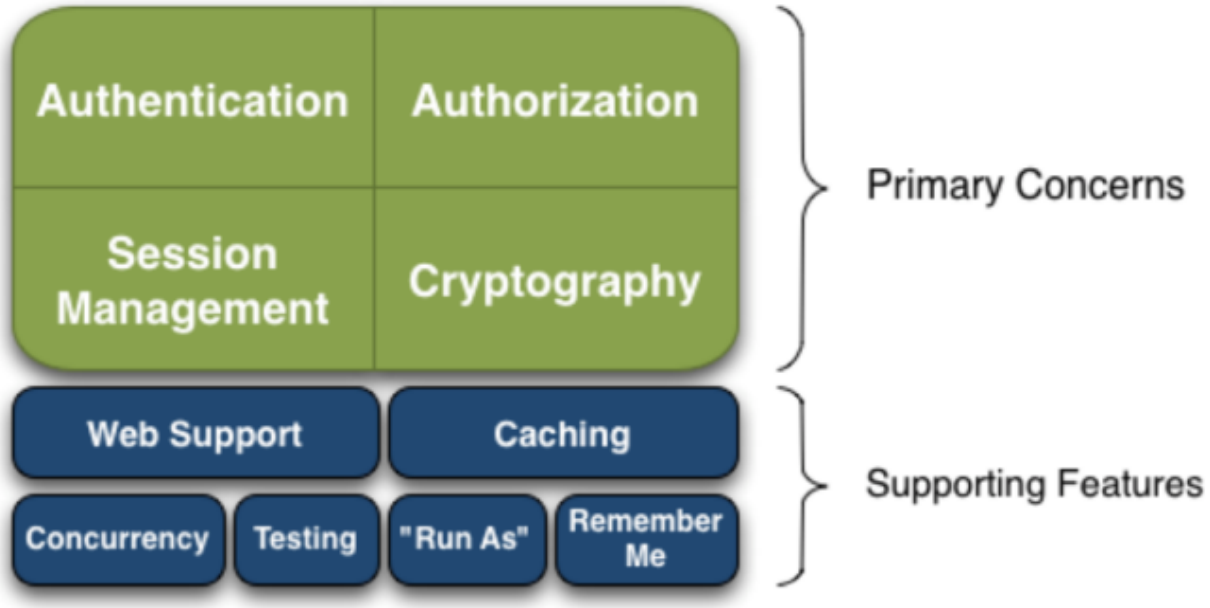
Shiro

概念

Apache Shiro 是 Java 的一个安全框架，实现简单，不仅可以用在 JavaSE 环境，也可以用在 JavaEE 环境。

模块



(1) Authentication：身份认证，验证用户是否拥有相应的身份；

(2) Authorization：授权，即权限验证，验证某个已认证的用户是否拥有某个权限；即判断用户是否能做事情，常见的如：验证某个用户是否拥有某个角色或者细粒度的验证某个用户对某个资源是否具有某个权限；

(3) Session Management：会话管理，即用户登录后就是一次会话，在没有退出之前，它的所有信息都在会话中；会话可以是普通 JavaSE 环境的，也可以是如 Web 环境的；

