout>

10.2 Spring Security框架实践

10.2.1 Spring Security HelloWorld

login表单可以自己进行定制

<https://docs.spring.io/spring-security/reference/servlet/authentication/passwords/form.html>

```
protected void configure(HttpSecurity http) throws Exception {  
    http  
        // ...  
        .formLogin(form -> form  
            .loginPage("/login")  
            .permitAll()  
        );  
}
```

```
@Controller  
class LoginController {  
    @GetMapping("/login")  
    String login() {  
        return "login";  
    }  
}
```

10.2 Spring Security框架实践

10.2.2 用户及密码存储

用户及密码存储需要我们使用某种方法把用户数据的来源告诉Spring Security框架。

方法1：使用内存中的用户，即我们通过框架提供的方法在内存中添加一些用户，并配置用户的密码及角色权限等。由于用户不方便进行添加删除修改密码等操作，所以主要是开发阶段测试用，缺少实用价值。

方法2：使用JDBC中的用户，即我们自己有用户、角色、权限等数据库表，然后通过JDBC SQL语句的形式告诉框架如何从我们的用户表中查询用户及用户权限等信息。

方法3：使用自定义用户服务，即通过实现**UserDetails**、**UserDetailsService**等接口的方式实现更通用的用户及角色权限等设置。

10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法1: 使用内存用户

编写WebSecurityConfig:

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
    }
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        auth.inMemoryAuthentication()
            .passwordEncoder(passwordEncoder)
            .withUser(username: "admin")
            .password(passwordEncoder.encode(charSequence: "123456"))
            .roles("USER", "ADMIN")
            .and()
            .withUser(username: "test")
            .password(passwordEncoder.encode(charSequence: "123456"))
            .roles("USER");
    }
}
```

认证管理器的Builder

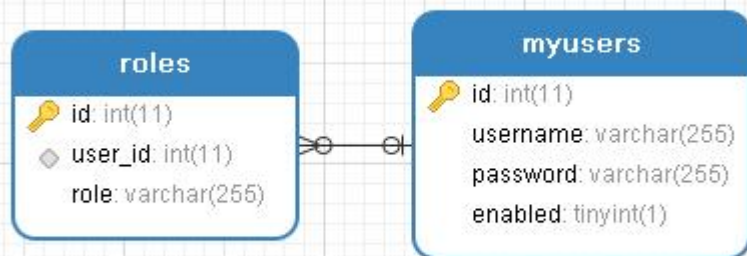
10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法2： 使用JDBC中的用户

创建好的表结构及数据如下图

Diagram 1



对象 无标题 - 模型 myusers @ch10_3 (localhost...)			
开始事务 备注 筛选 排序 导入 导出			
id	username	password	enabled
1	admin	\$2a\$10\$FtWPuxqh9/C2/NECJlxfTOv9α91Z2musbLsD8Zipxks•	1
2	test	\$2a\$10\$FtWPuxqh9/C2/NECJlxfTOv9α91Z2musbLsD8Zipxks•	1

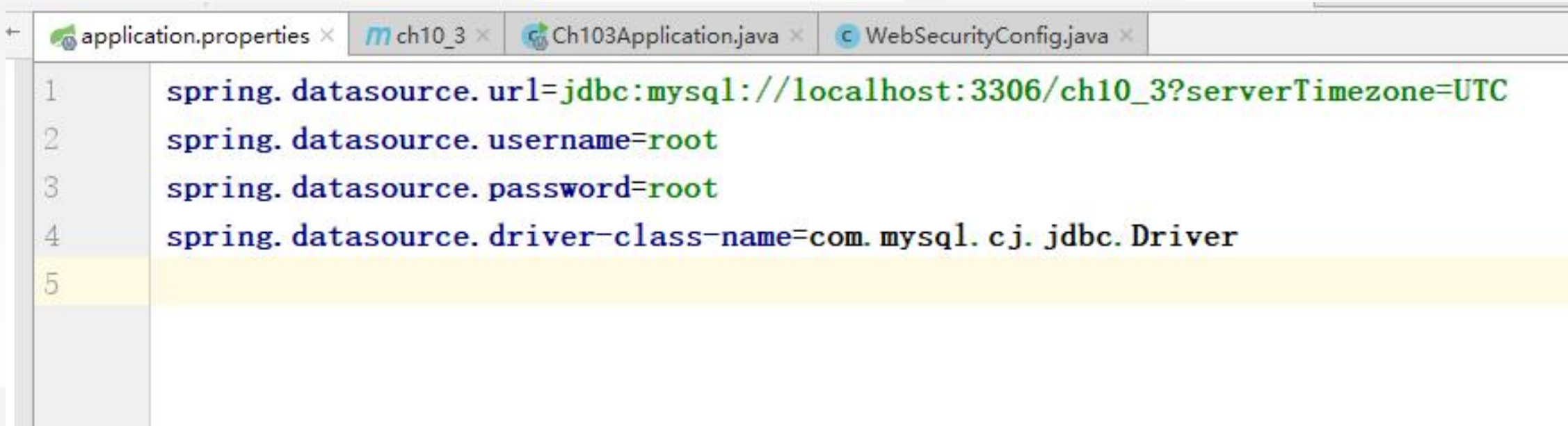
对象 无标题 - 模型 myusers @ch10_3 (localhost... roles @ch10_3 (localhost) - ...			
开始事务 备注 筛选 排序 导入 导出			
id	user_id	role	
1	1	ROLE_ADMIN	
2	2	ROLE_USER	

10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法2： 使用JDBC中的用户

在项目的application.properties文件中配置datasource数据源信息



