



# Spring Security：身份验证提供程序AuthenticationProvider介绍

原创 ITKaven 已于 2022-01-25 20:07:53 修改 阅读量4.8k 收藏 3 点赞数 4

分类专栏： Spring Security 文章标签： spring java 后端

Spring Security 专栏收录该内容

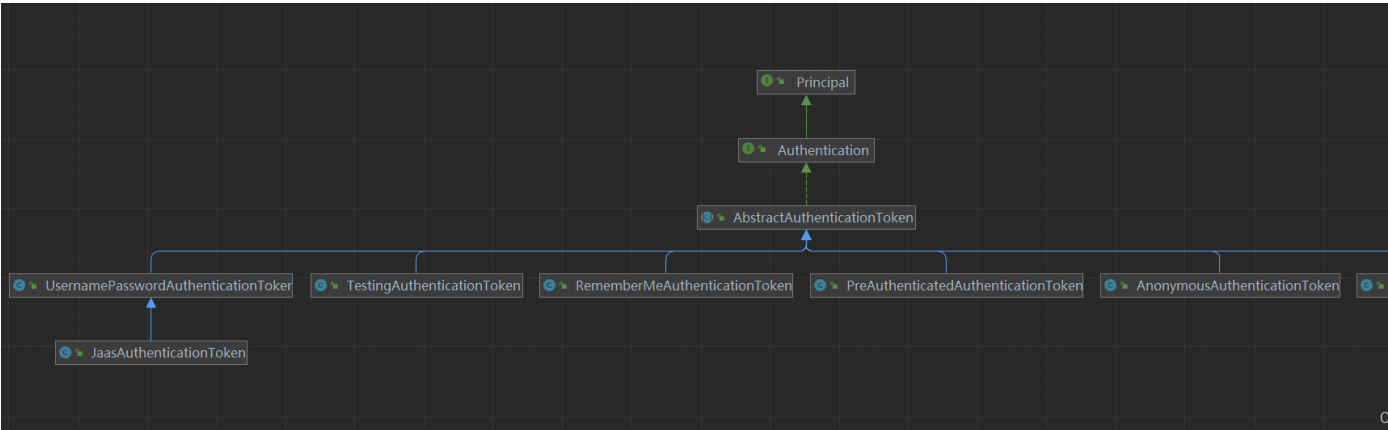
31 订阅 12 篇文章

## AuthenticationProvider

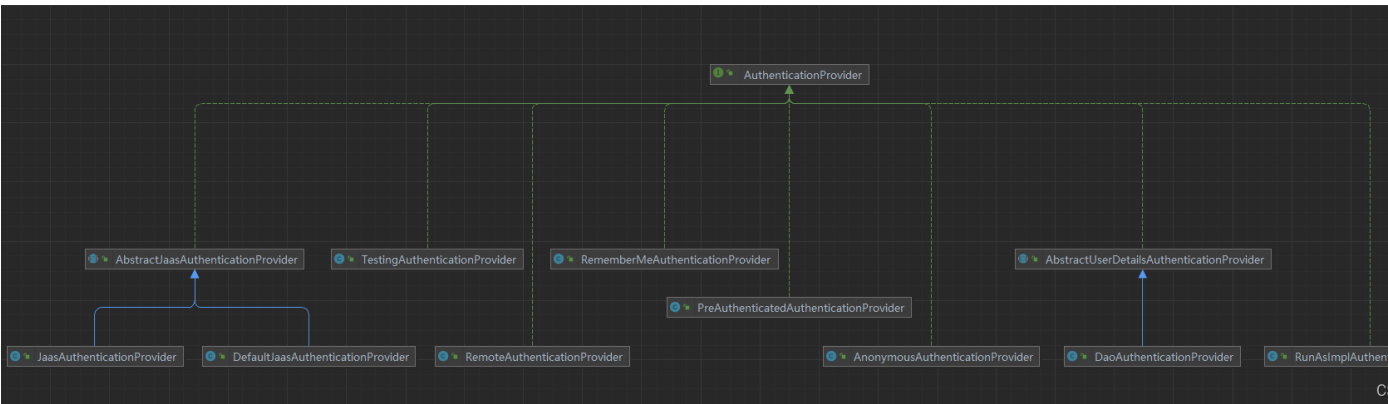
AuthenticationProvider 可以处理特验证的 Authentication 实现。

