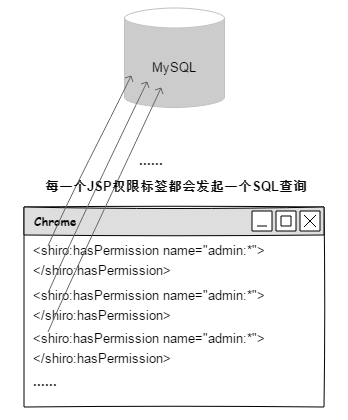


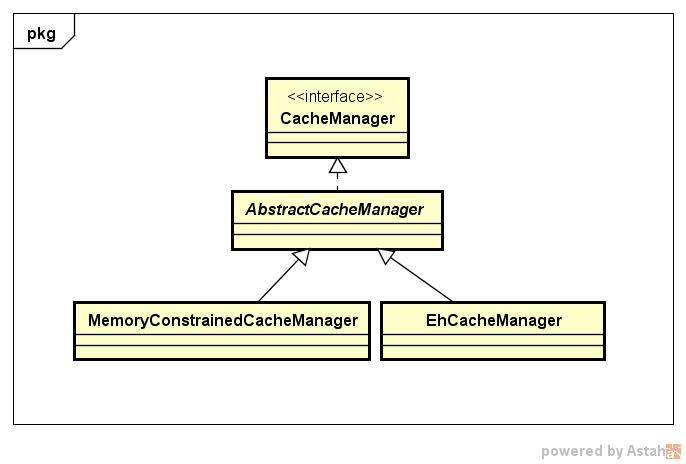
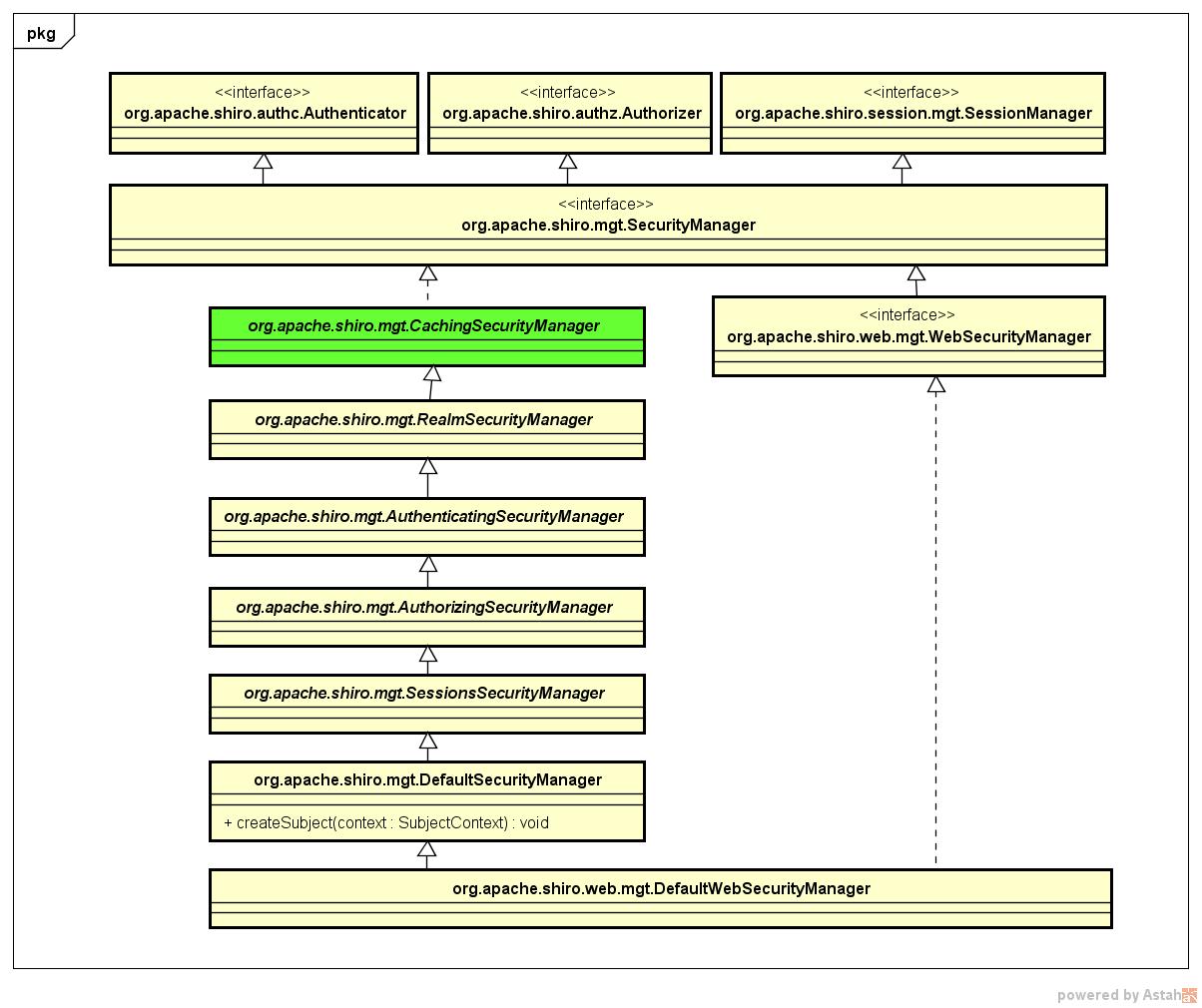
