[**第二十三章 多项目集中权限管理及分布式会话——《跟我学Shiro》**](http://jinnianshilongnian.iteye.com/blog/2047168)

**博客分类：**

* [跟我学Shiro](http://jinnianshilongnian.iteye.com/category/305053)

[跟我学Shiro](http://www.iteye.com/blogs/tag/%E8%B7%9F%E6%88%91%E5%AD%A6Shiro)

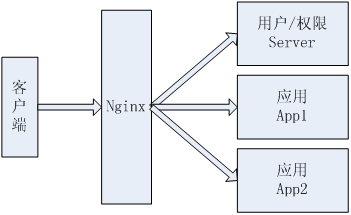
**目录贴：**[**跟我学Shiro目录贴**](http://jinnianshilongnian.iteye.com/blog/2018398)

在做一些企业内部项目时或一些互联网后台时；可能会涉及到集中权限管理，统一进行多项目的权限管理；另外也需要统一的会话管理，即实现单点身份认证和授权控制。

学习本章之前，请务必先学习《第十章 会话管理》和《第十六章 综合实例》，本章代码都是基于这两章的代码基础上完成的。

本章示例是同域名的场景下完成的，如果跨域请参考《第十五章 单点登录》和《第十七章 OAuth2集成》了解使用CAS或OAuth2实现跨域的身份验证和授权。另外比如客户端/服务器端的安全校验可参考《第二十章 无状态Web应用集成》。

**部署架构**



1、有三个应用：用于用户/权限控制的Server（端口：8080）；两个应用App1（端口9080）和App2（端口10080）；

2、使用Nginx反向代理这三个应用，nginx.conf的server配置部分如下：

**Java代码  [收藏代码](javascript:void())**

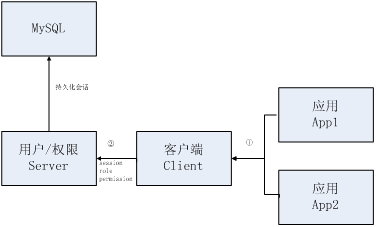
1. server {
2. listen 80;
3. server\_name  localhost;
4. charset utf-8;
5. location ~ ^/(chapter23-server)/ {
6. proxy\_pass http://127.0.0.1:8080;
7. index /;
8. proxy\_set\_header Host $host;
9. }
10. location ~ ^/(chapter23-app1)/ {
11. proxy\_pass http://127.0.0.1:9080;
12. index /;
13. proxy\_set\_header Host $host;
14. }
15. location ~ ^/(chapter23-app2)/ {
16. proxy\_pass http://127.0.0.1:10080;
17. index /;
18. proxy\_set\_header Host $host;
19. }
20. }

如访问<http://localhost/chapter23-server>会自动转发到<http://localhost:8080/chapter23-server>；

访问<http://localhost/chapter23-app1>会自动转发到<http://localhost:9080/chapter23-app1>；访问<http://localhost/chapter23-app3>会自动转发到<http://localhost:10080/chapter23-app3>；

Nginx的安装及使用请自行搜索学习，本文不再阐述。

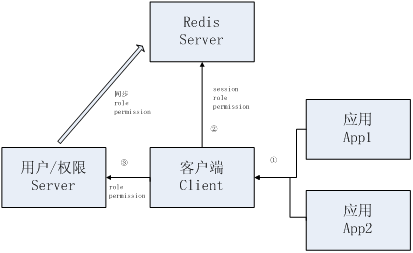
**项目架构**



1、首先通过用户/权限Server维护用户、应用、权限信息；数据都持久化到MySQL数据库中；

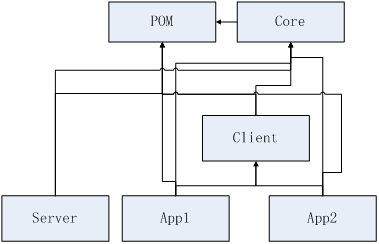
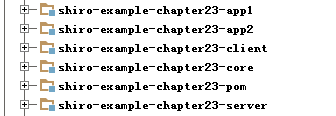
2、应用App1/应用App2使用客户端Client远程调用用户/权限Server获取会话及权限信息。

此处使用Mysql存储会话，而不是使用如Memcached/Redis之类的，主要目的是降低学习成本；如果换成如Redis也不会很难；如：



使用如Redis还一个好处就是无需在用户/权限Server中开会话过期调度器，可以借助Redis自身的过期策略来完成。

**模块关系依赖**

1、shiro-example-chapter23-pom模块：提供了其他所有模块的依赖；这样其他模块直接继承它即可，简化依赖配置，如shiro-example-chapter23-server：