- Spring Security：身份验证令牌Authentication介绍与Debug分析

Authentication 接口及其实现类如下图所示：



AuthenticationProvider 接口及其实现类如下图所示：



```
1 public interface AuthenticationProvider {
2
3     /**
4      * 使用与AuthenticationManager.authenticate(Authentication)相同的策略执行身份验证
5      * 参数:
6      * authentication - 身份验证请求对象
7      * 返回:
8      * 一个完全经过身份验证的对象，包括凭据
9      * 如果AuthenticationProvider无法支持传递的Authentication对象的身份验证
10     * 则可能返回null
11     * 在没动情况下 将尝试去支持提供的Authentication类的下一个AuthenticationProvider
```

## TestingAuthenticationProvider

验证 TestingAuthenticationToken 的 AuthenticationProvider 实现，此实现的目的是用于 单元测试

ITKaven 关注

```
1 public class TestingAuthenticationProvider implements AuthenticationProvider {
2     // 直接返回authentication实例, 用于单元测试
3     public Authentication authenticate(Authentication authentication)
4         throws AuthenticationException {
5         return authentication;
6     }
7
8     // 支持TestingAuthenticationToken类型
9     public boolean supports(Class<?> authentication) {
10         return TestingAuthenticationToken.class.isAssignableFrom(authentication);
11     }
12 }
```

## AnonymousAuthenticationProvider

验证 `AnonymousAuthenticationToken` 的 `AuthenticationProvider` 实现。要成功验证, `AnonymousAuthenticationToken.getKeyHash()` 必须与此类的 `getKeyHash()` 匹配。

```
1 public class AnonymousAuthenticationProvider implements AuthenticationProvider,
2     MessageSourceAware {
3     // 用于访问来自MessageSource的消息
4     protected MessageSourceAccessor messages = SpringSecurityMessageSource.getAccessor();
5     // 识别Authentication实例是否由授权客户生成的key
6     private String key;
7
8     // 使用key构造AnonymousAuthenticationProvider实例
9     public AnonymousAuthenticationProvider(String key) {
10         Assert.hasLength(key, "A Key is required");
11         this.key = key;
12     }
13 }
```



## RememberMeAuthenticationProvider

验证 `RememberMeAuthenticationToken` 的 `AuthenticationProvider` 实现。要成功验证, `RememberMeAuthenticationToken.getKeyHash()` 必须与此类的 `getKeyHash()` 匹配。

```
1 public class RememberMeAuthenticationProvider implements AuthenticationProvider,
2     InitializingBean, MessageSourceAware {
3     // 用于访问来自MessageSource的消息
4     protected MessageSourceAccessor messages = SpringSecurityMessageSource.getAccessor();
5     // 识别Authentication实例是否由授权客户生成的key
6     private String key;
7
8     // 使用key构造AnonymousAuthenticationProvider实例
9     public RememberMeAuthenticationProvider(String key) {
10         Assert.hasLength(key, "key must have a length");
11         this.key = key;
12     }
13 }
```



## RemoteAuthenticationProvider

使用 `RemoteAuthenticationManager` 验证身份验证请求的客户端对象。此类创建一个新的 `Authentication` 对象, 包括请求 `Authentication` 对象的 `principal` 和 `RemoteAuthenticationManager` 返回的 `GrantedAuthority` 集合。

```
1 public class RemoteAuthenticationProvider implements AuthenticationProvider,
2     InitializingBean {
3
4     // RemoteAuthenticationManager用于允许远程客户端尝试身份验证
5     private RemoteAuthenticationManager remoteAuthenticationManager;
6
7     // 检查remoteAuthenticationManager属性是否为null
8     public void afterPropertiesSet() {
9         Assert.notNull(this.remoteAuthenticationManager,
10             "remoteAuthenticationManager is required");
11     }
12 }
```



ITKaven

关注

```
9 |  
10 |  
11 |         "remoteAuthenticationManager is mandatory");
```



