spring boot整合shiro

安全框架Shiro和Spring Security比较,本文主要围绕Shiro进行学习

一 Shiro 是一个强大而灵活的开源安全框架,能够清晰的处理认证 授权 管理会话以及,密码加密

01 .认证与授权相关概念

安全实体: 系统需要保护的具体对象数据

权限: 系统相关的功能操作,例如基本的CRUD

Authentication:身份认证授权/登录,验证用户是否拥有相应的身份

Authorization: 授权,即权限的认证,认证某个已认证的用户是否拥有某个权限

Session Manager : 会话管理.即用户登录后就是一次会话,在没有退出之前,所有信息都在会话中

Cryptography: 加密,保护数据的安全性

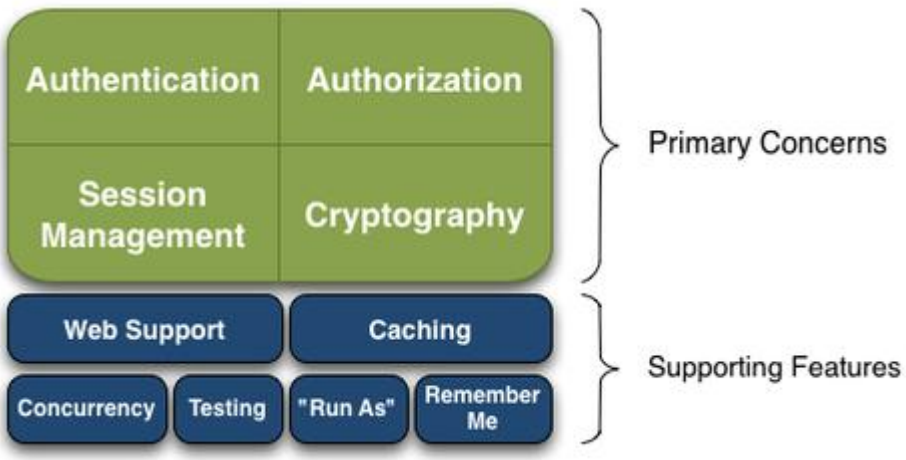
Web Support:web支持

Caching: 缓存

Concurrency: shiro支持多线程并发验证,一个线程中开启另一个线程,能把权限自动传播过去:

Remember Me:记住我

2. shiro四大核心,.如下是shiro架构

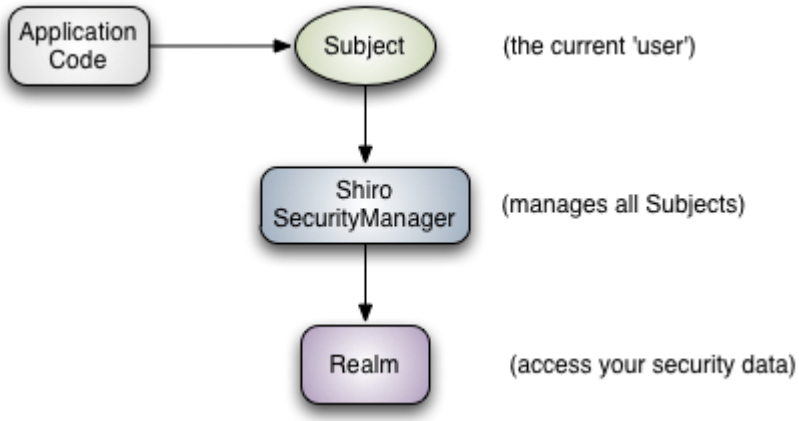


03 Shiro三个核心组件:Subject .SecurityManager和Realms

Subject: 主体,代表当前"用户",所有Subject都绑定到SecurityManager,是一个抽象的概念,与当前应用交互任何东西都是Subject,如网络爬虫,机器人,与Subject的所有交互都会委托给SecurityManager;

SecurityManager:安全管理器: 即所有与安全有关的操作都会与SecuityManager交互,管理所有的Subject;是Shiro的核心,它负责与后边介绍的其他组件进行交互

Realm: 域 , shiro从Realm获取安全数据(如 用户,角色,权限)SecurityManager要验证用户身份,那么他需要从Realm获取相应的用户,已确定使用户身份是否合法,也需要从Realm得到用户相应的角色/权限进行验证用于是否能进行操作,可以吧Realm看成DataSource.,即安全数据源