**Java代码  [收藏代码](javascript:void())**

1. <parent>
2. <artifactId>shiro-example-chapter23-pom</artifactId>
3. <groupId>com.github.zhangkaitao</groupId>
4. <version>1.0-SNAPSHOT</version>
5. </parent>

2、shiro-example-chapter23-core模块：提供给shiro-example-chapter23-server、shiro-example-chapter23-client、shiro-example-chapter23-app\*模块的核心依赖，比如远程调用接口等；

3、shiro-example-chapter23-server模块：提供了用户、应用、权限管理功能；

4、shiro-example-chapter23-client模块：提供给应用模块获取会话及应用对应的权限信息；

5、shiro-example-chapter23-app\*模块：各个子应用，如一些内部管理系统应用；其登录都跳到shiro-example-chapter23-server登录；另外权限都从shiro-example-chapter23-server获取（如通过远程调用）。

**shiro-example-chapter23-pom模块**

其pom.xml的packaging类型为pom，并且在该pom中加入其他模块需要的依赖，然后其他模块只需要把该模块设置为parent即可自动继承这些依赖，如shiro-example-chapter23-server模块：

**Java代码  [收藏代码](javascript:void())**

1. <parent>
2. <artifactId>shiro-example-chapter23-pom</artifactId>
3. <groupId>com.github.zhangkaitao</groupId>
4. <version>1.0-SNAPSHOT</version>
5. </parent>

简化其他模块的依赖配置等。

**shiro-example-chapter23-core模块**

提供了其他模块共有的依赖，如远程调用接口：

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** RemoteServiceInterface {
2. **public** Session getSession(String appKey, Serializable sessionId);
3. Serializable createSession(Session session);
4. **public** **void** updateSession(String appKey, Session session);
5. **public** **void** deleteSession(String appKey, Session session);
6. **public** PermissionContext getPermissions(String appKey, String username);
7. }

提供了会话的CRUD，及根据应用key和用户名获取权限上下文（包括角色和权限字符串）；shiro-example-chapter23-server模块服务端实现；shiro-example-chapter23-client模块客户端调用。

另外提供了com.github.zhangkaitao.shiro.chapter23.core.ClientSavedRequest，其扩展了org.apache.shiro.web.util.SavedRequest；用于shiro-example-chapter23-app\*模块当访问一些需要登录的请求时，自动把请求保存下来，然后重定向到shiro-example-chapter23-server模块登录；登录成功后再重定向回来；因为SavedRequest不保存URL中的[schema://domain:port](http://domainport/)部分；所以才需要扩展SavedRequest；使得ClientSavedRequest能保存[schema://domain:port](http://domainport/)；这样才能从一个应用重定向另一个（要不然只能在一个应用内重定向）：

**Java代码  [收藏代码](javascript:void())**

1. **public** String getRequestUrl() {
2. String requestURI = getRequestURI();
3. **if**(backUrl != **null**) {//1
4. **if**(backUrl.toLowerCase().startsWith("http://") || backUrl.toLowerCase().startsWith("https://")) {
5. **return** backUrl;
6. } **else** **if**(!backUrl.startsWith(contextPath)) {//2
7. requestURI = contextPath + backUrl;
8. } **else** {//3
9. requestURI = backUrl;
10. }
11. }
12. StringBuilder requestUrl = **new** StringBuilder(scheme);//4
13. requestUrl.append("://");
14. requestUrl.append(domain);//5
15. //6
16. **if**("http".equalsIgnoreCase(scheme) && port != 80) {
17. requestUrl.append(":").append(String.valueOf(port));
18. } **else** **if**("https".equalsIgnoreCase(scheme) && port != 443) {
19. requestUrl.append(":").append(String.valueOf(port));
20. }
21. //7
22. requestUrl.append(requestURI);
23. //8
24. **if** (backUrl == **null** && getQueryString() != **null**) {
25. requestUrl.append("?").append(getQueryString());
26. }
27. **return** requestUrl.toString();
28. }

1、如果从外部传入了successUrl（登录成功之后重定向的地址），且以http://或https://开头那么直接返回（相应的拦截器直接重定向到它即可）；

2、如果successUrl有值但没有上下文，拼上上下文；

3、否则，如果successUrl有值，直接赋值给requestUrl即可；否则，如果successUrl没值，那么requestUrl就是当前请求的地址；

5、拼上url前边的schema，如http或https；

6、拼上域名；

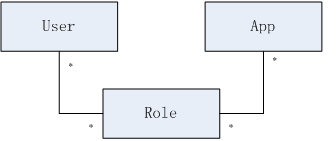
7、拼上重定向到的地址（带上下文）；

8、如果successUrl没值，且有查询参数，拼上；

9返回该地址，相应的拦截器直接重定向到它即可。

**shiro-example-chapter23-server模块**

**简单的实体关系图**



**简单数据字典**

用户(sys\_user)

