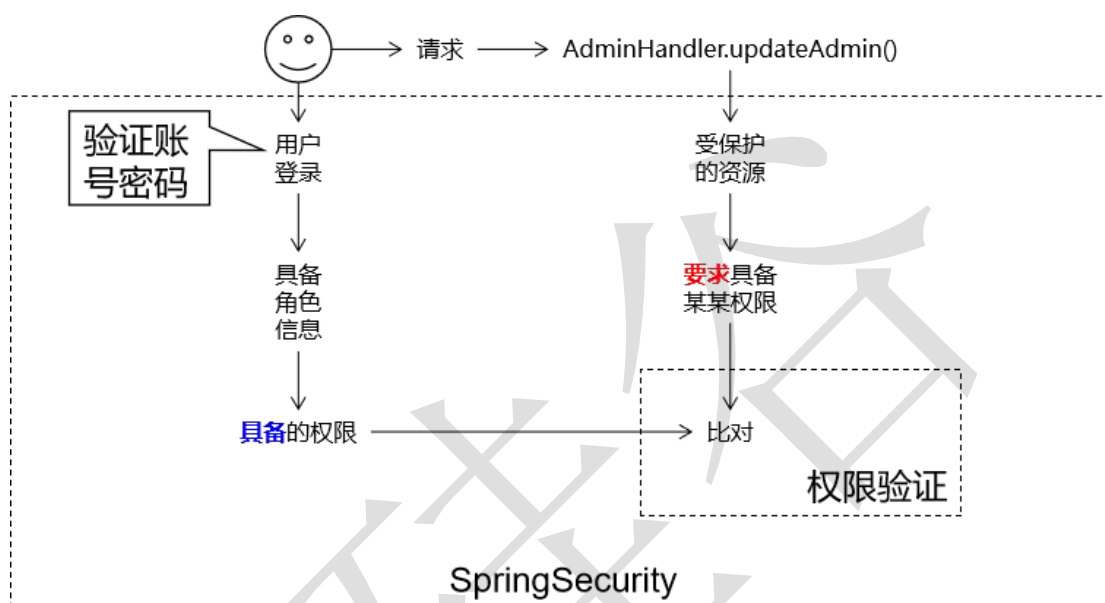


# SpringSecurity

## 1 SpringSecurity 框架用法简介



用户登录系统时我们协助 SpringSecurity 把用户对应的角色、权限组装好，同时把各个资源所要求的权限信息设定好，剩下的“登录验证”、“权限验证”等工作都交给 SpringSecurity。

## 2 权限管理过程中的相关概念

### 2.1 主体

英文单词：principal

使用系统的用户或设备或从其他系统远程登录的用户等等。简单说就是谁使用系统谁就是主体。

### 2.2 认证

英文单词：authentication

权限管理系统确认一个主体的身份，允许主体进入系统。简单说就是“主体”证明自己是谁。

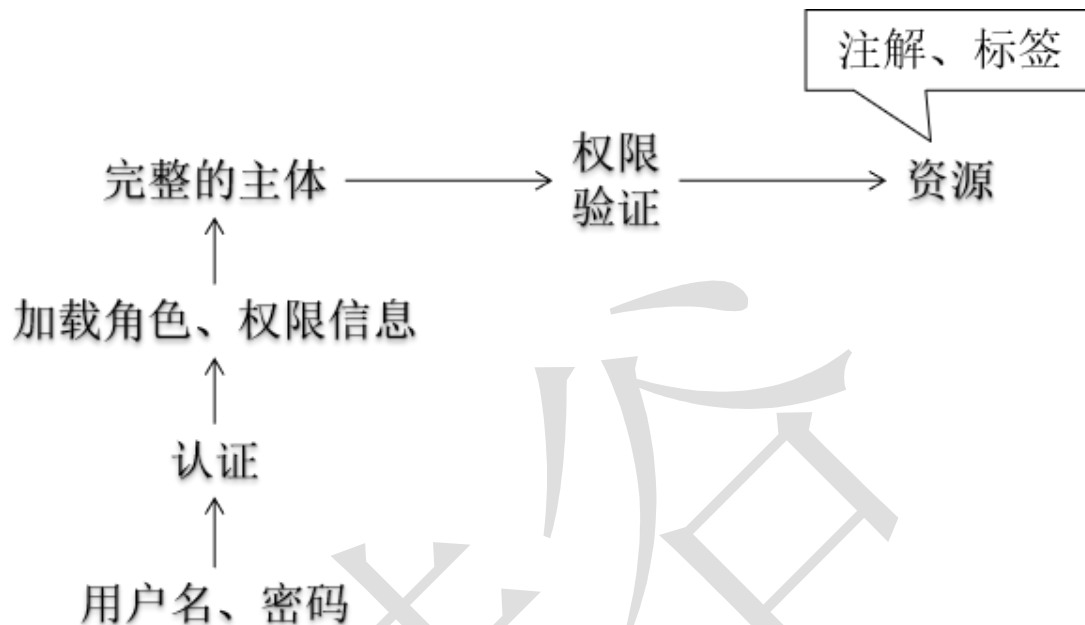
笼统的认为就是以前所做的登录操作。

### 2.3 授权

英文单词：authorization

将操作系统的“权力”“授予”“主体”，这样主体就具备了操作系统中特定功能的能力。

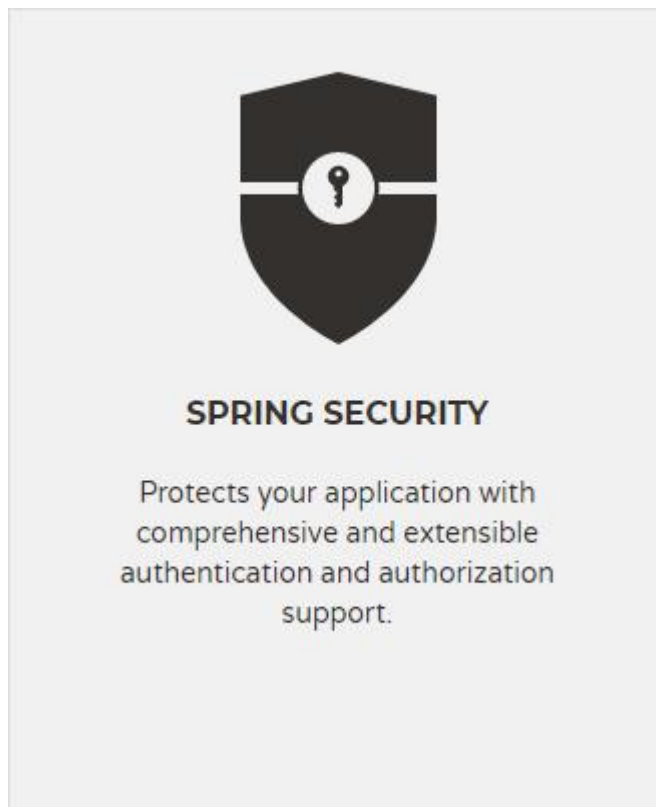
所以简单来说，授权就是给用户分配权限。



## 3 权限管理的主流框架

### 3.1 SpringSecurity

Spring 技术栈的组成部分。



通过提供完整可扩展的认证和授权支持保护你的应用程序。

<https://spring.io/projects/spring-security>

SpringSecurity 特点:

- 和 Spring 无缝整合。
- 全面的权限控制。
- 专门为 Web 开发而设计。
  - 旧版本不能脱离 Web 环境使用。
  - 新版本对整个框架进行了分层抽取，分成了核心模块和 Web 模块。单独引入核心模块就可以脱离 Web 环境。
- 重量级。

### 3.2 Shiro

Apache 旗下的轻量级权限控制框架。



特点:

