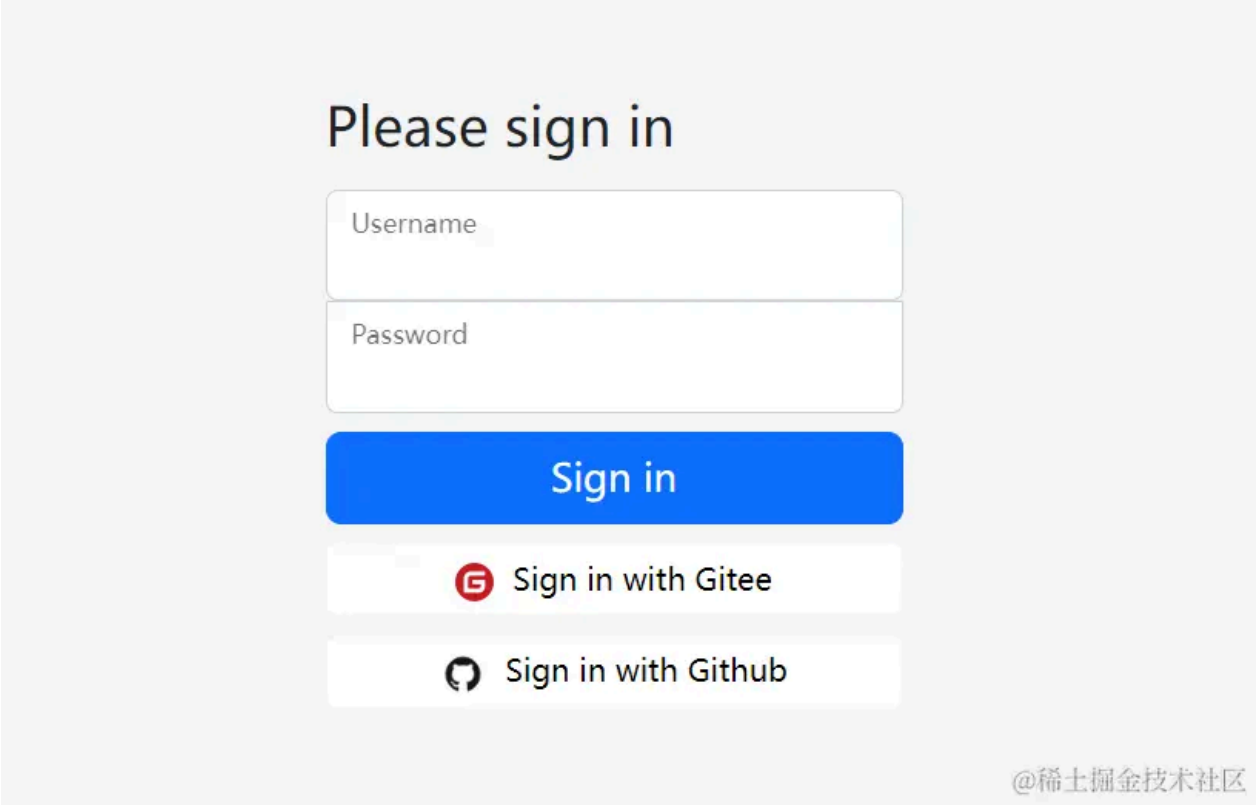
Spring Authorization Server (8) 授权服务的默认认证方式扩展

爱吃西瓜的胖娃 2023-10-15 489 阅读10分钟 关注

> 架构版本
>
> Spring Boot 3.1
>
> Spring Authorization Server 1.1.1
>
> spring-cloud 2022.0.3
>
> spring-cloud-alibaba 2022.0.0.0
>
> 完整代码👉 watermelon-cloud

授权服务的默认认证方式

> 默认的 `/login` ，且只有用户名和密码模式，就这个页面



> 密码我万一那天忘记了咋办，随便看看现在的谁家的网站，手机号+验证码没有，这也太low了，必须得加一个啊！

那先看看默认的 /login 它怎么玩的，我们就跟着它一样玩

授权服务默认登录

> 默认登录url `/login` ，能找到对应的filter `UsernamePasswordAuthenticationFilter` ，需要从这个地方先入手，点进去看看

UsernamePasswordAuthenticationFilter

> java 复制代码

```java
public class UsernamePasswordAuthenticationFilter extends >AbstractAuthenticationProcessi

    public static final String SPRING_SECURITY_FORM_USERNAME_KEY = "username";

    public static final String SPRING_SECURITY_FORM_PASSWORD_KEY = "password";

    private static final AntPathRequestMatcher DEFAULT_ANT_PATH_REQUEST_MATCHER = new >Antl
            "POST");

    private String usernameParameter = SPRING_SECURITY_FORM_USERNAME_KEY;

    private String passwordParameter = SPRING_SECURITY_FORM_PASSWORD_KEY;

    private boolean postOnly = true;

    public UsernamePasswordAuthenticationFilter() {
        super(DEFAULT_ANT_PATH_REQUEST_MATCHER);
    }

    public UsernamePasswordAuthenticationFilter(AuthenticationManager authenticationManager
        super(DEFAULT_ANT_PATH_REQUEST_MATCHER, authenticationManager);
    }

    @Override
    public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResp
            throws AuthenticationException {
        if (this.postOnly && !request.getMethod().equals("POST")) {
            throw new AuthenticationServiceException("Authentication method not supported:
        }
        String username = obtainUsername(request);
        username = (username != null) ? username.trim() : "";
        String password = obtainPassword(request);
        password = (password != null) ? password : "";
        UsernamePasswordAuthenticationToken authRequest = UsernamePasswordAuthenticationTol
                password);
        // Allow subclasses to set the "details" property
        setDetails(request, authRequest);
        return this.getAuthenticationManager().authenticate(authRequest);
    }
}
```

`UsernamePasswordAuthenticationFilter` 继承 `AbstractAuthenticationProcessingFilter` 的 重写了 `attemptAuthentication()` 方法

构造了 UsernamePasswordAuthenticationToken 对象

> `UsernamePasswordAuthenticationToken` 对于的 `DaoAuthenticationProvider` 处理， 最后返回了 `Authentication` 这个流程应该已经很熟悉了吧。
>
> 最后看看怎么走 `UsernamePasswordAuthenticationFilter` 的父类是一个抽象类 `AbstractAuthenticationProcessingFilter`

