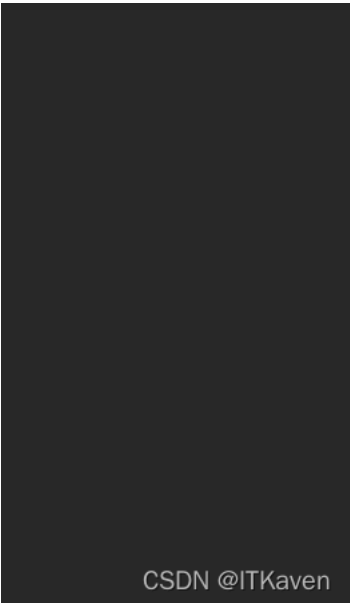


5

件中定义的用户，也可以是数据源中存储的用户，还可以是 Spring Security 自动创建的用户（ Spring Security 在没有用户或用户源相关配置时会自动创

`http://maven.apache.org/xsd/maven-4.0.0.xsd">`





ls;

sion;
ls;

/CoreVersion.SERIAL_VERSION_UID;

```
    s() {
```

的 `UserDetails` 实例。

`setter`、`getter` 代码)

```
    sion;
```



```
oderFactories;  
oder;  
  
{  
  
/CoreVersion.SERIAL_VERSION_UID;  
  
lass);
```

```
) {  
ities);  
  
led,  
cpired,  
itedAuthority> authorities) {  
  
sword == null)) {  
  
uctor");
```

```
orities(authorities));
```

```
{  
) {  
dAuthority collection");  
eSet<>(  
  
[  
  
null elements");
```

```
◁pired())
```



```
password -> password;
```

为每个角色条目添加“ROLE_”前缀

```
<>(
```

```
        role
        自動的に added)");
        .E_ " + role));
```

```
        authorities) {
```

```
        grantedAuthority> authorities) {
```

```
        {
            list(authorities));
```

```
    }
```

```
        /password);
        id, !accountExpired,
        vities);
```



```
ialsNonExpired, accountNonExpired: true credentialsNonExpired: true
ends GrantedAuthority> authorities) { accountNonLocked: true authorities: size = 2

) || (password == null)) {

to constructor");

username: "kaven"
tkaven" password: "{noop}itkaven"
led: true
accountNonExpired: true accountNonExpired: true
ired; credentialsNonExpired: true credentialsNonExpired: true
countNonLocked: true accountNonLocked: true
(sortAuthorities(authorities)); authorities: size = 2 authorities: size = 2
```

Variables

- this = (User@5207) Object is being initialized
 - password = "{noop}itkaven"
 - username = "kaven"
 - authorities = (Collections\$UnmodifiableSet@5214) size = 2
 - 0 = (SimpleGrantedAuthority@5216) "ROLE_ADMIN"
 - 1 = (SimpleGrantedAuthority@5217) "ROLE_USER"
 - accountNonExpired = true
 - accountNonLocked = true
 - credentialsNonExpired = true
 - enabled = true
- username = "kaven"
- password = "{noop}itkaven"
- enabled = true
- accountNonExpired = true
- credentialsNonExpired = true
- accountNonLocked = true
- authorities = (ArrayList@5210) size = 2
- this.authorities = (Collections\$UnmodifiableSet@5214) size = 2

in) , 是因为在创建 User 实例之前, 密码已经在 UserDetailsServiceAutoConfiguration 类的 getOrDeducePassword 方法中被修改了 (加 {noop} 前缀)。

```
der encoder) {
```