- 轻量级。Shiro 主张的理念是把复杂的事情变简单。针对对性能有更高要求的互联网应用有更好表现。

- 通用性。
  - 好处：不局限于 Web 环境，可以脱离 Web 环境使用。
  - 缺陷：在 Web 环境下一些特定的需求需要手动编写代码定制。

官网网址：<http://shiro.apache.org/>

学习视频网址：<http://www.gulixueyuan.com/course/45>

## 4 使用配置类代替 XML 配置文件

### 4.1 @Configuration 注解

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Component//由于当前注解带有@Component 注解，所以标记当前注解的类可以享受包的自动扫描
public @interface Configuration {

    /**
     * Explicitly specify the name of the Spring bean definition associated
     * with this Configuration class. If left unspecified (the common case),
     * a bean name will be automatically generated.
     *
     * <p>The custom name applies only if the Configuration class is picked up via
     * component scanning or supplied directly to a {@link
     AnnotationConfigApplicationContext}.
     * If the Configuration class is registered as a traditional XML bean definition,
     * the name/id of the bean element will take precedence.
     *
     * @return the specified bean name, if any
     * @see org.springframework.beans.factory.support.DefaultBeanNameGenerator
     */
    String value() default "";
}
```

类标记了这个注解就可以使用这个类代替 Spring 的 XML 配置文件。

### 4.2 @Bean 注解

用来代替 XML 配置文件中的 bean 标签。下面两种形式效果一致：

```
<bean id="empHandler" class="com.atguigu.component.EmpHandler">
    <property />
</bean>
```

```
</bean>
```

```
@Configuration
```

```
public class AnnotaionConfig{
```

```
    @Bean
```

```
    public EmpHandler getEmpHandler(){
```

```
        return new EmpHandler();
```

```
    }
```

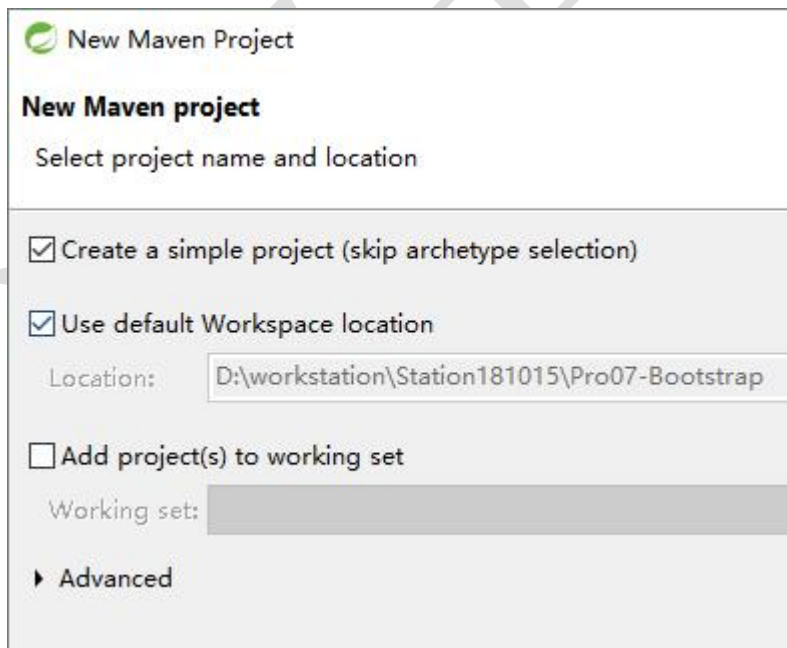
```
}
```

提示: Spring 通过调用标记了 @Bean 注解的方法将对象放入 IOC 容器行为不会重复调用方法。原因是 Spring 想要获取 bean 对应的实例对象时会查看 IOC 容器中是否已经有了这个对象, 如果有则不会执行这个方法, 从而保证这个 bean 是单一实例的。

如果希望对应的 bean 是多实例的, 则可以配合 @Scope 注解。

## 5 HelloWorld 工程创建步骤

### 5.1 创建 Maven 的 Web 工程



New Maven Project

**New Maven project**

Select project name and location

☒ Create a simple project (skip archetype selection)

☒ Use default Workspace location

Location: D:\workstation\Station181015\Pro07-Bootstrap

☐ Add project(s) to working set

Working set:

► Advanced

 New Maven Project**New Maven project**

Configure project

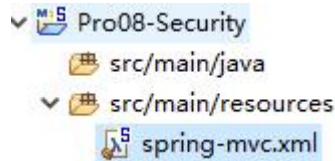
Artifact	
Group Id:	<input type="text" value="com.atguigu.security"/>
Artifact Id:	<input type="text" value="Pro08-Security"/>
Version:	<input type="text" value="0.0.1-SNAPSHOT"/>
Packaging:	<input type="text" value="war"/>

## 5.2 加入 SpringMVC 环境需要的依赖

```
<dependencies>
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>4.3.20.RELEASE</version>
  </dependency>
  <!-- 引入 Servlet 容器中相关依赖 -->
  <dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>servlet-api</artifactId>
    <version>2.5</version>
    <scope>provided</scope>
  </dependency>

  <!-- JSP 页面使用的依赖 -->
  <dependency>
    <groupId>javax.servlet.jsp</groupId>
    <artifactId>jsp-api</artifactId>
    <version>2.1.3-b06</version>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

### 5.3 创建 SpringMVC 配置文件



```
<context:component-scan
    base-package="com.atguigu.security"></context:component-scan>

<bean
    class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/WEB-INF/views/"></property>
    <property name="suffix" value=".jsp"></property>
</bean>

<mvc:annotation-driven></mvc:annotation-driven>
<mvc:default-servlet-handler />
```

### 5.4 在 web.xml 中配置 DispatcherServlet

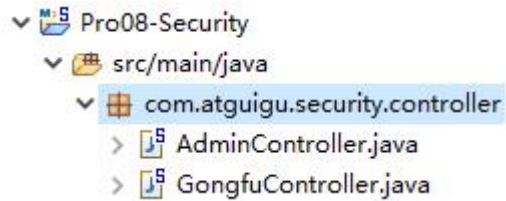
```
<!-- The front controller of this Spring Web application, responsible for handling all application
requests -->
<servlet>
    <servlet-name>springDispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>classpath:spring-mvc.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<!-- Map all requests to the DispatcherServlet for handling -->
<servlet-mapping>
    <servlet-name>springDispatcherServlet</servlet-name>
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

### 5.5 创建包

com.atguigu.security.controller

## 5.6 从例子工程中复制 Controller



## 5.7 加入 webapp 目录下文件



# 6 在 HelloWorld 基础上加入 SpringSecurity

## 6.1 加入 SpringSecurity 依赖

```
<!-- SpringSecurity 对 Web 应用进行权限管理 -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-web</artifactId>
    <version>4.2.10.RELEASE</version>
</dependency>

<!-- SpringSecurity 配置 -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-config</artifactId>
    <version>4.2.10.RELEASE</version>
</dependency>

<!-- SpringSecurity 标签库 -->
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>4.2.10.RELEASE</version>
</dependency>
```



## 6.2 加入 SpringSecurity 控制权限的 Filter

SpringSecurity 使用的是过滤器 Filter 而不是拦截器 Interceptor，意味着 SpringSecurity 能够管理的不仅仅是 SpringMVC 中的 handler 请求，还包含 Web 应用中所有请求。比如：项目中的静态资源也会被拦截，从而进行权限控制。

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

**特别注意：** <filter-name>springSecurityFilterChain</filter-name> 标签中必须是 springSecurityFilterChain。因为 springSecurityFilterChain 在 IOC 容器中对应真正执行权限控制的二十几个 Filter，只有叫这个名字才能够加载到这些 Filter。

## 6.3 加入配置类

```
com.atguigu.security.config.WebAppSecurityConfig

@Configuration
@EnableWebSecurity
public class WebAppSecurityConfig extends WebSecurityConfigurerAdapter {