(4) Cryptography：加密，保护数据的安全性，如密码加密存储到数据库，而不是明文存储；

(5) Web Support：web支持，不仅支持javase还支持javaee

(6) Caching：缓存，比如用户登录后，其用户信息、拥有的角色 / 权限不必每次去查，这样可以提高效率；

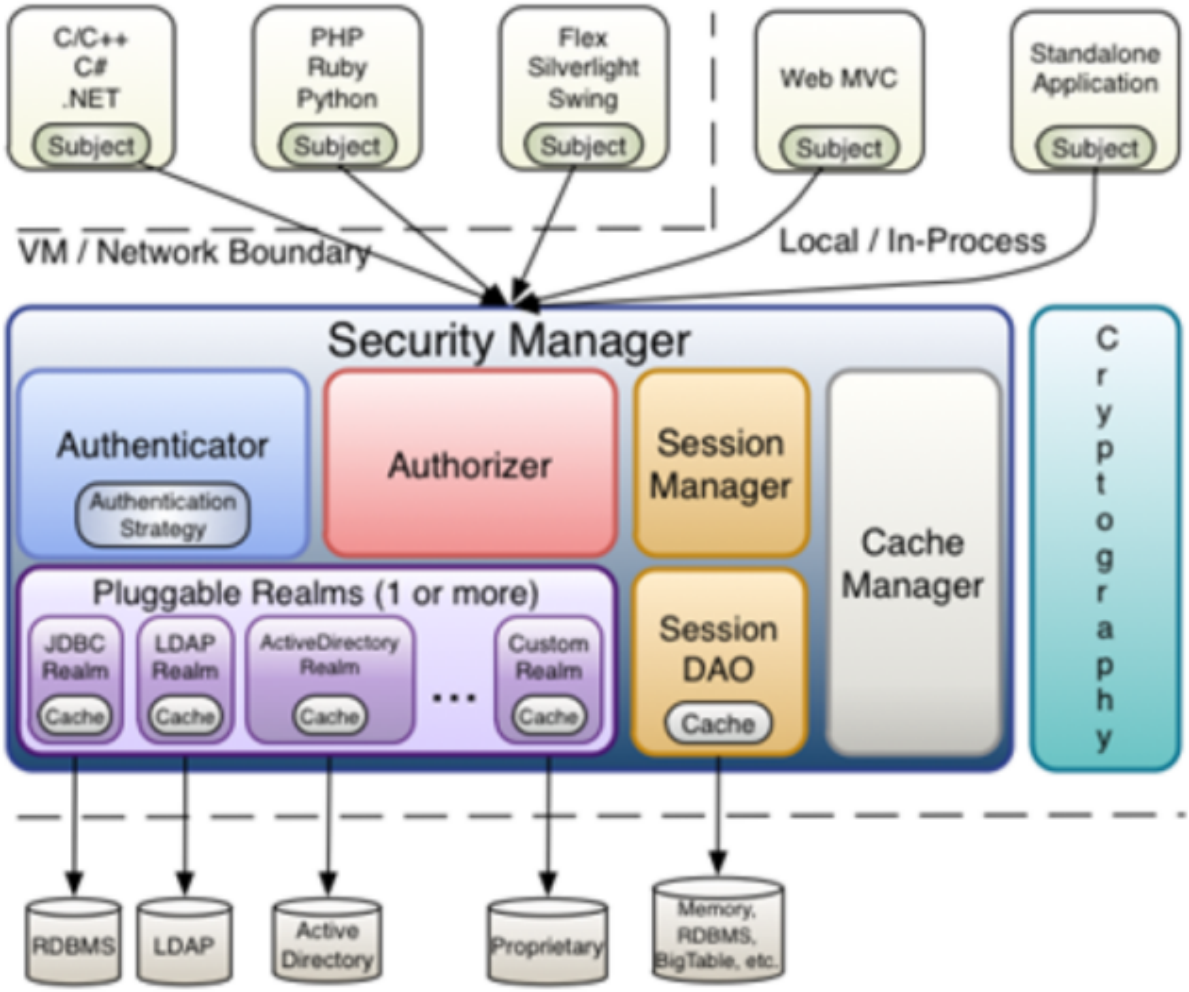
(7) Concurrency：shiro 支持多线程应用的并发验证，即如在一个线程中开启另一个线程，能把权限自动传播过去；

(8) Testing：提供测试支持；

(9) Run As：允许一个用户假装为另一个用户（如果他们允许）的身份进行访问；

(10) Remember Me：记住我，这个是非常常见的功能，即一次登录后，下次再来的话不用登录了。

架构



(1) Subject：主体，主体可以是任何可以与应用交互的 “用户”，简单来说任何携带用户登陆信息（用户名、密码等）和访问资源的对象

(2) Security Manager：安全管理器，相当于 SpringMVC 中的 DispatcherServlet 或者 Struts2 中的 FilterDispatcher；是 Shiro 的心脏；所有具体的交互都通过 SecurityManager 进行控制；它管理着所有 Subject、且负责进行认证和授权、及会话、缓存的管理等