```
application.properties × ch10_3 × Ch103Application.java × WebSecurityConfig.java ×  
1 spring.datasource.url=jdbc:mysql://localhost:3306/ch10_3?serverTimezone=UTC  
2 spring.datasource.username=root  
3 spring.datasource.password=root  
4 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
5
```

10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法2： 使用JDBC中的用户

创建WebSecurityConfig配置类，重写configure方法，配置使用jdbc验证时使用的两个SQL语句



The screenshot displays an IDE with a project structure on the left and a code editor on the right. The project structure shows a package hierarchy: `cn.e.b.ch10_3` containing `Ch103Application`, `cn.e.b.c.config` containing `WebSecurityConfig`, and `cn.e.b.c.controller` containing `IndexController`. The code editor shows the `WebSecurityConfig` class, which extends `WebSecurityConfigurerAdapter`. It includes an `@Autowired` `DataSource` and an `@Override` `configure` method. The `configure` method calls `auth.jdbcAuthentication().passwordEncoder(passwordEncoder).dataSource(dataSource).usersByUsernameQuery(...)` and `.authoritiesByUsernameQuery(...)`. A blue arrow points from the text "认证管理器的Builder" to the `auth` parameter in the `configure` method signature.

```
11
12
13 @Configuration
14 public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
15     @Autowired
16     DataSource dataSource;
17     @Override
18     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
19         PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
20         auth.jdbcAuthentication().passwordEncoder(passwordEncoder).dataSource(dataSource)
21             .usersByUsernameQuery("select username,password,enabled from myusers where
22             username= ?")
23             .authoritiesByUsernameQuery("select role, username from roles left join
24             myusers on roles.user_id = myusers.id where username = ?");
25     }
26 }
```

10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法2： 使用JDBC中的用户

下面这个语句：告诉框架当**已知用户名**时，如何能找到对应的**密码等信息**用来验证。

```
usersByUsernameQuery("select username,password,enabled from myusers where username=?")
```

下面这个语句：告诉框架**已知用户名**时，如何能找到它对应的**角色**用来授权

```
.authoritiesByUsernameQuery("select role, username from roles left join myusers on  
roles.user_id = myusers.id where username = ?");
```


10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法3：使用自定义用户服务

步骤1：实现框架用来查找用户的接口UserDetailsService，实现该接口的loadUserByUsername方法，该方法可以根据用户名返回对应用户的UserDetails对象。

```
public interface UserDetailsService {  
    UserDetails loadUserByUsername(String var1) throws UsernameNotFoundException;  
}
```

核心工作：告知框架根据用户名确定用户对象的途径

10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法3： 使用自定义用户服务

步骤2： 实现框架用来提取用户密码、权限、是否禁用、是否锁定等信息的接口UserDetails，实现该接口的一些列方法。

```
public interface UserDetails extends Serializable {  
    Collection<? extends GrantedAuthority> getAuthorities();
```