}
```

Enable 理解为启用。

@EnableWebSecurity 注解表示启用 Web 安全功能。

以后会接触到很多 @EnableXxx 注解，用来启用对应的功能。

## 6.4 效果

- 所有请求都被 SpringSecurity 拦截，要求登录才可以访问。
- 静态资源也都被拦截，要求登录。
- 登录失败有错误提示。

# 7 SpringSecurity 操作实验

下面的操作都是在 HelloWorld 的基础上逐步增加权限控制设置，循序渐进学习 SpringSecurity 用法。

## 7.1 实验 1：放行首页和静态资源

在配置类中重写父类的 `configure(HttpSecurity security)` 方法。

```
protected void configure(HttpSecurity security) throws Exception {  
    logger.debug("Using default configure(HttpSecurity). If subclassed this will potentially  
    override subclass configure(HttpSecurity).");  
  
    security  
        .authorizeRequests()  
            .anyRequest().authenticated()    //所有请求都需要进行认证  
            .and()  
            .formLogin()  
            .and()  
            .httpBasic();  
}
```

重写后

```
@Override  
protected void configure(HttpSecurity security) throws Exception {  
    //super.configure(security); 注释掉将取消父类方法中的默认规则  
  
    security.authorizeRequests()    //对请求进行授权  
        .antMatchers("/layui/**", "/index.jsp")    //使用 ANT 风格设置要授权的 URL 地  
        址  
        .permitAll()    //允许上面使用 ANT 风格设置的全部请求  
        .anyRequest()    //其他未设置的全部请求  
        .authenticated();    //需要认证  
}
```

效果：未认证的请求会跳转到 403 错误页面。

### HTTP Status 403 - Access Denied

**type** Status report

**message** Access Denied

**description** Access to the specified resource has been forbidden.

Apache Tomcat/7.0.57

## 7.2 实验 2：未认证请求跳转到登录页

```
@Override  
protected void configure(HttpSecurity security) throws Exception {  
    //super.configure(security); 注释掉将取消父类方法中的默认规则
```

```

security.authorizeRequests()           //对请求进行授权
    .antMatchers("/layui/**","/index.jsp") //使用 ANT 风格设置要授权的 URL 地址

    .permitAll()                       //允许上面使用 ANT 风格设置的全部请求
    .anyRequest()                       //其他未设置的全部请求
    .authenticated()                   //需要认证
    .and()
    .formLogin()                       //设置未授权请求跳转到登录页面
    .loginPage("/index.jsp")           //指定登录页
    .permitAll();                       //为登录页设置所有人都可以访问
}

```

指定登录页前后 SpringSecurity 登录地址变化：

指定前	/login GET - the login form /login POST - process the credentials and if valid authenticate the user /login?error GET - redirect here for failed authentication attempts /login?logout GET - redirect here after successfully logging out
指定后	/index.jsp GET - the login form /index.jsp POST - process the credentials and if valid authenticate the user /index.jsp?error GET - redirect here for failed authentication attempts /index.jsp?logout GET - redirect here after successfully logging out

#### Impact on other defaults

Updating this value, also impacts a number of other default values. For example, the following are the default values when only formLogin() was specified.

- /login GET - the login form
- /login POST - process the credentials and if valid authenticate the user
- /login?error GET - redirect here for failed authentication attempts
- /login?logout GET - redirect here after successfully logging out

If "/authenticate" was passed to this method it update the defaults as shown below:

- /authenticate GET - the login form
- /authenticate POST - process the credentials and if valid authenticate the user
- /authenticate?error GET - redirect here for failed authentication attempts
- /authenticate?logout GET - redirect here after successfully logging out

通过调用 loginProcessingUrl()方法指定登录地址。

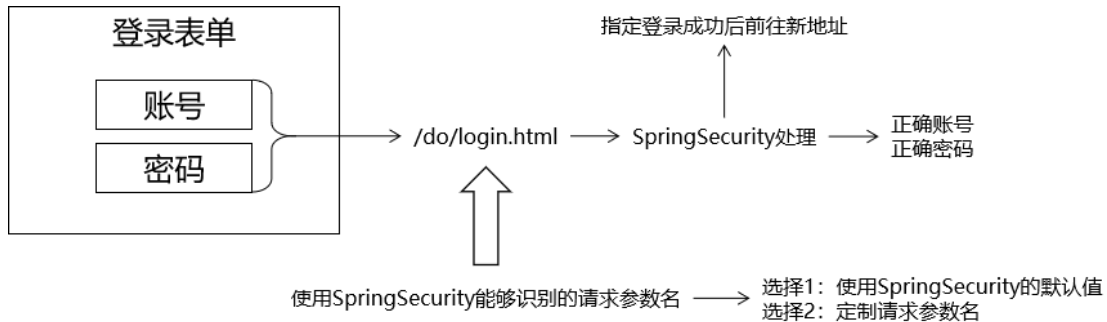
```

security
    .authorizeRequests()           // 对请求进行授权
    .....

    // loginProcessingUrl()方法指定了登录地址，就会覆盖 loginPage()方法中设置的默认值
    /index.jsp POST
    .loginProcessingUrl("/do/login.html") // 指定提交登录表单的地址
    ;

```

### 7.3 实验 3：设置登录系统的账号、密码



#### 7.3.1 页面设置

给 index.jsp 设置表单

```

<p>${SPRING_SECURITY_LAST_EXCEPTION.message}</p>
<form action="${pageContext.request.contextPath }/do/login.html" method="post">
    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
    .....
</form>
    
```

注意：要取消页面的“假”提交。不用管 layui 的语法。

```

/* form.on('submit(LAY-user-login-submit)', function(obj) {
    obj.elem.classList.add("layui-btn-disabled");//样式上的禁用效果
    obj.elem.disabled = true;//真正的禁用效果
    layer.msg("登陆成功，即将跳转");
    setTimeout(function(){
        location.href="main.html";
    }, 2000);
}); */
    
```

账号、密码的请求参数名

SpringSecurity 默认账号的请求参数名：username

SpringSecurity 默认密码的请求参数名：password

要么修改页面上的表单项的 name 属性值，要么修改配置。如果修改配置可以调用 usernameParameter() 和 passwordParameter() 方法。

#### 7.3.2 后端配置

设置登录成功后默认前往的页面

```

@Override
protected void configure(HttpSecurity security) throws Exception {
    //super.configure(security); 注释掉将取消父类方法中的默认规则

    security.authorizeRequests()           //对请求进行授权
        .antMatchers("/layui/**","/index.jsp") //使用 ANT 风格设置要授权的 URL 地
    }
    
```

```
址
    .permitAll()                //允许上面使用 ANT 风格设置的全部请求
    .anyRequest()               //其他未设置的全部请求
    .authenticated()            //需要认证
    .and()
    .formLogin()                //设置未授权请求跳转到登录页面：开启表单登
录功能

    .loginPage("/index.jsp")    //指定登录页
    .permitAll()                //为登录页设置所有人都可以访问
    .loginProcessingUrl("/do/login.html") // 指定提交登录表单的地址
    .usernameParameter("loginAcct") // 定制登录账号的请求参数名
    .passwordParameter("userPswd")  // 定制登录密码的请求参数名
    .defaultSuccessUrl("/main.html"); //设置登录成功后默认前往的 URL 地址
}
```

重写另外一个父类的方法，来设置登录系统的账号密码