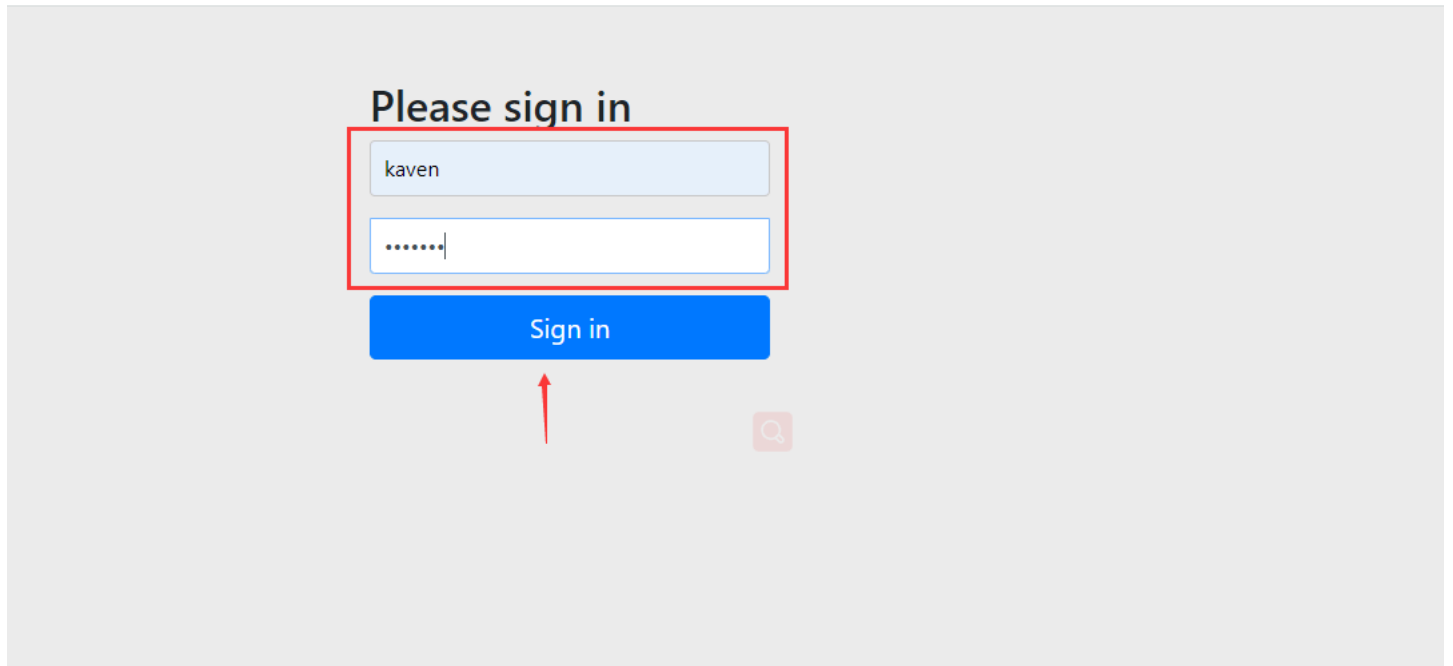
```
arity password: %s%n", user.getPassword());
```

```
natcher(password).matches() ? "{noop}" + password : password;
```

.E_ 前缀), 很显然是调用了 UserBuilder 类的 roles 方法 (在 UserDetailsServiceAutoConfiguration 类的 inMemoryUserDetailsManager 方法中被调用)。

```
er(SecurityProperties properties, ObjectProvider<PasswordEncoder> passwordEncoder) {
```

```
||org.springframework.security.core.userdetails.User.withUsername(user.getName()).password(this.getOrDeducePassword(user, (PasswordEncoder)p;
```

!，相当于获取用户的媒介，因此可以基于内存、数据库等方式来实现，下图的 `InMemoryUserDetailsManager` 类，是一种基于内存的用户服务（查找是否存在实例的委托对象正是应用启动时 `Spring Security` 基于配置文件创建的 `User` 实例）创建新的 `User` 实例。

The screenshot displays the following components:

- Source Code:**

```

loadUserByUsername(username);
authenticationManager.authenticate(authentication);
createUserDetails(username, password, user.isEnabled(),
    credentialsNonExpired(),
    authorities());