AbstractAuthenticationProcessingFilter

> java 复制代码

```java
public abstract class AbstractAuthenticationProcessingFilter extends GenericFilterBean
        implements ApplicationEventPublisherAware, MessageSourceAware {


    // UsernamePasswordAuthenticationFilter 是 对应就是  CompositeSessionAuthenticationStra
    private SessionAuthenticationStrategy sessionStrategy = new NullAuthenticatedSessionSt

    //
    private SecurityContextRepository securityContextRepository = new RequestAttributeSecu


    public abstract Authentication attemptAuthentication(HttpServletRequest request, HttpS
            throws AuthenticationException, IOException, ServletException;


    private void doFilter(HttpServletRequest request, HttpServletResponse response, Filter
            throws IOException, ServletException {
        if (!requiresAuthentication(request, response)) {
            chain.doFilter(request, response);
            return;
        }
        try {
            Authentication authenticationResult = attemptAuthentication(request, response)
            if (authenticationResult == null) {
                // return immediately as subclass has indicated that it hasn't completed
                return;
            }
            this.sessionStrategy.onAuthentication(authenticationResult, request, response)
            // Authentication success
            if (this.continueChainBeforeSuccessfulAuthentication) {
```

```
34          } catch (InternalAuthenticationServiceException failed) {
35              this.logger.error("An internal error occurred while trying to authenticate the
36              unsuccessfulAuthentication(request, response, failed);
37          } catch (AuthenticationException ex) {
38              // Authentication failed
39              unsuccessfulAuthentication(request, response, ex);
40          }
41      }
42  }
43
```

`Authentication authenticationResult = attemptAuthentication(request, response);` 这行会执行子类的 `attemptAuthentication()` 最后处理登录成功相关的了。\

那扩展肯定这个 attemptAuthentication() 方法是关键了

也同样继承 `AbstractAuthenticationProcessingFilter` 去实现一个 Filter 这样不行，因为 `AbstractAuthenticationProcessingFilter` 中有两个变量 `securityContextRepository` 、`sessionStrategy` 当是 `UsernamePasswordAuthenticationFilter` 时 初始化的值就不一样了，这样就导致我们和 `UsernamePasswordAuthenticationFilter` 的后续处理成功流程就不一样，导致一直认证失败的。

后续的filter 注入的时候代码中会体现出来。

扩展

手机号+验证码

UserAuthenticationProcessingFilter

自定义一个 `UserAuthenticationProcessingFilter` 与 `AbstractAuthenticationProcessingFilter` 类似

java 复制代码

```java
public abstract class UserAuthenticationProcessingFilter extends GenericFilterBean
    implements ApplicationEventPublisherAware, MessageSourceAware {

  private SecurityContextHolderStrategy securityContextHolderStrategy = SecurityContextHo
      .getContextHolderStrategy();

  protected ApplicationEventPublisher eventPublisher;

  protected AuthenticationDetailsSource<HttpServletRequest, ?> authenticationDetailsSource

  private AuthenticationManager authenticationManager;

  protected MessageSourceAccessor messages = SpringSecurityMessageSource.getAccessor();

  private RememberMeServices rememberMeServices = new NullRememberMeServices();

  private RequestMatcher requiresAuthenticationRequestMatcher;

  private boolean continueChainBeforeSuccessfulAuthentication = false;

  private SessionAuthenticationStrategy sessionStrategy = new NullAuthenticatedSessionStr

  private boolean allowSessionCreation = true;

  private AuthenticationSuccessHandler successHandler = new SavedRequestAwareAuthenticati

  private AuthenticationFailureHandler failureHandler = new SimpleUrlAuthenticationFailur

  private SecurityContextRepository securityContextRepository = new RequestAttributeSecur


  protected UserAuthenticationProcessingFilter(String defaultFilterProcessesUrl) {
    setFilterProcessesUrl(defaultFilterProcessesUrl);
  }


  protected UserAuthenticationProcessingFilter(RequestMatcher requiresAuthenticationReque
    Assert.notNull(requiresAuthenticationRequestMatcher, "requiresAuthenticationRequestM
    this.requiresAuthenticationRequestMatcher = requiresAuthenticationRequestMatcher;
  }


  protected UserAuthenticationProcessingFilter(String defaultFilterProcessesUrl,
                                               AuthenticationManager authenticationManage
    setFilterProcessesUrl(defaultFilterProcessesUrl);
    setAuthenticationManager(authenticationManager);
```

```
50  protected UserAuthenticationProcessingFilter(RequestMatcher requiresAuthenticationReque
51                                                AuthenticationManager authenticationManag
52     setRequiresAuthenticationRequestMatcher(requiresAuthenticationRequestMatcher);
53     setAuthenticationManager(authenticationManager);
54  }
55
56  @Override
57  public void afterPropertiesSet() {
58     Assert.notNull(this.authenticationManager, "authenticationManager must be specified"
59  }
60
61
62  @Override
63  public void doFilter(ServletRequest request, ServletResponse response, FilterChain chai
64        throws IOException, ServletException {
65     doFilter((HttpServletRequest) request, (HttpServletResponse) response, chain);
66  }
67
68  private void doFilter(HttpServletRequest request, HttpServletResponse response, FilterCh
69        throws IOException, ServletException {
70     if (!requiresAuthentication(request, response)) {
71        chain.doFilter(request, response);
72        return;
73     }
74     try {
75        Authentication authenticationResult = attemptAuthentication(request, response);
76        if (authenticationResult == null) {
77           // return immediately as subclass has indicated that it hasn't completed
78           return;
79        }
80        this.sessionStrategy.onAuthentication(authenticationResult, request, response);
81        // Authentication success
82        if (this.continueChainBeforeSuccessfulAuthentication) {
83           chain.doFilter(request, response);
84        }
85        successfulAuthentication(request, response, chain, authenticationResult);
86     }
87     catch (InternalAuthenticationServiceException failed) {
88        this.logger.error("An internal error occurred while trying to authenticate the use
89        unsuccessfulAuthentication(request, response, failed);
90     }
91     catch (AuthenticationException ex) {
92        // Authentication failed
93        unsuccessfulAuthentication(request, response, ex);
94     }
95  }
96
```

```
99      if (this.requiresAuthenticationRequestMatcher.matches(request)) {
100        return true;
101     }
102     if (this.logger.isTraceEnabled()) {
103        this.logger
104              .trace(LogMessage.format("Did not match request to %s", this.requiresAuthen
105     }
106     return false;
107  }
108
109
110  public abstract Authentication attemptAuthentication(HttpServletRequest request, HttpSe
111        throws AuthenticationException, IOException, ServletException;
112
113
114  protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse
115        Authentication authResult) throws IOException, ServletException {
116     SecurityContext context = this.securityContextHolderStrategy.createEmptyContext();
117     context.setAuthentication(authResult);
118     this.securityContextHolderStrategy.setContext(context);
119     this.securityContextRepository.saveContext(context, request, response);
120     if (this.logger.isDebugEnabled()) {
121        this.logger.debug(LogMessage.format("Set SecurityContextHolder to %s", authResult
122     }
123     this.rememberMeServices.loginSuccess(request, response, authResult);
124     if (this.eventPublisher != null) {
125        this.eventPublisher.publishEvent(new InteractiveAuthenticationSuccessEvent(authRe
126     }
127     this.successHandler.onAuthenticationSuccess(request, response, authResult);
128  }
129
130
131  protected void unsuccessfulAuthentication(HttpServletRequest request, HttpServletRespon
132        AuthenticationException failed) throws IOException, ServletException {
133     this.securityContextHolderStrategy.clearContext();
134     this.logger.trace("Failed to process authentication request", failed);
135     this.logger.trace("Cleared SecurityContextHolder");
136     this.logger.trace("Handling authentication failure");
137     this.rememberMeServices.loginFail(request, response);
138     this.failureHandler.onAuthenticationFailure(request, response, failed);
139  }
140
141  protected AuthenticationManager getAuthenticationManager() {
142     return this.authenticationManager;
143  }
144
145  public void setAuthenticationManager(AuthenticationManager authenticationManager) {
```

```
148
149
150   public void setFilterProcessesUrl(String filterProcessesUrl) {
151      setRequiresAuthenticationRequestMatcher(new AntPathRequestMatcher(filterProcessesUrl
152   }
153
154   public final void setRequiresAuthenticationRequestMatcher(RequestMatcher requestMatcher
155      Assert.notNull(requestMatcher, "requestMatcher cannot be null");
156      this.requiresAuthenticationRequestMatcher = requestMatcher;
157   }
158
159   public RememberMeServices getRememberMeServices() {
160      return this.rememberMeServices;
161   }
162
163   public void setRememberMeServices(RememberMeServices rememberMeServices) {
164      Assert.notNull(rememberMeServices, "rememberMeServices cannot be null");
165      this.rememberMeServices = rememberMeServices;
166   }
167
168
169   public void setContinueChainBeforeSuccessfulAuthentication(boolean continueChainBeforeSu
170      this.continueChainBeforeSuccessfulAuthentication = continueChainBeforeSuccessfulAuthe
171   }
172
173   @Override
174   public void setApplicationEventPublisher(ApplicationEventPublisher eventPublisher) {
175      this.eventPublisher = eventPublisher;
176   }
177
178   public void setAuthenticationDetailsSource(
179         AuthenticationDetailsSource<HttpServletRequest, ?> authenticationDetailsSource) {
180      Assert.notNull(authenticationDetailsSource, "AuthenticationDetailsSource required");
181      this.authenticationDetailsSource = authenticationDetailsSource;
182   }
183
184   @Override
185   public void setMessageSource(MessageSource messageSource) {
186      this.messages = new MessageSourceAccessor(messageSource);
187   }
188
189   protected boolean getAllowSessionCreation() {
190      return this.allowSessionCreation;
191   }
192
193   public void setAllowSessionCreation(boolean allowSessionCreation) {
194      this.allowSessionCreation = allowSessionCreation;
```

```
197   public void setSessionAuthenticationStrategy(SessionAuthenticationStrategy sessionStrat
198      this.sessionStrategy = sessionStrategy;
199   }
200
201   public void setAuthenticationSuccessHandler(AuthenticationSuccessHandler successHandler
202      Assert.notNull(successHandler, "successHandler cannot be null");
203      this.successHandler = successHandler;
204   }
205
206   public void setAuthenticationFailureHandler(AuthenticationFailureHandler failureHandler
207      Assert.notNull(failureHandler, "failureHandler cannot be null");
208      this.failureHandler = failureHandler;
209   }
210
211
212   public void setSecurityContextRepository(SecurityContextRepository securityContextRepos
213      Assert.notNull(securityContextRepository, "securityContextRepository cannot be null"
214      this.securityContextRepository = securityContextRepository;
215   }
216
217
218   public void setSecurityContextHolderStrategy(SecurityContextHolderStrategy securityCont
219      Assert.notNull(securityContextHolderStrategy, "securityContextHolderStrategy cannot
220      this.securityContextHolderStrategy = securityContextHolderStrategy;
221   }
222
223   protected AuthenticationSuccessHandler getSuccessHandler() {
224      return this.successHandler;
225   }
226
227   protected AuthenticationFailureHandler getFailureHandler() {
228      return this.failureHandler;
229   }
230
231 }
```