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 长度 | 描述 |
| id | bigint |  | 编号 主键 |
| username | varchar | 100 | 用户名 |
| password | varchar | 100 | 密码 |
| salt | varchar | 50 | 盐 |
| locked | bool |  | 账户是否锁定 |

应用(sys\_app)

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 长度 | 描述 |
| id | bigint |  | 编号 主键 |
| name | varchar | 100 | 应用名称 |
| app\_key | varchar | 100 | 应用key（唯一） |
| app\_secret | varchar | 100 | 应用安全码 |
| available | bool |  | 是否锁定 |

授权(sys\_authorization)

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 类型 | 长度 | 描述 |
| id | bigint |  | 编号 主键 |
| user\_id | bigint |  | 所属用户 |
| app\_id | bigint |  | 所属应用 |
| role\_ids | varchar | 100 | 角色列表 |

**用户**：比《第十六章 综合实例》少了role\_ids，因为本章是多项目集中权限管理；所以授权时需要指定相应的应用；而不是直接给用户授权；所以不能在用户中出现role\_ids了；

**应用**：所有集中权限的应用；在此处需要指定应用key(app\_key)和应用安全码（app\_secret），app在访问server时需要指定自己的app\_key和用户名来获取该app对应用户权限信息；另外app\_secret可以认为app的密码，比如需要安全访问时可以考虑使用它，可参考《第二十章 无状态Web应用集成》。另外available属性表示该应用当前是否开启；如果false表示该应用当前不可用，即不能获取到相应的权限信息。

**授权**：给指定的用户在指定的app下授权，即角色是与用户和app存在关联关系。

因为本章使用了《第十六章 综合实例》代码，所以还有其他相应的表结构（本章未使用到）。

**表/数据SQL**

具体请参考

sql/ shiro-schema.sql （表结构）

sql/ shiro-data.sql  （初始数据）

**实体**

具体请参考com.github.zhangkaitao.shiro.chapter23.entity包下的实体，此处就不列举了。

**DAO**

具体请参考com.github.zhangkaitao.shiro.chapter23.dao包下的DAO接口及实现。

**Service**

具体请参考com.github.zhangkaitao.shiro.chapter23.service包下的Service接口及实现。以下是出了基本CRUD之外的关键接口：

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** AppService {
2. **public** Long findAppIdByAppKey(String appKey);// 根据appKey查找AppId
3. }

**Java代码  [收藏代码](javascript:void())**

1. **public** **interface** AuthorizationService {
2. //根据AppKey和用户名查找其角色
3. **public** Set<String> findRoles(String appKey, String username);
4. //根据AppKey和用户名查找权限字符串
5. **public** Set<String> findPermissions(String appKey, String username);
6. }

根据AppKey和用户名查找用户在指定应用中对于的角色和权限字符串。

**UserRealm**

**Java代码  [收藏代码](javascript:void())**

1. **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
2. String username = (String)principals.getPrimaryPrincipal();
3. SimpleAuthorizationInfo authorizationInfo = **new** SimpleAuthorizationInfo();
4. authorizationInfo.setRoles(
5. authorizationService.findRoles(Constants.SERVER\_APP\_KEY, username));
6. authorizationInfo.setStringPermissions(
7. authorizationService.findPermissions(Constants.SERVER\_APP\_KEY, username));
8. **return** authorizationInfo;
9. }

此处需要调用AuthorizationService的findRoles/findPermissions方法传入AppKey和用户名来获取用户的角色和权限字符串集合。其他的和《第十六章 综合实例》代码一样。

**ServerFormAuthenticationFilter**

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ServerFormAuthenticationFilter **extends** FormAuthenticationFilter {
2. **protected** **void** issueSuccessRedirect(ServletRequest request, ServletResponse response) **throws** Exception {
3. String fallbackUrl = (String) getSubject(request, response)
4. .getSession().getAttribute("authc.fallbackUrl");
5. **if**(StringUtils.isEmpty(fallbackUrl)) {
6. fallbackUrl = getSuccessUrl();
7. }
8. WebUtils.redirectToSavedRequest(request, response, fallbackUrl);
9. }
10. }

因为是多项目登录，比如如果是从其他应用中重定向过来的，首先检查Session中是否有“authc.fallbackUrl”属性，如果有就认为它是默认的重定向地址；否则使用Server自己的successUrl作为登录成功后重定向到的地址。