```
    String getPassword();
```

```
    String getUsername();
```

```
    boolean isAccountNonExpired();
```

核心工作：告知框架怎么从用户对象中读到它需要的属性



```
    boolean isAccountNonLocked();
```

10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法3： 使用自定义用户服务

步骤3： 创建WebSecurityConfig配置类，重写configure方法，将自己实现的UserDetailsService告知框架。

`@Override`

```
protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
    PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();  
    auth.userDetailsService(customUserService).passwordEncoder(passwordEncoder);  
}
```

认证管理器的Builder

注册自己使用的UserDetailsService以及使用的密码编码器

10.2 Spring Security框架实践

<https://gitee.com/buptnetwork/spring-security-userdetails-demo>

Name:

Location:

Language: ☒ Java ☐ Kotlin ☐ Groovy

Type: ☒ Maven ☐ Gradle

Group:

Artifact:

Package name:

Project SDK:

Java:

Packaging: ☒ Jar ☐ War

Added dependencies:

- × Spring Web
- × Thymeleaf
- × Spring Security
- × H2 Database
- × MyBatis Plus Framework
- × Lombok

10.2 Spring Security框架实践

<https://gitee.com/buptnetwork/spring-security-userdetails-demo>

```
DROP TABLE IF EXISTS `authorities`;
DROP TABLE IF EXISTS `users`;
create table users(
    username varchar_ignorecase(50) not null primary key,
    password varchar_ignorecase(500) not null,
    enabled boolean not null
);
create table authorities (
    `id` bigint(20) NOT NULL AUTO_INCREMENT,
    username varchar_ignorecase(50) not null,
    authority varchar_ignorecase(50) not null,
    PRIMARY KEY (`id`),
    constraint fk_authorities_users foreign key(username)
        references users(username)
);
```

schema.sql建表语句，建立用户表和用户权限表

10.2 Spring Security框架实践

<https://gitee.com/buptnetwork/spring-security-userdetails-demo>

```
INSERT INTO `Users` VALUES ('admin', '$2a$10$xAw3A/Dan0.h5QGJGcP0.5L1Y825TUpwUIh.pc9kZUv7vseHGxK2',1);  
INSERT INTO `Authorities` VALUES (1,'admin', 'ROLE_ADMIN');
```

data.sql初始化数据语句，创建了一个用户admin，密码为123456，权限代码为'ROLE_ADMIN'

```
//用户表  
@Data  
public class Users {  
    @TableId(type = IdType.INPUT)  
    private String username;  
  
    private String password;  
    private Boolean enabled;  
  
    public Users(String username, String password, Boolean enabled) {  
        this.username = username;  
        this.password = password;  
        this.enabled = enabled;  
    }  
}
```

```
//用户权限表  
@Data  
public class Authorities {  
    //主键注解,数据库 ID 自增  
    @TableId(type = IdType.AUTO)  
    private Long id;  
  
    private String username;  
    private String authority;  
}
```

和两张表对应的实体类

10.2 Spring Security框架实践

自己实现的UserDetailsService
中的loadUserByUsername方法

根据用户名查user表

根据用户名查authorities表
获得权限

构造出返回的用户详情对象

```
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
    //selectById是按主键搜索, 用户表的主键就是username
    Users user = usersMapper.selectById(username);
    if (user == null) {
        throw new UsernameNotFoundException("用户名不存在");
    }
    //查找当前用户名对应的权限列表
    Map<String, Object> map = new HashMap<>();
    map.put("username", user.getUsername());
    List<Authorities> authoritiesList = authoritiesMapper.selectByMap(map);

    List<GrantedAuthority> auths = new ArrayList<>();
    for(Authorities authorities:authoritiesList){
        auths.add(new SimpleGrantedAuthority(authorities.getAuthority()));
    }
    //构造出返回的用户详情对象
    CustomUserDetails customUserDetails = new CustomUserDetails(user, auths);
    return customUserDetails;
}
```

10.2 Spring Security框架实践

自己实现了UserDetails接口的类

根据用户名查user表

构造函数

返回该用户详情对象的权限

```
public class CustomUserDetails implements UserDetails {  
    private String username;  
    private String password;  
    private Boolean enabled;  
    private List<GrantedAuthority> auths;  
  
    public CustomUserDetails(Users users, List<GrantedAuthority> auths) {  
        this.username = users.getUsername();  
        this.password = users.getPassword();  
        this.enabled = users.getEnabled();  
        this.auths = auths;  
    }  
  