```
- Annotations:**
 - Red box around `loadUserByUsername(username);`: 根据用户名获取已经创建的用户
 - Red box around `authenticationManager.authenticate(authentication);`: 没有该用户名的用户
 - Red box around `createUserDetails(...)`: 根据获取的用户创建新的UserDetails实例 (User实例)
- Variables Panel:**
 - `this = (InMemoryUserDetailsManager@6792)`
 - `username = "kaven"`
 - `user = (MutableUser@6794)`
 - `password = "(noop)itkaven"`
 - `delegate = (User@5207)` (highlighted with a red box and an arrow pointing to the text "应用启动时Spring Security创建的用户实例 (User@5207)")
 - `password = "(noop)itkaven"`
 - `username = "kaven"`
 - `authorities = (Collections$UnmodifiableSet@5214) size = 2`
 - `accountNonExpired = true`
 - `accountNonLocked = true`
 - `credentialsNonExpired = true`
 - `enabled = true`



```
ialsNonExpired, credentialsNonExpired: true accountNonExpired: true
ends GrantedAuthority> authorities) { authorities: size = 2 accountNonLocked: true

) || (password == null)) {

to constructor");

    username: "kaven"
    password: "{noop}itkaven"
    enabled: true
    accountNonExpired: true accountNonExpired: true
    credentialsNonExpired: true credentialsNonExpired: true
    accountNonLocked: true accountNonLocked: true
    (sortAuthorities(authorities)); authorities: size = 2 authorities: size = 2
```

Variables

- > this = (User@6192) Object is being initialized
- > username = "kaven"
- > password = "{noop}itkaven"
- > enabled = true
- > accountNonExpired = true
- > credentialsNonExpired = true
- > accountNonLocked = true
- > authorities = (Collections\$UnmodifiableSet@5211) size = 2
- > this.authorities = (Collections\$UnmodifiableSet@6193) size = 2

密码目前很显然还不匹配 ({noop}itkaven 和 itkaven) 。

The screenshot shows the 'Variables' window in an IDE, displaying the state of variables during a Spring Security authentication process. The 'userDetails' variable is highlighted, showing its structure as a `User@611b289` object. It contains fields for `password`, `username`, `authorities`, `accountNonExpired`, `accountNonLocked`, `credentialsNonExpired`, and `enabled`. The `authentication` variable is also shown, containing a `UsernamePasswordAuthenticationToken@6208` object with fields for `principal`, `credentials`, `authorities`, `details`, and `authenticated`. The `passwordEncoder` variable is shown as a `DelegatingPasswordEncoder@7071` object, which contains a `passwordEncoderForEncode` and a `passwordEncoderForMatch` map. The `passwordEncoderForMatch` map contains various password encoders like `noop`, `pbkdf2`, `sha1`, `sha256`, `ldap`, `md4`, `md5`, `bcrypt`, and `argon2`.

Variables

- `userDetails = (User@611b289) "org.springframework.security.core.userdetails.User@611b289: Username: kaven; Password: [REDACTED]; authorities=[ROLE_USER]; accountNonExpired=true; accountNonLocked=true; credentialsNonExpired=true; enabled=true"`
 - `password = "(noop)itkaven"`
 - `username = "kaven"`
 - `authorities = (Collections$UnmodifiableSet@6193) size = 2`
 - `accountNonExpired = true`
 - `accountNonLocked = true`
 - `credentialsNonExpired = true`
 - `enabled = true`
- `authentication = (UsernamePasswordAuthenticationToken@6208) "org.springframework.security.authentication.UsernamePasswordAuthenticationToken@6208: Principal: kaven; Credentials: [REDACTED]; Authorities: []; Details: (WebAuthenticationDetails@7086) "org.springframework.security.web.authentication.WebAuthenticationDetails@7086: RemoteIpAddress: null; RemotePort: null; SessionId: null; authenticated=false"`
 - `principal = "kaven"`
 - `credentials = "itkaven"`
 - `authorities = (Collections$EmptyList@7085) size = 0`
 - `details = (WebAuthenticationDetails@7086) "org.springframework.security.web.authentication.WebAuthenticationDetails@7086: RemoteIpAddress: null; RemotePort: null; SessionId: null; authenticated=false"`
 - `authenticated = false`
- `passwordEncoder = (DelegatingPasswordEncoder@7071) "org.springframework.security.crypto.password.DelegatingPasswordEncoder@7071: supportedEncoders={noop, pbkdf2, sha1, sha256, ldap, md4, md5, bcrypt, argon2}; defaultPasswordEncoder=org.springframework.security.crypto.password.DefaultPasswordEncoder$HmacSha256PasswordEncoder@7090"`
 - `idForEncode = "bcrypt"`
 - `passwordEncoderForEncode = (BCryptPasswordEncoder@7095) "org.springframework.security.crypto.password.BCryptPasswordEncoder@7095"`
 - `passwordEncoderForMatch = (HashMap@7096) size = 11`
 - `"argon2" -> (Argon2PasswordEncoder@7117) "org.springframework.security.crypto.password.Argon2PasswordEncoder@7117"`
 - `"noop" -> (NoOpPasswordEncoder@7119) "org.springframework.security.crypto.password.NoOpPasswordEncoder@7119"`
 - `"pbkdf2" -> (Pbkdf2PasswordEncoder@7121) "org.springframework.security.crypto.password.Pbkdf2PasswordEncoder@7121"`
 - `"sha1" -> (MessageDigestPasswordEncoder@7123) "org.springframework.security.crypto.password.MessageDigestPasswordEncoder@7123"`
 - `"sha256" -> (StandardPasswordEncoder@7125) "org.springframework.security.crypto.password.StandardPasswordEncoder@7125"`
 - `"ldap" -> (LdapShaPasswordEncoder@7127) "org.springframework.security.crypto.password.LdapShaPasswordEncoder@7127"`
 - `"md4" -> (Md4PasswordEncoder@7131) "org.springframework.security.crypto.password.Md4PasswordEncoder@7131"`
 - `"md5" -> (MessageDigestPasswordEncoder@7135) "org.springframework.security.crypto.password.MessageDigestPasswordEncoder@7135"`
 - `"bcrypt" -> (BCryptPasswordEncoder@7095) "org.springframework.security.crypto.password.BCryptPasswordEncoder@7095"`
 - `"argon2" -> (Argon2PasswordEncoder@7117) "org.springframework.security.crypto.password.Argon2PasswordEncoder@7117"`