## RemoteAuthenticationManager

允许远程客户端尝试身份验证。

```
1 | public interface RemoteAuthenticationManager {  
2 |     /**  
3 |      * 尝试使用提供的用户名和密码对远程客户端进行身份验证  
4 |      * 如果身份验证成功，将返回一组GrantedAuthority对象  
5 |      * 参数：  
6 |      * 用户名 - 远程客户端希望进行身份验证的用户名  
7 |      * 密码 - 远程客户端希望验证的密码  
8 |      * 返回：  
9 |      * 指定的用户名和密码可以访问的所有授予权限  
10 |     */  
11 |     Collection<? extends GrantedAuthority> attemptAuthentication(String username,  
12 |         String password) throws RemoteAuthenticationException;  
13 | }
```

## RemoteAuthenticationManagerImpl

RemoteAuthenticationManager 接口的实现类。

```
1 | public class RemoteAuthenticationManagerImpl implements RemoteAuthenticationManager,  
2 |     InitializingBean {  
3 |     // AuthenticationManager实例，用于处理身份验证请求  
4 |     private AuthenticationManager authenticationManager;  
5 |  
6 |     // 检查authenticationManager属性是否为null  
7 |     public void afterPropertiesSet() {  
8 |         Assert.notNull(this.authenticationManager, "authenticationManager is required");  
9 |     }  
10 |  
11 |     // 尝试验证
```



## PreAuthenticatedAuthenticationProvider

处理预验证的验证请求。该请求通常来自 `AbstractPreAuthenticatedProcessingFilter` 的子类。此身份验证提供程序不会对身份验证请求执行任何检查，因为他们应该已经过预身份验证。但是，例如 `AuthenticationUserDetailsService` 实现（允许基于 `Authentication` 对象加载 `UserDetails` 对象的接口，用于验证的用户加载 `UserDetails`）可能仍会引发 `UsernameNotFoundException`。

```
1 | public class PreAuthenticatedAuthenticationProvider implements AuthenticationProvider,  
2 |     InitializingBean, Ordered {  
3 |     private static final Log logger = LoggerFactory  
4 |         .getLog(PreAuthenticatedAuthenticationProvider.class);  
5 |  
6 |     // 用于为经过身份验证的用户加载UserDetails  
7 |     private AuthenticationUserDetailsService<PreAuthenticatedAuthenticationToken> preAuthenticatedUserDetailsService = null;  
8 |     // 用于检查加载的UserDetails对象的状态  
9 |     private UserDetailsChecker userDetailsChecker = new AccountStatusUserDetailsChecker();  
10 |    // 当令牌（验证请求）被拒绝（验证失败）时是否抛出异常  
11 |    private boolean throwExceptionWhenTokenRejected = false;
```



## RunAsImplAuthenticationProvider



ITKaven

关注

可以对 `RunAsUserToken` 进行身份验证的 `AuthenticationProvider` 实现。与 `RunAsImplAuthenticationProvider` 的密钥匹配的哈希码的任何 `RunAsUserToken` 为有效。如果密钥不匹配，则会抛出 `BadCredentialsException`。

```
1 public class RunAsImplAuthenticationProvider implements InitializingBean,
2     AuthenticationProvider, MessageSourceAware {
3     // 用于访问来自MessageSource的消息
4     protected MessageSourceAccessor messages = SpringSecurityMessageSource.getAccessor();
5     // 识别Authentication实例是否由授权客户生成的key
6     private String key;
7
8     // 检查key属性是否为null
9     public void afterPropertiesSet() {
10         Assert.notNull(key,
11             "A key is required and should match that configured for the RunAsManagerImpl");
12     }
13 }
```



## AbstractUserDetailsAuthenticationProvider

一个基本的 `AuthenticationProvider`，它允许子类覆盖和使用 `UserDetails` 对象。该类旨在响应 `UsernamePasswordAuthenticationToken` 身份验证请求后，将创建一个 `UsernamePasswordAuthenticationToken` 实例，并将其返回给调用者。