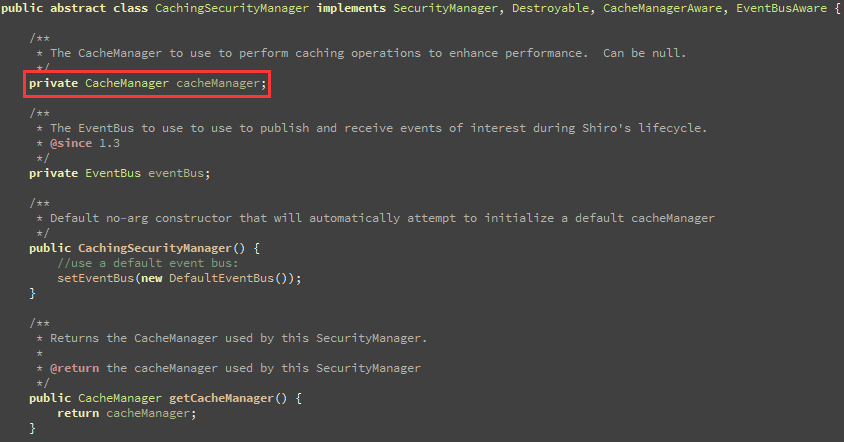
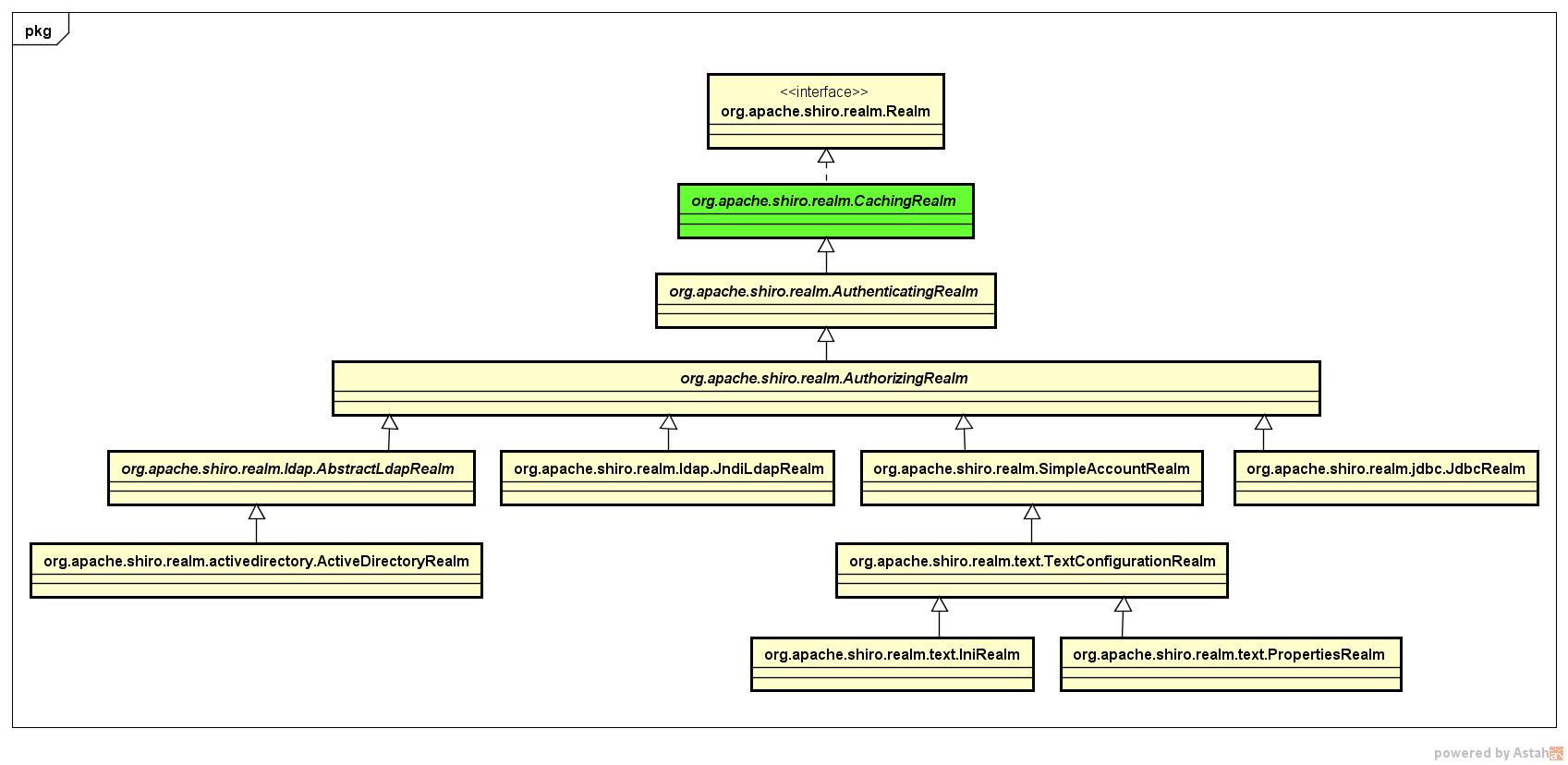
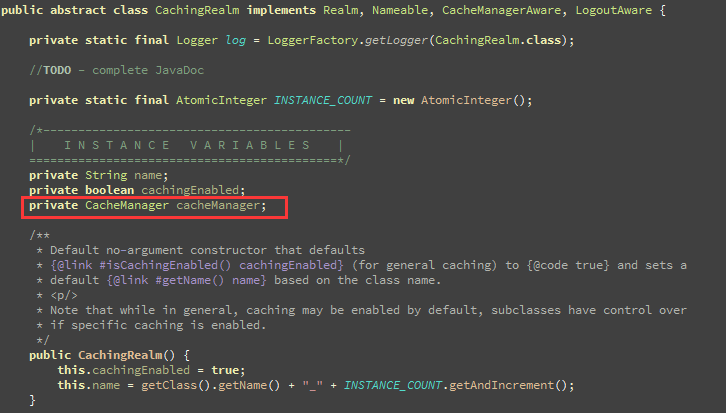
**一、概述**

