**实现Spring Boot、 Redis、 Shiro集群**

　为实现Web应用的分布式集群部署，要解决登录session的统一。本文利用shiro做权限控制，redis做session存储，结合spring boot快速配置实现session共享。

　　1、引入相关依赖

<dependency>  
    <groupId>org.springframework.data</groupId>  
    <artifactId>spring-data-redis</artifactId>  
    <version>1.8.4.RELEASE</version>  
</dependency>  
<dependency>  
    <groupId>redis.clients</groupId>  
    <artifactId>jedis</artifactId>  
    <version>2.9.0</version>  
</dependency>

<dependency>

<groupId>org.apache.shiro</groupId>

<artifactId>shiro-spring</artifactId>

<version>1.3.2</version>

</dependency>

　　2、Redis相关

　　2.1.redis配置

　　spring.redis.host=localhost #redis所在服务器ip

　　spring.redis.port=6379 #redis端口

spring.redis.password= #redis密码

spring.redis.database=0 #redis数据库id

　　2.2.redis缓存的对象必须序列化，通用序列化

import org.springframework.core.convert.converter.Converter;

import org.springframework.core.serializer.support.DeserializingConverter;

import org.springframework.core.serializer.support.SerializingConverter;

import org.springframework.data.redis.serializer.RedisSerializer;

import org.springframework.data.redis.serializer.SerializationException;

/\*\*

\* redis序列化对象

\*/

public class RedisObjectSerializer implements RedisSerializer<Object> {

private Converter<Object, byte[]> serializer = new SerializingConverter();

private Converter<byte[], Object> deserializer = new DeserializingConverter();

static final byte[] EMPTY\_ARRAY = new byte[0];

public Object deserialize(byte[] bytes) {

if (isEmpty(bytes)) {

return null;

}

try {

return deserializer.convert(bytes);

} catch (Exception ex) {

throw new SerializationException("Cannot deserialize", ex);

}

}

public byte[] serialize(Object object) {

if (object == null) {

return EMPTY\_ARRAY;

}

try {

return serializer.convert(object);

} catch (Exception ex) {

return EMPTY\_ARRAY;

}

}

private boolean isEmpty(byte[] data) {

return (data == null || data.length == 0);

}

}

　　2.3 RedisTemplate 配置

import org.springframework.cache.CacheManager;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.redis.cache.RedisCacheManager;

import org.springframework.data.redis.connection.RedisConnectionFactory;

import org.springframework.data.redis.core.RedisTemplate;

import org.springframework.data.redis.serializer.StringRedisSerializer;

/\*\*

\* redis 配置

\*/

@Configuration

public class RedisConfig {

@Bean

public CacheManager cacheManager(RedisTemplate<Object, Object> redisTemplate) {

RedisCacheManager cacheManager = new RedisCacheManager(redisTemplate);

cacheManager.setDefaultExpiration(1800);

return cacheManager;

}

@Bean

public RedisTemplate<Object, Object> redisTemplate(RedisConnectionFactory factory) {

RedisTemplate<Object, Object> template = new RedisTemplate<Object, Object>();

template.setConnectionFactory(factory);

template.setKeySerializer(new StringRedisSerializer());

template.setValueSerializer(new RedisObjectSerializer());

return template;

}

}

 　　3.Redis实现shiro的SessionDao存取session

import java.io.Serializable;

import java.util.concurrent.TimeUnit;

import javax.annotation.Resource;

import org.apache.shiro.session.Session;

import org.apache.shiro.session.mgt.eis.EnterpriseCacheSessionDAO;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.data.redis.core.RedisTemplate;

import org.springframework.stereotype.Component;

/\*\*

\* redis实现共享session

\*/

@Component

public class RedisSessionDAO extends EnterpriseCacheSessionDAO {

private static Logger logger = LoggerFactory.getLogger(RedisSessionDAO.class);

// session 在redis过期时间是30分钟30\*60

private static int expireTime = 1800;

private static String prefix = "weiyou-shiro-session:";

@Resource

private RedisTemplate<String, Object> redisTemplate;

// 创建session，保存到数据库

@Override

protected Serializable doCreate(Session session) {

Serializable sessionId = super.doCreate(session);

logger.debug("创建session:{}", session.getId());

redisTemplate.opsForValue().set(prefix + sessionId.toString(), session);

return sessionId;

}

// 获取session

@Override

protected Session doReadSession(Serializable sessionId) {

logger.debug("获取session:{}", sessionId);

// 先从缓存中获取session，如果没有再去数据库中获取

Session session = super.doReadSession(sessionId);

if (session == null) {

session = (Session) redisTemplate.opsForValue().get(prefix + sessionId.toString());

}

return session;

}

// 更新session的最后一次访问时间

@Override

protected void doUpdate(Session session) {

super.doUpdate(session);

logger.debug("获取session:{}", session.getId());

String key = prefix + session.getId().toString();

if (!redisTemplate.hasKey(key)) {

redisTemplate.opsForValue().set(key, session);

}

redisTemplate.expire(key, expireTime, TimeUnit.SECONDS);