**MySqlSessionDAO**

将会话持久化到Mysql数据库；此处大家可以将其实现为如存储到Redis/Memcached等，实现策略请参考《第十章 会话管理》中的会话存储/持久化章节的MySessionDAO，完全一样。

**MySqlSessionValidationScheduler**

和《第十章 会话管理》中的会话验证章节部分中的MySessionValidationScheduler完全一样。如果使用如Redis之类的有自动过期策略的DB，完全可以不用实现SessionValidationScheduler，直接借助于这些DB的过期策略即可。

**RemoteService**

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** RemoteService **implements** RemoteServiceInterface {
2. @Autowired  **private** AuthorizationService authorizationService;
3. @Autowired  **private** SessionDAO sessionDAO;
5. **public** Session getSession(String appKey, Serializable sessionId) {
6. **return** sessionDAO.readSession(sessionId);
7. }
8. **public** Serializable createSession(Session session) {
9. **return** sessionDAO.create(session);
10. }
11. **public** **void** updateSession(String appKey, Session session) {
12. sessionDAO.update(session);
13. }
14. **public** **void** deleteSession(String appKey, Session session) {
15. sessionDAO.delete(session);
16. }
17. **public** PermissionContext getPermissions(String appKey, String username) {
18. PermissionContext permissionContext = **new** PermissionContext();
19. permissionContext.setRoles(authorizationService.findRoles(appKey, username));
20. permissionContext.setPermissions(authorizationService.findPermissions(appKey, username));
21. **return** permissionContext;
22. }
23. }

将会使用HTTP调用器暴露为远程服务，这样其他应用就可以使用相应的客户端调用这些接口进行Session的集中维护及根据AppKey和用户名获取角色/权限字符串集合。此处没有实现安全校验功能，如果是局域网内使用可以通过限定IP完成；否则需要使用如《第二十章 无状态Web应用集成》中的技术完成安全校验。

然后在spring-mvc-remote-service.xml配置文件把服务暴露出去：

**Java代码  [收藏代码](javascript:void())**

1. <bean id="remoteService"
2. **class**="com.github.zhangkaitao.shiro.chapter23.remote.RemoteService"/>
3. <bean name="/remoteService"
4. **class**="org.springframework.remoting.httpinvoker.HttpInvokerServiceExporter">
5. <property name="service" ref="remoteService"/>
6. <property name="serviceInterface"
7. value="com.github.zhangkaitao.shiro.chapter23.remote.RemoteServiceInterface"/>
8. </bean>

**Shiro配置文件spring-config-shiro.xml**

和《第十六章 综合实例》配置类似，但是需要在shiroFilter中的filterChainDefinitions中添加如下配置，即远程调用不需要身份认证：

**Java代码  [收藏代码](javascript:void())**

1. /remoteService = anon

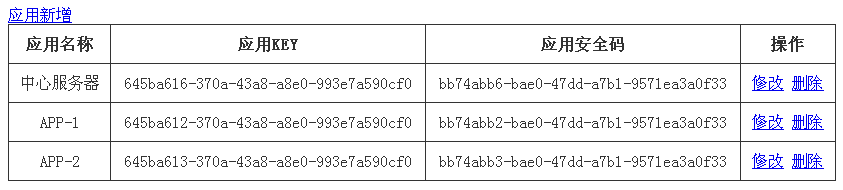
对于userRealm的缓存配置直接禁用；因为如果开启，修改了用户权限不会自动同步到缓存；另外请参考《第十一章 缓存机制》进行缓存的正确配置。

**服务器端数据维护**

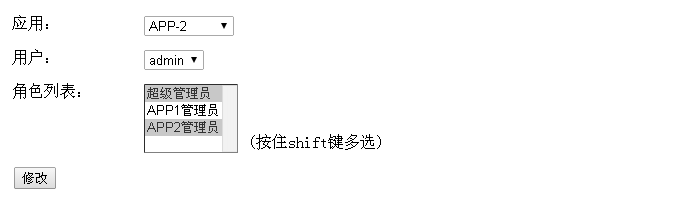
1、首先开启ngnix反向代理；然后就可以直接访问<http://localhost/chapter23-server/>；

2、输入默认的用户名密码：admin/123456登录

3、应用管理，进行应用的CRUD，主要维护应用KEY（必须唯一）及应用安全码；客户端就可以使用应用KEY获取用户对应应用的权限了。



4、授权管理，维护在哪个应用中用户的角色列表。这样客户端就可以根据应用KEY及用户名获取到对应的角色/权限字符串列表了。

**shiro-example-chapter23-client模块**

Client模块提供给其他应用模块依赖，这样其他应用模块只需要依赖Client模块，然后再在相应的配置文件中配置如登录地址、远程接口地址、拦截器链等等即可，简化其他应用模块的配置。