(3) Authenticator：认证器，负责主体认证的，这是一个扩展点，如果用户觉得 Shiro 默认的不好，可以自定义实现；其需要认证策略（Authentication Strategy），即什么情况下算用户认证通过了；

(4) Authrizer：授权器，或者访问控制器，用来决定主体是否有权限进行相应的操作；即控制着用户能访问应用中的哪些功能；

(5) Realm：域，可以有 1 个或多个 Realm，可以认为是安全实体数据源，即用于获取安全实体的；可以是 JDBC 实现，也可以是 LDAP 实现，或者内存实现等等；由用户提供；注意：Shiro 不知道你的用户 / 权限存储在哪及以何种格式存储；所以我们一般在应用中都需要实现自己的 Realm；

(6) SessionManager：如果写过 Servlet 就应该知道 Session 的概念，Session 呢需要有人去管理它的生命周期，这个组件就是 SessionManager；Shiro 就抽象了一个自己的 Session 来管理主体与应用之间交互的数据；

(7) SessionDAO：DAO 大家都用过，数据访问对象，用于会话的 CRUD，比如我们想把 Session 保存到数据库，那么可以实现自己的 SessionDAO，通过如 JDBC 写到数据库；比如想把 Session 放到 Memcached中，可以实现自己的 Memcached SessionDAO；另外 SessionDAO 中可以使用 Cache 进行缓存，以提高性能；

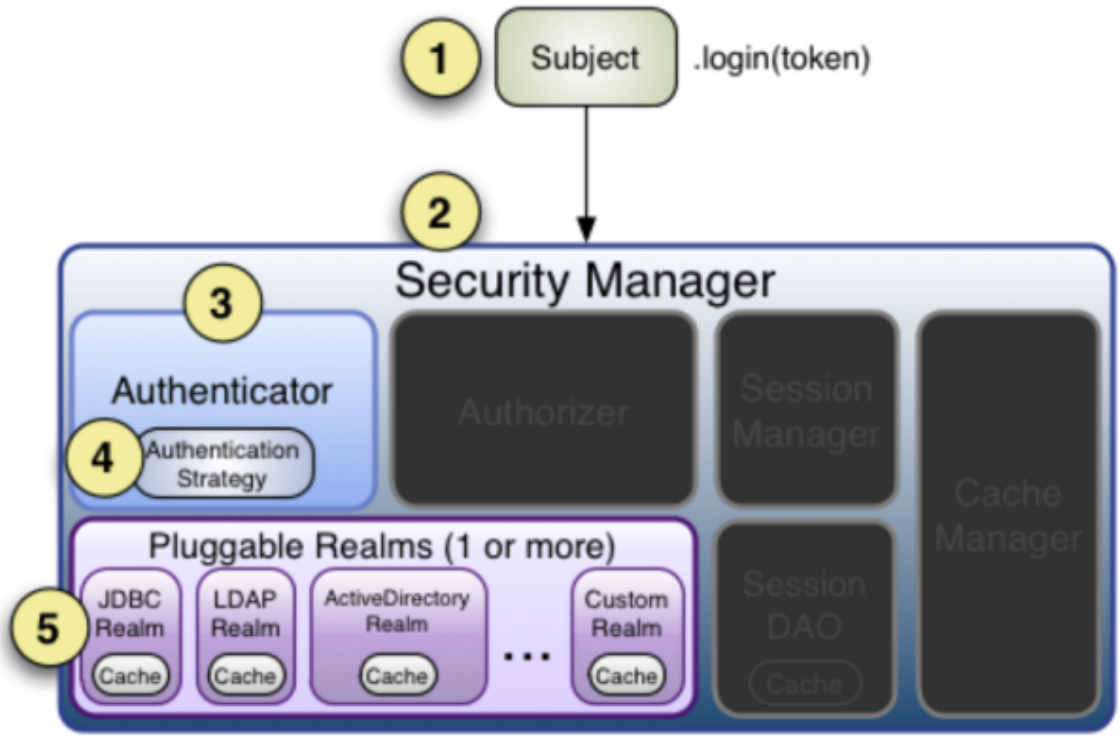
(8) CacheManager：缓存控制器，来管理如用户、角色、权限等的缓存的；因为这些数据基本上很少去改变，放到缓存中后可以提高访问的性能；

