解决Shiro频繁访问Redis读取和更新session(十二)

该博客是接着上一篇博客: Shiro使用redis作为缓存(解决shiro频繁访问Redis) 请将两篇博客同时打开，方便查看。

前言

关于shiro 频繁访问redis,共分为两种,一种是频繁的去redis读取session， 一种是频繁的去更新redis 中的session，针对两种不同情况,分别写出解决方案。

频繁读取session 解决有两种方案：

第一种方案：本地缓存

上面的RedisSessionDAO类中依赖一个叫SessionInMemory的类,是shiro-redis作者为了解决一次请求频繁访问redis读取session的解决方案,基于本地cache,如果是在一秒内的请求,都会从本地cache中获取request。下面有更好的方案,但是这个代码我也没有删除。共存互不影响。

SessionInMemory.java

```
package com.springboot.test.shiro.config.shiro;

import org.apache.shiro.session.Session;

import java.util.Date;

/**
 * Use ThreadLocal as a temporary storage of Session, so that shiro wouldn't keep read redis several times while a request coming.
 */
public class SessionInMemory {
    private Session session;
    private Date createTime;

    public Session getSession() {
        return session;
    }

    public void setSession(Session session) {
        this.session = session;
    }

    public Date getCreateTime() {
        return createTime;
    }

    public void setCreateTime(Date createTime) {
        this.createTime = createTime;
    }
}
```

然后在RedisSessionDAO的doReadSession先去本地缓存查询是否有，有的话从本地获取,并对比时间,在规定时间内,则使用缓存中的session。详细代码,请参考上篇博客。

另一种方案：从request中获取

参考博客：http://www.hillfly.com/2017/182.html
关于频繁去Redis中读取Session有一个更好的解决方案，重写
DefaultWebSessionManager 的 `retrieveSession()` 方法。在 Web 下使用 shiro 时这个 sessionKey 是 WebSessionKey 类型的，这个类有个我们很熟悉的属性：servletRequest。小伙伴们应该都灵光一现了！直接把 session 对象怼进 request 里去！那么在单次请求周期内我们都可以从 request 中取 session 了，而且请求结束后 request 被销毁，作用域和生命周期的问题都需要我们考虑了。
显然我们要 Override 这个retrieveSession方法，为此我们需要使用自定义的 SessionManager，如下：

ShiroSessionManager.java

```
package com.springboot.test.shiro.config.shiro;

import org.apache.shiro.session.Session;
import org.apache.shiro.session.UnknownSessionException;
import org.apache.shiro.session.mgt.SessionKey;
import org.apache.shiro.web.session.mgt.DefaultWebSessionManager;
import org.apache.shiro.web.session.mgt.WebSessionKey;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import javax.servlet.ServletRequest;
import java.io.Serializable;

/**
```

```
 * @author: wangsaichao
 * @date: 2018/6/23
 * @description: 解决单次请求需要多次访问redis
 */
public class ShiroSessionManager extends DefaultWebSessionManager {

    private static Logger logger = LoggerFactory.getLogger(DefaultWebSessionManager.class);
    /**
     * 获取session
     * 优化单次请求需要多次访问redis的问题
     * @param sessionKey
     * @return
     * @throws UnknownSessionException
     */
    @Override
    protected Session retrieveSession(SessionKey sessionKey) throws UnknownSessionException {
        Serializable sessionId = getSessionId(sessionKey);

        ServletRequest request = null;
        if (sessionKey instanceof WebSessionKey) {
            request = ((WebSessionKey) sessionKey).getServletRequest();
        }

        if (request != null && null != sessionId) {
            Object sessionObj = request.getAttribute(sessionId.toString());
            if (sessionObj != null) {
                logger.debug("read session from request");
                return (Session) sessionObj;
            }
        }

        Session session = super.retrieveSession(sessionKey);
        if (request != null && null != sessionId) {
            request.setAttribute(sessionId.toString(), session);
        }
        return session;
    }
}
```

记得在ShiroConfig中配置SessionManager为自定义的ShiroSessionManager

解决频繁更新session解决方案

参考博客：https://blog.csdn.net/zsg88/article/details/74806374
由于SimpleSession lastAccessTime更改后也会调用SessionDao update方法，更新的字段只有LastAccessTime（最后一次访问时间），由于会话失效是由Redis数据过期实现的，这个字段意义不大，为了减少对Redis的访问，降低网络压力，实现自己的Session，在SimpleSession上套一层，增加一个标识位，如果Session除lastAccessTime意外其它字段修改，就标识一下，只有标识为修改的才可以通过doUpdate访问Redis，否则直接返回。

ShiroSession.java

```
package com.springboot.test.shiro.config.shiro;

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

import java.io.Serializable;
import java.util.Date;
import java.util.Map;

/**
 * @author: wangsaichao
 * @date: 2018/6/23
 * @description: 由于SimpleSession lastAccessTime更改后也会调用SessionDao update方法,
 * 增加标识位，如果只是更新lastAccessTime SessionDao update方法直接返回
 */
public class ShiroSession extends SimpleSession implements Serializable {
    // 除lastAccessTime以外其他字段发生改变时为true
    private boolean isChanged = false;

    public ShiroSession() {
        super();
        this.setChanged(true);
    }

    public ShiroSession(String host) {
        super(host);
        this.setChanged(true);
    }

