springboot整合shiro-在线人数以及并发登录人数控制(七)

项目中有时候会遇到统计当前在线人数的需求，也有这种情况当A 用户在邯郸地区登录 ， 然后A用户在北京地区再登录 ， 要踢出邯郸登录的状态。如果用户在北京重新登录，那么又要踢出邯郸的用户，这样反复。
这样保证了一个帐号只能同时一个人使用。那么下面来讲解一下 Shiro 怎么实现在线人数统计 以及 并发人数控制这个功能。

并发 人数控制

参考开涛大神博客：http://jinnianshilongnian.iteye.com/blog/2039760
使用的技术其实是 shiro的自定义filter，在 springboot整合shiro -快速入门 中 我们已经了解到,在shiroConfig的ShiroFilterFactoryBean中使用的过滤规则，如： `anon` ， `authc` ， `user` 等本质上是通过调用各自对应的filter方式集成的，也就是说，它是遵循过滤器链规则的。

如何使用自定义filter实现并发人数的控制?

写一个KickoutSessionControlFilter类继承AccessControlFilter类

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

import java.io.Serializable;
import java.util.Deque;
import java.util.LinkedList;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

import com.springboot.test.shiro.modules.user.dao.entity.User;
import org.apache.shiro.cache.Cache;
import org.apache.shiro.cache.CacheManager;
import org.apache.shiro.session.Session;
import org.apache.shiro.session.mgt.DefaultSessionKey;
import org.apache.shiro.session.mgt.SessionManager;
import org.apache.shiro.subject.Subject;
import org.apache.shiro.web.filter.AccessControlFilter;
import org.apache.shiro.web.util.WebUtils;

/**
 * @author: WangSaiChao
 * @date: 2018/5/23
 * @description: shiro 自定义filter 实现 并发登录控制
 */
public class KickoutSessionControlFilter  extends AccessControlFilter{

    /** 踢出后到的地址 */
    private String kickoutUrl;

    /**  踢出之前登录的/之后登录的用户 默认踢出之前登录的用户 */
    private boolean kickoutAfter = false;

    /**  同一个帐号最大会话数 默认1 */
    private int maxSession = 1;
    private SessionManager sessionManager;
    private Cache<String, Deque<Serializable>> cache;

    public void setKickoutUrl(String kickoutUrl) {
        this.kickoutUrl = kickoutUrl;
    }

    public void setKickoutAfter(boolean kickoutAfter) {
        this.kickoutAfter = kickoutAfter;
    }

    public void setMaxSession(int maxSession) {
        this.maxSession = maxSession;
    }

    public void setSessionManager(SessionManager sessionManager) {
        this.sessionManager = sessionManager;
    }

    public void setCacheManager(CacheManager cacheManager) {
        this.cache = cacheManager.getCache("shiro-activeSessionCache");
    }
    /**
     * 是否允许访问，返回true表示允许
     */
    @Override
```

```
    protected boolean isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue) throws Exception {
        return false;
    }
    /**
     * 表示访问拒绝时是否自己处理，如果返回true表示自己不处理且继续拦截器链执行，返回false表示自己已经处理了（比如重定向到另一个页面）。
     */
    @Override
    protected boolean onAccessDenied(ServletRequest request, ServletResponse response) throws Exception {
        Subject subject = getSubject(request, response);
        if(!subject.isAuthenticated() && !subject.isRemembered()) {
            //如果没有登录，直接进行之后的流程
            return true;
        }

        Session session = subject.getSession();
        //这里获取的User是实体 因为我在 自定义ShiroRealm中的doGetAuthenticationInfo方法中
        //new SimpleAuthenticationInfo(user, password, getName()); 传的是 User实体 所以这里拿到的也是实体,如果传的是userName 这里拿到的就是user
        String username = ((User) subject.getPrincipal()).getUsername();
        Serializable sessionId = session.getId();

        // 初始化用户的队列放到缓存里
        Deque<Serializable> deque = cache.get(username);
        if(deque == null) {
            deque = new LinkedList<Serializable>();
            cache.put(username, deque);
        }