(9) Cryptography：密码模块，Shiro 提高了一些常见的加密组件用于如密码加密 / 解密的

权限数据库设计

1. 用户表
2. 用户角色表
3. 角色表
4. 角色权限表
5. 权限表

身份认证流程



流程如下：

1. 首先调用 Subject.login(token) 进行登录，其会自动委托给 Security Manager，调用之前必须通过 SecurityUtils.setSecurityManager() 设置；
2. SecurityManager 负责真正的身份验证逻辑；它会委托给 Authenticator 进行身份验证；
3. Authenticator 才是真正的身份验证者，Shiro API 中核心的身份认证入口点，此处可以自定义插入自己的实现；
4. Authenticator 可能会委托给相应的 AuthenticationStrategy 进行多 Realm 身份验证，默认 ModularRealmAuthenticator 会调用 AuthenticationStrategy 进行多 Realm 身份验证；
5. Authenticator 会把相应的 token 传入 Realm，从 Realm 获取身份验证信息，如果没有返回 / 抛出异常表示身份验证失败了。此处可以配置多个 Realm，将按照相应的顺序及策略进行访问。

身份认证异常（AuthenticationException）

Shiro身份验证通过与否是以异常的方式提示，需要根据异常来判断，异常的种类如下：

(1) AccountException(账户异常)：

① ConcurrentAccessException：多线程并发身份验证异常

② DisabledAccountException：账号已禁用异常

(2) LockedAccountException：账号已锁定异常

③ ExcessiveAttemptsException：尝试认证次数过多异常

④ UnknownAccountException：未知账号异常

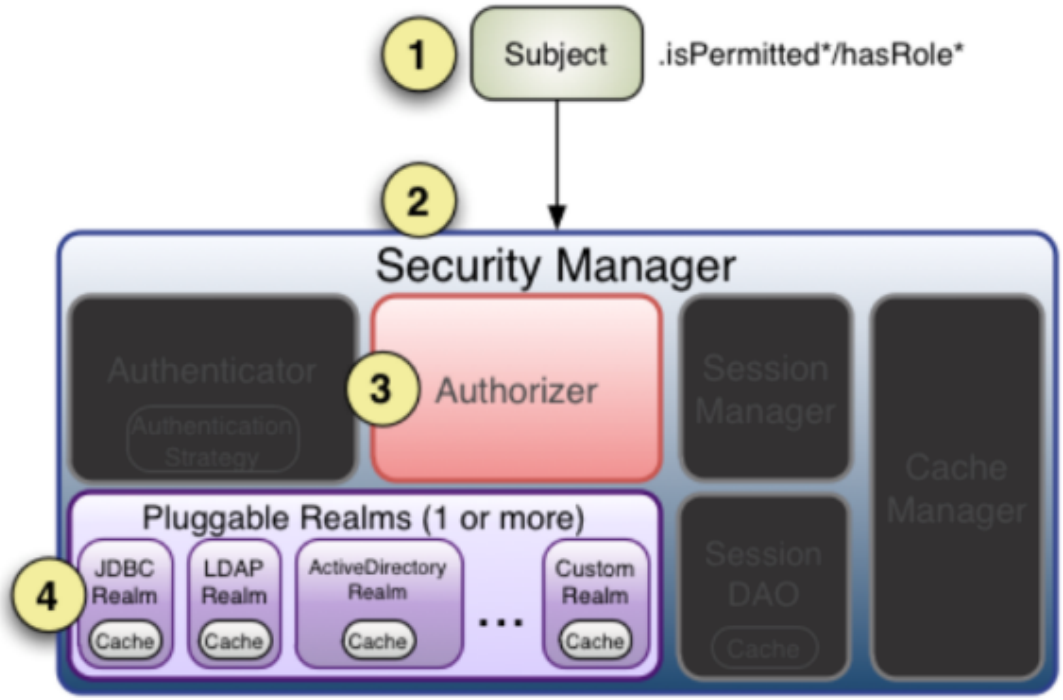
(3) CredentialsException：凭证异常

① ExpiredCredentialsException：密码已过期异常

② IncorrectCredentialsException：密码错误异常

(4) UnsupportedTokenException：不支持令牌异常

鉴权流程



流程如下：

1. 首先调用 Subject.isPermitted/*hasRole* 接口，其会委托给 SecurityManager，而 SecurityManager接着会委托给 Authorizer；
2. Authorizer 是真正的授权者，如果我们调用如 isPermitted(“user:view”)，其首先会通过 PermissionResolver 把字符串转换成相应的 Permission 实例；
3. 在进行授权之前，其会调用相应的 Realm 获取 Subject 相应的角色/权限用于匹配传入的角色/权限；
4. Authorizer 会判断 Realm 的角色/权限是否和传入的匹配，如果有多个 Realm，会委托给 ModularRealm Authorizer 进行循环判断，如果匹配如 isPermitted/*hasRole* 会返回 true，否则返回 false 表示授权失败。

加密方式

(1) 散列算法

① **MD5**

② SHA

③ DES

(2) 非对称加密算法

① **RSA**

② ECC

③ Elgamal

实例

自定义Realm

```
/**
 * 自定义realm
 *
 * @version 2018年3月13日下午4:07:15
 * @author zhuwenbin
 */
public class ShiroRealm extends AuthorizingRealm {

    @Autowired
    private SysUserService sysUserService;
    @Autowired
    private SysRoleService sysRoleService;
    @Autowired
    private SysFunctionService sysFunctionService;

    /**
     * 资源授权
     */
    @Override
    protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
        // 获取身份信息
        ShiroUser user = (ShiroUser) principals.getPrimaryPrincipal();
        // 获取用户id
        int userId = user.getUserId();

        // 授权信息
        SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();

        // 根据用户id获取用户角色信息
        List<SysRole> sysRoles = sysRoleService.getSysRoleByUserId(userId);
        // 将角色列表添加到授权信息内
        Set<String> roles = new HashSet<>();
        for (SysRole sysRole : sysRoles) {
            roles.add(sysRole.getRoleName());
        }
        authorizationInfo.setRoles(roles);

        // 根据用户id获取用户权限信息
        List<SysFunction> sysFunctions = sysFunctionService.getSysFunctionByUserId(userId);
        // 将权限列表添加到授权信息内
        Set<String> functions = new HashSet<>();
        for (SysFunction sysFunction : sysFunctions) {
            functions.add(sysFunction.getPermission());
        }
        authorizationInfo.setStringPermissions(functions);
```

```
        return authorizationInfo;