    @Override  
    public Collection<? extends GrantedAuthority> getAuthorities() {  
        return this.auths;  
    }  
}
```


10.2 Spring Security框架实践

自己实现了UserDetails接口的类

用来实现更高级的账号控制
返回false时禁止登录，在本demo中
简化处理，直接返回true

```
@Override
public String getPassword() { return password; }

@Override
public String getUsername() { return username; }

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

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

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

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

10.2 Spring Security框架实践

编写配置类:

```
@Configuration
@EnableWebSecurity
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    CustomUserService customUserService;

    @Override
    public void configure(WebSecurity web) throws Exception {
        //不拦截静态资源
        web.ignoring().antMatchers(...antPatterns: "/js/**", "/css/**", "/img/**");
    }
}
```

10.2 Spring Security框架实践

```
@Override
protected void configure(HttpSecurity http) throws Exception {
    //允许h2-console的frameset界面
    http.headers().headersConfigurer<HttpSecurity>()
        .frameOptions().disable()
        //针对h2-console关闭csrf验证
        .and().csrf().csrfConfigurer<HttpSecurity>()
            .ignoringAntMatchers("/h2-console/**", "/user/**")
            .and().formLogin().formLoginConfigurer<HttpSecurity>()
                .loginPage("/user/login")
                .permitAll()
                .successHandler((httpServletRequest, httpServletResponse, authentication) -> {
                    System.out.println("登陆成功处理=====");
                    //跳转到首页
                    httpServletResponse.sendRedirect(s: "/");
                })
                .failureHandler((httpServletRequest, httpServletResponse, e) -> {
                    System.out.println("登陆失败处理=====");
                    //返回到登陆页面
                    httpServletResponse.sendRedirect(s: "/user/login");
                })
}
```

10.2 Spring Security框架实践

```
.and().logout() LogoutConfigurer<HttpSecurity>  
    .logoutUrl("/user/logout")  
    .logoutSuccessHandler(((HttpServletRequest, HttpServletResponse, authentication) -  
        System.out.println("登出成功处理=====");  
        HttpServletResponse.sendRedirect(s: "/user/login");  
    )))
```

```
.and().authorizeRequests() ExpressionUrlAuthorizationConfigurer<H>.ExpressionInterceptUrlRegistry  
    .antMatchers(...antPatterns: "/h2-console/**", "/user/register").permitAll()  
    .antMatchers(...antPatterns: "/admin/**").hasRole("ADMIN")  
    .anyRequest().authenticated();
```

针对/h2-console/**, 以及注册url可不登录访问

针对/admin/**, 必须具备ADMIN角色才能访问

10.2 Spring Security框架实践

```
@PostMapping("/register")
String registerDone(Model model, String username, String password) {
    Users users = usersMapper.selectById(username);
    if (users != null) {
        model.addAttribute("msg", "用户名已存在!");
        return "user/register";
    } else {
        PasswordEncoder passwordEncoder = new BCryptPasswordEncoder();
        users = new Users(username, passwordEncoder.encode(password), enabled: true);
        usersMapper.insert(users);
        return "redirect:/user/login";
    }
}
```

处理用户注册POST请求的控制器方法

10.2 Spring Security框架实践

@Controller

```
public class IndexController {
```

```
    @GetMapping("/")
```

```
    String index(Model model, Authentication auth){
```

```
        model.addAttribute(s: "username", auth.getName());
```

```
        model.addAttribute(s: "principal", auth.getPrincipal().toString());
```

```
        model.addAttribute(s: "authorities", auth.getAuthorities().toString());
```

```
        return "index";
```

```
    }
```