PhoneCaptchaAuthenticationToken

java 复制代码

```
1  public class PhoneCaptchaAuthenticationToken extends AbstractAuthenticationToken {
2
3     private final Object principal;
```

```
6
7   public PhoneCaptchaAuthenticationToken(Object principal, Object credentials) {
8       super(null);
9       this.principal = principal;
10      this.credentials = credentials;
11      setAuthenticated(false);
12  }
13
14
15  public PhoneCaptchaAuthenticationToken(Object principal, Object credentials,
16                                         Collection<? extends GrantedAuthority> authc
17      super(authorities);
18      this.principal = principal;
19      this.credentials = credentials;
20      super.setAuthenticated(true); // must use super, as we override
21  }
22
23
24  public static PhoneCaptchaAuthenticationToken unauthenticated(Object principal, Object
25      return new PhoneCaptchaAuthenticationToken(principal, credential);
26  }
27
28
29  public static PhoneCaptchaAuthenticationToken authenticated(Object principal, Object cr
30                                                              Collection<? extends Gr
31      return new PhoneCaptchaAuthenticationToken(principal, credentials, authorities);
32  }
33
34  @Override
35  public Object getCredentials() {
36      return this.credentials;
37  }
38
39  @Override
40  public Object getPrincipal() {
41      return this.principal;
42  }
43
44  @Override
45  public void setAuthenticated(boolean isAuthenticated) throws IllegalArgumentException {
46      Assert.isTrue(!isAuthenticated,
47              "Cannot set this token to trusted - use constructor which takes a GrantedAu
48      super.setAuthenticated(false);
49  }
50
51  @Override
52  public void eraseCredentials() {
```

```
55     }
56 }
```