```
@Override
protected void configure(AuthenticationManagerBuilder builder) throws Exception {
    //super.configure(auth); 一定要禁用默认规则

    builder.inMemoryAuthentication()
        .withUser("tom").password("123123") //设置账号密码
        .roles("ADMIN")                    //设置角色
        .and()
        .withUser("jerry").password("456456")//设置另一个账号密码
        .authorities("SAVE","EDIT");        //设置权限
}
```

Cannot pass a null GrantedAuthority collection 问题是由于没有设置 roles() 或 authorities()方法导致的。

实现的最后效果：登录成功后具体资源都可以访问了。

### 7.3.3 ※了解：\_csrf 如何防止跨站请求伪造？

Cross-site request forgery 跨站请求伪造

发送登录请求时没有携带\_csrf 值，则返回下面错误：

**HTTP Status 403 - Invalid CSRF Token**  
'null' was found on the request parameter '\_csrf' or header 'X-CSRF-TOKEN'.

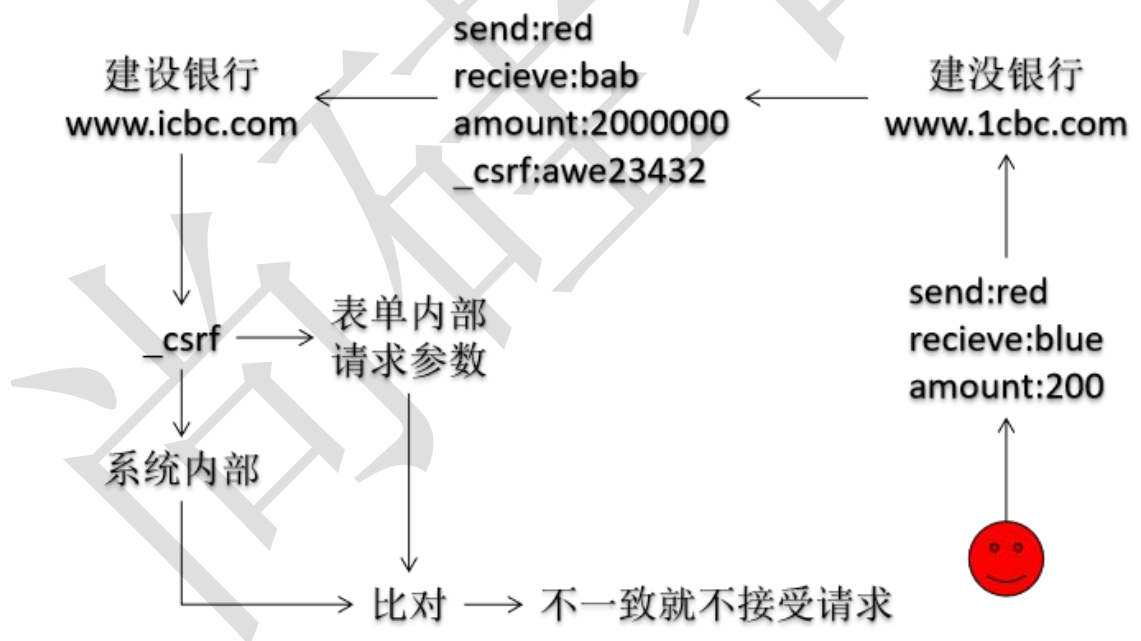
**type** Status report

**message** Invalid CSRF Token 'null' was found on the request parameter '\_csrf' or header 'X-CSRF-TOKEN'.

**description** Access to the specified resource has been forbidden.

Apache Tomcat/7.0.57

从钓鱼网站的页面提交的请求无法携带正确、被承认的令牌。



## 7.4 实验 4：用户注销

通过调用 `HttpSecurity` 对象的一系列方法设置注销功能。

`logout()`方法：开启注销功能

`logoutUrl()`方法：自定义注销功能的 URL 地址



● LogoutConfigurer<HttpSecurity>  
org.springframework.security.config.annotation.web.configurers.LogoutConfigurer.logoutUrl  
(String logoutUrl)

The URL that triggers log out to occur (default is "/logout"). If CSRF protection is enabled (default), then the request must also be a POST. This means that by default POST "/logout" is required to trigger a log out. If CSRF protection is disabled, then any HTTP method is allowed.

It is considered best practice to use an HTTP POST on any action that changes state (i.e. log out) to protect against [CSRF attacks](#). If you really want to use an HTTP GET, you can use `logoutRequestMatcher(new AntPathRequestMatcher(logoutUrl, "GET"))`;

如果 CSRF 功能没有禁用，那么退出请求必须是 POST 方式。如果禁用了 CSRF 功能则任何请求方式都可以。

logoutSuccessUrl()方法：退出成功后前往的 URL 地址

addLogoutHandler()方法：添加退出处理器

logoutSuccessHandler()方法：退出成功处理器

退出的表单

```
<form id="logoutForm" action="${pageContext.request.contextPath}/my/logout"
method="post">
    <input type="hidden" name="${_csrf.parameterName}" value="${_csrf.token}"/>
</form>

<script type="text/javascript">
    window.onload = function(){
        var anchor = document.getElementById("logoutAnchor");
        anchor.onclick = function(){
            document.getElementById("logoutForm").submit();
            return false;
        };
    };
</script>

<a id="logoutAnchor" href="">退出</a>
```

## 7.5 实验 5：基于角色或权限进行访问控制

通过 HttpSecurity 对象设置资源的角色要求

```
security.authorizeRequests() //对请求进行授权
    .antMatchers("/layui/**","/index.jsp") //使用 ANT 风格设置要授权的 URL 地址
    .permitAll() //允许上面使用 ANT 风格设置的全部请求
    .antMatchers("/level1/**")
    .hasRole("学徒")
```

能

```
.antMatchers("/level2/**")
.hasRole("大师")
.antMatchers("/level3/**")
.hasRole("宗师")
.anyRequest()           //其他未设置的全部请求
.authenticated()        //需要认证
.and()
.formLogin()           //设置未授权请求跳转到登录页面：开启表单登录功

.loginPage("/index.jsp") //指定登录页
.permitAll()            //为登录页设置所有人都可以访问
.defaultSuccessUrl("/main.html") //设置登录成功后默认前往的 URL 地址
.and()
.logout()
.logoutUrl("/my/logout")
.logoutSuccessUrl("/index.jsp");
```

通过 AuthenticationManagerBuilder 对象设置用户登录时具备的角色

```
builder.inMemoryAuthentication()
    .withUser("tom").password("123123") //设置账号密码
    .roles("ADMIN","学徒","宗师")      //设置角色
    .and()
    .withUser("jerry").password("456456")//设置另一个账号密码
    .authorities("SAVE","EDIT");        //设置权限
```

访问被拒绝后看到 403 错误页面：

## HTTP Status 403 - Access is denied

**type** Status report

**message** Access is denied

**description** Access to the specified resource has been forbidden.

Apache Tomcat/7.0.57

注意：调用顺序

```
.antMatchers("/level1/**")           //设置匹配/level1/**的地址
.hasRole("学徒")                     //要求具备“学徒”角色
.antMatchers("/level2/**")
.hasRole("大师")
.antMatchers("/level3/**")
.hasRole("宗师")
.anyRequest()                       //其实未设置的所有请求
.authenticated()                     //需要认证才可以访问
```



蓝色代码设置范围更大

红色代码设置范围相对小

如果蓝色代码先调用，会把后面红色代码的设置覆盖，导致红色代码无效。所以要  
先做具体小范围设置，再做大范围模糊设置。

**注意：** SpringSecurity 会在角色字符串前面加 “ROLE\_” 前缀

```
private static String hasRole(String role) {  
    Assert.notNull(role, "role cannot be null");  
    if (role.startsWith("ROLE_")) {  
        throw new IllegalArgumentException(  
            "role should not start with 'ROLE_' since  
            + role + """);  
    }  
    return "hasRole('ROLE_" + role + "')";  
}
```

之所以要强调这个事情，是因为将来从数据库查询得到的用户信息、角色信息、权限信息需要我们自己手动组装。手动组装时需要我们自己给角色字符串前面加 “ROLE\_” 前缀。

## 7.6 实验 6：自定义 403 错误页面

由 main.jsp 复制得到 no\_auth.jsp。修改如下：