    }

    /**
     * 身份认证
     */
    @Override
    protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws
AuthenticationException {
        //判断token是否是UsernamePasswordToken
        if (!(token instanceof UsernamePasswordToken)) {
            throw new UnsupportedTokenException();
        }
        // 将token强转为UsernamePasswordToken
        UsernamePasswordToken usernamePasswordToken = (UsernamePasswordToken) token;

        // 从token中得到用户名密码
        String username = usernamePasswordToken.getUsername();
        String password = new String(usernamePasswordToken.getPassword());

        // 根据用户名查询用户信息
        SysUser sysUser = sysUserService.getSysUserByUserName(username);
        // 判断用户是否存在
        if (Objects.isNull(sysUser)) {
            throw new UnknownAccountException();
        }

        // 用户被冻结
        if (Objects.equals(sysUser.getStatus(), 1)) {
            throw new LockedAccountException();
        }

        // 用户被删除
        if (Objects.equals(sysUser.getStatus(), 2)) {
            throw new DisabledAccountException();
        }

        // 判断密码是否正确
        if (!Objects.equals(password, sysUser.getPassword())) {
            throw new IncorrectCredentialsException();
        }
        // 返回认证信息，后续授权通过认证信息得到用户的身份信息
        return new SimpleAuthenticationInfo(
                new ShiroUser(sysUser.getUserId(), sysUser.getUserName(),
sysUser.getRealName()), password, getName());
    }

    /**
     * 清空用户关联权限认证，待下次使用时重新加载
     */
    public void clearCachedAuthorizationInfo(String principal) {
        SimplePrincipalCollection principals = new SimplePrincipalCollection(principal,
getName());
```

```
        clearCachedAuthorizationInfo(principals);
    }

    /**
     *
     * 自定义Authentication对象，使得Subject除了携带用户的登录名外还可以携带更多信息
     *
     * @version 2018年3月13日下午6:44:15
     * @author zhuwenbin
     */
    class ShiroUser {

        private Integer userId;

        private String userName;

        private String realName;

        public ShiroUser() {
        }

        public ShiroUser(Integer userId, String userName, String realName) {
            this.userId = userId;
            this.userName = userName;
            this.realName = realName;
        }

        public Integer getUserId() {
            return userId;
        }

        public void setUserId(Integer userId) {
            this.userId = userId;
        }

        public String getUserName() {
            return userName;
        }

        public void setUserName(String userName) {
            this.userName = userName;
        }

        public String getRealName() {
            return realName;
        }

        public void setRealName(String realName) {
            this.realName = realName;
        }

        /**
         * 本函数输出将作为默认的<shiro:principal/>输出
```

```
         *
         * @return
         */
        @Override
        public String toString() {
            return realName;
        }

    }

}
```

