链滴

Shiro 与分布式 Session 与 Redis 的那些坑

作者：matthewhan
原文链接：https://ld246.com/article/1585556428661
来源网站：链滴

需要知道的点

Shiro的Session支持企业级的特性，例如分布式缓存。我们在Spring Data Redis + Shiro的方案中要注意下以下几点：

1. 无论Redis服务是单机还是集群模式，都需要注意Session对象的序列化与反序列化的问题；

2. Shiro的 **Session**：定义好的一个接口；**Simple Session**：一个它的简单实现，我们想要实现持化就需要对它进行维护；

3. **EnterpriseCacheSessionDAO**：Session对象的增删改查，可以对**Session对象**进行下一步的定化操作（个人理解），所以我们可以通过覆写它的方法来达到我们想要的持久化效果。以下4个方法对Session的持久化处理：

- doCreate
- doUpdate
- doReadSession
- doDelete

4. **SessionManager**：对**EnterpriseCacheSessionDAO**创建好的Session对象交给**SessionManag r**。它管理着Session的创建、操作以及清除等；**DefaultSessionManager**：具体实现，默认的web用Session管理器，主要是涉及到Session和Cookies，涉及到的行为：添加、删除SessionId到Cooki、读取Cookie获得SessionId；

5. **SessionId**：得到Session的关键

6. **securityManager**：这是Shiro框架的核心组件，可以把他看做是一个Shiro框架的全局管理组件用于调度各种Shiro框架的服务。我们需要将自定义的**sessionManager**交给它

Session持久化

上面写到如果想定制化我们的持久化效果，就必须覆写它的方法，所以我们需要新创建一个类 SessioRedisDao来继承 EnterpriseCacheSessionDAO类：

```
@Component
public class SessionRedisDao extends EnterpriseCacheSessionDAO {

    /**
     * 注入的是byteRedisTemplate，只用于byte[]类型的序列化存储在redis中
     */
    private final RedisTemplate<String, byte[]> redisTemplate;
    public SessionRedisDao(@Qualifier("byteRedisTemplate") RedisTemplate<String, byte[]> r
disTemplate) {
        this.redisTemplate = redisTemplate;
    }

    /**
     * 创建session，保存到数据库
     * @param session
     * @return
     */
    @Override
    protected Serializable doCreate(Session session) {
```

```
        Serializable sessionId = super.doCreate(session);
        System.out.println("===============【 " + sessionId + " 】 创建了session！ ====
===========");

        BoundValueOperations<String, byte[]> boundValueOperations = redisTemplate.boundV
lueOps(SHIRO_SESSION + sessionId.toString());
        boundValueOperations.set(sessionToByte(session), 240, TimeUnit.MINUTES);

        return sessionId;
    }

    /**
     * 获取session
     * @param sessionId
     * @return
     */
    @Override
    protected Session doReadSession(Serializable 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);
        //System.out.println("===============【 " + sessionId + " 】 获取了session！ ===
============");

        if(session == null){
            BoundValueOperations<String, byte[]> boundValueOperations = redisTemplate.boun
ValueOps(SHIRO_SESSION + sessionId.toString());
            byte[] bytes = (boundValueOperations.get());
            if(bytes != null && bytes.length > 0){
                session = byteToSession(bytes);
            }
        }
        return session;
    }

    /**
     * 更新session的最后一次访问时间
     * @param session
     */
    @Override
    protected void doUpdate(Session session) {
        super.doUpdate(session);
        System.out.println("===============【 " + session.getId() + " 】 更新了session！ =
==============");

        BoundValueOperations<String, byte[]> boundValueOperations = redisTemplate.boundV
lueOps(SHIRO_SESSION + session.getId().toString());
        boundValueOperations.set(sessionToByte(session), 240, TimeUnit.MINUTES);
    }

    /**
     * 删除session
     * @param session
     */
    @Override
```

```
    protected void doDelete(Session session) {
        System.out.println("===============【 " + session.getId() + " 】 删除了session！ =
==============");
        super.doDelete(session);
        redisTemplate.delete(SHIRO_SESSION + session.getId().toString());
    }

    /**
     * 把session对象转化为byte保存到redis中
     * @param session
     * @return
     */
    public byte[] sessionToByte(Session session){
        ByteArrayOutputStream bo = new ByteArrayOutputStream();
        byte[] bytes = null;
        try {
            ObjectOutputStream oo = new ObjectOutputStream(bo);
            oo.writeObject(session);
            bytes = bo.toByteArray();
        } catch (IOException e) {
            e.printStackTrace();
        }
        return bytes;
    }

    /**
     * 把byte还原为session
     * @param bytes
     * @return
     */
    public Session byteToSession(byte[] bytes){

        ByteArrayInputStream bi = new ByteArrayInputStream(bytes);
        ObjectInputStream in;
        SimpleSession session = null;
        try {
            in = new ObjectInputStream(bi);
            session = (SimpleSession) in.readObject();
        } catch (ClassNotFoundException | IOException e) {
            e.printStackTrace();
        }

        return session;
    }

}
```