PhoneCaptchaAuthenticationProvider

java 复制代码

```java
@Component
public class PhoneCaptchaAuthenticationProvider
          implements AuthenticationProvider {

  protected final Log logger = LogFactory.getLog(getClass());

  protected MessageSourceAccessor messages = SpringSecurityMessageSource.getAccessor();

  private UserCache userCache = new NullUserCache();

  private final boolean forcePrincipalAsString = false;

  protected boolean hideUserNotFoundExceptions = true;

  private UserDetailsChecker preAuthenticationChecks = new DefaultPreAuthenticationChecks

  private UserDetailsChecker postAuthenticationChecks = new DefaultPostAuthenticationChecl

  private GrantedAuthoritiesMapper authoritiesMapper = new NullAuthoritiesMapper();

  @Lazy
  @Autowired
  public UserDetailsService userDetailsService;


  @Override
  public Authentication authenticate(Authentication authentication) throws AuthenticationI
          Assert.isInstanceOf(PhoneCaptchaAuthenticationToken.class, authentication,
                  () -> this.messages.getMessage("PhoneCaptchaAuthenticationProvi
                          "Only PhoneCaptchaAuthenticationToken is suppor
          String username = determineUsername(authentication);
          boolean cacheWasUsed = true;
          UserDetails user = this.userCache.getUserFromCache(username);
```

```
36                  try {
37                          user = retrieveUser(username, (PhoneCaptchaAuthenticationToken)
38                  }
39                  catch (UsernameNotFoundException ex) {
40                          this.logger.debug("Failed to find phone '" + username + "'");
41                          if (!this.hideUserNotFoundExceptions) {
42                                  throw ex;
43                          }
44                          throw new BadCredentialsException(this.messages
45                                          .getMessage("PhoneCaptchaAuthenticationProvider
46                  }
47                  Assert.notNull(user, "retrieveUser returned null - a violation of the i
48          }
49          try {
50                  this.preAuthenticationChecks.check(user);
51          }
52          catch (AuthenticationException ex) {
53                  if (!cacheWasUsed) {
54                          throw ex;
55                  }
56                  // There was a problem, so try again after checking
57                  // we're using latest data (i.e. not from the cache)
58                  cacheWasUsed = false;
59                  user = retrieveUser(username, (PhoneCaptchaAuthenticationToken) authent
60                  this.preAuthenticationChecks.check(user);
61          }
62          this.postAuthenticationChecks.check(user);
63          if (!cacheWasUsed) {
64                  this.userCache.putUserInCache(user);
65          }
66          Object principalToReturn = user;
67          if (this.forcePrincipalAsString) {
68                  principalToReturn = user.getUsername();
69          }
70          Authentication successAuthentication = createSuccessAuthentication(principalToR
71          return successAuthentication;
72  }
73
74  private String determineUsername(Authentication authentication) {
75          return (authentication.getPrincipal() == null) ? "NONE_PROVIDED" : authenticati
76  }
77
78
79  protected Authentication createSuccessAuthentication(Object principal, Authentication au
80                  UserDetails user) {
81          PhoneCaptchaAuthenticationToken result = PhoneCaptchaAuthenticationToken.authen
82                          authentication.getCredentials(), this.authoritiesMapper.mapAuth
```

```
85          return result;
86  }
87
88
89
90  public UserCache getUserCache() {
91          return this.userCache;
92  }
93
94  public boolean isForcePrincipalAsString() {
95          return this.forcePrincipalAsString;
96  }
97
98  public boolean isHideUserNotFoundExceptions() {
99          return this.hideUserNotFoundExceptions;
100 }
101
102
103 protected  UserDetails retrieveUser(String username, PhoneCaptchaAuthenticationToken au
104                 throws AuthenticationException{
105         try {
106                 UserDetails loadedUser = this.getUserDetailsService().loadUserByUsernam
107                 if (loadedUser == null) {
108                         throw new InternalAuthenticationServiceException(
109                                         "UserDetailsService returned null, which is an
110                 }
111                 return loadedUser;
112         }
113         catch (UsernameNotFoundException ex) {
114                 throw ex;
115         }
116         catch (InternalAuthenticationServiceException ex) {
117                 throw ex;
118         }
119         catch (Exception ex) {
120                 throw new InternalAuthenticationServiceException(ex.getMessage(), ex);
121         }
122
123 }
124
125
126 protected UserDetailsService getUserDetailsService() {
127         return this.userDetailsService;
128 }
129
130
131 public void setUserCache(UserCache userCache) {
```

```
134
135  @Override
136  public boolean supports(Class<?> authentication) {
137          return (PhoneCaptchaAuthenticationToken.class.isAssignableFrom(authentication))
138  }
139
140  protected UserDetailsChecker getPreAuthenticationChecks() {
141          return this.preAuthenticationChecks;
142  }
143
144  /**
145   * Sets the policy will be used to verify the status of the loaded
146   * <tt>UserDetails</tt> <em>before</em> validation of the credentials takes place.
147   * @param preAuthenticationChecks strategy to be invoked prior to authentication.
148   */
149  public void setPreAuthenticationChecks(UserDetailsChecker preAuthenticationChecks) {
150          this.preAuthenticationChecks = preAuthenticationChecks;
151  }
152
153  protected UserDetailsChecker getPostAuthenticationChecks() {
154          return this.postAuthenticationChecks;
155  }
156
157  public void setPostAuthenticationChecks(UserDetailsChecker postAuthenticationChecks) {
158          this.postAuthenticationChecks = postAuthenticationChecks;
159  }
160
161  public void setAuthoritiesMapper(GrantedAuthoritiesMapper authoritiesMapper) {
162          this.authoritiesMapper = authoritiesMapper;
163  }
164
165
166  private class DefaultPreAuthenticationChecks implements UserDetailsChecker {
167
168          @Override
169          public void check(UserDetails user) {
170                  if (!user.isAccountNonLocked()) {
171                          PhoneCaptchaAuthenticationProvider.this.logger
172                                          .debug("Failed to authenticate since user accou
173                          throw new LockedException(PhoneCaptchaAuthenticationProvider.th
174                                          .getMessage("AbstractUserDetailsAuthenticationP
175                  }
176                  if (!user.isEnabled()) {
177                          PhoneCaptchaAuthenticationProvider.this.logger
178                                          .debug("Failed to authenticate since user accou
179                          throw new DisabledException(PhoneCaptchaAuthenticationProvider.
180                                          .getMessage("AbstractUserDetailsAuthenticationP
```

```
183                             PhoneCaptchaAuthenticationProvider.this.logger
184                                     .debug("Failed to authenticate since user accou
185                             throw new AccountExpiredException(PhoneCaptchaAuthenticationPro
186                                     .getMessage("AbstractUserDetailsAuthenticationP
187                     }
188             }
189
190     }
191
192     private class DefaultPostAuthenticationChecks implements UserDetailsChecker {
193
194             @Override
195             public void check(UserDetails user) {
196                     if (!user.isCredentialsNonExpired()) {
197                             logger.debug("Failed to authenticate since user account credent
198                             throw new CredentialsExpiredException(PhoneCaptchaAuthenticatio
199                                     .getMessage("PhoneCaptchaAuthenticationProvider
200                                             "User credentials have expired"
201                     }
202             }
203
204     }
205
206 }
```

UserAuthenticationFilter

自定义 `UserAuthenticationFilter` 继承 `UserAuthenticationProcessingFilter` ，与 `UsernamePasswordAuthenticationFilter` 类似