当权限信息存放在数据库中时，对于每次前端的访问请求都需要进行一次数据库查询。特别是在大量使用shiro的jsp标签的场景下，对应前端的一个页面访问请求会同时出现很多的权限查询操作，这对于权限信息变化不是很频繁的场景，每次前端页面访问都进行大量的权限数据库查询是非常不经济的。因此，非常有必要对权限数据使用缓存方案。  
  
关于shiro权限数据的缓存方式，可以分为2类：

其一，**将权限数据缓存到集中式存储中间件中**，比如redis或者memcached；

其二，**将权限数据缓存到本地**。使用集中式缓存方案，页面的每次访问都会向缓存发起一次网络请求，如果大量使用了shiro的jsp标签，那么对应一个页面访问将会出现N个到集中缓存的网络请求，会给集中缓存组件带来一定的瞬时请求压力。另外，每个标签都需要经过一个网络查询，其实效率并不高。而采用本地缓存方式均不存在这些问题。所以，针对shiro的缓存方案，需要根据实际的使用场景进行权衡。如果在项目中并未使用shiro的jsp标签库，那么使用集中式的缓存方案也未尝不妥；但是，**如果大量使用shiro的jsp标签库，那么采用本地缓存才是最佳选择**。

**二、如何在shiro中使用缓存**

根据Shiro官方的说法，虽然缓存在权限框架中非常重要，但是如果实现一套完整的缓存机制会使得shiro偏离了核心的功能（认证和授权）。因此，Shiro只提供了一个可以支持具体缓存实现（如：Hazelcast,Ehcache,OSCache,Terracotta,Coherence,GigaSpaces,JBossCache等）的抽象API接口，这样就允许Shiro用户根据自己的需求灵活地选择具体的CacheManager。当然，其实Shiro也自带了一个本地内存CacheManager：org.apache.shiro.cache.MemoryConstrainedCacheManager。  
  