通过存储放置在 `UserCache` 中的 `UserDetails` 对象来处理缓存，这确保可以验证具有相同用户名的后续请求，而无需查询 `UserDetailsService`。需要时，如果用户出现密码错误，将查询 `UserDetailsService` 以确认是否使用了最新密码进行比较。只有无状态应用程序才可能需要缓存。例如，在普通的 Web 应用中，`SecurityContext` 存储在用户的会话中，并且用户不会在每个请求上重新进行身份验证。因此，默认缓存实现是 `NullUserCache`。

```
1 public abstract class AbstractUserDetailsAuthenticationProvider implements
2     AuthenticationProvider, InitializingBean, MessageSourceAware {
3
4     protected final Log logger = LoggerFactory.getLog(getClass());
5     protected MessageSourceAccessor messages = SpringSecurityMessageSource.getAccessor();
6     private UserCache userCache = new NullUserCache();
7     private boolean forcePrincipalAsString = false;
8     protected boolean hideUserNotFoundExceptions = true;
9     private UserDetailsChecker preAuthenticationChecks = new DefaultPreAuthenticationChecks();
10    private UserDetailsChecker postAuthenticationChecks = new DefaultPostAuthenticationChecks();
11    private GrantedAuthoritiesMapper authoritiesMapper = new NullAuthoritiesMapper();
12 }
```



## DaoAuthenticationProvider

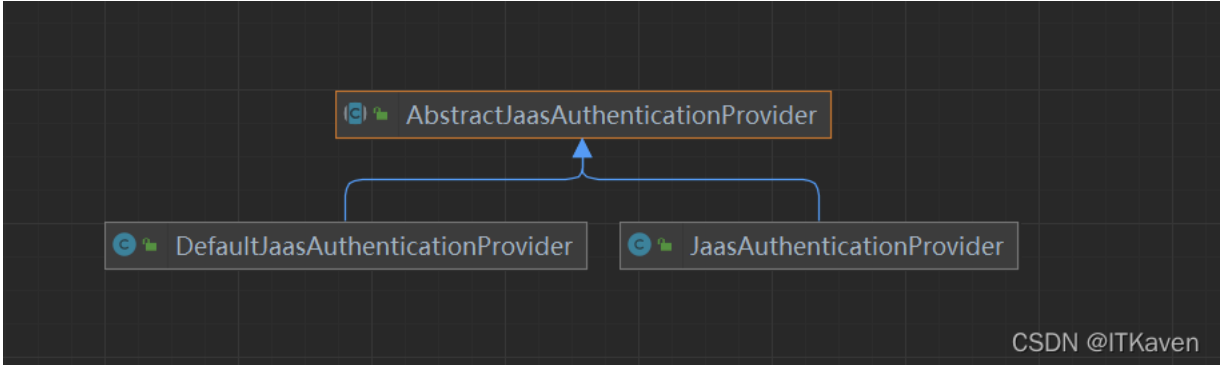
从 `UserDetailsService` 加载用户详细信息的 `AuthenticationProvider` 实现。

- Spring Security: 用户服务UserDetailsService源码分析

```
1 public class DaoAuthenticationProvider extends AbstractUserDetailsAuthenticationProvider {
2
3     /**
4      * 用于在未找到用户时执行PasswordEncoder.matches(CharSequence, String) 的明文密码
5      * 以避免SEC-2056 (黑客嗅探用户是否存在)
6      */
7     private static final String USER_NOT_FOUND_PASSWORD = "userNotFoundPassword";
8     // 密码编码器
9     private PasswordEncoder passwordEncoder;
10
11     /**
12      *
13      */
14 }
```



还有三个 `AuthenticationProvider` 接口的实现类，用于验证 `JaasAuthenticationToken`，大家可以自行阅读源码，这里就不进行介绍了。



身份验证提供程序 `AuthenticationProvider` 介绍就到这里，如果博主有说错的地方或者大家有不同的见解，欢迎大家评论补充。

文章知识点与官方知识档案匹配，可进一步学习相关知识

Java技能树 首页 概览 149725 人正在系统学习中

spring Security 类结构了解-作用细分

helloWorldAndYou的

`Authentication` 用于身份验证。 `RememberMeAuthenticationToken` 记住&保持登录状态 `UsernamePasswordAuthenticationToken` 一种简单的实现，顾名思义，这个类包含

springSecurity---AuthenticationProvider解析

池

首先进入到`AuthenticationProvider`源码中可以看到它只是个简单的接口里面也只有两个方法：`public interface AuthenticationProvider { // 具体认证流程 Authentication aut`