java 复制代码

```
1  public class UserAuthenticationFilter extends UserAuthenticationProcessingFilter {
2
3    public static final String SPRING_SECURITY_FORM_USERNAME_KEY = "username";
4
5    public static final String SPRING_SECURITY_FORM_PASSWORD_KEY = "password";
6
7    public static final String SPRING_SECURITY_FORM_CODE_KEY = "code";
```

```
11  private String passwordParameter = SPRING_SECURITY_FORM_PASSWORD_KEY;
12
13  private final String codeParameter = SPRING_SECURITY_FORM_CODE_KEY;
14
15  private boolean postOnly = true;
16
17  private static final AntPathRequestMatcher DEFAULT_ANT_PATH_REQUEST_MATCHER = new AntPa
18                  "POST");
19
20  public UserAuthenticationFilter() {
21          super(DEFAULT_ANT_PATH_REQUEST_MATCHER);
22  }
23
24  public UserAuthenticationFilter(AuthenticationManager authenticationManager) {
25          super(DEFAULT_ANT_PATH_REQUEST_MATCHER, authenticationManager);
26  }
27
28  @Override
29  public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResp
30                  throws AuthenticationException {
31          if (this.postOnly && !request.getMethod().equals("POST")) {
32                  throw new AuthenticationServiceException("Authentication method not sup
33          }
34          String username = obtainUsername(request);
35          username = (username != null) ? username.trim() : "";
36          String password = obtainPassword(request);
37          if(StringUtils.hasText(password)){
38                  password = (password != null) ? password : "";
39                  UsernamePasswordAuthenticationToken authRequest = UsernamePasswordAuthe
40                                  password);
41                  // Allow subclasses to set the "details" property
42                  setDetails(request, authRequest);
43                  return this.getAuthenticationManager().authenticate(authRequest);
44          }else {
45                  String code = obtainCode(request);
46                  assert code != null;
47                  if (!code.equals("000000")) {
48                          throw new UsernameNotFoundException("验证码错误！");
49                  }
50                  PhoneCaptchaAuthenticationToken authRequest = PhoneCaptchaAuthenticatio
51                  // Allow subclasses to set the "details" property
52                  this.setDetails(request, authRequest);
53                  return this.getAuthenticationManager().authenticate(authRequest);
54          }
55
56  }
57
```

```
60              return request.getParameter(this.passwordParameter);
61      }
62
63      @Nullable
64      protected String obtainUsername(HttpServletRequest request) {
65              return request.getParameter(this.usernameParameter);
66      }
67
68      @Nullable
69      protected String obtainCode(HttpServletRequest request) {
70              return request.getParameter(this.codeParameter);
71      }
72
73      protected void setDetails(HttpServletRequest request, UsernamePasswordAuthenticationTok
74              authRequest.setDetails(this.authenticationDetailsSource.buildDetails(request));
75      }
76
77      protected void setDetails(HttpServletRequest request, PhoneCaptchaAuthenticationToken au
78              authRequest.setDetails(this.authenticationDetailsSource.buildDetails(request));
79      }
80
81      public void setPasswordParameter(String passwordParameter) {
82              this.passwordParameter = passwordParameter;
83      }
84
85      public void setPostOnly(boolean postOnly) {
86              this.postOnly = postOnly;
87      }
88
89      public final String getUsernameParameter() {
90              return this.usernameParameter;
91      }
92
93  }
```

以上都扩展好了

UserAuthenticationProcessingFilter

UserAuthenticationFilter

PhoneCaptchaAuthenticationToken

PhoneCaptchaAuthenticationProvider

在 `AbstractAuthenticationProcessingFilter` 的找到了一个地方被

`AbstractAuthenticationFilterConfigurer`

java 复制代码

```java
public abstract class AbstractAuthenticationFilterConfigurer<B extends HttpSecurityBuilder<B>, T
        extends AbstractHttpConfigurer<T, B> {
    @Override
    public void configure(B http) throws Exception {
        PortMapper portMapper = http.getSharedObject(PortMapper.class);
        if (portMapper != null) {
            this.authenticationEntryPoint.setPortMapper(portMapper);
        }
        RequestCache requestCache = http.getSharedObject(RequestCache.class);
        if (requestCache != null) {
            this.defaultSuccessHandler.setRequestCache(requestCache);
        }
        this.authFilter.setAuthenticationManager(http.getSharedObject(AuthenticationManager.clas
        this.authFilter.setAuthenticationSuccessHandler(this.successHandler);
        this.authFilter.setAuthenticationFailureHandler(this.failureHandler);
        if (this.authenticationDetailsSource != null) {
            this.authFilter.setAuthenticationDetailsSource(this.authenticationDetailsSource);
        }
        SessionAuthenticationStrategy sessionAuthenticationStrategy = http
                .getSharedObject(SessionAuthenticationStrategy.class);
        if (sessionAuthenticationStrategy != null) {
            this.authFilter.setSessionAuthenticationStrategy(sessionAuthenticationStrategy);
        }
        RememberMeServices rememberMeServices = http.getSharedObject(RememberMeServices.class);
        if (rememberMeServices != null) {
            this.authFilter.setRememberMeServices(rememberMeServices);
        }
        SecurityContextConfigurer securityContextConfigurer = http.getConfigurer(SecurityContext
        if (securityContextConfigurer != null && securityContextConfigurer.isRequireExplicitSave
            SecurityContextRepository securityContextRepository = securityContextConfigurer
                    .getSecurityContextRepository();
            this.authFilter.setSecurityContextRepository(securityContextRepository);
        }
        this.authFilter.setSecurityContextHolderStrategy(getSecurityContextHolderStrategy());
        F filter = postProcess(this.authFilter);
        http.addFilter(filter);
    }
}
```

在里面就看到 `SessionAuthenticationStrategy` 、 `SecurityContextRepository` 初始化与 `AbstractAuthenticationProcessingFilter` 中为 `UsernamePasswordAuthenticationFilter` 时的一致，那么 `UserAuthenticationProcessingFilter` 中的 `SessionAuthenticationStrategy` 、 `SecurityContextRepository` 初始化也用这样的方式进行注入吧。

DefaultSecurityConfig 中添加到 SecurityFilterChain

java 复制代码

```java
@EnableWebSecurity
@Configuration(proxyBeanMethods = false)
public class DefaultSecurityConfig {


  @Autowired
  private PhoneCaptchaAuthenticationProvider phoneCaptchaAuthenticationProvider;



  // 过滤器链
  @Bean
  public SecurityFilterChain defaultSecurityFilterChain(HttpSecurity http) throws Exceptio
          UserAuthenticationFilter userAuthenticationFilter = new UserAuthenticationFilte
          http
                          .addFilterAt(userAuthenticationFilter, UsernamePasswordAuthenti
                          .authorizeHttpRequests(authorize ->//① 配置鉴权的
                                          authorize
                                                          .requestMatchers(
                                                                          "/assets/**",
                                                                          "/webjars/**",
                                                                          AuthorizationSe
                                                                          "/oauth2/**",
                                                                          "/oauth2/token"
                                                          ).permitAll() //② 忽略鉴权的url
                                                          .anyRequest().authenticated()//
                          )
                          .csrf(AbstractHttpConfigurer::disable)
                          .authenticationProvider(phoneCaptchaAuthenticationProvider)
                          .formLogin(login->
                                          login.loginPage(AuthorizationServerConfiguratio
                                                          .defaultSuccessUrl("/test") //
                          )
                          .oauth2Login(oauth2Login->
                                          oauth2Login.loginPage(AuthorizationServerConfig
```

