springboot整合shiro-ehcache缓存(五)

快速入门篇中,我们最后有两个问题,其中关于无权限页面的问题已经解决了,还有一个问题就是每次访问服务,都会去查询数据库,真实的情况是权限并不会经常出现变化,所以最好的办法就是做缓存处理,如果只是公司后台运营管理系统,可能只启动一个单节点就够了,这个时候我们就可以用到ehcache缓存,如果是分布式多节点部署,最好使用redis缓存,关于使用redis缓存在下篇中讲

shiro 使用ehcache缓存

pom添加依赖

```
<dependency>
    <groupId>org.apache.shiro</groupId>
    <artifactId>shiro-ehcache</artifactId>
    <version>1.4.0</version>
</dependency>
```

在ShiroConfig中添加缓存管理器

```
/**
 * shiro缓存管理器;
 * 需要添加到securityManager中
 * @return
 */
@Bean
public EhCacheManager ehCacheManager(){
    EhCacheManager cacheManager = new EhCacheManager();
    cacheManager.setCacheManagerConfigFile("classpath:config/ehcache-shiro.xml");
    return cacheManager;
}
```

将缓存管理器添加到SecurityManager中

```
/**
 * 配置核心安全事务管理器
 * @param shiroRealm
 * @return
 */
@Bean(name="securityManager")
public SecurityManager securityManager() {
    DefaultWebSecurityManager securityManager =  new DefaultWebSecurityManager();
    //设置自定义realm.
    securityManager.setRealm(shiroRealm());
    //配置记住我 参考博客:
    securityManager.setRememberMeManager(rememberMeManager());

    //配置 ehcache缓存管理器 参考博客:
    securityManager.setCacheManager(ehCacheManager());

    //配置自定义session管理，使用redis 参考博客:
    //securityManager.setSessionManager(sessionManager());

    return securityManager;
}
```

修改shiroRealm开启缓存

```
/**
 *  身份认证realm; (这个需要自己写，账号密码校验；权限等)
 * @return
 */
@Bean
public ShiroRealm shiroRealm(){
    ShiroRealm shiroRealm = new ShiroRealm();
    shiroRealm.setCachingEnabled(true);
    //启用身份验证缓存，即缓存AuthenticationInfo信息，默认false
    shiroRealm.setAuthenticationCachingEnabled(true);
    //缓存AuthenticationInfo信息的缓存名称 在ehcache-shiro.xml中有对应缓存的配置
    shiroRealm.setAuthenticationCacheName("authenticationCache");
    //启用授权缓存，即缓存AuthorizationInfo信息，默认false
    shiroRealm.setAuthorizationCachingEnabled(true);
    //缓存AuthorizationInfo信息的缓存名称  在ehcache-shiro.xml中有对应缓存的配置
    shiroRealm.setAuthorizationCacheName("authorizationCache");

```

```
    return shiroRealm;
}
```

在config下添加ehcache-shiro.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache name="es">

    <!--
        缓存对象存放路径
        java.io.tmpdir: 默认的临时文件存放路径。
        user.home: 用户的主目录。
        user.dir: 用户的当前工作目录，即当前程序所对应的工作路径。
        其它通过命令行指定的系统属性，如“java -DdiskStore.path=D:\\abc ……”。
    -->
    <diskStore path="java.io.tmpdir"/>