```
<div class="layui-body">  
    <!-- 内容主体区域 -->  
    <div style="padding: 15px;">  
        <h1>抱歉！您没有权限访问此功能！ </h1>  
    </div>  
</div>
```

前往自定义页面方式一：

```
@RequestMapping("/to/no/auth/page")  
public String toNoAuthPage() {  
    return "no_auth";  
}
```

HttpSecurity 对象.exceptionHandling().accessDeniedPage("/to/no/auth/page");

前往自定义页面方式二：

```
HttpSecurity 对象.exceptionHandling().accessDeniedHandler(new AccessDeniedHandler() {  
  
    @Override  
    public void handle(HttpServletRequest request, HttpServletResponse response,
```

```

        AccessDeniedException accessDeniedException) throws IOException,
ServletException {
    request.setAttribute("message", accessDeniedException.getMessage());
    request.getRequestDispatcher("/WEB-INF/views/no_auth.jsp").forward(request,
response);
}
});

```

## 7.7 实验 7：记住我-内存版（不重要）

HttpSecurity 对象调用 rememberMe()方法。

登录表单携带名为 remember-me 的请求参数。具体做法是将登录表单中的 checkbox 的 name 设置为 remember-me

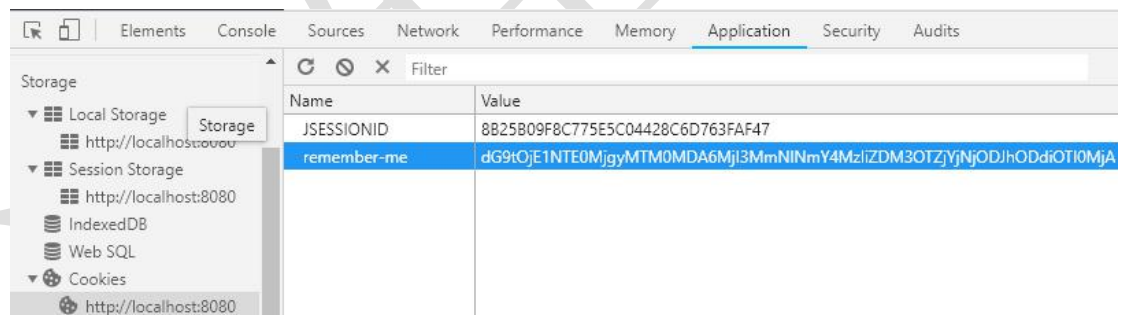
```

<input type="checkbox" name="remember-me"
        lay-skin="primary"
        title="记住密码">

```

如果不能使用 “remember-me” 作为请求参数名称，可以使用 rememberMeParameter()方法定制。

记住我原理简要分析：

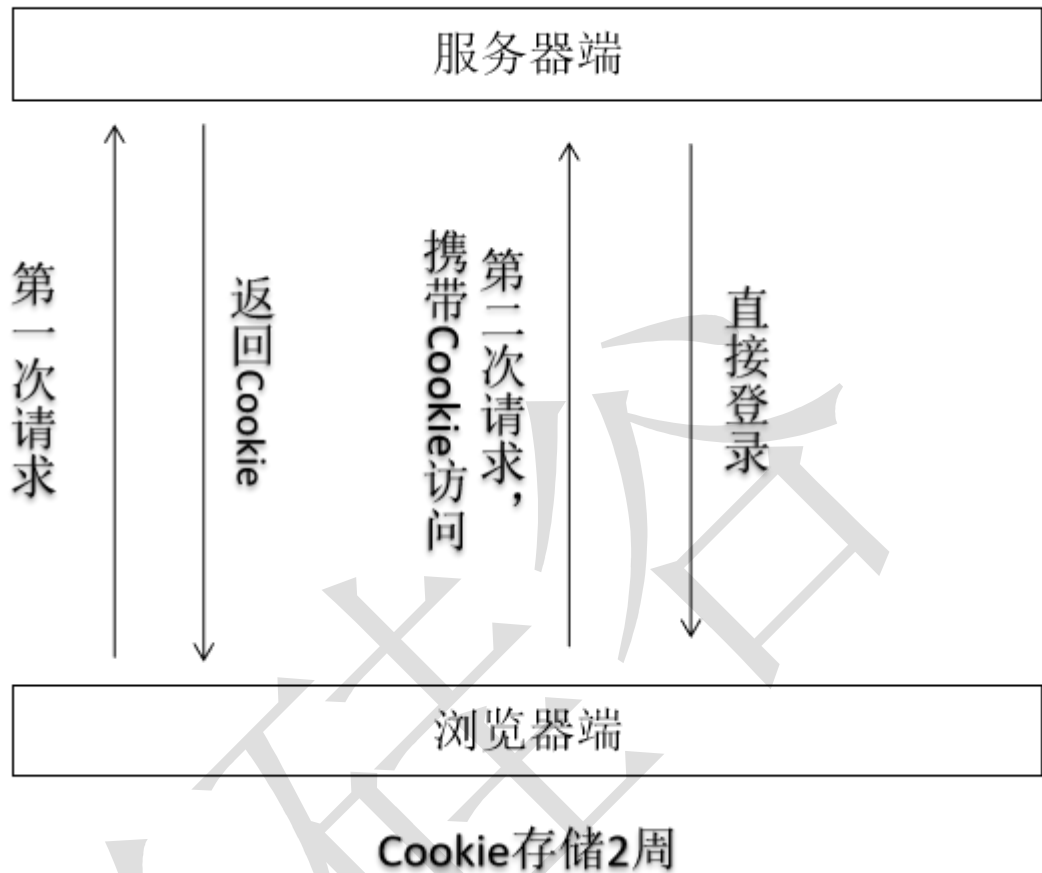


通过开发者工具看到浏览器端存储了名为 remember-me 的 Cookie。根据这个 Cookie 的 value 在服务器端找到以前登录的 User。

Expires / Max-Age
1969-12-31T23:59:59.000Z
2019-03-01T08:16:53.405Z

而且这个 Cookie 被设置为存储 2 个星期的时间。

## 根据Cookie找到对应User



## 7.8 实验 8：记住我-数据库版（不重要）

为了让服务器重启也不影响记住登录状态，将用户登录状态信息存入数据库。

### 7.8.1 建立数据库连接

依赖
<pre> &lt;!-- https://mvnrepository.com/artifact/com.alibaba/druid --&gt; &lt;dependency&gt;   &lt;groupId&gt;com.alibaba&lt;/groupId&gt;   &lt;artifactId&gt;druid&lt;/artifactId&gt;   &lt;version&gt;1.1.12&lt;/version&gt; &lt;/dependency&gt;  &lt;!-- mysql 驱动 --&gt; &lt;!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java --&gt; &lt;dependency&gt;   &lt;groupId&gt;mysql&lt;/groupId&gt;   &lt;artifactId&gt;mysql-connector-java&lt;/artifactId&gt; </pre>

```
<version>5.1.47</version>
</dependency>
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-orm</artifactId>
  <version>4.3.20.RELEASE</version>
</dependency>
```

#### 配置数据源

```
<!-- 配置数据源 -->
<bean id="dataSource" class="com.alibaba.druid.pool.DruidDataSource">
  <property name="username" value="root"></property>
  <property name="password" value="root"></property>
  <property name="url"
value="jdbc:mysql://localhost:3306/security?useSSL=false"></property>
  <property name="driverClassName" value="com.mysql.jdbc.Driver"></property>
</bean>

<!-- jdbcTemplate-->
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
  <property name="dataSource" ref="dataSource"></property>
</bean>
```

#### 创建数据库

```
CREATE DATABASE `security` CHARACTER SET utf8;
```

#### 在 WebAppSecurityConfig 类中注入数据源