配置xml

spring-shiro.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop.xsd">

    <!-- shiro过滤器，指定哪些资源需要权限验证 -->
    <bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
        <!-- 指定安全管理器 -->
        <property name="securityManager" ref="securityManager" />
        <!-- 指定登陆地址，如果用户未登陆会跳转到该地址 -->
        <property name="loginUrl" value="/admin/login" />
        <!-- 指定成功地址，如果用户登陆成功后会跳转到该地址 -->
        <property name="successUrl" value="/admin/home" />
        <!-- 指定未授权地址，如果用户访问未授权的资源会跳转到该地址 -->
        <property name="unauthorizedUrl" value="/admin/login" />
        <!-- 指定过滤资源，哪些需要权限验证 -->
        <property name="filterChainDefinitions">
            <!-- authc指该资源需要授权，anon指该资源不需要授权，先写不需要授权资源再写需要授权的资源 --
>
            <value>
                /js/** = anon
                /css/** = anon
                /images/** = anon
                /include/** = anon
                /admin/login = anon
                /admin/doLogin = anon
                /admin/logout = anon
```

```
                /admin/** = authc
                /** = authc
            </value>
        </property>
    </bean>

    <!-- 安全管理器，shiro核心 -->
    <bean id="securityManager" class="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
        <!-- 指定realm，如果有多个realm则使用realms属性 -->
        <property name="realm" ref="shiroRealm" />
        <!-- 指定会话管理器 -->
        <property name="sessionManager" ref="sessionManager" />
        <!-- 指定缓存管理器 -->
        <property name="cacheManager" ref="shiroCacheManager" />
        <!-- 指定rememberMe管理器加密串 -->
        <property name="rememberMeManager.cipherKey" value="kPH+bIxk5D2deZiIxcaaaA==" />
    </bean>

    <!-- realm，自定义 -->
    <bean id="shiroRealm" class="com.qhcs.ssm.common.auth.ShiroRealm"></bean>