04 以下已springboot的一个项目中shiro运用为例

下面是pom.xml中相关jar

```
<!--shiro -->
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-core</artifactId>
            <version>1.3.2</version>
        </dependency>
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-spring</artifactId>
            <version>1.3.2</version>
        </dependency>
        <!-- shiro ehcache -->
        <dependency>
            <groupId>org.apache.shiro</groupId>
            <artifactId>shiro-ehcache</artifactId>
            <version>1.3.2</version>
        </dependency>
        <dependency>
            <groupId>com.github.theborakompanioni</groupId>
            <artifactId>thymeleaf-extras-shiro</artifactId>
            <version>1.2.1</version>
        </dependency>
```

ShiroConfig配置文件

```
import at.pollux.thymeleaf.shiro.dialect.ShiroDialect;
import com.prostate.common.config.Constant;

import com.prostate.common.redis.shiro.RedisCacheManager;
```

```
import com.prostate.common.redis.shiro.RedisManager;
import com.prostate.common.redis.shiro.RedisSessionDAO;
import com.prostate.system.shiro.UserRealm;
//import org.apache.shiro.cache.CacheManager;
import net.sf.ehcache.CacheManager;
import org.apache.shiro.cache.ehcache.EhCacheManager;
import org.apache.shiro.mgt.SecurityManager;
import org.apache.shiro.session.SessionListener;
import org.apache.shiro.session.mgt.eis.MemorySessionDAO;
import org.apache.shiro.session.mgt.eis.SessionDAO;
import org.apache.shiro.spring.LifecycleBeanPostProcessor;
import org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor;
import org.apache.shiro.spring.web.ShiroFilterFactoryBean;
import org.apache.shiro.web.mgt.DefaultWebSecurityManager;
import org.apache.shiro.web.session.mgt.DefaultWebSessionManager;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.cache.ehcache.EhCacheCacheManager;
import org.springframework.cache.ehcache.EhCacheManagerFactoryBean;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;

import java.util.ArrayList;
import java.util.Collection;
import java.util.LinkedHashMap;

/**
 * @author*/


@Configuration
public class ShiroConfig {
    @Value("${spring.redis.host}")
    private String host;
    @Value("${spring.redis.password}")
    private String password;
    @Value("${spring.redis.port}")
    private int port;
    @Value("${spring.redis.timeout}")
    private int timeout;

    @Value("${spring.cache.type}")
    private String cacheType;

    @Value("${server.session-timeout}")
    private int tomcatTimeout;

//    @Autowired
//    CacheManager cacheManager;

    @Bean

    public static LifecycleBeanPostProcessor getLifecycleBeanPostProcessor() {
```

```
        return new LifecycleBeanPostProcessor();
    }

    /**
     * ShiroDialect，为了在thymeleaf里使用shiro的标签的bean
     *
     * @return
     */
    @Bean
    public ShiroDialect shiroDialect() {
        return new ShiroDialect();
    }
//  ShiroFilterFactoryBean  为了生成ShiroFilter,处理拦截资源文件问题
//它主要保持三项数据,securityManager,filters,fiterChainDefinition
//  Shiro验证URL时,匹配成功就不在匹配,自上而下
//
    @Bean
    ShiroFilterFactoryBean shiroFilterFactoryBean(SecurityManager securityManager) {
        ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();
        shiroFilterFactoryBean.setSecurityManager(securityManager);
        shiroFilterFactoryBean.setLoginUrl("/login");
        shiroFilterFactoryBean.setSuccessUrl("/index");
        shiroFilterFactoryBean.setUnauthorizedUrl("/403");
        LinkedHashMap<String, String> filterChainDefinitionMap = new LinkedHashMap<>();
        filterChainDefinitionMap.put("/css/**", "anon");
        filterChainDefinitionMap.put("/js/**", "anon");
        filterChainDefinitionMap.put("/fonts/**", "anon");
        filterChainDefinitionMap.put("/img/**", "anon");
        filterChainDefinitionMap.put("/docs/**", "anon");
        filterChainDefinitionMap.put("/druid/**", "anon");
        filterChainDefinitionMap.put("/upload/**", "anon");
        filterChainDefinitionMap.put("/files/**", "anon");
        filterChainDefinitionMap.put("/logout", "logout");
        filterChainDefinitionMap.put("/", "anon");
        filterChainDefinitionMap.put("/blog", "anon");
        filterChainDefinitionMap.put("/blog/open/**", "anon");
        filterChainDefinitionMap.put("/**", "authc");
        shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);
        return shiroFilterFactoryBean;
    }


    @Bean
    public SecurityManager securityManager() {
        DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
        //设置realm.
        securityManager.setRealm(userRealm());
        // 自定义缓存实现 使用redis
        if (Constant.CACHE_TYPE_REDIS.equals(cacheType)) {
            securityManager.setCacheManager(cacheManager());
        } else {
            securityManager.setCacheManager(ehCacheManager());

        }
```

```
        securityManager.setSessionManager(sessionManager());
        return securityManager;
    }

    @Bean
    UserRealm userRealm() {
        UserRealm userRealm = new UserRealm();
        return userRealm;
    }

    /**
     * 开启shiro aop注解支持.
     * 使用代理方式;所以需要开启代码支持;
     *
     * @param securityManager
     * @return
     */
    @Bean
    public AuthorizationAttributeSourceAdvisor
authorizationAttributeSourceAdvisor(SecurityManager securityManager) {
        AuthorizationAttributeSourceAdvisor authorizationAttributeSourceAdvisor = new
AuthorizationAttributeSourceAdvisor();
        authorizationAttributeSourceAdvisor.setSecurityManager(securityManager);
        return authorizationAttributeSourceAdvisor;
    }

    /**
     * 配置shiro redisManager
     *
     * @return
     */
    @Bean
    public RedisManager redisManager() {
        RedisManager redisManager = new RedisManager();
        redisManager.setHost(host);
        redisManager.setPort(port);
        redisManager.setExpire(1800);// 配置缓存过期时间
        //redisManager.setTimeout(1800);
        redisManager.setPassword(password);
        return redisManager;
    }

    /**
     * cacheManager 缓存 redis实现
     * 使用的是shiro-redis开源插件
     *
     * @return
     */
    public RedisCacheManager cacheManager() {
        RedisCacheManager redisCacheManager = new RedisCacheManager();
        redisCacheManager.setRedisManager(redisManager());
        return redisCacheManager;

    }
```

```
    /**
     * RedisSessionDAO shiro sessionDao层的实现 通过redis
     * 使用的是shiro-redis开源插件
     */
    @Bean
    public RedisSessionDAO redisSessionDAO() {
        RedisSessionDAO redisSessionDAO = new RedisSessionDAO();
        redisSessionDAO.setRedisManager(redisManager());
        return redisSessionDAO;
    }

    @Bean
    public SessionDAO sessionDAO() {
        if (Constant.CACHE_TYPE_REDIS.equals(cacheType)) {
            return redisSessionDAO();
        } else {
            return new MemorySessionDAO();
        }
    }

    /**
     * shiro session的管理
     */
    @Bean
    public DefaultWebSessionManager sessionManager() {
        DefaultWebSessionManager sessionManager = new DefaultWebSessionManager();
        sessionManager.setGlobalSessionTimeout(tomcatTimeout * 1000);
        sessionManager.setSessionDAO(sessionDAO());
        Collection<SessionListener> listeners = new ArrayList<SessionListener>();
        listeners.add(new BDSessionListener());
        sessionManager.setSessionListeners(listeners);
        return sessionManager;
    }

    @Bean
    public EhCacheManager ehCacheManager() {
        EhCacheManager em = new EhCacheManager();
        em.setCacheManager(CacheManager.create());
        return em;
    }


}
```