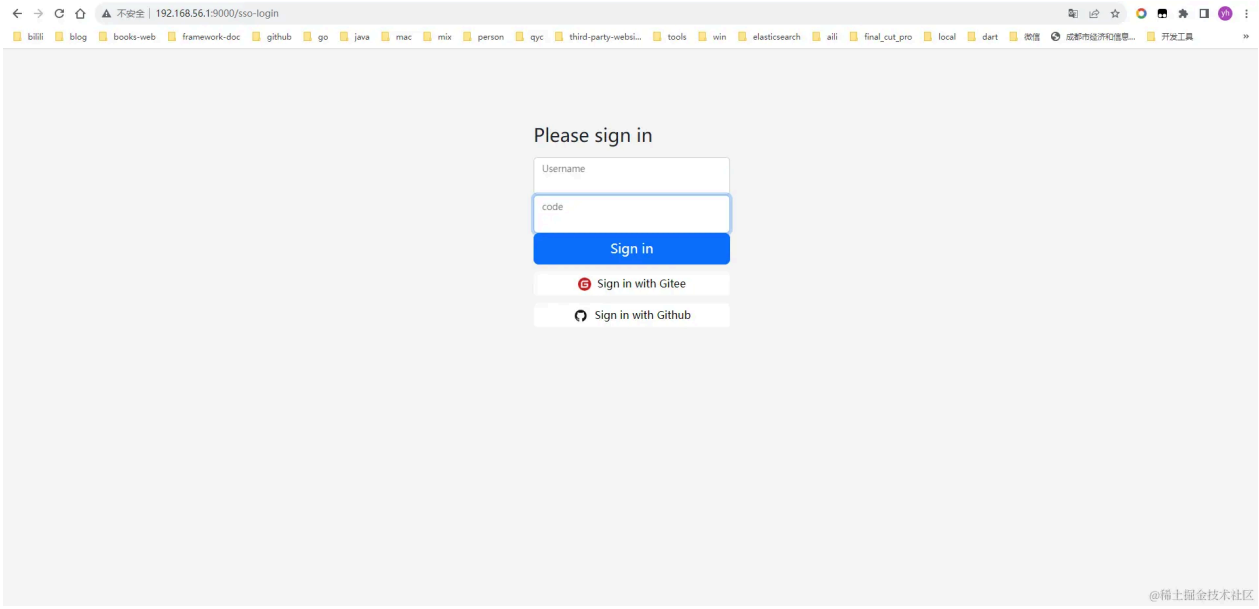
```
38            DefaultSecurityFilterChain build = http.build();
39            userAuthenticationFilter.setAuthenticationManager(http.getSharedObject(Authenti
40            userAuthenticationFilter.setSessionAuthenticationStrategy(http.getSharedObject(
41            SecurityContextRepository securityContextRepository =new DelegatingSecurityCont
42                        new RequestAttributeSecurityContextRepository(), new HttpSessio
43            userAuthenticationFilter.setSecurityContextRepository(securityContextRepository
44
45            return build;
46    }
47 }
```

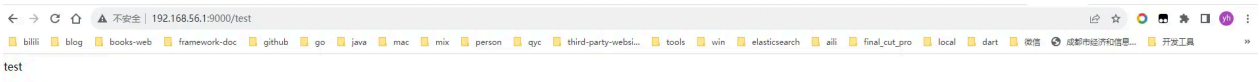
sso-login.html

```
html                                                    复制代码
```

```
1  <!DOCTYPE html>
2  <html lang="en" xmlns="http://www.w3.org/1999/xhtml" xmlns:th="https://www.thymeleaf.org"
3  <head>
4     <meta charset="utf-8" />
5     <meta name="viewport" content="width=device-width, initial-scale=1">
6     <title>Spring Authorization Server sample</title>
7     <link rel="stylesheet" href="/webjars/bootstrap/css/bootstrap.css" th:href="@{/webjars,
8     <link rel="stylesheet" href="/assets/css/signin.css" th:href="@{/assets/css/signin.css
9  </head>
10 <body>
11 <div class="container">
12    <form class="form-signin w-100 m-auto" method="post" th:action="@{/sso-login}">
13        <div th:if="${param.error}" class="alert alert-danger" role="alert">
14            Invalid username or password.
15        </div>
16        <div th:if="${param.logout}" class="alert alert-success" role="alert">
17            You have been logged out.
18        </div>
19        <h1 class="h3 mb-3 fw-normal">Please sign in</h1>
20        <div class="form-floating">
21            <input type="text" id="username" name="username" class="form-control" required
22            <label for="username">Username</label>
23        </div>
24        <div class="form-floating">
25            <input type="text" id="code" name="code" class="form-control" required>
```

```
28          <div>
29              <button class="w-100 btn btn-lg btn-primary btn-block" type="submit">Sign in</
30              <a class="w-100 btn btn-light btn-block bg-white" href="/oauth2/authorization/
31                  <img src="/assets/img/gitee.png" th:src="@{/assets/img/gitee.png}" width="
32                  Sign in with Gitee
33              </a>
34              <a class="w-100 btn btn-light btn-block bg-white" href="/oauth2/authorization/
35                  <img src="/assets/img/github.png" th:src="@{/assets/img/github.png}" width
36                  Sign in with Github
37              </a>
38          </div>
39      </form>
40  </div>
41  </body>
42  </html>
```





用spring-authorization-server的目的一定是要非常明确，就是为了不再去重复造轮子完成oauth2的认证流程，前端代码都不用写的，spring-boot-starter-oauth2-client 里面都去做了拿code去换token的整套流程，所以一定好好理解为什么要用spring-authorization-server的目的，别跑偏了。

如果出现以下异常

```java
java.lang.IllegalArgumentException: The class with com.watermelon.authorization.filter.Ph
and name of com.watermelon.authorization.filter.PhoneCaptchaAuthenticationToken is not in
If you believe this class is safe to deserialize, please provide an explicit mapping usin
If the serialization is only done by a trusted source, you can also enable default typing
See https://github.com/spring-projects/spring-security/issues/4370 for details \
  at org.springframework.security.oauth2.server.authorization.JdbcOAuth2AuthorizationServ
```