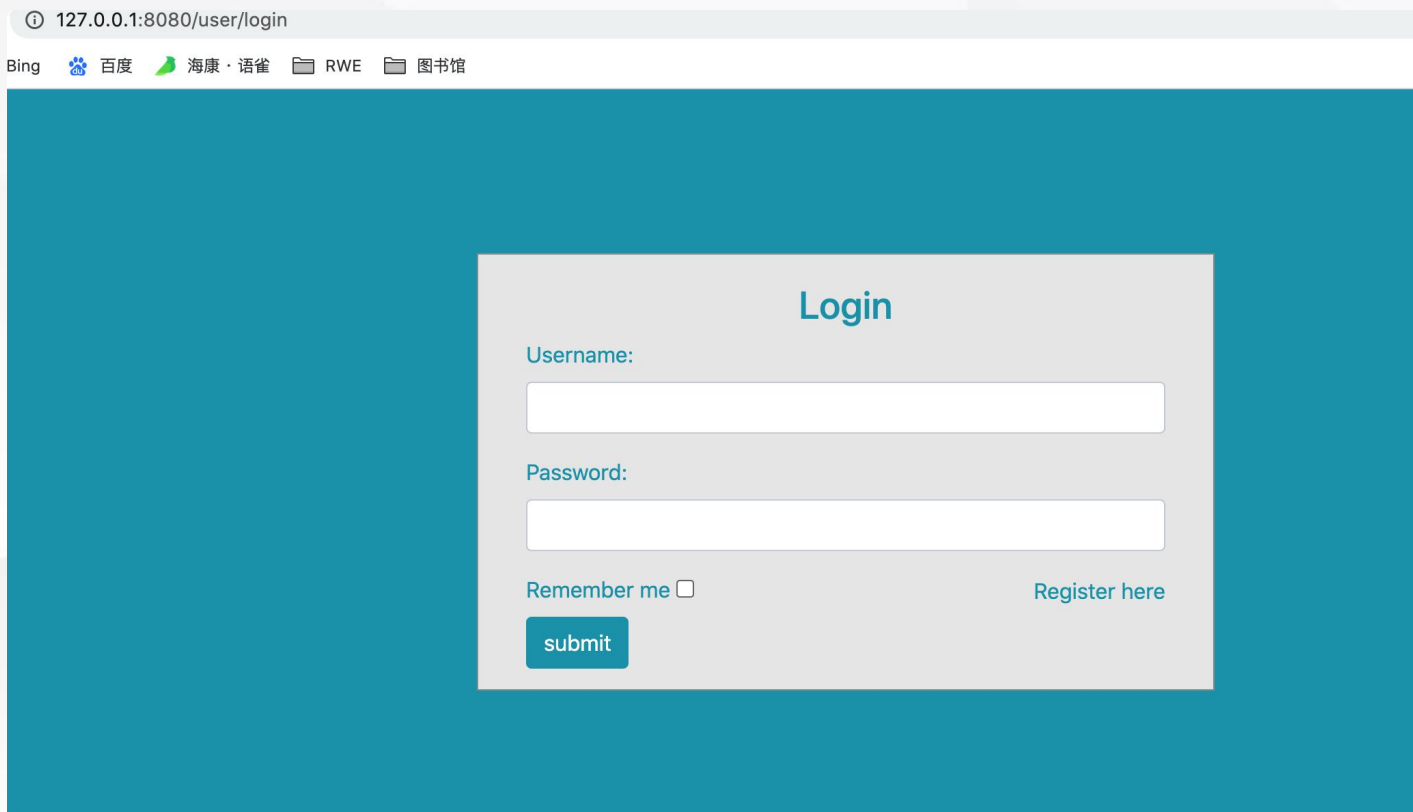
在控制器中获取当前用户信息的示例

10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法3：使用自定义用户服务，运行效果如下：

1) 访问`http://127.0.0.1:8080/`，由于未登陆被重定向到登录页面



127.0.0.1:8080/user/login

Bing 百度 海康·语雀 RWE 图书馆

Login

Username:

Password:

Remember me ☐

[Register here](#)

submit

10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法3： 使用自定义用户服务, 运行效果如下：

2) 点击register here, 进行注册

SignUp

Username:

Password:

10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法3：使用自定义用户服务，运行效果如下：

3) 注册成功后，重定向到登录页面，使用刚注册的账号、密码登录

Login

Username:

Password:

Remember me ☐

[Register here](#)

10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法3：使用自定义用户服务，运行效果如下：

4) 登录成功后进入首页，点击goto admin page，发现403。需要注册一个用户名为admin的用户才具备ADMIN角色，才能访问/admin/**

hello spring security

username: user

principal: cn.edu.bupt.springsecurity.userdetails.demo.bo

authorities: []

[goto admin page](#)

[logout](#)

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this default page.

Mon Dec 14 23:34:27 CST 2020

There was an unexpected error (type=Forbidden, status=403).