好像看起来和网上的其他技术文章的实现差不多，但是还有差啦（迷之台湾腔？）

首先是关于 RedisTemplate客户端的注入使用：

```
private final RedisTemplate<String, byte[]> redisTemplate;
public SessionRedisDao(@Qualifier("byteRedisTemplate") RedisTemplate<String, byte[]> redi
Template) {
```

```
        this.redisTemplate = redisTemplate;
}
```

在这里看到key和value的类型 <String, byte[]>，不是常规的 <String, Object>，因为我在以下序化工具中以json字符串的形式存储在Redis：

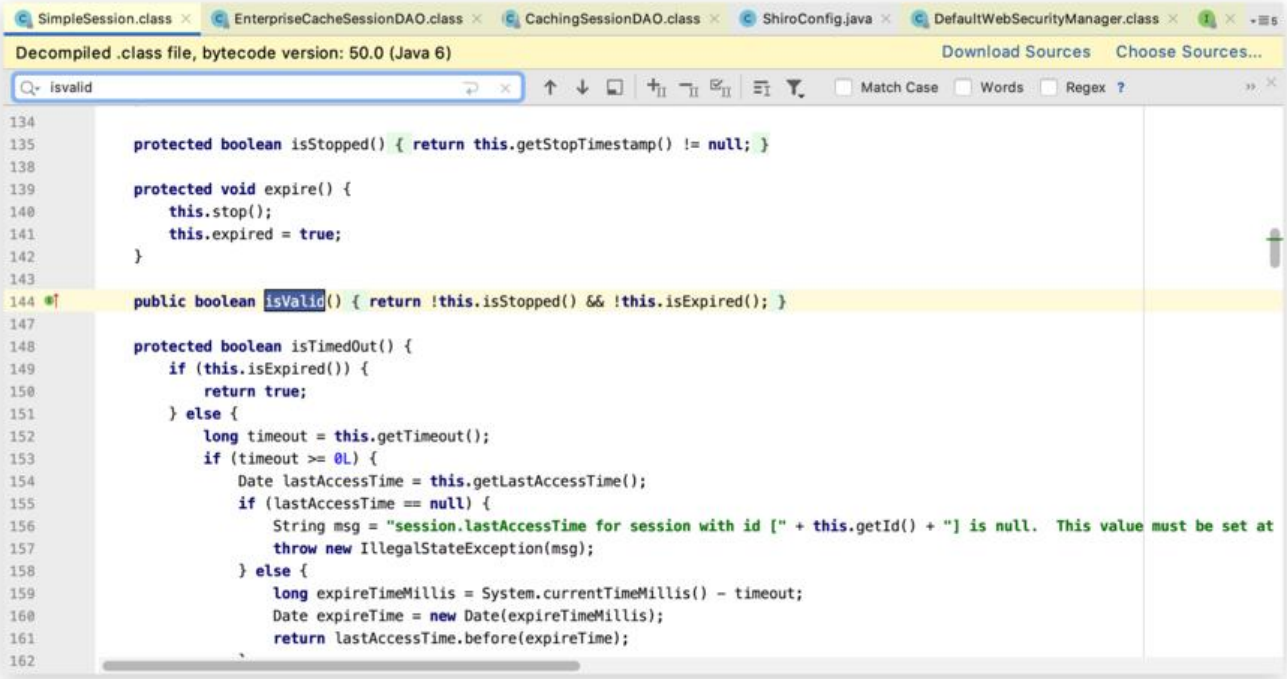
1. FastJsonRedisSerializer
2. GenericJackson2JsonRedisSerializer
3. Jackson2JsonRedisSerializer