其实，从Shiro缓存组件类图可以看到，Shiro提供的缓存抽象API接口正是：org.apache.shiro.cache.CacheManager。  
那么，我们应该如何配置和使用CacheManager呢？如下我们以使用Shiro提供的MemoryConstrainedCacheManager组件为例进行说明。  
我们知道，SecurityManager是Shiro的核心控制器，我们来看一下其类图：  
  
org.apache.shiro.mgt.CachingSecurityManager是Shiro中SecurityManager接口的基础抽象类，我们来看一下其源码结构：  
  
从图中我们看到，在CachingSecurityManager中存在一个CacheManager类型的成员变量。  
另外，接口org.apache.shiro.realm.Realm定义了权限数据的存储方式，我们看一下其类图：  
  
显然，org.apache.shiro.realm.CachingRealm是Shiro中Realm接口的基础实现类，我们同样来看一下其源码结构：  
  
同样，在CachingRealm也存在一个CacheManager类型的成员变量。  
从以上分析我们知道：Shiro支持在2个地方定义缓存管理器，既可以在SecurityManager中定义，也可以在Realm中定义，任选其一即可。  
通常我们都会自定义Realm实现，例如将权限数据存放在数据库中，那么在Realm实现中定义缓存管理器再合适不过了。  
举个例子，我们扩展了org.apache.shiro.realm.jdbc.JdbcRealm，在其中定义一个缓存组件。

<!--Define the Shiro Realm implementation you want to use to connect to your back-end--><!--security datasource:--><bean id="myRealm"class="org.chench.test.shiro.spring.dao.ShiroCacheJdbcRealm">

<property name="dataSource"ref="dataSource"/>

<property name="permissionsLookupEnabled"value="true"/>

<property name="cacheManager"ref="cacheManager"/></bean>

<bean id="cacheManager"class="org.apache.shiro.cache.MemoryConstrainedCacheManager"/>

当然，同样可以在SecurityManager中定义缓存组件：

<bean id="securityManager"class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">

<!--Single realm app.If you have multiple realms,use the'realms'property instead.-->

<property name="realm"ref="myRealm"/>

<property name="cacheManager"ref="cacheManager"/></bean><bean id="cacheManager"class="org.apache.shiro.cache.MemoryConstrainedCacheManager"/>

那么，我们不禁要问了：  
第一：为什么Shiro要设计成既可以在Realm，也可以在SecurityManager中设置缓存管理器呢？  
第二：分别在Realm和SecurityManager定义的缓存管理器，他们有什么区别或联系吗？  
下面，我们追踪一下org.apache.shiro.mgt.RealmSecurityManage的源码实现：

protected void applyCacheManagerToRealms(){

CacheManager cacheManager=getCacheManager();

Collection<Realm>realms=getRealms();

if(cacheManager!=null&&realms!=null&&!realms.isEmpty()){

for(Realm realm:realms){

if(realm instanceof CacheManagerAware){

((CacheManagerAware)realm).setCacheManager(cacheManager);

}

}

}

}

这下终于真相大白了吧！其实在SecurityManager中设置的CacheManager组中都会给Realm使用，即：真正使用CacheManager的组件是Realm。

**三、缓存方案**

1.集中式缓存

我们在前面分析了，使用集中式缓存方案只适用于那些没有使用shiro的jsp标签的场景，比如：前后端完全分离的项目。目前比较流行的集中式缓存组件有：Redis，Memcache等，我们可以借助于这样的集中式缓存实现shiro的缓存方案。  
虽然使用了集中式缓存组件，但是不必要直接把权限数据本身存放到集中式缓存中，而是通过在集中式缓存中存放缓存标志即可。这样可以避免直接从集中式缓存中取权限数据，当权限数据比较大时，大量权限数据查询所占用的带宽也是比较可观的。

基于Redis的集中式缓存方案：https://github.com/alexxiyang/shiro-redis

基于Memcached的集中式缓存方案：https://github.com/mythfish/shiro-memcached

基于Ehcache集群模式的存放方案：http://www.ehcache.org/

2.本地缓存

本地缓存的实现有几种方式：（1）直接存放到JVM堆内存（2）使用NIO存放在堆外内存，自定义实现或者借助于第三方缓存组件。  
不论是采用集中式缓存还是使用本地缓存，shiro的权限数据本身都是直接存放在本地的，不同的是缓存标志的存放位置。采用本地缓存方案是，我们将缓存标志也存放在本地，这样就避免了查询缓存标志的网络请求，能更进一步提升缓存效率。

**四、缓存更新**