Forbidden

10.2 Spring Security框架实践

10.2.2 用户及密码存储

方法3：使用自定义用户服务，运行效果如下：

5) 点击logout，注销登录，重定向到登录页面

hello spring security

username: user

principal: cn.edu.bupt.springsecurity.userdetails.demo.bo.Cu

authorities: []

[goto admin page](#)

logout

Login

Username:

Password:

Remember me ☐

[Register here](#)

10.3 OAuth 2.0认证

<https://www.ruanyifeng.com/blog/2019/04/oauth-grant-types.html>

- OAuth 2.0(开放授权): 为用户资源的授权提供了一个安全的、开放而又简易的标准。OAuth 的核心就是向第三方应用颁发令牌。第三方应用要想使用某个系统提供的OAuth认证服务, 需要先进行备案, 获得客户端 ID (client ID) 和客户端密钥 (client secret)

OAuth2.0标准定义了获得令牌的四种授权方式 (authorization grant)

- 授权码 (authorization-code) ,先申请授权码, 再用该码获取令牌
- 隐藏式 (implicit) : 没有授权码, 直接向前端颁发令牌
- 密码式 (password) : 用户把用户名和密码直接告诉该应用, 应用使用用户名密码申请令牌
- 客户端凭证 (client credentials) : 应用获取不针对特定用户的令牌



A user registration form with the following fields and elements:

- Input field for "你的昵称" (Your Nickname)
- Input field for "手机号" (Mobile Number)
- Input field for "设置密码" (Set Password)
- A green "注册" (Register) button
- A section for "社交帐号直接登录" (Direct login with social accounts) featuring three circular icons: WeChat (green), QQ (blue), and Weibo (red).
- Links at the bottom: "已有帐号登录" (Already have an account, login) and "随便看看" (Browse randomly).

10.3 OAuth 2.0认证

<https://www.ruanyifeng.com/blog/2019/04/oauth-grant-types.html>

➤ 如需使用Gitee提供的OAuth认证服务，需要在以下界面进行应用备案

我的应用 / 应用详情

oauth-test (今日请求次数: 46 次)

应用名称 *

oauth-test

Client ID

df72b56595ae11f9b1a27a056243d69cf30c6c5fbce1c

Client Secret

7a8ab9d12a66ab62abcf3652d717de55817d96b

重置 Client Secret

移除已授权用户的有效 Token

上传 Logo *



JPG或PNG格式，文件大小不超过2M

上传

应用描述

应用描述

应用主页 *

http://127.0.0.1:8080

应用回调地址 * (+)