```

```
29
30      @Override
31      public void setId(Serializable id) {
32          super.setId(id);
33          this.setChanged(true);
34      }
35
36      @Override
37      public void setStopTimestamp(Date stopTimestamp) {
38          super.setStopTimestamp(stopTimestamp);
39          this.setChanged(true);
40      }
41
42      @Override
43      public void setExpired(boolean expired) {
44          super.setExpired(expired);
45          this.setChanged(true);
46      }
47
48      @Override
49      public void setTimeout(long timeout) {
50          super.setTimeout(timeout);
51          this.setChanged(true);
52      }
53
54      @Override
55      public void setHost(String host) {
56          super.setHost(host);
57          this.setChanged(true);
58      }
59
60      @Override
61      public void setAttributes(Map<Object, Object> attributes) {
62          super.setAttributes(attributes);
63          this.setChanged(true);
64      }
65
66      @Override
67      public void setAttribute(Object key, Object value) {
68          super.setAttribute(key, value);
69          this.setChanged(true);
70      }
71
72      @Override
73      public Object removeAttribute(Object key) {
74          this.setChanged(true);
75          return super.removeAttribute(key);
76      }
77
78      /**
79       * 停止
80       */
81      @Override
82      public void stop() {
83          super.stop();
84          this.setChanged(true);
85      }
86
87      /**
88       * 设置过期
89       */
90      @Override
91      protected void expire() {
92          this.stop();
93          this.setExpired(true);
94      }
95
96      public boolean isChanged() {
97          return isChanged;
98      }
99
100     public void setChanged(boolean isChanged) {
101         this.isChanged = isChanged;
102     }
103
104     @Override
105     public boolean equals(Object obj) {
106         return super.equals(obj);
107     }
```

```
108
109     @Override
110     protected boolean onEquals(SimpleSession ss) {
111         return super.onEquals(ss);
112     }
113
114     @Override
115     public int hashCode() {
116         return super.hashCode();
117     }
118
119     @Override
120     public String toString() {
121         return super.toString();
122     }
123 }
```

ShiroSessionFactory.java

```
1  package com.springboot.test.shiro.config.shiro;
2
3  import org.apache.commons.lang3.StringUtils;
4  import org.apache.shiro.session.Session;
5  import org.apache.shiro.session.mgt.SessionContext;
6  import org.apache.shiro.session.mgt.SessionFactory;
7  import org.apache.shiro.web.session.mgt.DefaultWebSessionContext;
8  import org.slf4j.Logger;
9  import org.slf4j.LoggerFactory;
10
11 import javax.servlet.http.HttpServletRequest;
12
13 /**
14  * @author: wangsaichao
15  * @date: 2018/6/23
16  * @description:
17  */
18 public class ShiroSessionFactory implements SessionFactory {
19     private static final Logger logger = LoggerFactory.getLogger(ShiroSessionFactory.class);
20
21     @Override
22     public Session createSession(SessionContext initData) {
23         ShiroSession session = new ShiroSession();
24         HttpServletRequest request = (HttpServletRequest)initData.get(DefaultWebSessionContext.class.getName() + ".SERVLET_REQUEST");
25         session.setHost(getIpAddress(request));
26         return session;
27     }
28
29     public static String getIpAddress(HttpServletRequest request) {
30         String localIP = "127.0.0.1";
31         String ip = request.getHeader("x-forwarded-for");
32         if (StringUtils.isBlank(ip) || (ip.equalsIgnoreCase(localIP)) || "unknown".equalsIgnoreCase(ip)) {
33             ip = request.getHeader("Proxy-Client-IP");
34         }
35         if (StringUtils.isBlank(ip) || (ip.equalsIgnoreCase(localIP)) || "unknown".equalsIgnoreCase(ip)) {
36             ip = request.getHeader("WL-Proxy-Client-IP");
37         }
38         if (StringUtils.isBlank(ip) || (ip.equalsIgnoreCase(localIP)) || "unknown".equalsIgnoreCase(ip)) {
39             ip = request.getRemoteAddr();
40         }
41         return ip;
42     }
43 }
```

然后在ShiroConfig配置该Bean，并赋值给sessionManager

```
1  @Bean("sessionFactory")
2  public ShiroSessionFactory sessionFactory(){
3      ShiroSessionFactory sessionFactory = new ShiroSessionFactory();
4      return sessionFactory;
5  }
6
7  @Bean("sessionManager")
8  public SessionManager sessionManager() {
9      ShiroSessionManager sessionManager = new ShiroSessionManager();
10      ... 该出省略其他配置
11     sessionManager.setSessionFactory(sessionFactory());
12     return sessionManager;
13
```

```
14
    }
```

然后在RedisSessionDAO的update方法上判断如果只是更改session的lastAccessTime,则直接返回。

```
1  @Override
2  public void update(Session session) throws UnknownSessionException {
3      //如果会话过期/停止 没必要再更新了
4      try {
5          if (session instanceof ValidatingSession && !((ValidatingSession) session).isValid()) {
6              return;
7          }
8
9          if (session instanceof ShiroSession) {
10             // 如果没有主要字段(除lastAccessTime以外其他字段)发生改变
11             ShiroSession ss = (ShiroSession) session;
12             if (!ss.isChanged()) {
13                 return;
14             }
15             //如果没有返回 证明有调用 setAttribute往redis 放的时候永远设置为false
16             ss.setChanged(false);
17         }
18
19         this.saveSession(session);
20     } catch (Exception e) {
21         logger.warn("update Session is failed", e);
22     }
23 }
```

这里注意：在操作redis更新session时候, changed属性一定是false, 原博客中并没有说明,我在测试的时候,发现如果只是更改lastAccessTime也不会直接返回,因为从redis拿出来的是true 。所以,既然走到往redis更新session这一步，那一定有setAttributes等方法被调用.所以往redis放的时候设置为false。下次从redis获取session是false 只是更改lastAccessTime 那么 changed属性就是false,将不会操作redis。