    <!-- 会话管理器 -->
    <bean id="sessionManager" class="org.apache.shiro.web.session.mgt.DefaultWebSessionManager">
        <!-- session存储的实现 -->
        <property name="sessionDAO" ref="sessionDAO" />
        <!-- 全局超时时间 -->
        <property name="globalSessionTimeout" value="${shiro.globalSessionTimeout}" />
        <!-- 定时清理失效会话，清理用户直接关闭浏览器造成的孤立会话   -->
        <property name="sessionValidationInterval" value="${shiro.sessionValidationInterval}"/>
        <!-- 是否启用定时清理失效会话 -->
        <property name="sessionValidationSchedulerEnabled" value="true"/>
        <!-- session的id名字 -->
        <property name="sessionIdCookie" ref="sessionIdCookie" />
    </bean>

    <!-- 会话DAO -->
    <bean id="sessionDAO"
          class="org.apache.shiro.session.mgt.eis.EnterpriseCacheSessionDAO">
        <!-- 指定ehcache中cache的名字 -->
        <property name="activeSessionsCacheName" value="shiro-activeSessionCache"/>
        <!-- 会话id生成器 -->
        <property name="sessionIdGenerator" ref="sessionIdGenerator"/>
    </bean>

    <!-- 缓存管理器，使用Ehcache实现 -->
    <bean id="shiroCacheManager" class="org.apache.shiro.cache.ehcache.EhCacheManager">
        <!-- 注入ehcache -->
        <property name="cacheManager" ref="ehcache" />
    </bean>

    <!-- sessionIdCookie的实现,用于重写覆盖容器默认的JSESSIONID -->
    <bean id="sessionIdCookie" class="org.apache.shiro.web.servlet.SimpleCookie">
        <!-- cookie的name,对应的默认是 JSESSIONID -->
```

```
        <constructor-arg name="name" value="SHAREJSESSIONID" />
        <!-- jsessionId的path为 / 用于多个系统共享jsessionId -->
        <!-- <property name="path" value="/qhcs" /> -->
    </bean>

    <!-- 会话ID生成器 -->
    <bean id="sessionIdGenerator"
        class="org.apache.shiro.session.mgt.eis.JavaUuidSessionIdGenerator" />

    <!-- hash凭证匹配器 -->
    <bean id="hashedCredentialsMatcher"
        class="org.apache.shiro.authc.credential.HashedCredentialsMatcher">
        <!-- hash算法名字 -->
        <property name="hashAlgorithmName" value="MD5" />
        <!-- 是否保存凭证hex值 -->
        <property name="storedCredentialsHexEncoded" value="true" />
        <!-- 加密次数 -->
        <property name="hashIterations" value="1" />
    </bean>

    <!-- 保证实现了Shiro内部lifecycle函数的bean执行 -->
    <bean id="lifecycleBeanPostProcessor"
class="org.apache.shiro.spring.LifecycleBeanPostProcessor" />

</beans>
```

注意：允许shiro注解的配置应该首先执行

spring-mvc.xml

```
    <!-- 允许shiro注解，一定要写在最先加载的xml中，写在后面加载的xml中也不起作用 -->
        <bean
            class="org.springframework.aop.framework.autoproxy.DefaultAdvisorAutoProxyCreator"
            depends-on="lifecycleBeanPostProcessor" />
        <bean

class="org.apache.shiro.spring.security.interceptor.AuthorizationAttributeSourceAdvisor">
            <property name="securityManager" ref="securityManager" />
        </bean>
```

spring-ehcache.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:cache="http://www.springframework.org/schema/cache"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/cache
```

```
        http://www.springframework.org/schema/cache/spring-cache.xsd">

        <!-- ehcache缓存工厂 -->
        <bean id="ehcache"
            class="org.springframework.cache.ehcache.EhCacheManagerFactoryBean">
            <!-- 加载ehcache配置 -->
            <property name="configLocation" value="classpath:ehcache.xml" />
        </bean>

        <!-- ehcache缓存管理器 -->
        <bean id="cacheManager" class="org.springframework.cache.ehcache.EhCacheCacheManager">
            <property name="cacheManager" ref="ehcache" />
        </bean>

        <!-- 启用缓存注解开关 -->
        <cache:annotation-driven cache-manager="cacheManager" />

</beans>
```

ehcache.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<ehcache xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="http://ehcache.org/ehcache.xsd">