        //如果队列里没有此sessionId，且用户没有被踢出；放入队列
        if(!deque.contains(sessionId) && session.getAttribute("kickout") == null) {
            deque.push(sessionId);
        }

        //如果队列里的sessionId数超出最大会话数，开始踢人
        while(deque.size() > maxSession) {
            Serializable kickoutSessionId = null;
            if(kickoutAfter) { //如果踢出后者
                kickoutSessionId=deque.getFirst();
                kickoutSessionId = deque.removeFirst();
            } else { //否则踢出前者
                kickoutSessionId = deque.removeLast();
            }
            try {
                Session kickoutSession = sessionManager.getSession(new DefaultSessionKey(kickoutSessionId));
                if(kickoutSession != null) {
                    //设置会话的kickout属性表示踢出了
                    kickoutSession.setAttribute("kickout", true);
                }
            } catch (Exception e) {//ignore exception
                e.printStackTrace();
            }
        }

        //如果被踢出了，直接退出，重定向到踢出后的地址
        if (session.getAttribute("kickout") != null) {
            //会话被踢出了
            try {
                subject.logout();
            } catch (Exception e) {
            }
            WebUtils.issueRedirect(request, response, kickoutUrl);
            return false;
        }
        return true;
    }
}
```

注意： 我们首先看一下 isAccessAllowed() 方法，在这个方法中，如果返回 true，则表示“通过”，走到下一个过滤器。如果没有下一个过滤器的话，表示具有了访问某个资源的权限。如果返回 false，则会调用 onAccessDenied 方法，去实现相应的当过滤不通过的时候执行的操作，例如检查用户是否已经登陆过,如果登陆过,根据自定义规则选择踢出前一个用户 还是 后一个用户。
onAccessDenied方法 返回 true 表示 自己处理完成,然后继续拦截器链执行。
只有当两者都返回false时,才会终止后面的filter执行。

在shiroConfig中配置该Bean

```
/**
 * 并发登录控制
 * @return
```

```
 */
@Bean
public KickoutSessionControlFilter kickoutSessionControlFilter(){
    KickoutSessionControlFilter kickoutSessionControlFilter = new KickoutSessionControlFilter();
    //用于根据会话ID，获取会话进行踢出操作的；
    kickoutSessionControlFilter.setSessionManager(sessionManager());
    //使用cacheManager获取相应的cache来缓存用户登录的会话；用于保存用户—会话之间的关系的；
    kickoutSessionControlFilter.setCacheManager(ehCacheManager());
    //是否踢出后来登录的，默认是false；即后者登录的用户踢出前者登录的用户；
    kickoutSessionControlFilter.setKickoutAfter(false);
    //同一个用户最大的会话数，默认1；比如2的意思是同一个用户允许最多同时两个人登录；
    kickoutSessionControlFilter.setMaxSession(1);
    //被踢出后重定向到的地址；
    kickoutSessionControlFilter.setKickoutUrl("/login?kickout=1");
    return kickoutSessionControlFilter;
}
```

修改shiroConfig中shirFilter中配置KickoutSessionControlFilter 并修改过滤规则

```
/**
 * ShiroFilterFactoryBean 处理拦截资源文件问题。
 * 注意：初始化ShiroFilterFactoryBean的时候需要注入：SecurityManager
 * Web应用中,Shiro可控制的Web请求必须经过Shiro主过滤器的拦截
 * @param securityManager
 * @return
 */
@Bean(name = "shirFilter")
public ShiroFilterFactoryBean shiroFilter(@Qualifier("securityManager") SecurityManager securityManager) {

    ShiroFilterFactoryBean shiroFilterFactoryBean = new ShiroFilterFactoryBean();

    ......