这里补充一下ehcache和redis比较

ehcache直接在jvm虚拟机中缓存,速度快,效率高,但是缓存共享麻烦,集群分布式应用不方便

redis是通过socket访问缓存务,效率比ehcachedi,处理集群和分布式缓存方便,有成熟的方案

UserRealm

```
import java.util.HashMap;
import java.util.Map;
import java.util.Set;

import com.prostate.common.config.ApplicationContextRegister;
import com.prostate.system.domain.UserToken;
import org.apache.shiro.authc.AuthenticationException;
import org.apache.shiro.authc.AuthenticationInfo;
import org.apache.shiro.authc.AuthenticationToken;
import org.apache.shiro.authc.IncorrectCredentialsException;
import org.apache.shiro.authc.LockedAccountException;
import org.apache.shiro.authc.SimpleAuthenticationInfo;
import org.apache.shiro.authc.UnknownAccountException;
import org.apache.shiro.authz.AuthorizationInfo;
import org.apache.shiro.authz.SimpleAuthorizationInfo;
import org.apache.shiro.realm.AuthorizingRealm;
import org.apache.shiro.subject.PrincipalCollection;
import org.springframework.beans.factory.annotation.Autowired;

import com.prostate.common.utils.ShiroUtils;
import com.prostate.system.dao.UserDao;
import com.prostate.system.domain.UserDO;
import com.prostate.system.service.MenuService;

/**
 * 自定义realm 安全的数据库
 */
public class UserRealm extends AuthorizingRealm {
/*    @Autowired
    UserDao userMapper;
    @Autowired
    MenuService menuService;*/

    /**
     *
     * @param arg0
     * @return
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection arg0) {
        Long userId = ShiroUtils.getUserId();
        MenuService menuService = ApplicationContextRegister.getBean(MenuService.class);
        Set<String> perms = menuService.listPerms(userId);
        SimpleAuthorizationInfo info = new SimpleAuthorizationInfo();
        info.setStringPermissions(perms);
        return info;
    }

    @Override
```

```
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws
AuthenticationException {
        String username = (String) token.getPrincipal();
        Map<String, Object> map = new HashMap<>(16);
        map.put("username", username);
        String password = new String((char[]) token.getCredentials());

        UserDao userMapper = ApplicationContextRegister.getBean(UserDao.class);
        // 查询用户信息
        UserDO user = userMapper.list(map).get(0);

        // 账号不存在
        if (user == null) {
            throw new UnknownAccountException("账号或密码不正确");
        }

        // 密码错误
        if (!password.equals(user.getPassword())) {
            throw new IncorrectCredentialsException("账号或密码不正确");
        }

        // 账号锁定
        if (user.getStatus() == 0) {
            throw new LockedAccountException("账号已被锁定,请联系管理员");
        }
        SimpleAuthenticationInfo info = new SimpleAuthenticationInfo(user, password, getName());
        return info;
    }

}
```