**配置远程服务spring-client-remote-service.xml**

**Java代码  [收藏代码](javascript:void())**

1. <bean id="remoteService"
2. **class**="org.springframework.remoting.httpinvoker.HttpInvokerProxyFactoryBean">
3. <property name="serviceUrl" value="${client.remote.service.url}"/>
4. <property name="serviceInterface"
5. value="com.github.zhangkaitao.shiro.chapter23.remote.RemoteServiceInterface"/>
6. </bean>

client.remote.service.url是远程服务暴露的地址；通过相应的properties配置文件配置，后续介绍。然后就可以通过remoteService获取会话及角色/权限字符串集合了。

**ClientRealm**

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ClientRealm **extends** AuthorizingRealm {
2. **private** RemoteServiceInterface remoteService;
3. **private** String appKey;
4. **public** **void** setRemoteService(RemoteServiceInterface remoteService) {
5. **this**.remoteService = remoteService;
6. }
7. **public** **void** setAppKey(String appKey) {
8. **this**.appKey = appKey;
9. }
10. **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
11. String username = (String) principals.getPrimaryPrincipal();
12. SimpleAuthorizationInfo authorizationInfo = **new** SimpleAuthorizationInfo();
13. PermissionContext context = remoteService.getPermissions(appKey, username);
14. authorizationInfo.setRoles(context.getRoles());
15. authorizationInfo.setStringPermissions(context.getPermissions());
16. **return** authorizationInfo;
17. }
18. **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {
19. //永远不会被调用
20. **throw** **new** UnsupportedOperationException("永远不会被调用");
21. }
22. }

ClientRealm提供身份认证信息和授权信息，此处因为是其他应用依赖客户端，而这些应用不会实现身份认证，所以doGetAuthenticationInfo获取身份认证信息直接无须实现。另外获取授权信息，是通过远程暴露的服务RemoteServiceInterface获取，提供appKey和用户名获取即可。

**ClientSessionDAO**

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ClientSessionDAO **extends** CachingSessionDAO {
2. **private** RemoteServiceInterface remoteService;
3. **private** String appKey;
4. **public** **void** setRemoteService(RemoteServiceInterface remoteService) {
5. **this**.remoteService = remoteService;
6. }
7. **public** **void** setAppKey(String appKey) {
8. **this**.appKey = appKey;
9. }
10. **protected** **void** doDelete(Session session) {
11. remoteService.deleteSession(appKey, session);
12. }
13. **protected** **void** doUpdate(Session session) {
14. remoteService.updateSession(appKey, session);
15. }
16. **protected** Serializable doCreate(Session session) {
17. Serializable sessionId = remoteService.createSession(session);
18. assignSessionId(session, sessionId);
19. **return** sessionId;
20. }
21. **protected** Session doReadSession(Serializable sessionId) {
22. **return** remoteService.getSession(appKey, sessionId);
23. }
24. }

Session的维护通过远程暴露接口实现，即本地不维护会话。

**ClientAuthenticationFilter**

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ClientAuthenticationFilter **extends** AuthenticationFilter {
2. **protected** **boolean** isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue) {
3. Subject subject = getSubject(request, response);
4. **return** subject.isAuthenticated();
5. }
6. **protected** **boolean** onAccessDenied(ServletRequest request, ServletResponse response) **throws** Exception {
7. String backUrl = request.getParameter("backUrl");
8. saveRequest(request, backUrl, getDefaultBackUrl(WebUtils.toHttp(request)));
9. **return** **false**;
10. }
11. **protected** **void** saveRequest(ServletRequest request, String backUrl, String fallbackUrl) {
12. Subject subject = SecurityUtils.getSubject();
13. Session session = subject.getSession();
14. HttpServletRequest httpRequest = WebUtils.toHttp(request);
15. session.setAttribute("authc.fallbackUrl", fallbackUrl);
16. SavedRequest savedRequest = **new** ClientSavedRequest(httpRequest, backUrl);
17. session.setAttribute(WebUtils.SAVED\_REQUEST\_KEY, savedRequest);
18. }
19. **private** String getDefaultBackUrl(HttpServletRequest request) {
20. String scheme = request.getScheme();
21. String domain = request.getServerName();
22. **int** port = request.getServerPort();
23. String contextPath = request.getContextPath();
24. StringBuilder backUrl = **new** StringBuilder(scheme);
25. backUrl.append("://");
26. backUrl.append(domain);
27. **if**("http".equalsIgnoreCase(scheme) && port != 80) {
28. backUrl.append(":").append(String.valueOf(port));
29. } **else** **if**("https".equalsIgnoreCase(scheme) && port != 443) {
30. backUrl.append(":").append(String.valueOf(port));
31. }
32. backUrl.append(contextPath);
33. backUrl.append(getSuccessUrl());
34. **return** backUrl.toString();
35. }
36. }