截取成 `itkaven`，这样密码就匹配了，Spring Security 提供了多种密码编码器（根据不同的应用场景），以后博主会详细介绍，因此不要以为密码匹配



```
ing prefixEncodedPassword) {  
    if (password == null) {  
        return null;  
    }  
    User user = userDao.getById(id); idToPasswordEncoder: size = 11  
    PasswordEncoder passwordEncoder = user.getPasswordEncoder();  
    if (passwordEncoder instanceof DelegatingPasswordEncoder$UnmappedIdPasswordEncoder@7055) {  
        return passwordEncoder.encode(password);  
    }  
    return passwordEncoder.encode(prefixEncodedPassword);  
}
```

Variables

- > this = (DelegatingPasswordEncoder@7034)
- > rawPassword = "itkaven"
- > prefixEncodedPassword = "{noop}itkaven"
- > id = "noop"
- > delegate = (NoOpPasswordEncoder@7050)
- > encodedPassword = "itkaven"

Build Python Packages Spring

```
password) {
```



```
edAuthority</code> collection

ean enabled, username: "user" password: "{noop}813f4852-9a13-4173-a578-91a2fe6c6c3e" enabled: true
ialsNonExpired, accountNonExpired: true credentialsNonExpired: true
ends GrantedAuthority> authorities) { accountNonLocked: true authorities: size = 0

) || (password == null)) {

to constructor");

username: "user"
13f4852-9a13-4173-a578-91a2fe6c6c3e" password: "{noop}813f4852-9a13-4173-a578-91a2fe6c6c3e"
led: true
accountNonExpired: true accountNonExpired: true
ired; credentialsNonExpired: true credentialsNonExpired: true
countNonLocked: true accountNonLocked: true
(sortAuthorities(authorities)); authorities: size = 0 authorities: size = 0
```

Variables

```
this = (User@5036) Object is being initialized
> password = "{noop}813f4852-9a13-4173-a578-91a2fe6c6c3e"
> username = "user"
> authorities = (Collections$UnmodifiableSet@5044) size = 0
> accountNonExpired = true
> accountNonLocked = true
> credentialsNonExpired = true
> enabled = true

> username = "user"
> password = "{noop}813f4852-9a13-4173-a578-91a2fe6c6c3e"
> enabled = true
> accountNonExpired = true
> credentialsNonExpired = true
> accountNonLocked = true
> authorities = (ArrayList@5039) size = 0
this.authorities = (Collections$UnmodifiableSet@5044) size = 0
```