```
@Autowired
private DataSource dataSource;
```

### 7.8.2 启用令牌仓库功能



```
JdbcTokenRepositoryImpl repository = new JdbcTokenRepositoryImpl();
repository.setDataSource(dataSource);
```

```
HttpSecurity 对象.tokenRepository(repository);
```

注意：需要进入 JdbcTokenRepositoryImpl 类中找到创建 persistent\_logins 表的 SQL 语句创建

persistent\_logins 表。

```
CREATE TABLE persistent_logins (  
    username VARCHAR (64) NOT NULL,  
    series VARCHAR (64) PRIMARY KEY,  
    token VARCHAR (64) NOT NULL,  
    last_used TIMESTAMP NOT NULL  
);
```

## 7.9 实验 9：查询数据库完成认证

### 7.9.1 了解：SpringSecurity 默认实现

```
builder.jdbcAuthentication().usersByUsernameQuery("tom");
```

在 `usersByUsernameQuery("tom")` 等方法中最终调用 `JdbcDaoImpl` 类的方法查询数据库。

```
public class JdbcDaoImpl extends JdbcDaoSupport  
    implements UserDetailsService, MessageSourceAware {  
    // ~ Static fields/initializers  
    // =====  
  
    public static final String DEF_USERS_BY_USERNAME_QUERY = "select username,password,enabled "  
        + "from users " + "where username = ?";  
    public static final String DEF_AUTHORITIES_BY_USERNAME_QUERY = "select username,authority "  
        + "from authorities " + "where username = ?";  
    public static final String DEF_GROUP_AUTHORITIES_BY_USERNAME_QUERY = "select g.id, g.group_name, ga.authority "  
        + "from groups g, group_members gm, group_authorities ga "  
        + "where gm.username = ? " + "and g.id = ga.group_id "  
        + "and g.id = gm.group_id";
```

SpringSecurity 的默认实现已经将 SQL 语句硬编码在了 `JdbcDaoImpl` 类中。这种情况下，我们有下面三种选择：

- 按照 `JdbcDaoImpl` 类中 SQL 语句设计表结构。
- 修改 `JdbcDaoImpl` 类的源码。
- 不使用 `jdbcAuthentication()`。

### 7.9.2 自定义数据库查询方式

```
builder.userDetailsService(userDetailsService)
```

其中 `userDetailsService` 需要自定义实现 `UserDetailsService` 接口的类并自动装配。

```
com.atguigu.security.service.AppUserDetailService  
  
@Override  
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException  
{  
  
    //1.使用 SQL 语句根据用户名查询用户对象  
    String sql = "SELECT id,loginacct,userpswd,username,email,createtime FROM t_admin  
WHERE loginacct = ?";  
  
    //2.获取查询结果  
    Map<String, Object> resultMap = jdbcTemplate.queryForMap(sql, username);
```

```
//3.获取用户名、密码数据
String loginacct = resultMap.get("loginacct").toString();
String userpswd = resultMap.get("userpswd").toString();

//4.创建权限列表
List<GrantedAuthority> list = AuthorityUtils.createAuthorityList("ADMIN","USER");

//5.创建用户对象
User user = new User(loginacct, userpswd, list);

return user;
}

create table t_admin
(
    id                int not null auto_increment,
    loginacct         varchar(255) not null,
    userpswd          char(32) not null,
    username          varchar(255) not null,
    email             varchar(255) not null,
    createtime        char(19),
    primary key (id)
);
```

### 7.9.3 使用自定义 UserDetailsService 完成登录

```
// builder
//     .inMemoryAuthentication()
//     .withUser("tom")           // 指定登录系统的账号
//     .password("123123")        // 指定账号对应的密码
//     .roles("大师");           // 必须设置角色或权限，否则会出现 Cannot pass a null
//                               GrantedAuthority collection 错误

builder.userDetailsService(userDetailService);
```

### 7.9.4 “ROLE\_” 前缀问题

```
AuthorityUtils.createAuthorityList("ROLE_学徒", "ROLE_大师");
                ^           ^
                |           |
            手动添加前缀   自动加前缀

authorities.add(new SimpleGrantedAuthority(role));
                ^
                |
            "hasRole('ROLE_' + role + '');"
                ^
                |
            在自定义的 UserDetailsService 中，使用
```

org.springframework.security.core.authority.AuthorityUtils.createAuthorityList(String...) 工具方法获取创建 SimpleGrantedAuthority 对象添加角色时需要手动在角色名称前加 “ROLE\_” 前缀。

## 7.10 实验 10：应用自定义密码加密规则

自定义类实现 org.springframework.security.crypto.password.PasswordEncoder（使用没有过时的）接口。

```
com.atguigu.security.service.PasswordEncoderService

@Override
public String encode(CharSequence rawPassword) {
    return CrowdfundingStringUtils.md5(rawPassword.toString());
}

@Override
public boolean matches(CharSequence rawPassword, String encodedPassword) {

    String result = CrowdfundingStringUtils.md5(rawPassword.toString());

    return Objects.equals(result, encodedPassword);
}
```

encode()方法对明文进行加密。

matches()方法对明文加密后和密文进行比较。

```
在配置类中的 configure(AuthenticationManagerBuilder)方法中应用自定义密码加密规则
builder.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);
```

※SpringSecurity 提供的 BCryptPasswordEncoder 加密规则。

BCryptPasswordEncoder 创建对象后代替自定义 passwordEncoder 对象即可。

BCryptPasswordEncoder 在加密时通过加入随机盐值让每一次的加密结果都不同。能够避免密码的明文被猜到。

而在对明文和密文进行比较时，BCryptPasswordEncoder 会在密文的固定位置取出盐值，重新进行加密。



```
// 测试代码
BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

CharSequence rawPassword = "123123";

for(int i = 0; i < 10; i++) {
```



```
String encodedPassword = encoder.encode(rawPassword);

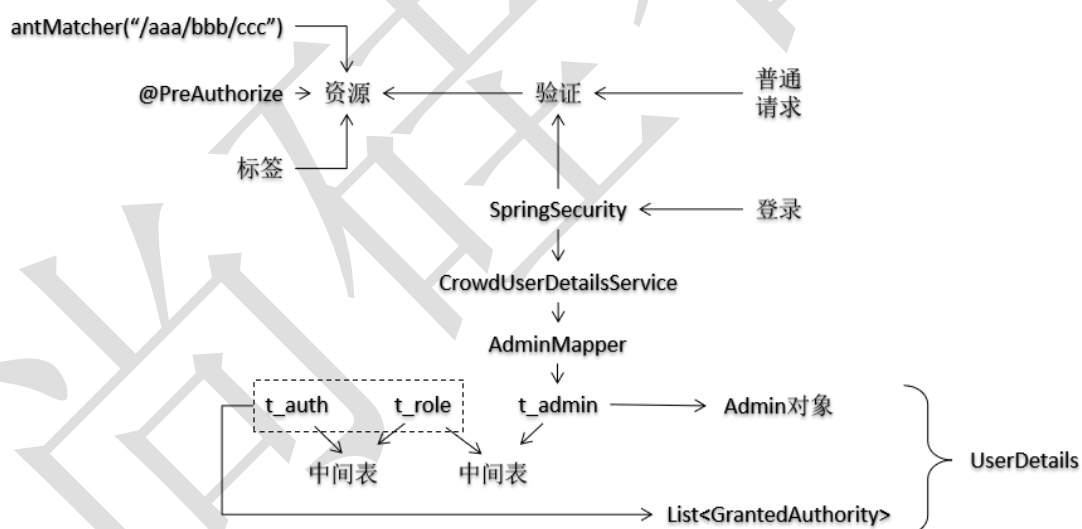
System.out.println(encodedPassword);