    <!--

       name:缓存名称。
       maxElementsOnDisk: 硬盘最大缓存个数。0表示不限制
       maxEntriesLocalHeap: 指定允许在内存中存放元素的最大数量，0表示不限制。
       maxBytesLocalDisk: 指定当前缓存能够使用的硬盘的最大字节数，其值可以是数字加单位，单位可以是K、M或者G，不区分大小写，
                          如: 30G。当在CacheManager级别指定了该属性后，Cache级别也可以用百分比来表示，
                          如: 60%，表示最多使用CacheManager级别指定硬盘容量的60%。该属性也可以在运行期指定。当指定了该属性后会隐式的使当前Cache的overf
       maxEntriesInCache: 指定缓存中允许存放元素的最大数量。这个属性也可以在运行期动态修改。但是这个属性只对Terracotta分布式缓存有用。
       maxBytesLocalHeap: 指定当前缓存能够使用的堆内存的最大字节数，其值的设置规则跟maxBytesLocalDisk是一样的。
       maxBytesLocalOffHeap: 指定当前Cache允许使用的非堆内存的最大字节数。当指定了该属性后，会使当前Cache的overflowToOffHeap的值变为true，
                             如果我们需要关闭overflowToOffHeap，那么我们需要显示的指定overflowToOffHeap的值为false。
       overflowToDisk:boolean类型，默认为false。当内存里面的缓存已经达到预设的上限时是否允许将按驱除策略驱除的元素保存在硬盘上，默认是LRU（最近最少使用）。
                      当指定为false的时候表示缓存信息不会保存到磁盘上，只会保存在内存中。
                      该属性现在已经废弃，推荐使用cache元素的子元素persistence来代替，如: <persistence strategy=”localTempSwap”/>。
       diskSpoolBufferSizeMB: 当往磁盘上写入缓存信息时缓冲区的大小，单位是MB，默认是30。
       overflowToOffHeap: boolean类型，默认为false。表示是否允许Cache使用非堆内存进行存储，非堆内存是不受Java GC影响的。该属性只对企业版Ehcache有用。
       copyOnRead: 当指定该属性为true时，我们在从Cache中读数据时取到的是Cache中对应元素的一个copy副本，而不是对应的一个引用。默认为false。
       copyOnWrite: 当指定该属性为true时，我们在往Cache中写入数据时用的是原对象的一个copy副本，而不是对应的一个引用。默认为false。
       timeToIdleSeconds: 单位是秒，表示一个元素所允许闲置的最大时间，也就是说一个元素在不被请求的情况下允许在缓存中待的最大时间。默认是0，表示不限制。
       timeToLiveSeconds: 单位是秒，表示无论一个元素闲置与否，其允许在Cache中存在的最大时间。默认是0，表示不限制。
       eternal: boolean类型，表示是否永恒，默认为false。如果设为true，将忽略timeToIdleSeconds和timeToLiveSeconds，Cache内的元素永远都不会过期，也就
       diskExpiryThreadIntervalSeconds : 单位是秒，表示多久检查元素是否过期的线程多久运行一次，默认是120秒。
       clearOnFlush: boolean类型。表示在调用Cache的flush方法时是否要清空MemoryStore。默认为true。
       diskPersistent: 是否缓存虚拟机重启期数据 Whether the disk store persists between restarts of the Virtual Machine. The default value
       maxElementsInMemory:缓存最大数目
       memoryStoreEvictionPolicy: 当达到maxElementsInMemory限制时，Ehcache将会根据指定的策略去清理内存。默认策略是LRU（最近最少使用）。你可以设置为F
            memoryStoreEvictionPolicy:
               Ehcache的三种清空策略;
               FIFO, first in first out, 这个是大家最熟的，先进先出。
               LFU,  Less Frequently Used, 就是上面例子中使用的策略，直白一点就是讲一直以来最少被使用的。如上面所讲，缓存的元素有一个hit属性，hit值最小的
               LRU, Least Recently Used, 最近最少使用的，缓存的元素有一个时间戳，当缓存容量满了，而又需要腾出地方来缓存新的元素的时候，那么现有缓存元素中
    -->
    <defaultCache
            maxElementsInMemory="10000"
            eternal="false"
            timeToIdleSeconds="0"
            timeToLiveSeconds="0"
            overflowToDisk="false"
            diskPersistent="false"
            diskExpiryThreadIntervalSeconds="120"
    />

    <!-- 授权缓存 -->
    <cache name="authorizationCache"
           maxEntriesLocalHeap="2000"
           eternal="false"
           timeToIdleSeconds="0"
           timeToLiveSeconds="0"
           overflowToDisk="false"
           statistics="true">
    </cache>

    <!-- 认证缓存 -->
    <cache name="authenticationCache"
           maxEntriesLocalHeap="2000"
           eternal="false"
           timeToIdleSeconds="0"
           timeToLiveSeconds="0"
           overflowToDisk="false"
           statistics="true">
```

```
    </cache>

</ehcache>
```

在ShiroRealm的doGetAuthorizationInfo方法中添加日志，或者直接看控制台sql打印也行。
启动项目,使用admin登录,访问http://localhost:9090/userInfo/view 第一次访问,查看控制台，有sql查询,再次访问将不再打印查询数据库的日志，证明缓存生效了。上面的缓存配置时间配置为永久,请根据需求自己更改值来进行测试。
关于在ShiroRealm中对 用户信息 角色信息 权限信息的查询,如果需要添加缓存 请自行处理。

如果用户的权限发生改变怎么办?

上面已经启用了缓存,第一次请求走数据库查询,后续请求将直接查询ehcache缓存，假如这个时候在权限控制台分配了某个权限给某个角色,那么拥有这个角色的所有用户在下次请求之前都需要从数据库查询最新的权限信息。下面开始进行在权限发生改变时,该如何做:

在ShiroRealm中添加以下方法(清理缓存)