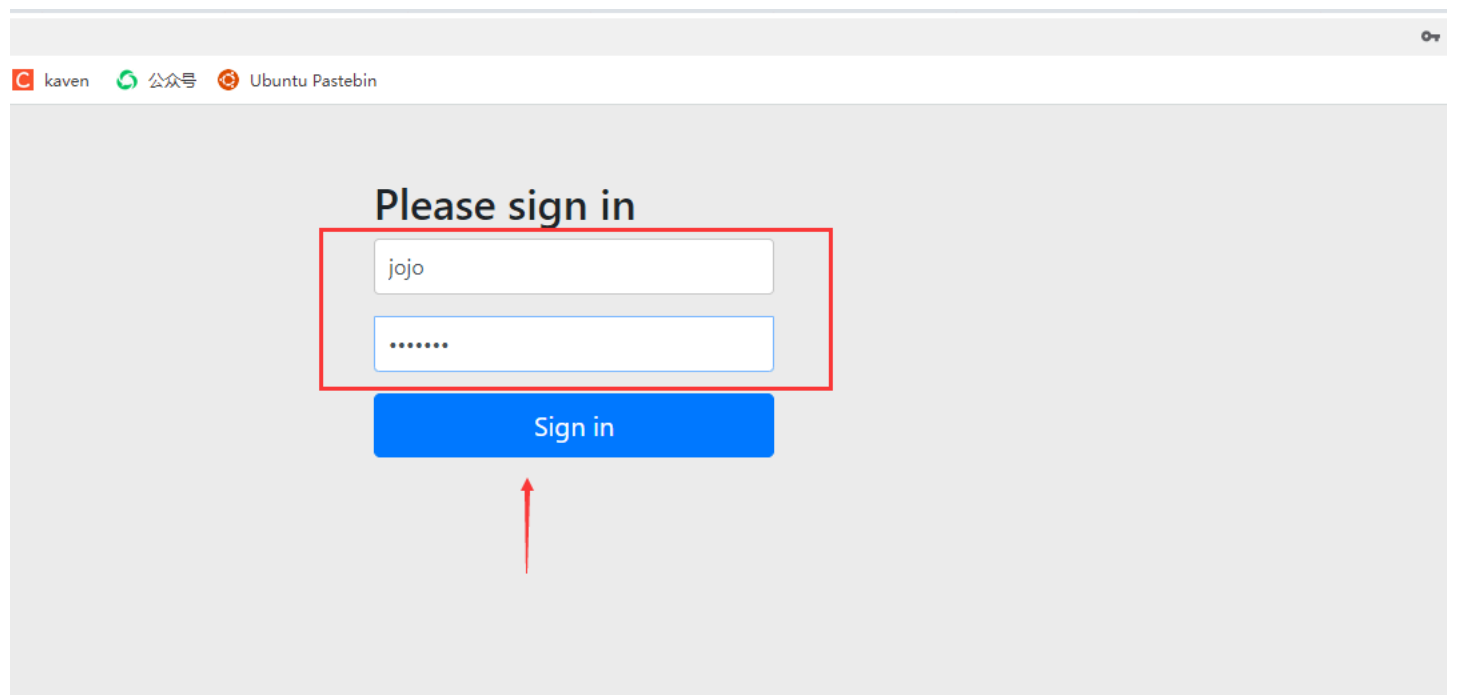
```
verTimezone=Asia/Shanghai&useUnicode=true&characterEncoding=UTF-8
```