ClientAuthenticationFilter是用于实现身份认证的拦截器（authc），当用户没有身份认证时；

1、首先得到请求参数backUrl，即登录成功重定向到的地址；

2、然后保存保存请求到会话，并重定向到登录地址（server模块）；

3、登录成功后，返回地址按照如下顺序获取：backUrl、保存的当前请求地址、defaultBackUrl（即设置的successUrl）；

**ClientShiroFilterFactoryBean**

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ClientShiroFilterFactoryBean **extends** ShiroFilterFactoryBean **implements** ApplicationContextAware {
2. **private** ApplicationContext applicationContext;
3. **public** **void** setApplicationContext(ApplicationContext applicationContext) {
4. **this**.applicationContext = applicationContext;
5. }
6. **public** **void** setFiltersStr(String filters) {
7. **if**(StringUtils.isEmpty(filters)) {
8. **return**;
9. }
10. String[] filterArray = filters.split(";");
11. **for**(String filter : filterArray) {
12. String[] o = filter.split("=");
13. getFilters().put(o[0], (Filter)applicationContext.getBean(o[1]));
14. }
15. }
16. **public** **void** setFilterChainDefinitionsStr(String filterChainDefinitions) {
17. **if**(StringUtils.isEmpty(filterChainDefinitions)) {
18. **return**;
19. }
20. String[] chainDefinitionsArray = filterChainDefinitions.split(";");
21. **for**(String filter : chainDefinitionsArray) {
22. String[] o = filter.split("=");
23. getFilterChainDefinitionMap().put(o[0], o[1]);
24. }
25. }
26. }

1、setFiltersStr：设置拦截器，设置格式如“filterName=filterBeanName; filterName=filterBeanName”；多个之间分号分隔；然后通过applicationContext获取filterBeanName对应的Bean注册到拦截器Map中；

2、setFilterChainDefinitionsStr：设置拦截器链，设置格式如“url=filterName1[config],filterName2; url=filterName1[config],filterName2”；多个之间分号分隔；

**Shiro客户端配置spring-client.xml**

提供了各应用通用的Shiro客户端配置；这样应用只需要导入相应该配置即可完成Shiro的配置，简化了整个配置过程。

**Java代码  [收藏代码](javascript:void())**

1. <context:property-placeholder location=
2. "classpath:client/shiro-client-default.properties,classpath:client/shiro-client.properties"/>

提供给客户端配置的properties属性文件，client/shiro-client-default.properties是客户端提供的默认的配置；classpath:client/shiro-client.properties是用于覆盖客户端默认配置，各应用应该提供该配置文件，然后提供各应用个性配置。

**Java代码  [收藏代码](javascript:void())**

1. <bean id="remoteRealm" **class**="com.github.zhangkaitao.shiro.chapter23.client.ClientRealm">
2. <property name="cachingEnabled" value="false"/>
3. <property name="appKey" value="${client.app.key}"/>
4. <property name="remoteService" ref="remoteService"/>
5. </bean>

appKey：使用${client.app.key}占位符替换，即需要在之前的properties文件中配置。

**Java代码  [收藏代码](javascript:void())**

1. <bean id="sessionIdCookie" **class**="org.apache.shiro.web.servlet.SimpleCookie">
2. <constructor-arg value="${client.session.id}"/>
3. <property name="httpOnly" value="true"/>
4. <property name="maxAge" value="-1"/>
5. <property name="domain" value="${client.cookie.domain}"/>
6. <property name="path" value="${client.cookie.path}"/>
7. </bean>

Session Id Cookie，cookie名字、域名、路径等都是通过配置文件配置。

**Java代码  [收藏代码](javascript:void())**

1. <bean id="sessionDAO"
2. **class**="com.github.zhangkaitao.shiro.chapter23.client.ClientSessionDAO">
3. <property name="sessionIdGenerator" ref="sessionIdGenerator"/>
4. <property name="appKey" value="${client.app.key}"/>
5. <property name="remoteService" ref="remoteService"/>
6. </bean>