    <!-- 磁盘缓存位置 -->
    <diskStore path="java.io.tmpdir/ehcache" />

    <!-- 默认缓存 -->
    <!-- timeToIdleSeconds:置对象在失效前的允许闲置时间（单位：秒）。仅当eternal=false对象不是永久有效时使用，可选属性，默认值是0，也就是可闲置时间无穷大 -->
    <!-- timeToLiveSeconds:缓存数据的生存时间（TTL），也就是一个元素从构建到消亡的最大时间间隔值，这只能在元素不是永久驻留时有效，如果该值是0就意味着元素可以停顿无穷长的时间 -->
    <!-- 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FIFO（先进先出）或是LFU（较少使用） -->
    <defaultCache maxEntriesLocalHeap="10000" eternal="false"
        timeToIdleSeconds="120" timeToLiveSeconds="120" maxEntriesLocalDisk="10000000"
        diskExpiryThreadIntervalSeconds="120" memoryStoreEvictionPolicy="LRU" />

    <!-- 保存shiro会话的缓存 -->
    <cache name="shiro-activeSessionCache" maxEntriesLocalHeap="10000"
        eternal="false" timeToIdleSeconds="600" timeToLiveSeconds="1800"
        maxEntriesLocalDisk="10000000" diskExpiryThreadIntervalSeconds="120"
        memoryStoreEvictionPolicy="LRU" />

</ehcache>
```

编写页面

```
```

```
<!-- shiro标签库 -->
<%@ taglib uri="http://shiro.apache.org/tags" prefix="shiro"%>

<aside class="lt_aside_nav content mCustomScrollbar">
        <h2>
            <a href="index.html">起始页</a>
        </h2>
        <ul>
            <!-- 设置访问需要的权限标识 -->
            <shiro:hasPermission name="user:users">
                <li>
                    <dl>
                        <dt>用户管理</dt>
                        <dd>
                            <a href="${ctx }/admin/users">用户列表</a>
                        </dd>
                    </dl>
                </li>
            </shiro:hasPermission>
            <!-- 设置访问需要的权限标识 -->
            <shiro:hasPermission name="role:roles">
                <li>
                    <dl>
                        <dt>角色管理</dt>
                        <dd>
                            <a href="user_list.html">角色列表</a>
                        </dd>
                    </dl>
                </li>
            </shiro:hasPermission>
            <!-- 设置访问需要的权限标识 -->
            <shiro:hasPermission name="function:functions">
                <li>
                    <dl>
                        <dt>权限管理</dt>
                        <dd>
                            <a href="user_list.html">权限列表</a>
                        </dd>
                    </dl>
                </li>
            </shiro:hasPermission>
            <!-- 设置访问需要的权限标识 -->
            <shiro:hasPermission name="article:articles">
                <li>
                    <dl>
                        <dt>文章管理</dt>
                        <dd>
                            <a href="${ctx }/admin/articles">文章列表</a>
                        </dd>
                    </dl>
                </li>
            </shiro:hasPermission>
        </ul>
```

```
    </aside>
```

编写Controller代码

```
    /**
     *
     * 文章列表
     *
     * @version 2018年3月14日下午6:15:07
     * @author zhuwenbin
     * @return
     */
    @GetMapping("/articles")
    @RequiresPermissions("article:articles")
    public String articleList(Model model) {
        List<Article> articles = articleService.getAllArticle();
        model.addAttribute("articles", articles);
        return "articleList";
    }

    /**
     *
     * 删除文章
     *
     * @version 2018年3月14日下午8:19:35
     * @author zhuwenbin
     * @param id
     * @return
     */
    @GetMapping("/articles/{id}")
    @RequiresPermissions("article:delete")
    public String deleteArticle(@PathVariable("id") int id) {
        articleService.deleteArticleById(id);
        return "redirect:/admin/articles";
    }
```