我们将 `OAuth2AuthorizationService` 实现替换为redis存储即可

```java
@Component
public class RedisOAuth2AuthorizationServiceImpl implements OAuth2AuthorizationService {


    private final static String AUTHORIZATION_TYPE = "authorization_type";

    private final static String OAUTH2_PARAMETER_NAME_ID = "id";

    private final static Long TIMEOUT = 600L;


    @Resource
    private RedisTemplate<String, Object> redisTemplate;

    @Override
    public void save(OAuth2Authorization authorization) {
        Assert.notNull(authorization, "authorization cannot be null");
        redisTemplate.setValueSerializer(RedisSerializer.java());
        redisTemplate.opsForValue().set(buildAuthorizationKey(OAUTH2_PARAMETER_NAME_ID, au
        if (isState(authorization)) {
            String state = authorization.getAttribute(OAuth2ParameterNames.STATE);
            String isStateKey = buildAuthorizationKey(OAuth2ParameterNames.STATE, state);
```

```
25          }
26          if (isAuthorizationCode(authorization)) {
27              OAuth2Authorization.Token<OAuth2AuthorizationCode> authorizationCode =
28                      authorization.getToken(OAuth2AuthorizationCode.class);
29              String tokenValue = authorizationCode.getToken().getTokenValue();
30              String isAuthorizationCodeKey = buildAuthorizationKey(OAuth2ParameterNames.CODI
31              Instant expiresAt = authorizationCode.getToken().getExpiresAt();//过期时间
32              Instant issuedAt = authorizationCode.getToken().getIssuedAt();//发放token的时间
33              Date expiresAtDate = Date.from(expiresAt);
34              Date issuedAtDate = Date.from(issuedAt);
35              redisTemplate.setValueSerializer(RedisSerializer.java());
36              redisTemplate.opsForValue().set(isAuthorizationCodeKey, authorization, TIMEOUT,
37          }
38          if (isAccessToken(authorization)) {
39              OAuth2Authorization.Token<OAuth2AccessToken> accessToken =
40                      authorization.getToken(OAuth2AccessToken.class);
41              String tokenValue = accessToken.getToken().getTokenValue();
42              String isAccessTokenKey = buildAuthorizationKey(OAuth2ParameterNames.ACCESS_TOI
43              Instant expiresAt = accessToken.getToken().getExpiresAt();//过期时间
44              Instant issuedAt = accessToken.getToken().getIssuedAt();//发放token的时间
45              Date expiresAtDate = Date.from(expiresAt);
46              Date issuedAtDate = Date.from(issuedAt);
47              redisTemplate.setValueSerializer(RedisSerializer.java());
48              redisTemplate.opsForValue().set(isAccessTokenKey, authorization, TIMEOUT, Timel
49          }
50          if (isRefreshToken(authorization)) {
51              OAuth2Authorization.Token<OAuth2RefreshToken> refreshToken =
52                      authorization.getToken(OAuth2RefreshToken.class);
53              String tokenValue = refreshToken.getToken().getTokenValue();
54              String isRefreshTokenKey = buildAuthorizationKey(OAuth2ParameterNames.REFRESH_
55              Instant expiresAt = refreshToken.getToken().getExpiresAt();//过期时间
56              Instant issuedAt = refreshToken.getToken().getIssuedAt();//发放token的时间
57              Date expiresAtDate = Date.from(expiresAt);
58              Date issuedAtDate = Date.from(issuedAt);
59              redisTemplate.setValueSerializer(RedisSerializer.java());
60              redisTemplate.opsForValue().set(isRefreshTokenKey, authorization, TIMEOUT, Time
61
62          }
63          if (isIdToken(authorization)) {
64              OAuth2Authorization.Token<OidcIdToken> idToken =
65                      authorization.getToken(OidcIdToken.class);
66              String tokenValue = idToken.getToken().getTokenValue();
67              String isIdTokenKey = buildAuthorizationKey(OidcParameterNames.ID_TOKEN, token
68              Instant expiresAt = idToken.getToken().getExpiresAt();//过期时间
69              Instant issuedAt = idToken.getToken().getIssuedAt();//发放token的时间
70              Date expiresAtDate = Date.from(expiresAt);
71              Date issuedAtDate = Date.from(issuedAt);
```

```
74          }
75          if (isDeviceCode(authorization)) {
76              OAuth2Authorization.Token<OAuth2DeviceCode> deviceCode =
77                      authorization.getToken(OAuth2DeviceCode.class);
78
79              String tokenValue = deviceCode.getToken().getTokenValue();
80              String isDeviceCodeKey = buildAuthorizationKey(OAuth2ParameterNames.DEVICE_CODI
81              Instant expiresAt = deviceCode.getToken().getExpiresAt();//过期时间
82              Instant issuedAt = deviceCode.getToken().getIssuedAt();//发放token的时间
83              Date expiresAtDate = Date.from(expiresAt);
84              Date issuedAtDate = Date.from(issuedAt);
85              redisTemplate.setValueSerializer(RedisSerializer.java());
86              redisTemplate.opsForValue().set(isDeviceCodeKey, authorization, TIMEOUT, TimeUr
87          }
88          if (isUserCode(authorization)) {
89              OAuth2Authorization.Token<OAuth2UserCode> userCode =
90                      authorization.getToken(OAuth2UserCode.class);
91              String tokenValue = userCode.getToken().getTokenValue();
92              String isUserCodeKey = buildAuthorizationKey(OAuth2ParameterNames.USER_CODE, to
93              Instant expiresAt = userCode.getToken().getExpiresAt();//过期时间
94              Instant issuedAt = userCode.getToken().getIssuedAt();//发放token的时间
95              Date expiresAtDate = Date.from(expiresAt);
96              Date issuedAtDate = Date.from(issuedAt);
97              redisTemplate.setValueSerializer(RedisSerializer.java());
98              redisTemplate.opsForValue().set(isUserCodeKey, authorization, TIMEOUT, TimeUnit
99          }
100     }
101
102     @Override
103     public void remove(OAuth2Authorization authorization) {
104         List<String> keys = new ArrayList<>();
105         String idKey = buildAuthorizationKey(OAUTH2_PARAMETER_NAME_ID, authorization.getId
106         keys.add(idKey);
107         if (isState(authorization)) {
108             String state = authorization.getAttribute(OAuth2ParameterNames.STATE);
109             String isStateKey = buildAuthorizationKey(OAuth2ParameterNames.STATE, state);
110             keys.add(isStateKey);
111         }
112         if (isAuthorizationCode(authorization)) {
113             OAuth2Authorization.Token<OAuth2AuthorizationCode> authorizationCode =
114                     authorization.getToken(OAuth2AuthorizationCode.class);
115             String tokenValue = authorizationCode.getToken().getTokenValue();
116             String isAuthorizationCodeKey = buildAuthorizationKey(OAuth2ParameterNames.CODI
117             keys.add(isAuthorizationCodeKey);
118         }
119         if (isAccessToken(authorization)) {
120             OAuth2Authorization.Token<OAuth2AccessToken> accessToken =
```

```
123             String isAccessTokenKey = buildAuthorizationKey(OAuth2ParameterNames.ACCESS_TOI
124             keys.add(isAccessTokenKey);
125         }
126         if (isRefreshToken(authorization)) {
127             OAuth2Authorization.Token<OAuth2RefreshToken> refreshToken =
128                     authorization.getToken(OAuth2RefreshToken.class);
129             String tokenValue = refreshToken.getToken().getTokenValue();
130             String isRefreshTokenKey = buildAuthorizationKey(OAuth2ParameterNames.REFRESH_T
131             keys.add(isRefreshTokenKey);
132         }
133         if (isIdToken(authorization)) {
134             OAuth2Authorization.Token<OidcIdToken> idToken =
135                     authorization.getToken(OidcIdToken.class);
136             String tokenValue = idToken.getToken().getTokenValue();
137             String isIdTokenKey = buildAuthorizationKey(OidcParameterNames.ID_TOKEN, token\
138             keys.add(isIdTokenKey);
139         }
140         if (isDeviceCode(authorization)) {
141             OAuth2Authorization.Token<OAuth2DeviceCode> deviceCode =
142                     authorization.getToken(OAuth2DeviceCode.class);
143
144             String tokenValue = deviceCode.getToken().getTokenValue();
145             String isDeviceCodeKey = buildAuthorizationKey(OAuth2ParameterNames.DEVICE_CODI
146             keys.add(isDeviceCodeKey);
147         }
148         if (isUserCode(authorization)) {
149             OAuth2Authorization.Token<OAuth2UserCode> userCode =
150                     authorization.getToken(OAuth2UserCode.class);
151             String tokenValue = userCode.getToken().getTokenValue();
152             String isUserCodeKey = buildAuthorizationKey(OAuth2ParameterNames.USER_CODE, to
153             keys.add(isUserCodeKey);
154         }
155         redisTemplate.delete(keys);
156     }
157
158     @Override
159     public OAuth2Authorization findById(String id) {
160         return (OAuth2Authorization) Optional.ofNullable(redisTemplate.opsForValue().get(bu
161     }
162
163     @Override
164     public OAuth2Authorization findByToken(String token, OAuth2TokenType tokenType) {
165         Assert.hasText(token, "token cannot be empty");
166         Assert.notNull(tokenType, "tokenType cannot be empty");
167         redisTemplate.setValueSerializer(RedisSerializer.java());
168         return (OAuth2Authorization) redisTemplate.opsForValue().get(buildAuthorizationKey(
169     }
```

```
172    private boolean isState(OAuth2Authorization authorization) {
173        return Objects.nonNull(authorization.getAttribute(OAuth2ParameterNames.STATE));
174    }
175
176
177    private boolean isAuthorizationCode(OAuth2Authorization authorization) {
178        OAuth2Authorization.Token<OAuth2AuthorizationCode> authorizationCode =
179                authorization.getToken(OAuth2AuthorizationCode.class);
180        return Objects.nonNull(authorizationCode);
181    }
182
183
184    private boolean isAccessToken(OAuth2Authorization authorization) {
185        OAuth2Authorization.Token<OAuth2AccessToken> accessToken =
186                authorization.getToken(OAuth2AccessToken.class);
187        return Objects.nonNull(accessToken) && Objects.nonNull(accessToken.getToken().getT
188    }
189
190    private boolean isRefreshToken(OAuth2Authorization authorization) {
191        OAuth2Authorization.Token<OAuth2RefreshToken> refreshToken =
192                authorization.getToken(OAuth2RefreshToken.class);
193        return Objects.nonNull(refreshToken) && Objects.nonNull(refreshToken.getToken().get
194    }
195
196    private boolean isIdToken(OAuth2Authorization authorization) {
197        OAuth2Authorization.Token<OidcIdToken> idToken =
198                authorization.getToken(OidcIdToken.class);
199        return Objects.nonNull(idToken) && Objects.nonNull(idToken.getToken().getTokenValue
200    }
201
202    private boolean isDeviceCode(OAuth2Authorization authorization) {
203        OAuth2Authorization.Token<OAuth2DeviceCode> deviceCode =
204                authorization.getToken(OAuth2DeviceCode.class);
205        return Objects.nonNull(deviceCode) && Objects.nonNull(deviceCode.getToken().getToke
206    }
207
208    private boolean isUserCode(OAuth2Authorization authorization) {
209        OAuth2Authorization.Token<OAuth2UserCode> userCode =
210                authorization.getToken(OAuth2UserCode.class);
211        return Objects.nonNull(userCode) && Objects.nonNull(userCode.getToken().getTokenVa
212    }
213
214
215    /**
216     * redis key 构建
217     *
218     * @param type  授权类型
```

```
221     */
222    private String buildAuthorizationKey(String type, String value) {
223        return AUTHORIZATION_TYPE.concat("::").concat(type).concat("::").concat(value);
224    }
225 }
```