SessionDAO的appKey，也是通过${ client.app.key }占位符替换，需要在配置文件配置。

**Java代码  [收藏代码](javascript:void())**

1. <bean id="sessionManager"
2. **class**="org.apache.shiro.web.session.mgt.DefaultWebSessionManager">
3. <property name="sessionValidationSchedulerEnabled" value="false"/>//省略其他
4. </bean>

其他应用无须进行会话过期调度，所以sessionValidationSchedulerEnabled=false。

**Java代码  [收藏代码](javascript:void())**

1. <bean id="clientAuthenticationFilter"
2. **class**="com.github.zhangkaitao.shiro.chapter23.client.ClientAuthenticationFilter"/>

应用的身份认证使用ClientAuthenticationFilter，即如果没有身份认证，则会重定向到Server模块完成身份认证，身份认证成功后再重定向回来。

**Java代码  [收藏代码](javascript:void())**

1. <bean id="shiroFilter"
2. **class**="com.github.zhangkaitao.shiro.chapter23.client.ClientShiroFilterFactoryBean">
3. <property name="securityManager" ref="securityManager"/>
4. <property name="loginUrl" value="${client.login.url}"/>
5. <property name="successUrl" value="${client.success.url}"/>
6. <property name="unauthorizedUrl" value="${client.unauthorized.url}"/>
7. <property name="filters">
8. <util:map>
9. <entry key="authc" value-ref="clientAuthenticationFilter"/>
10. </util:map>
11. </property>
12. <property name="filtersStr" value="${client.filters}"/>
13. <property name="filterChainDefinitionsStr" value="${client.filter.chain.definitions}"/>
14. </bean>

ShiroFilter使用我们自定义的ClientShiroFilterFactoryBean，然后loginUrl（登录地址）、successUrl（登录成功后默认的重定向地址）、unauthorizedUrl（未授权重定向到的地址）通过占位符替换方式配置；另外filtersStr和filterChainDefinitionsStr也是使用占位符替换方式配置；这样就可以在各应用进行自定义了。

**默认配置client/ shiro-client-default.properties**

**Java代码  [收藏代码](javascript:void())**

1. #各应用的appKey
2. client.app.key=
3. #远程服务URL地址
4. client.remote.service.url=http://localhost/chapter23-server/remoteService
5. #登录地址
6. client.login.url=http://localhost/chapter23-server/login
7. #登录成功后，默认重定向到的地址
8. client.success.url=/
9. #未授权重定向到的地址
10. client.unauthorized.url=http://localhost/chapter23-server/unauthorized
11. #session id 域名
12. client.cookie.domain=
13. #session id 路径
14. client.cookie.path=/
15. #cookie中的session id名称
16. client.session.id=sid
17. #cookie中的remember me名称
18. client.rememberMe.id=rememberMe
19. #过滤器 name=filter-ref;name=filter-ref
20. client.filters=
21. #过滤器链 格式 url=filters;url=filters
22. client.filter.chain.definitions=/\*\*=anon

在各应用中主要配置client.app.key、client.filters、client.filter.chain.definitions。

**shiro-example-chapter23-app\*模块**

**继承shiro-example-chapter23-pom模块**

**Java代码  [收藏代码](javascript:void())**

1. <parent>
2. <artifactId>shiro-example-chapter23-pom</artifactId>
3. <groupId>com.github.zhangkaitao</groupId>
4. <version>1.0-SNAPSHOT</version>
5. </parent>

**依赖shiro-example-chapter23-client模块**

<dependency>

<groupId>com.github.zhangkaitao</groupId>

<artifactId>shiro-example-chapter23-client</artifactId>

<version>1.0-SNAPSHOT</version>

</dependency>

**客户端配置client/shiro-client.properties**

**配置shiro-example-chapter23-app1**

**Java代码**

1. client.app.key=645ba612-370a-43a8-a8e0-993e7a590cf0
2. client.success.url=/hello
3. client.filter.chain.definitions=/hello=anon;/login=authc;/\*\*=authc

client.app.key是server模块维护的，直接拷贝过来即可；client.filter.chain.definitions定义了拦截器链；比如访问/hello，匿名即可。

**配置shiro-example-chapter23-app2**

**Java代码**

1. client.app.key=645ba613-370a-43a8-a8e0-993e7a590cf0
2. client.success.url=/hello
3. client.filter.chain.definitions=/hello=anon;/login=authc;/\*\*=authc

和app1类似，client.app.key是server模块维护的，直接拷贝过来即可；client.filter.chain.definitions定义了拦截器链；比如访问/hello，匿名即可。