}

System.out.println();

boolean matches = encoder.matches(rawPassword,
"$2a$10$Y2Cq8iIT21ME.lvu6bwcPO/RMkU7ucAZpmFzx7GDTXK9KNxHyEM1e");
System.out.println(matches);
```

## 8 众筹项目加入 SpringSecurity 环境



### 8.1 加入依赖

```
<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-web</artifactId>
  <version>4.2.10.RELEASE</version>
</dependency>

<dependency>
  <groupId>org.springframework.security</groupId>
  <artifactId>spring-security-config</artifactId>
  <version>4.2.10.RELEASE</version>
```



```
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-taglibs</artifactId>
    <version>4.2.10.RELEASE</version>
</dependency>
```

## 8.2 Filter

```
<!-- SpringSecurity 的 Filter -->
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>
<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

## 8.3 配置类

```
@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(prePostEnabled=true)
public class CrowdfundingSecurityConfig extends WebSecurityConfigurerAdapter {

}
```

@EnableGlobalMethodSecurity(prePostEnabled=true)注解表示启用全局方法权限管理功能。

## 8.4 自动扫描的包

考虑到权限控制系统更多的需要控制 Web 请求，而且有些请求没有经过 Service 方法，所以在 SpringMVC 的 IOC 容器中扫描 CrowdfundingSecurityConfig。但是，SpringSecurity 是有管理 Service、Dao 方法的能力的。

```
/atcrowdfunding-admin-1-webui/src/main/resources/spring-web-mvc.xml
<context:component-scan
base-package="com.atguigu.crowd.funding.handler,com.atguigu.crowd.funding.exeption,com.atguigu.crowd.funding.config"/>
```

## 8.5 多个 IOC 容器之间的关系

问题描述：项目启动时控制台抛异常说找不到“springSecurityFilterChain”的 bean。

```
org.springframework.beans.factory.NoSuchBeanDefinitionException: No bean named
```

'springSecurityFilterChain' is defined

问题分析：

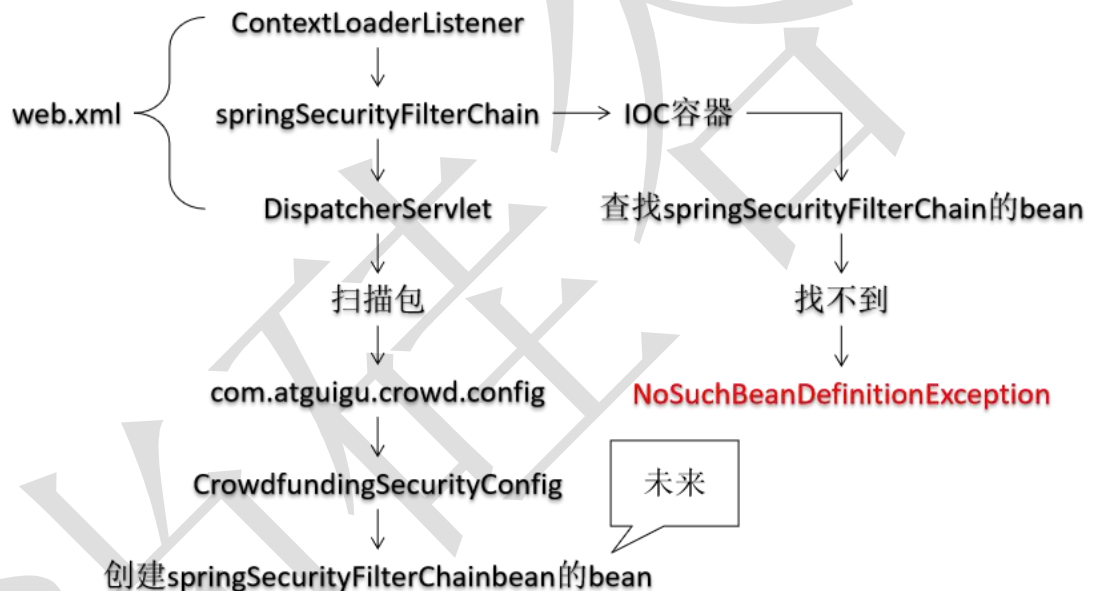
Web 组件加载顺序：Listener→Filter→Servlet

Spring IOC 容器：ContextLoaderListener 创建

SpringMVC IOC 容器：DispatcherServlet 创建

springSecurityFilterChain：从 IOC 容器中找到对应的 bean

ContextLoaderListener 初始化后，springSecurityFilterChain 就在 ContextLoaderListener 创建的 IOC 容器中查找所需要的 bean，但是我们没有在 ContextLoaderListener 的 IOC 容器中扫描 SpringSecurity 的配置类，所以 springSecurityFilterChain 对应的 bean 找不到。



问题解决：

将 ContextLoaderListener 取消，原本由 ContextLoaderListener 读取的 Spring 配置文件交给 DispatcherServlet 负责读取。

```

<!-- 配置 ContextLoaderListener 来加载 Spring 配置文件 -->
<!-- needed for ContextLoaderListener -->
<!-- <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:spring-main-*.xml</param-value>
</context-param> -->

<!-- Bootstraps the root web application context before servlet initialization -->
<!-- <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
</listener> -->
  
```

```
<!-- 配置 DispatcherServlet 来加载 SpringMVC 配置文件 -->
<!-- The front controller of this Spring Web application, responsible for
      handling all application requests -->
<servlet>
    <servlet-name>springDispatcherServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>

        <param-value>classpath:spring-web-mvc.xml,classpath:spring-persist-*.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>

<!-- Map all requests to the DispatcherServlet for handling -->
<servlet-mapping>
    <servlet-name>springDispatcherServlet</servlet-name>
    <!-- <url-pattern>/</url-pattern> -->
    <url-pattern>*.html</url-pattern>
    <url-pattern>*.json</url-pattern>
</servlet-mapping>
```

## 8.6 SpringSecurity 初始设置

放行首页、静态资源。

```
security.authorizeRequests()
    .antMatchers("/index.html", "/bootstrap/**", "/css/**", "/fonts/**", "/img/**", "/jquery/**",
        "/layer/**", "/script/**", "/ztree/**")
    .permitAll()
    .anyRequest()
    .authenticated();
```

# 9 登录

## 9.1 SpringSecurity 开启表单登录功能并前往登录表单页面

```
.formLogin()
.loginPage("/admin/toLoginPage.html")
.permitAll()
```

## 9.2 循环重定向问题



该网页无法正常运作

localhost 将您重定向的次数过多。

[尝试清除 Cookie.](#)

ERR\_TOO\_MANY\_REDIRECTS

去登录页面和登录请求本身都需要 `permitAll()` 否则登录和去登录页面本身都需要登录，形成死循环。

## 9.3 提交登录表单

注意：我们以前自己写的登录 `handler` 方法以后就不使用了。使用 `SpringSecurity` 之后，登录请求由 `SpringSecurity` 处理。

```
security
    .authorizeRequests()
    .antMatchers("/index.html")
    .permitAll()
    .antMatchers("/bootstrap/**")
    .permitAll()
    .antMatchers("/css/**")
    .permitAll()
    .antMatchers("/fonts/**")
    .permitAll()
    .antMatchers("/img/**")
    .permitAll()
    .antMatchers("/jquery/**")
    .permitAll()
    .antMatchers("/layer/**")
    .permitAll()
    .antMatchers("/script/**")
    .permitAll()
    .antMatchers("/ztree/**")
    .permitAll()
    .anyRequest()
    .authenticated()
    .and()
```

```
.formLogin()
.loginPage("/admin/to/login/page.html")
.permitAll()
.loginProcessingUrl("/admin/security/login.html")
.permitAll()
.usernameParameter("loginacct")
.passwordParameter("userpswd")
.defaultSuccessUrl("/admin/to/main/page.html")
.and()
.logout()
.logoutUrl("/admin/security/logout.html")
.logoutSuccessUrl("/index.html")
.and()
.csrf()
.disable();    // 禁用 CSRF 功能
```