ShiroUtils

```
package com.prostate.common.utils;

import com.prostate.system.domain.UserToken;
import org.apache.commons.beanutils.BeanUtils;
import org.apache.shiro.SecurityUtils;
import org.apache.shiro.session.Session;
import org.apache.shiro.session.mgt.eis.SessionDAO;
import org.apache.shiro.subject.Subject;

import com.prostate.system.domain.UserDO;
import org.springframework.beans.factory.annotation.Autowired;

import java.lang.reflect.InvocationTargetException;
import java.security.Principal;
import java.util.Collection;
import java.util.List;
```

```
public class ShiroUtils {
    @Autowired
    private static SessionDAO sessionDAO;

    public static Subject getSubjct() {
        return SecurityUtils.getSubject();
    }
    public static UserDO getUser() {
        Object object = getSubjct().getPrincipal();
        return (UserDO)object;
    }
    public static Long getUserId() {
        return getUser().getUserId();
    }
    public static void logout() {
        getSubjct().logout();
    }

    public static List<Principal> getPrinciples() {
        List<Principal> principals = null;
        Collection<Session> sessions = sessionDAO.getActiveSessions();
        return principals;
    }
}
```

Shiro 分析

①: Shiro常见的抛出异常

```
UnknownAccountException（账号不存在）
IncorrectCredentialsException(密码不存在)
DisabledAccountException（帐号被禁用）
LockedAccountException（帐号被锁定）
ExcessiveAttemptsException（登录失败次数过多）
ExpiredCredentialsException（凭证过期）等
```

②:Shiro标签

```
<shiro:authenticated> 登录之后
<shiro:notAuthenticated> 不在登录状态时
<shiro:guest> 用户在没有RememberMe时
<shiro:user> 用户在RememberMe时
<shiro:hasAnyRoles name="abc,123" > 在有abc或者123角色时
<shiro:hasRole name="abc"> 拥有角色abc
<shiro:lacksRole name="abc"> 没有角色abc
<shiro:hasPermission name="abc"> 拥有权限abc
<shiro:lacksPermission name="abc"> 没有权限abc
<shiro:principal> 显示用户登录名
```

③ Shiro过滤器链中的几种内置过滤器

```
--(1)认证过滤器
anon               匿名过滤器
authc              需要认证
authcBasic      表示httpBasic认证
user                表示必须存在用户,当登入操作时不做检查

--(2)授权过滤器
roles             /admins/user/**=authc,roles[admin] (必须认证过，并拥有admin 角色)
perms
port
rest
ssl

--(3) shiro 登录退出
shiro 内置的logout 过滤器（先配置logout，再在过滤器链中配置）
<bean id="logout" class="org.apache.shiro.web.filter.authc.LogoutFilter">
        <property name="redirectUrl" value="/loginform" />
</bean>
filterChainDefinitions 配置  /logout = logout

--(4) 关于登录过滤器的配置
通常将登录请求和使用的资源(js.css等)放开设置为anon,将其他的所有资源/** 设置为authc

--(5)关于Session 失效的问题
每次请求都会经过过滤器链的过滤，对于authc的资源，如果登录失效，自动返回到默认登录首页 配置的loginUrl
中；
或者可以自定义个session 拦截的过滤器放入shiro 的过滤器链
```

二 Spring Security是一个能够为Spring的企业应用系统提供声明式安全访问控制解决方案的安全框架.它提供一组Spring应用上下文中配置Bean,充分利用Spring Ioc,DI,AOP,减少了为企业系统安全控制编写大量重复代码,它是一个轻量级的安全框架,他确保基于Spring的应用程序提供身份验证和授权支持,并配备了流行的安全算法实现捆绑在一起