1 条评论



zm199711 热评 你这是什么包下面的，是原生的吗

...AuthenticationProvider解析\_authenticationprovider

1.`AuthenticationProvider`是接口,抽象方法是`authenticate()`,之前的博文介绍过,`AuthenticationProvider`是认证逻辑的提供者,每一个实现`AuthenticationProvider`接口的类,都将

Spring Security(七)-- AuthenticationProvider

@ComponentpublicclassCustomAuthenticationProviderimplementsAuthenticationProvider{@AutowiredprivateMybatisUserDetailsServiceuserServiceDetailsService;@Autowiredrec

Spring Security：概念模型 AuthenticationProvider 认证提供者

Details Inside S

Spring Security中`AuthenticationProvider`接口抽象建模了认证提供者这一概念，某个认证提供者能认证符合某种特征的认证令牌`Authentication`。 `Spring Security`针对常见

Spring Boot Security自定义AuthenticationProvider 最新发布

2201\_75600005的

在配置器中我们去实例化一个认证管理器`AuthenticationManager`，这个认证管理器中包含了两个认证器，分别是`MobilecodeAuthenticationProvider`（手机验证码）、`Dao`

...Security(学习笔记) --AuthenticationProvider案例演示梳理!\_spri...

response,authentication)->{response.getWriter().write(authentication.getName());}).permitAll()).csrf(c->c.disable());returnhttp.build();}@Bean@Order(1)SecurityFilterChe

轻松搞定自定义Oauth2授权模式-AuthenticationProvider功能介绍...

publicbooleansupports(Class<?> authenticationType){ returnauthenticationType.equals(UsernamePasswordAuthenticationToken.class); } } 然后,将该自定义`Authenticatio`

sso与spring security整合 预认证场景 PreAuthenticated

cmqn6968的

1. 当前项目的有个要求，sso与spring security结合使用2. 使用spring security的“预认证场景” `PreAuthenticated`参考文档<http://www.family168.com/tuto...>

Spring Security Oauth2配置类AuthorizationServerConfigurerAdapter

wangooo的

`AuthorizationServerConfigurerAdapter`中: `ClientDetailsServiceConfigurer`：用来配置客户端详情服务（`ClientDetailsService`），客户端详情信息在这里进行初始化，你能

AuthenticationManager、AuthenticationProvider、UserDetailsService的...

一、`AuthenticationManager`里面保存有很多实例,其中一个就是`AuthenticationProvider`;而每个`AuthenticationProvider`里面都有一个`UserDetailsService`。 二、默认情况下,3

AuthenticationProvider类authenticate方法的调用,以及令牌的保持\_aut...

`PasswordAuthenticationProvider`类实现了`AuthenticationProvider`接口,这是 `Spring Security` 中用于自定义身份验证逻辑的方式。`authenticate`方法是`AuthenticationProvide`

Spring Security Authentication Provider

neweastsun的

Spring Security Authentication Provider 介绍 本文介绍spring security如何相比简单的`UserDetailService`上实现灵活的认证。 `Authentication Provider spring security`提供了

spring security 5 (5)-自定义认证

JAVAT

第3篇讲过，用户登录时，系统会读取用户的`UserDetails`对其认证，认证过程是由系统自动完成的，主要是检查用户密码。而在现实中，登录方式并不一定是检查密码，也

spring security自定义AuthenticationProvider,验证规则xml配置方式-CS...