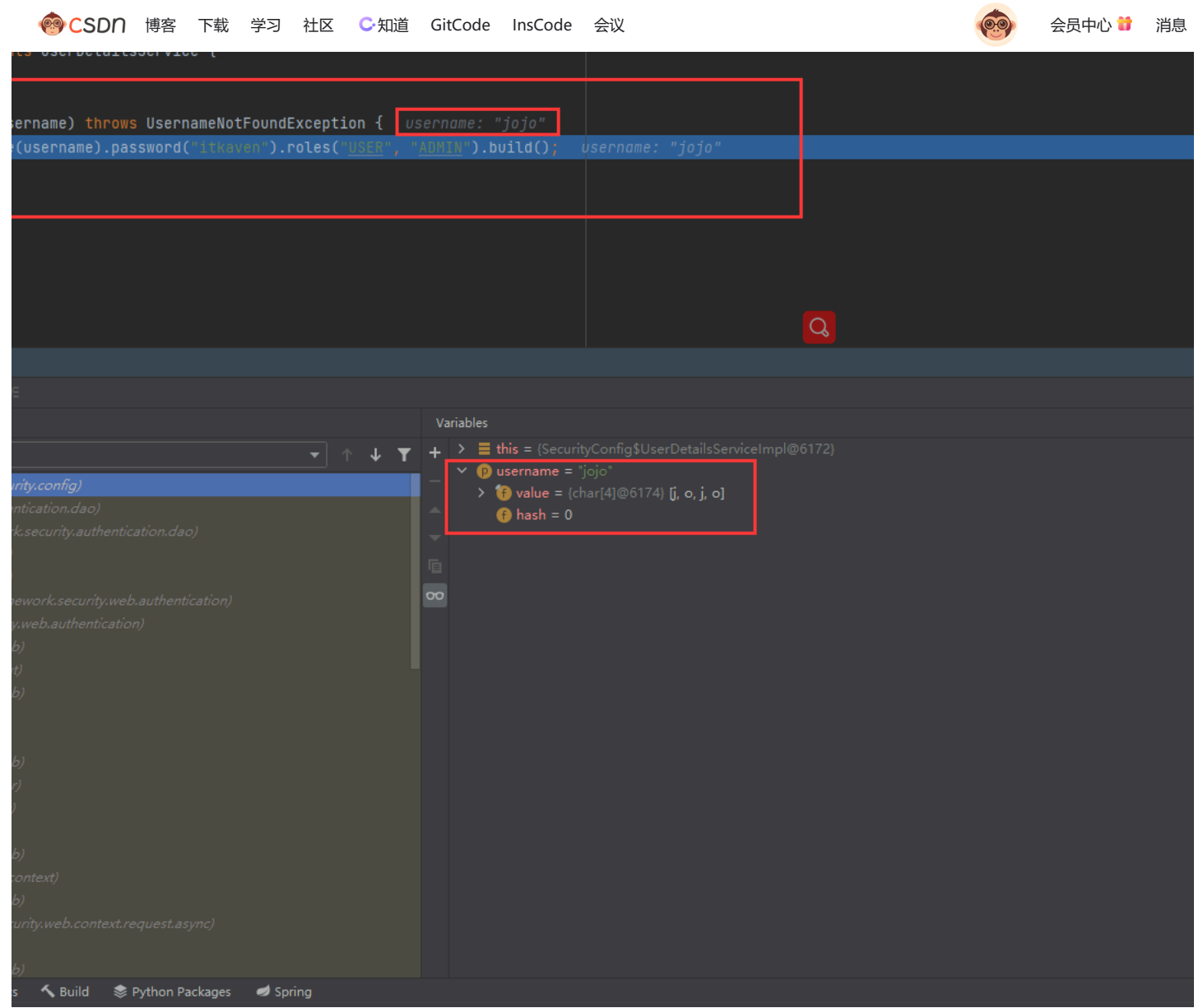
```
er {
```



```
    throws Exception {  
  
        passwordEncoder(NoOpPasswordEncoder.getInstance());  
  
        detailsService {  
  
            throws UsernameNotFoundException {  
  
                MIN  
                ne).password("itkaven").roles("USER", "ADMIN").build();  
            }  
        }  
    }  
}
```

就创建用户（前两种方式会在启动时就创建用户），因为 Spring Security 不可能在启动时就将用户源中的所有用户都创建一次（饿汉式），这是不现实的，关于 UserDetailsService 接口及其实现类的内容以后会详细介绍。





验证用户才被创建的（饿汉式与懒汉式），用户进行验证时，Spring Security 会通过 UserDetailsService 实例加载与该用户的用户名匹配的 UserDetails 实例进行匹配，如果匹配成功，则验证成功，否则验证失败。

家有不同的见解，欢迎大家评论补充。

UserDetailsService接口负责从数据源中加载用户信息,例如用户名、密码和权限信息。详细步骤如下：自定义UserDetailsService类 创建一个名为CustomUserDetailsService的

根据自己的需求来实现对应的方法。GrantedAuthority

用户信息。在这个类中，我们可以根据用户类型（例如，用户的角色或权限）执行不同的逻辑。以下是一个简化的实现：``java @...