http://127.0.0.1:8080/login/oauth2/code/gitee

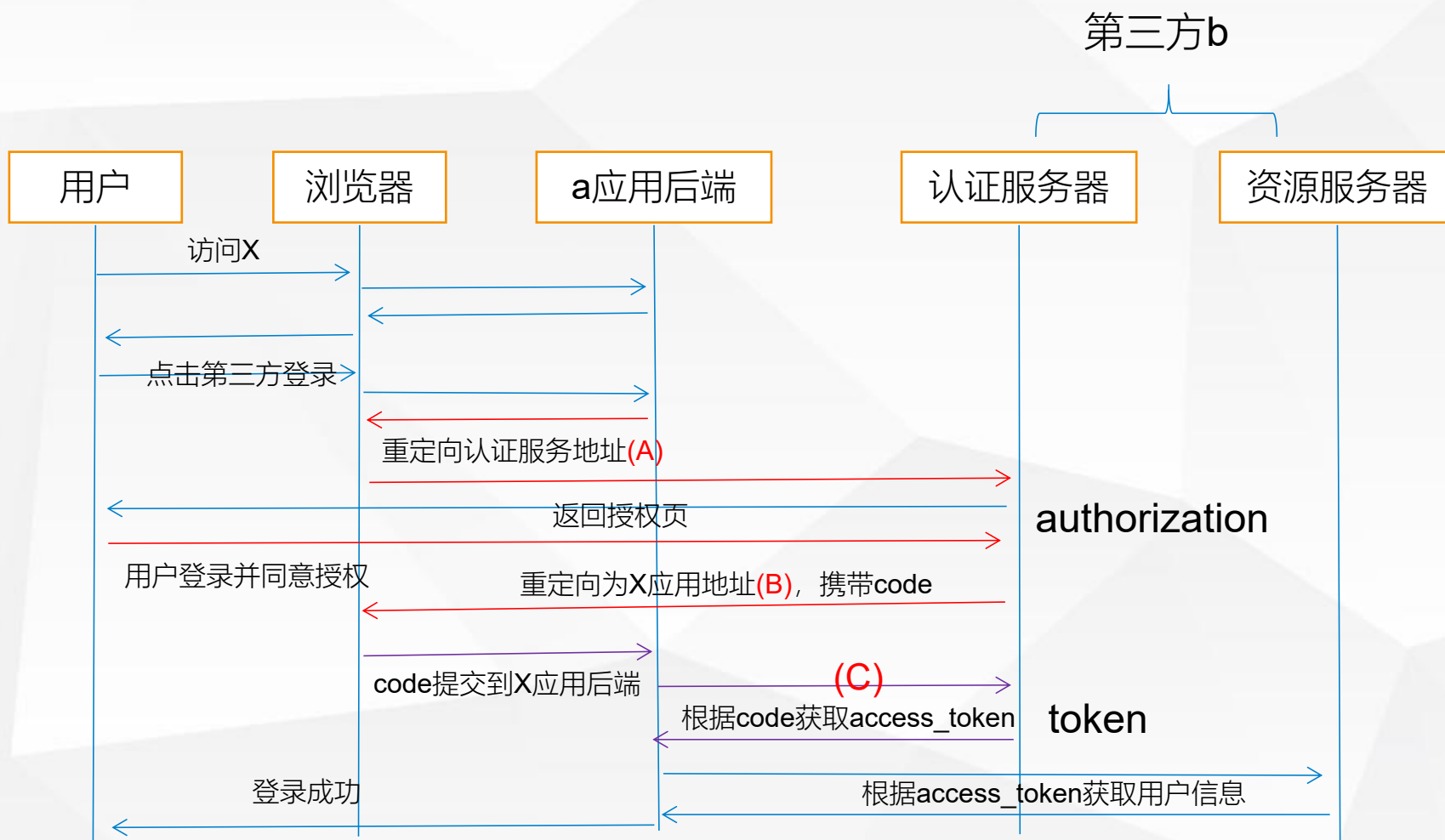
权限 (请慎重选择所需权限，过高的权限用户可能拒绝授权)

☐ 全选

☒ user_info

访问用户的个人信息、最新动态等

10.3 OAuth 2.0认证



A: `https://b.com/oauth/authorize?response_type=code&client_id=CLIENT_ID&redirect_uri=CALLBACK_URL&scope=read`

B: `https://a.com/callback?code=AUTHORIZATION_CODE`

C: `https://b.com/oauth/token?client_id=CLIENT_ID&client_secret=CLIENT_SECRET&grant_type=authorization_code&code=AUTHORIZATION_CODE&redirect_uri=CALLBACK_URL`

10.3 OAuth 2.0认证

Spring OAuth 2.0 Demo

Server URL: start.aliyun.com ⚙️

Name:

Location:

Language: ☒ Java ☐ Kotlin ☐ Groovy

Type: ☒ Maven ☐ Gradle

Group:

Artifact:

Package name:

Project SDK:

Java:

Packaging: ☒ Jar ☐ War

Previous

Next

OAuth2 Client

用于 Spring Security 的 OAuth2/OpenID Connect 客户端功能的 Spring Boot 集成。

<https://gitee.com/buptnetwork/spring-security-oauth-demo>

Added dependencies:

- × Spring Web
- × Spring Security
- × OAuth2 Client

Previous

Finish

10.3 OAuth 2.0认证

Spring OAuth 2.0 Demo

<https://gitee.com/buptnetwork/spring-security-oauth-demo>

```
spring:
  security:
    oauth2:
      client:
        registration:
          gitee:
            clientId: df72b56595ae11f9b1a27a056243d69cf30c6c5fbce1c805de3cecec60bec07d
            clientSecret: 7a8ab9d12a66ab62abcf3652d717de55817d96b4423dbbd5be16d40de36e0233
            redirectUri: http://127.0.0.1:8080/login/oauth2/code/gitee
            authorizationGrantType: authorization_code
        provider:
          gitee:
            authorizationUri: https://gitee.com/oauth/authorize
            tokenUri: https://gitee.com/oauth/token
            userInfoUri: https://gitee.com/api/v5/user
            userNameAttribute: name
```