//禁用 CSRF 功能。注意：这仅仅是我们学习过程中偷懒的做法，实际开发时还是不要禁用。  
`security.csrf().disable();`

## 9.4 登录操作查询相关数据的 SQL

// 1.根据用户名从数据库查询 Admin 对象

```
AdminExample adminExample = new AdminExample();
```

```
adminExample
    .createCriteria()
    .andLoginacctEqualTo(username);
```

```
List<Admin> adminList = adminMapper.selectByExample(adminExample);
```

```
List<Role> roleList = roleMapper.selectAssignRoleList(adminId);
```

```
<select id="selectAssignedAuthList" resultType="string">
    SELECT
        a.`name`
    FROM
        t_auth a
    LEFT JOIN inner_role_auth ra ON ra.auth_id = a.id
    LEFT JOIN inner_admin_role ar ON ar.role_id = ra.role_id
    WHERE
        ar.admin_id = #{adminId}
        AND a.`name` != ""
</select>
```

## 9.5 SecurityAdmin 封装

```
/**
 * 扩展 User 类
 * 创建 SecurityAdmin 对象时调用构造器，传入 originalAdmin 和 authorities
 * 可以通过 getOriginalAdmin()方法获取原始 Admin 对象
 *
 */
public class SecurityAdmin extends User {

    private static final long serialVersionUID = 1L;

    private Admin originalAdmin;

    public SecurityAdmin(Admin admin,Collection<GrantedAuthority> authorities) {
        super(admin.getLoginacct(), admin.getUserpswd(), true, true, true, true, authorities);

        this.originalAdmin = admin;
    }

    public Admin getOriginalAdmin() {
        return originalAdmin;
    }

}
```

## 9.6 loadUserByUsername(String username)方法

```
@Override
public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException
{

    // 1.根据用户名从数据库查询 Admin 对象
    AdminExample adminExample = new AdminExample();

    adminExample
        .createCriteria()
        .andLoginacctEqualTo(username);

    List<Admin> adminList = adminMapper.selectByExample(adminExample);

    if(adminList == null || adminList.size() != 1) {
        return null;
    }
}
```

```
}

Admin admin = adminList.get(0);

// 2.获取数据库中密码
// String userpswd = admin.getUserpswd();

// 3.查询 Admin 对应的权限信息（包括角色、权限）
Integer adminId = admin.getId();

// ①创建集合用来存放权限信息
Collection<GrantedAuthority> authorities = new ArrayList<>();

// ②根据 adminId 查询对应的角色
List<Role> roleList = roleMapper.selectAssignRoleList(adminId);
for (Role role : roleList) {

    String roleName = role.getName();

    // 注意：一定要加“ROLE_”
    authorities.add(new SimpleGrantedAuthority("ROLE_"+roleName));
}

// ③根据 adminId 查询对应的权限
List<String> authNameList = authMapper.selectAssignedAuthList(adminId);
for (String authName : authNameList) {

    authorities.add(new SimpleGrantedAuthority(authName));
}

// 4.封装到 User 的子类 SecurityAdmin 类型的对象中
// User user = new User(username, userpswd, authorities);
SecurityAdmin securityAdmin = new SecurityAdmin(admin, authorities);

return securityAdmin;
}
```

## 10 认证功能问题调整

### 10.1 取消手动进行登录检查的拦截器

/atcrowdfunding-1-ui/src/main/resources/spring-web.xml

```
<!-- 配置登录拦截器 -->
<!-- 使用了 SpringSecurity 之后，不使用手动进行判断的登录拦截器 -->
<!-- <mvc:interceptors>
    <mvc:interceptor>
        配置要拦截的请求的路径
        <mvc:mapping path="/**"/>

        配置不拦截的请求的路径
        <mvc:exclude-mapping path="/admin/toLoginPage.html"/>
        <mvc:exclude-mapping path="/admin/doLogin/async.json"/>
        <mvc:exclude-mapping path="/admin/logout.html"/>

        登录拦截器类
        <bean
class="com.atguigu.crowd.funding.component.interceptors.LoginInterceptor"/>
        </mvc:interceptor>
    </mvc:interceptors> -->
```

### 10.2 登录成功后显示实际登录的用户名

第一步：导入 SpringSecurity 标签库

```
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="security" %>
```

第二步：使用 security:authentication 标签

```
<!-- 使用 SpringSecurity 提供的 JSP 标签获取已登录用户的用户名 -->
```

```
<security:authentication property="name"/>
```

```
<!-- 通过访问当前对象的 principal.originalAdmin.userName 属性可以获取用户的昵称 -->
```

```
<security:authentication property="principal.originalAdmin.userName"/>
```

### 10.3 加入关联关系假数据

页面操作或者直接将数据插入到数据库中即可。

### 10.4 保存 Admin 时使用 SpringSecurity 加密方式

```
@Autowired
```

```
private BCryptPasswordEncoder passwordEncoder;
```



```
@RequestMapping("/admin/save")
public String saveAdmin(Admin admin) {

    String userpswd = admin.getUserpswd();

    userpswd = passwordEncoder.encode(userpswd);

    admin.setUserpswd(userpswd);

    adminService.saveAdmin(admin);

    return "redirect:/admin/query.html?pageNo="+Integer.MAX_VALUE;
}
```

## 11 权限控制

### 11.1 handler 方法的权限控制

```
linda:
    ROLE_总裁
    role:get
peiqi:
    ROLE_经理
    user:get
```

需要进行权限控制的 handler 方法

```
com.atguigu.crowd.handler.AdminHandler

@PreAuthorize(value="hasRole('PM - 项目经理')")
@RequestMapping("/admin/query")
public String queryWithSearch(
    @RequestParam(value="keyword", defaultValue="") String keyword,
    @RequestParam(value="pageNo", defaultValue="1") int pageNo,
    Model model
){

    // 1.调用 Service 方法获取分页数据
    PageInfo<Admin> pageInfo = adminService.getAdminPageInfoWithKeyword(keyword,
pageNo, ArgumentsConstant.PAGE_SIZE);
```

```
// 2.将分页数据存入模型
model.addAttribute(AttrNameConstant.PAGE, pageInfo);

// 3.跳转页面
return "admin_page";
}
```

注 意 : @PreAuthorize 注解生效需要 @EnableGlobalMethodSecurity(prePostEnabled=true)注解支持。

## 11.2 使用全局配置控制

```
.antMatchers("/admin/query/for/search.html")
.hasRole("董事长")

.....

.and()
.exceptionHandling()
.accessDeniedHandler(new CrowdFundingAccessDeniedHandler())
```

accessDeniedHandler()方法指定了检测到权限不匹配时的处理方式。

```
public class CrowdFundingAccessDeniedHandler implements AccessDeniedHandler {

    @Override
    public void handle(HttpServletRequest request, HttpServletResponse response,
        AccessDeniedException accessDeniedException) throws IOException,
        ServletException {
        request.setAttribute("exception", accessDeniedException);
        request.getRequestDispatcher("/WEB-INF/system-error.jsp").forward(request,
        response);
    }
}
```

## 11.3 页面元素权限控制

使用 SpringSecurity 提供的标签可以详细对页面元素进行权限控制。

第一步：导入标签库

```
<%@ taglib uri="http://www.springframework.org/security/tags" prefix="security" %>
```

第二步：使用 security:authorize 标签

```
<security:authorize access="hasRole('经理')">
    <a href="assign/to/assign/role/page/${admin.id }.html" class="btn btn-success btn-xs">
        <i class="glyphicon glyphicon-check"></i>
    </a>
```

```
</security:authorize>
```