标签： Spring Boot Spring Cloud 话题： 1024一起掘金

本文收录于以下专栏

Spring Authorization Server 精讲 专栏目录

本专栏将深入讲解Spring Authorization Server在实践中的扩展点，希...

56 订阅 · 11 篇文章

订阅

上一篇 Spring Authorization Ser...

下一篇 Spring Authorization Ser...

评论 0

登录 / 注册 即可发布评论！

暂无评论数据

目录 收起

相关推荐

spring-authorization-server系列--Oauth介绍
28阅读 · 0点赞

Spring Authorization Server 入门教程
1.1k阅读 · 1点赞

spring-authorization-server系列--authorization_code模式实现流程
76阅读 · 0点赞

Spring Authorization Server 全新授权服务器整合使用
3.3k阅读 · 16点赞

Spring Authorization Server 快速上手
444阅读 · 0点赞

为你推荐

Spring Authorization Server (4) 客户端、资源服务、授权服务 源码加流程细讲 再也...

Spring Authorization Server (3) so seasy 集成第三方【gitee、github】oauth2登录

爱吃西瓜的胖娃 11月前 1.1k 3 3 Spring ... Spring ...

Spring Authorization Server (10) 授权服务的JWK密钥对生成和JWT信息扩展

爱吃西瓜的胖娃 9月前 796 1 评论 Spring ... Spring ...

Spring Authorization Server 授权服务器

程序员Mark 1年前 6.5k 25 9 Spring ...

Spring Authorization Server入门 (二) Spring Boot整合Spring Authorization Server

叹雪飞花 1年前 9.4k 35 107 Java

Spring Authorization Server 全新授权服务器整合使用

冷冷zz 3年前 3.3k 16 3 Java Spring B...

Spring Authorization Server的使用

huan1993 3年前 7.5k 21 6 Spring 后端

Spring Authorization Server Password授权扩展

hundanli 5月前 386 1 2 Java Spring B...

Spring Authorization Server入门 (十六) Spring Cloud Gateway对接认证服务

叹雪飞花 11月前 3.5k 19 47 Spring ... Spring ... 安全

Spring Authorization Server入门 (九) Spring Boot引入Resource Server对接认证服务

叹雪飞花 1年前 2.4k 16 9 Spring Spring ...

Spring Authorization Server入门 (八) Spring Boot引入Security OAuth2 Client对接认...

叹雪飞花 1年前 3.7k 17 54 Spring ... Spring

Spring 官方发起Spring Authorization Server 项目

码农小胖哥 4年前 4.6k 7 11 Java Spring B...

Spring Authorization Server + Oauth2 配置认证服务器与资源服务器

张蕊是胖胖 1年前 3.3k 11 6 后端

Spring Authorization Server入门 (十七) Vue项目使用授权码模式对接认证服务

叹雪飞花 11月前 1.2k 9 18 Vue.js 安全 Spring ...

爱吃西瓜的胖娃 9月前 877 3 3 Spring … Spring …