对OAuth客户端的配置
包括在认证服务器上备案的
clientId和clientSecret

认证服务器返回授权码时的回调

认证并获取code的端点

获取token的端点

获取用户信息的端点

授权类型设置为“授权码”

返回的用户信息中哪个属性是
userName

10.3 OAuth 2.0认证

Spring OAuth 2.0 Demo

<https://gitee.com/buptnetwork/spring-security-oauth-demo>

```
@RestController
```

```
public class IndexController {
```

```
    @GetMapping("/")
```

```
    public String hello(Principal principal,
```

```
                        @RegisteredOAuth2AuthorizedClient OAuth2AuthorizedClient authorizedClient,
```

```
                        @AuthenticationPrincipal OAuth2User oauth2User) {
```

```
        System.out.println(principal.toString());
```

```
        System.out.println(authorizedClient.getClientRegistration().getClientName());
```

```
        System.out.println(oauth2User.getAttributes().toString());
```

```
        return "hello " + principal.getName();
```

```
    }
```

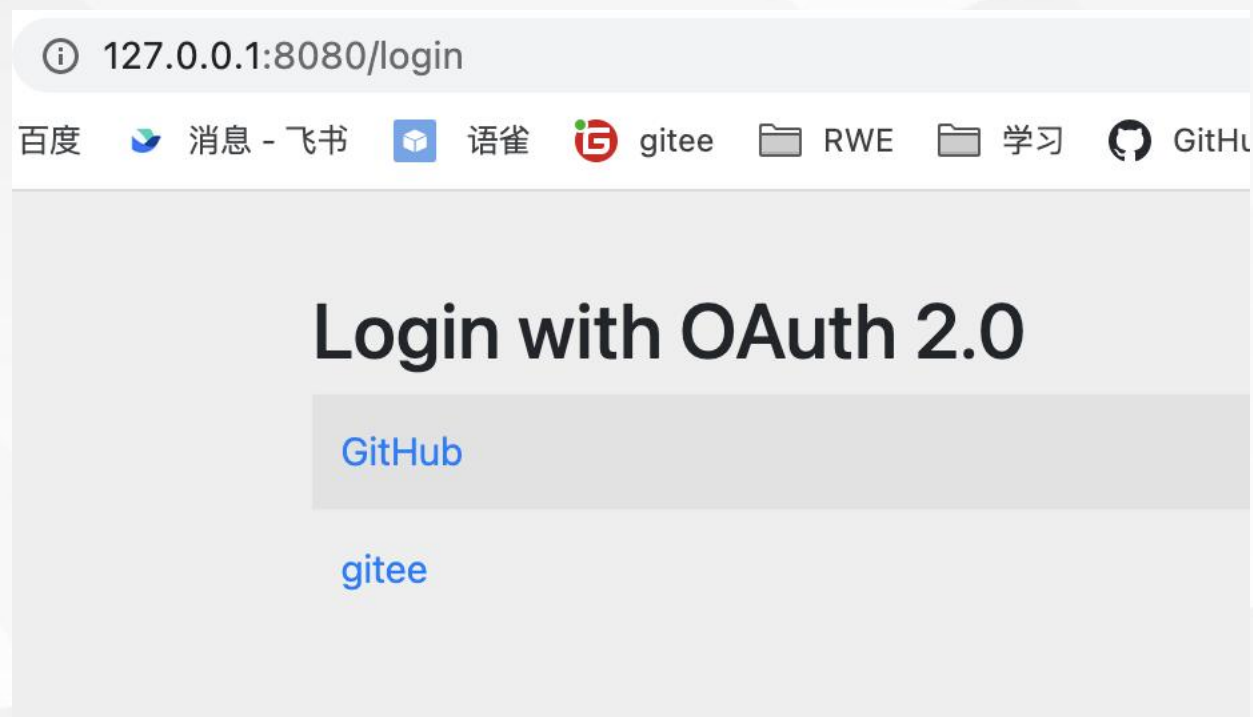
```
}
```

OAuth认证通过后可以进入首页，显示获取到的用户名

10.3 OAuth 2.0认证

Spring OAuth 2.0 Demo

<https://gitee.com/buptnetwork/spring-security-oauth-demo>



登录页面

10.3 OAuth 2.0认证

Spring OAuth 2.0 Demo

<https://gitee.com/buptnetwork/spring-security-oauth-demo>



OAuth授权页面