是如何工作的。我们又该如何驾驭它。请多多关注公众号：Felordcn。本篇将通过 Spring Boot 2.x 来讲解 Spring Security 中的用户主体UserDetails。以及从中找点乐子。2.

的组件都可以使用默认的,因为UserDetailsService是SpringSecurity获取数据库中的认证信息的媒介,而如何才能从数据库中获取认证信息只有我们才知道。在入门案例中我们使

```
rd());String getUsername();//账户是否过期,过期无法验证boolean isAccountNonExpired();//指定用户是否被锁定或者解锁,锁定的用户无法进行身份验证boolean isAccountNonLo
```

去实现UserDetails接口,重写UserDetails接口方法时,直接写死一个用户信息,一会儿new这个自定义的SecurityUser类,也就和上面的User.builder一个意思。重写loadUserByUsern

础上改用了自定义UserDetails实现用户登录访问,因为这里只修改了实体类User和UserService中的代码,其他代码请参见示例四 地址:https://blog.csdn.net/qq_32224047/article/de

身份验证和授权框架，在大多数Web应用程序中都使用它来保护用户的身份验证和授权。今天，我们将讨论如何在Spring Security中...

```
lUserDetailsService。 publicvoidconfigure(AuthenticationManagerBuilder auth)throwsException { auth.userDetailsService(userDetailsService())//自定义构建 .passwordEncode
  ...
```

生成的,而实际中我们需要从数据库中获取,所以我们需要自定义。通俗一点就是把查询数据库的过程,写到这个接口中 如何用? 1.创建类继承UsernamePasswordAuthenticationFi

se`实现, 或者利用Spring Security提供的其他扩展点, 如 `AuthenticationProvider` 或 `UserDetailsService` 的实现类。...

3, 并提供一个完整的源码示例。这将特别适用于初学者和开发者,帮助他们更好地理解Spring Security的安全特性。首先,让我们理解...

on; 1 该方法很容易理解:通过用户名来加载用户。这个方法主要用于从系统数据中查询并加载具体的用户到Spring Security中。 3.2 UserDetails 从上面UserDetailsService可以知

实例。对于我们只需要使用里面的User类即可。注意User的全限定路径是: org.springframework.security.core.userdetails.User 此处经常和系统中自己开发的User类弄混。

解
ity定义生成的。而在实际项目中账号和密码都是从数据库中查询出来的。所以我们要通过自定义逻辑控制认证逻辑。如果需要自定义逻辑时,只需要实现UserDetailsService并

是从数据 库中查询出来的。所以我们要通过自定义逻辑控制认证逻辑。如果需要自定义逻辑时,只需要实现接口即可。接口定义如下: 直接看源码 要想返回 UserDetails 的实例

流程) 最新发布

ls接口方法时,直接写死一个用户信息,一会儿new这个自定义的SecurityUser类,也就和上面的User.builder一个意思。重写loadUserByUsername方法,并当用户名等于自定义

achingUserDetailsService。该类的构造接收一个用于真正加载UserDetails的UserDetailsService实现类。当需要加载UserDetails时,其首先会从缓存中获取,如果缓存中没有对

ity使用UserDetails实例(实现类的实例)表示用户,当客户端进行验证时(提供用户名和密码),Spring Security会通过用户服务(UserDetailsService接口及其实现)来获取

1这个方法会根据用户名提取用户信息 UserDetails(包含密码),一般是从数据库中提取信息进行组装,最后返回一个UserDetails实例.loadUserByUsername方法返回之后的UserDe

例子,分离接口可以提供更好的灵活性,因为框架不会强迫我们在应用程序不需要的时候实现行为,如果应用程序只需要验证用户,那么实现UserDetailsService契约就足以覆

会又来啦! 1. 前言前一篇介绍了Spring Security入门的基础准备。从今天开始我们来一步步窥探它是如何工作的。我们又该如何驾驭它。请多多关注公众号: Felordcn。本篇将

又用户进行登录。虽然该方式的灵活性相较于静态账号密码的方式灵活了许多,但是将数据库的结构暴露在明显的位置上,绝对不是一个明智的做法。本文通过Java代码实现Us