认证是由 `AuthenticationManager` 来管理的,但是真正进行认证的是 `AuthenticationManager` 中定义的 `Authentica`

spring boot中spring security实现单点登录，传统模式(一) 热门推荐



ITKaven

关注

单点登录是什么？一个系统中可能会引用别的很多系统。单点登录就是解决，一次登录，就可以访问所有的系统。每次浏览器向一个域名发送http请求，会去查找域名的

springboot （十五）安全框架 Spring Security zzkeung的  
1.pom依赖 2.代码演示 官方参考 Spring Security是一个提供身份验证，授权和保护以防止常见攻击的框架。凭借对命令式和响应式应用程序的一流支持，它是用于保护基

Spring Security：身份验证令牌Authentication介绍与Debug分析 ka  
在Spring Security中，通过Authentication来封装用户的验证信息以及用户验证实现，Authentication可以是需要验证和已验证的用户信息封装。接下来，博主介绍Authentic

spring security OAuth2AuthenticationProcessingFilter的token认证流程解析 a807719447的  
在ResourceServer中需要对token进行校验需要走如下过滤器，我们来分析下token校验的认证过程是什么样的 OAuth2AuthenticationProcessingFilter.doFilter public void c

第 15 章 预先认证 weixin\_33734785的  
第15章预先认证 预先认证是指用户在进入系统给钱，就已经通过某种机制进行过身份认证，请求中已经附带了身份认证的信息，这时我们只需要从获得这些身份认证信息

Spring Security# Multiple DaoAuthenticationProvider 来啊 快  
正文有三种情况我们需要配置多个AuthenticationProvider，甚至是自定义AuthenticationProvider： 1. 用户认证信息存储在多个地方 2. 在同一个地方但是需要多种授权方

关于我们 招贤纳士 商务合作 寻求报道 400-660-0108 kefu@csdn.net 在线客服 工作时间 8:30-22:00  
公安备案号11010502030143 京ICP备19004658号 京网文〔2020〕1039-165号 经营性网站备案信息 北京互联网违法和不良信息举报中心  
家长监护 网络110报警服务 中国互联网举报中心 Chrome商店下载 账号管理规范 版权与免责声明 版权申诉 出版物许可证 营业执照  
©1999-2024北京创新乐知网络技术有限公司



ITKaven  
码龄7年 暂无认证

542 2万+ 105万+ 227万+ 等级  
原创 周排名 总排名 访问

2万+ 1368 1992 937 3844  
积分 粉丝 获赞 评论 收藏



私信

关注

AI圈早知道，每日最新动态  
了解全球AI新鲜事！  
立即参与

大额流量券免费送  
发布一篇就可获得！  
去查看

搜博主文章



热门文章

- MyBatis-Plus 之分页查询 167673
- 同步和异步的区别 89904
- 怎么保存退出 vim 编辑 70673
- MySQL的driverClassName、url 50048
- Spring 中 ClassPathXmlApplicationContext 类的简单使用 40177






分类专栏

JAVA 30篇



ITKaven

关注

	Spring	9篇
	Spring Boot	31篇
	Spring Security	12篇
	Spring Cloud	12篇
	Spring Cloud Alibaba	12篇

最新评论

- ElasticJob-Lite：作业监听器  
一只烤鸭朝北走啊: 好久没看了，有点忘记了，我那会儿做的是定时任务在分布式 ...
- ElasticJob-Lite：作业监听器  
AKKO111: 那感觉没必要用这个Distribute的listener了，直接用ElasticJobServiceLo ...
- Nginx：Nginx添加SSL实现HTTPS访问  
风度翩翩609: 我这边配完 http+域名可以，但是https+域名就不行了
- 使用Vue和Spring Boot实现文件下载  
一入程序无退路: 下载按钮点第二次就不太好使怎么回事
- Docker容器中使用PING命令报错：bash:...  
桂哥317: 干脆利落、实测有效

最新文章

- MySQL: 备份 & 导入备份
- RabbitMQ：Docker Compose部署RabbitMQ集群
- VirtualBox：系统安装重启后又要重新安装

2022年 80篇	2021年 55篇
2020年 206篇	2019年 10篇
2018年 220篇	2017年 31篇





目录

- AuthenticationProvider
- TestingAuthenticationProvider
- AnonymousAuthenticationProvider
- RememberMeAuthenticationProvider
- RemoteAuthenticationProvider
  - RemoteAuthenticationManager
  - RemoteAuthenticationManagerImpl
- PreAuthenticatedAuthenticationProvider
- RunAsImplAuthenticationProvider
- AbstractUserDetailsAuthenticationProvider
- DaoAuthenticationProvider



ITKaven

关注