```
/**
 * 重写方法,清除当前用户的的 授权缓存
 * @param principals
 */
@Override
public void clearCachedAuthorizationInfo(PrincipalCollection principals) {
    super.clearCachedAuthorizationInfo(principals);
}

/**
 * 重写方法，清除当前用户的 认证缓存
 * @param principals
 */
@Override
public void clearCachedAuthenticationInfo(PrincipalCollection principals) {
    super.clearCachedAuthenticationInfo(principals);
}

@Override
public void clearCache(PrincipalCollection principals) {
    super.clearCache(principals);
}

/**
 * 自定义方法：清除所有 授权缓存
 */
public void clearAllCachedAuthorizationInfo() {
    getAuthorizationCache().clear();
}

/**
 * 自定义方法：清除所有 认证缓存
 */
public void clearAllCachedAuthenticationInfo() {
    getAuthenticationCache().clear();
}

/**
 * 自定义方法：清除所有的  认证缓存  和 授权缓存
 */
public void clearAllCache() {
    clearAllCachedAuthenticationInfo();
    clearAllCachedAuthorizationInfo();
}
```

在shiroConfig中添加以下bean

```
/**
 * 让某个实例的某个方法的返回值注入为Bean的实例
 * Spring静态注入
 * @return
 */
@Bean
public MethodInvokingFactoryBean getMethodInvokingFactoryBean(){
    MethodInvokingFactoryBean factoryBean = new MethodInvokingFactoryBean();
    factoryBean.setStaticMethod("org.apache.shiro.SecurityUtils.setSecurityManager");
    factoryBean.setArguments(new Object[]{securityManager()});
    return factoryBean;
}
```

在UserController中添加下面两个方法

```
/**
 * 给admin用户添加 userInfo:del 权限
 * @param model
 * @return
 */
@RequestMapping(value = "/addPermission",method = RequestMethod.GET)
@ResponseBody
public String addPermission(Model model) {

    //在sys_role_permission 表中  将 删除的权限 关联到admin用户所在的角色
    roleMapper.addPermission(1,3);

    //添加成功之后 清除缓存
    DefaultWebSecurityManager securityManager = (DefaultWebSecurityManager)SecurityUtils.getSecurityManager();
    ShiroRealm shiroRealm = (ShiroRealm) securityManager.getRealms().iterator().next();
    //清除权限 相关的缓存
    shiroRealm.clearAllCache();
    return "给admin用户添加 userInfo:del 权限成功";

}

/**
 * 删除admin用户 userInfo:del 权限
 * @param model
 * @return
 */
@RequestMapping(value = "/delPermission",method = RequestMethod.GET)
@ResponseBody
public String delPermission(Model model) {

    //在sys_role_permission 表中  将 删除的权限 关联到admin用户所在的角色
    roleMapper.delPermission(1,3);
    //添加成功之后 清除缓存
    DefaultWebSecurityManager securityManager = (DefaultWebSecurityManager)SecurityUtils.getSecurityManager();
    ShiroRealm shiroRealm = (ShiroRealm) securityManager.getRealms().iterator().next();
    //清除权限 相关的缓存
    shiroRealm.clearAllCache();

    return "删除admin用户userInfo:del 权限成功";

}
```

注意：在添加权限 或者 删除权限之后 都有调用shiroRealm.clearAllCache();来清除所有的缓存。

测试步骤：

1.两个浏览器： 一个谷歌浏览器登录 admin账户，另一个360浏览器登录 test账户，每个账户登录跳转到idnex页面之后 ,刷新两下，发现之后无论怎么刷新都不再打印查询数据库的sql

2.在谷歌浏览器地址调用添加权限的方法http://localhost:9090/userInfo/addPermission 显示添加完成,然后再次访问http://localhost:9090/index 页面上已经显示刚添加的权限，查看日志发现走数据库查询了最新的权限信息

3.这个时候在360浏览器刷新http://localhost:9090/index 查看控制台日志发现test的用户也有走数据库查询权限信息