}

// 删除session

@Override

protected void doDelete(Session session) {

logger.debug("删除session:{}", session.getId());

super.doDelete(session);

redisTemplate.delete(prefix + session.getId().toString());

}

}

　　4.实现cache共享

import java.util.ArrayList;

import java.util.Collection;

import java.util.List;

import java.util.Set;

import java.util.concurrent.TimeUnit;

import org.apache.shiro.cache.Cache;

import org.apache.shiro.cache.CacheException;

import org.springframework.data.redis.core.RedisTemplate;

@SuppressWarnings("unchecked")

public class ShiroCache<K, V> implements Cache<K, V> {

private static final String REDIS\_SHIRO\_CACHE = "weiyou-shiro-cache:";

private String cacheKey;

private RedisTemplate<K, V> redisTemplate;

private long globExpire = 30;

@SuppressWarnings("rawtypes")

public ShiroCache(String name, RedisTemplate client) {

this.cacheKey = REDIS\_SHIRO\_CACHE + name + ":";

this.redisTemplate = client;

}

@Override

public V get(K key) throws CacheException {

redisTemplate.boundValueOps(getCacheKey(key)).expire(globExpire, TimeUnit.MINUTES);

return redisTemplate.boundValueOps(getCacheKey(key)).get();

}

@Override

public V put(K key, V value) throws CacheException {

V old = get(key);

redisTemplate.boundValueOps(getCacheKey(key)).set(value);

return old;

}

@Override

public V remove(K key) throws CacheException {

V old = get(key);

redisTemplate.delete(getCacheKey(key));

return old;

}

@Override

public void clear() throws CacheException {

redisTemplate.delete(keys());

}

@Override

public int size() {

return keys().size();

}

@Override

public Set<K> keys() {

return redisTemplate.keys(getCacheKey("\*"));

}

@Override

public Collection<V> values() {

Set<K> set = keys();

List<V> list = new ArrayList<>();

for (K s : set) {

list.add(get(s));

}

return list;

}

private K getCacheKey(Object k) {

return (K) (this.cacheKey + k);

}

}

　　实现shiro 的CacheManager

import javax.annotation.Resource;

import org.apache.shiro.cache.Cache;

import org.apache.shiro.cache.CacheException;

import org.apache.shiro.cache.CacheManager;

import org.springframework.data.redis.core.RedisTemplate;

public class RedisCacheManager implements CacheManager {

@Resource

private RedisTemplate<String, Object> redisTemplate;

@Override

public <K, V> Cache<K, V> getCache(String name) throws CacheException {

return new ShiroCache<K, V>(name, redisTemplate);

}

public RedisTemplate<String, Object> getRedisTemplate() {

return redisTemplate;

}

public void setRedisTemplate(RedisTemplate<String, Object> redisTemplate) {

this.redisTemplate = redisTemplate;

}

}

 5.配置

import java.util.HashMap;

import java.util.Map;

import javax.annotation.Resource;

import org.apache.shiro.session.mgt.SessionManager;

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.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

/\*\*

\*

\* @author April.Chen

\*/

//@Configuration

public class ShiroConfig {

@Resource

private RedisSessionDAO sessionDAO;

@Bean

public UserRealm getUserRealm() {

return new UserRealm();

}

@Bean

public LifecycleBeanPostProcessor getLifecycleBeanPostProcessor() {

return new LifecycleBeanPostProcessor();

}

@Bean

public RedisCacheManager redisCacheManager() {

return new RedisCacheManager();

}

@Bean

public SessionManager sessionManager() {

DefaultWebSessionManager sessionManager = new DefaultWebSessionManager();

sessionManager.setSessionDAO(sessionDAO);

sessionManager.setGlobalSessionTimeout(1800);

sessionManager.setCacheManager(redisCacheManager());

return sessionManager;

}

@Bean

public DefaultWebSecurityManager securityManager() {

DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();

securityManager.setSessionManager(sessionManager());

securityManager.setCacheManager(redisCacheManager());

return securityManager;

}

@Bean

public AuthorizationAttributeSourceAdvisor getAuthorizationAttributeSourceAdvisor() {

AuthorizationAttributeSourceAdvisor aasa = new AuthorizationAttributeSourceAdvisor();

aasa.setSecurityManager(securityManager());

return new AuthorizationAttributeSourceAdvisor();

}

@Bean

public DefaultAdvisorAutoProxyCreator getDefaultAdvisorAutoProxyCreator() {

DefaultAdvisorAutoProxyCreator daap = new DefaultAdvisorAutoProxyCreator();

daap.setProxyTargetClass(true);

return daap;

}

@Bean

public ShiroFilterFactoryBean getShiroFilterFactoryBean() {

Map<String, String> filterChainDefinitionMap = new HashMap<>();

ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();

shiroFilterFactoryBean.setSecurityManager(securityManager());

shiroFilterFactoryBean.setLoginUrl("/login");

shiroFilterFactoryBean.setSuccessUrl("/index");

filterChainDefinitionMap.put("/sa/\*\*", "authc");

filterChainDefinitionMap.put("/\*\*", "anon");

shiroFilterFactoryBean.setFilterChainDefinitionMap(filterChainDefinitionMap);

return shiroFilterFactoryBean;

}

}