链滴

spring security 支持多个认证方法

作者：wenyl

原文链接：https://ld246.com/article/1629791519490

来源网站：链滴

spring security通过定义多个AuthenticationProvider来实现不同的认证方式。

1、自定义认证器

自定义认证器可以通过实现AuthenticationProvider接口来实现，这个接口，一共有两个方法

```
public interface AuthenticationProvider {

    /**
     * Performs authentication with the same contract as
     * {@link org.springframework.security.authentication.AuthenticationManager#authenticate
Authentication)}
     * .
     * @param authentication the authentication request object.
     * @return a fully authenticated object including credentials. May return
     * <code>null</code> if the <code>AuthenticationProvider</code> is unable to support
     * authentication of the passed <code>Authentication</code> object. In such a case,
     * the next <code>AuthenticationProvider</code> that supports the presented
     * <code>Authentication</code> class will be tried.
     * @throws AuthenticationException if authentication fails.
     */
    Authentication authenticate(Authentication authentication) throws AuthenticationException


    /**
     * Returns <code>true</code> if this <Code>AuthenticationProvider</code> supports th

     * indicated <Code>Authentication</code> object.
     * <p>
     * Returning <code>true</code> does not guarantee an
     * <code>AuthenticationProvider</code> will be able to authenticate the presented
```

```
     * instance of the <code>Authentication</code> class. It simply indicates it can
     * support closer evaluation of it. An <code>AuthenticationProvider</code> can still
     * return <code>null</code> from the {@link #authenticate(Authentication)} method to
     * indicate another <code>AuthenticationProvider</code> should be tried.
     * </p>
     * <p>
     * Selection of an <code>AuthenticationProvider</code> capable of performing
     * authentication is conducted at runtime the <code>ProviderManager</code>.
     * </p>
     * @param authentication
     * @return <code>true</code> if the implementation can more closely evaluate the
     * <code>Authentication</code> class presented
     */
    boolean supports(Class<?> authentication);

}
```

第一个方法参数是Authencation，通过Authencation我们又可以自定义token，也是通过实现Authecation来自定义token，这个token会在第二个方法中使用；

第二个方法用于判断当前的认证其是否支持指定类型的token；

2、自定义实现一个认证器

```
/**
 * @author Mr.Wen
 * @version 1.0
 * @date 2021-08-19 14:20
 */
@Service
public class SecurityAuthenticationProvider implements AuthenticationProvider {
    @Resource
    private UserDetailsServiceImpl userDetailsService;
    @Resource
    private SecurityPasswordEncoder passwordEncoder;
    @Override
    public Authentication authenticate(Authentication authentication) throws AuthenticationEx
eption {
        String username = authentication.getName();
        String password = (String) authentication.getCredentials();
        if(StringUtils.isBlank(username)){
            throw new UsernameNotFoundException("username用户名不可以为空");
        }
        if(StringUtils.isBlank(password)){
            throw new BadCredentialsException("密码不可以为空");
        }
        //获取用户信息
        SecurityUserDetails user = (SecurityUserDetails)userDetailsService.loadUserByUsername(
sername);
        //比较前端传入的密码明文和数据库中加密的密码是否相等
        if (!passwordEncoder.matches(password, user.getPassword())) {
            //发布密码不正确事件
            throw new BadCredentialsException("password密码不正确");
```

```
        }
        //获取用户权限信息
        Collection<? extends GrantedAuthority> authorities = user.getAuthorities();
        return new SecurityAuthenticationToken(user, password, authorities);
    }

    @Override
    public boolean supports(Class<?> authentication) {
        return authentication.equals(SecurityAuthenticationToken.class);
    }
}
```

UserDetails和对应的service是自己实现的，密码处理器也是自己实现的，这些都无需关心，换成自系统的实现就可以了，第一个方法，就是认证，返回一个token交给spring security后续流程处理；二个方法判断当前token是不是认证其支持的token。

3、配置多认证器

再spring security的配置类中，配置认证器。

```
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        // 加入自定义的安全认证
        auth.
            authenticationProvider(securityAuthenticationProvider)
            .authenticationProvider(getKmairCasAuthenticationProvider());
    }
```

第一个provider是用spring注入的，第二个则是再方法中创建了provider；

到此就可以实现多认证器。