发现在执行 doUpdate方法后，Redis当中会增加一些Simple Session没有字段，比如 "valid":true 等，所以在反序列化获取Session对象的过程中会抛出如下异常：

"Could not read JSON: Cannot construct instance of`org.apache.shiro.web.util.SavedRequest` no Creators, like default construct, exist): cannot deserialize from Object value (no delegate- o property-based Creator)□ at [Source: (byte[])"["org.apache.shiro.session.mgt.SimpleSession"

我突然就想到了《码出高效》里面，有说过POJO类不要使用isXxx作为变量的形式

当然这里也没发现存在 isXxx的成员变量，只看到了 isValid()方法，以及 isStoped()方法也没有对应的 toped 成员属性。可能是在反序列化的过程中，通过Redis里的键值对发现，**SimpleSession**并没有个 boolean 类型的 valid变量而导致错误。不知道是不是算Shiro的Simple Session一个Bug。



解决手段

目前个人找到的解决的方法是使用 byte[]字节流存储，且用默认的JDK序列化工具 JdkSerialization edisSerializer。

Redis配置序列化工具

因为可能在代码其他处已经使用了其他序列化工具操作Redis了，所以这里建议重新写一个Bean的方

专门用于Shiro安全框架的Session操作：

```
@Bean(name = "byteRedisTemplate")
public RedisTemplate<String, byte[]> byteRedisTemplate(RedisConnectionFactory redisConn
ctionFactory) {
    RedisTemplate<String, byte[]> redisTemplate = new RedisTemplate<>();
    redisTemplate.setConnectionFactory(redisConnectionFactory);

    JdkSerializationRedisSerializer jdkSerializationRedisSerializer = new JdkSerializationRedisSer
alizer();
    // 全局开启AutoType，不建议使用
    // ParserConfig.getGlobalInstance().setAutoTypeSupport(true);
    // 建议使用这种方式，小范围指定白名单
    ParserConfig.getGlobalInstance().addAccept("com.zrtg.");

    // 设置值（value）的序列化采用jdkSerializationRedisSerializer。
    redisTemplate.setValueSerializer(jdkSerializationRedisSerializer);
    redisTemplate.setHashValueSerializer(jdkSerializationRedisSerializer);
    // 设置键（key）的序列化采用StringRedisSerializer。
    redisTemplate.setKeySerializer(new StringRedisSerializer());
    redisTemplate.setHashKeySerializer(new StringRedisSerializer());

    redisTemplate.afterPropertiesSet();
    log.info("MatthewHan: [ byteRedisTemplate启动，鸡你实在是太美! ] ");
    return redisTemplate;
}
```

然后在需要注入的地方，加入 @Qualifier注解即可，像这样：@Qualifier("byteRedisTemplate") ReisTemplate<String, byte[]> redisTemplate。

并入管理

```
@Bean
public DefaultWebSessionManager sessionManager(SessionRedisDao sessionRedisDao) {
    DefaultWebSessionManager sessionManager = new DefaultWebSessionManager();

    sessionManager.setSessionIdCookie(remeberMeCookie());
    sessionManager.setGlobalSessionTimeout(14400000L);
    sessionManager.setDeleteInvalidSessions(true);
    // 将写好的缓存sessionDao注入
    sessionManager.setSessionDAO(sessionRedisDao);
    sessionManager.setSessionValidationSchedulerEnabled(true);


    return sessionManager;
}

/**
 * 注入 securityManager
 * 将写好的缓存sessionDao注入
 * @param sessionRedisDao
 * @param customRealmConfig
 * @return
 */
```

```
@Bean
public SecurityManager securityManager(SessionRedisDao sessionRedisDao, CustomRealmC
nfig customRealmConfig) {
    DefaultWebSecurityManager securityManager = new DefaultWebSecurityManager();
    // 设置realm
    securityManager.setRealm(customRealmConfig);
    // 注入Cookie记住我管理器
    securityManager.setRememberMeManager(null);
    securityManager.setSessionManager(sessionManager(sessionRedisDao));
    return securityManager;
}
```

这里注意rememberMe的Cookies管理，以及 sessionManager.setGlobalSessionTimeout(1440000L);和Redis设置的过期时间保持一致即可。