    //自定义拦截器限制并发人数,参考博客
    LinkedHashMap<String, Filter> filtersMap = new LinkedHashMap<>();
    //限制同一帐号同时在线的个数
    filtersMap.put("kickout", kickoutSessionControlFilter());
    shiroFilterFactoryBean.setFilters(filtersMap);

    // 配置访问权限 必须是LinkedHashMap，因为它必须保证有序
    // 过滤链定义，从上向下顺序执行，一般将 /**放在最为下边 --> : 这是一个坑，一不小心代码就不好使了
    LinkedHashMap<String, String> filterChainDefinitionMap = new LinkedHashMap<>();
    //配置不登录可以访问的资源，anon 表示资源都可以匿名访问
    //配置记住我或认证通过可以访问的地址
    filterChainDefinitionMap.put("/login", "kickout,anon");

    ......

    //其他资源都需要认证  authc 表示需要认证才能进行访问 user表示配置记住我或认证通过可以访问的地址
    filterChainDefinitionMap.put("/**", "kickout,user");

    return shiroFilterFactoryBean;
}
```

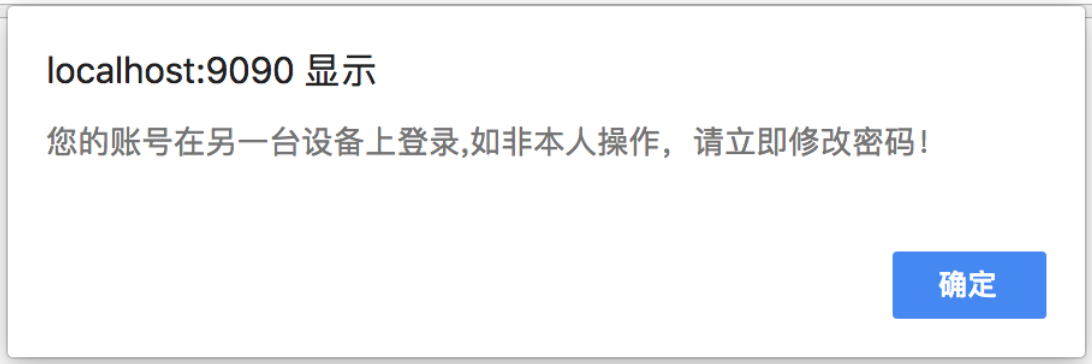
解释： `filterChainDefinitionMap.put("/**", "kickout,user");` 表示 访问/**下的资源 首先要通过 kickout 后面的filter，然后再通过user后面对应的filter才可以访问。

login.html添加踢出登录的信息提示

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:th="http://www.thymeleaf.org"
      xmlns:sec="http://www.thymeleaf.org/thymeleaf-extras-springsecurity3"
      xmlns:shiro="http://www.pollix.at/thymeleaf/shiro">
<head>
    <meta charset="UTF-8" />
    <title>Insert title here</title>
</head>
<body>
<h1>欢迎登录</h1>
<h1 th:if="${msg != null }" th:text="${msg}" style="color: red"></h1>
<form action="/login" method="post">
    用户名：<input type="text" name="username"/><br/>
    密码：<input type="password" name="password"/><br/>
    <input type="checkbox" name="rememberMe" />记住我<br/>
    <input type="submit" value="提交"/>
</form>
</body>
<script type="text/javascript" th:src="@{/js/jquery.js}"></script>
```

```
<script type="text/javascript">
    function kickout(){
        var href=location.href;
        if(href.indexOf("kickout")>0){
            alert("您的账号在另一台设备上登录,如非本人操作，请立即修改密码！");
        }
    }
    window.onload=kickout();
</script>
</html>
```

测试结果：



统计在线人数

springboot整合shiro-session管理 博客中，我们有配置过一个监听类 ,在该类中有统计session创建个数,我们也就用session的个数来统计在线的人数，但是这个统计人数是不准确的，存在这样一种情况，用户登录之后，强制退出浏览器,再次打开浏览器重新登录,在线人数一直在增加。暂时也没有想到特别好的方案,有的话留言共同学习。

在LoginController中注入ShiroSessionListener，然后在 index方法中 获取session 自增数量

```
model.addAttribute("count",shiroSessionListener.getSessionCount());
```