**web.xml**

**Java代码**

1. <context-param>
2. <param-name>contextConfigLocation</param-name>
3. <param-value>
4. classpath:client/spring-client.xml
5. </param-value>
6. </context-param>
7. <listener>
8. <listener-**class**>
9. org.springframework.web.context.ContextLoaderListener
10. </listener-**class**>
11. </listener>

指定加载客户端Shiro配置，client/spring-client.xml。

**Java代码**

1. <filter>
2. <filter-name>shiroFilter</filter-name>
3. <filter-**class**>org.springframework.web.filter.DelegatingFilterProxy</filter-**class**>
4. <init-param>
5. <param-name>targetFilterLifecycle</param-name>
6. <param-value>**true**</param-value>
7. </init-param>
8. </filter>
9. <filter-mapping>
10. <filter-name>shiroFilter</filter-name>
11. <url-pattern>/\*</url-pattern>
12. </filter-mapping>

 配置ShiroFilter拦截器。

**控制器**

shiro-example-chapter23-app1

**Java代码**

1. @Controller
2. **public** **class** HelloController {
3. @RequestMapping("/hello")
4. **public** String hello() {
5. **return** "success";
6. }
7. @RequestMapping(value = "/attr", method = RequestMethod.POST)
8. **public** String setAttr(
9. @RequestParam("key") String key, @RequestParam("value") String value) {
10. SecurityUtils.getSubject().getSession().setAttribute(key, value);
11. **return** "success";
12. }
13. @RequestMapping(value = "/attr", method = RequestMethod.GET)
14. **public** String getAttr(
15. @RequestParam("key") String key, Model model) {
16. model.addAttribute("value",
17. SecurityUtils.getSubject().getSession().getAttribute(key));
18. **return** "success";
19. }
20. @RequestMapping("/role1")
21. @RequiresRoles("role1")
22. **public** String role1() {
23. **return** "success";
24. }
25. }

shiro-example-chapter23-app2的控制器类似，role2方法使用@RequiresRoles("role2")注解，即需要角色2。

其他配置请参考源码。

**测试**

**1、安装配置启动nginx**

1、首先到<http://nginx.org/en/download.html>下载，比如我下载的是windows版本的；

2、然后编辑conf/nginx.conf配置文件，在server部分添加如下部分：

**Java代码**

1. location ~ ^/(chapter23-server)/ {
2. proxy\_pass http://127.0.0.1:8080;
3. index /;
4. proxy\_set\_header Host $host;
5. }
6. location ~ ^/(chapter23-app1)/ {
7. proxy\_pass http://127.0.0.1:9080;
8. index /;
9. proxy\_set\_header Host $host;
10. }
11. location ~ ^/(chapter23-app2)/ {
12. proxy\_pass http://127.0.0.1:10080;
13. index /;
14. proxy\_set\_header Host $host;
15. }

3、最后双击nginx.exe启动Nginx即可。

已经配置好的nginx请到shiro-example-chapter23-nginx模块下下周nginx-1.5.11.rar即可。

**2、安装依赖**

1、首先安装shiro-example-chapter23-core依赖，到shiro-example-chapter23-core模块下运行mvn install安装core模块。

2、接着到shiro-example-chapter23-client模块下运行mvn install安装客户端模块。

**3、启动Server模块**

到shiro-example-chapter23-server模块下运行mvn jetty:run启动该模块；使用<http://localhost:8080/chapter23-server/>即可访问，因为启动了nginx，那么可以直接访问<http://localhost/chapter23-server/>。

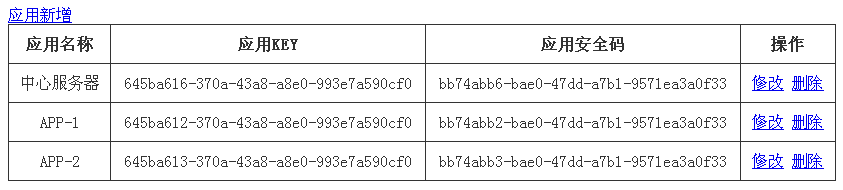
**4、启动App\*模块**

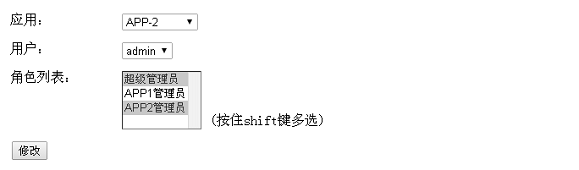
到shiro-example-chapter23-app1和shiro-example-chapter23-app2模块下分别运行mvn jetty:run启动该模块；使用<http://localhost:9080/chapter23-app1/>和<http://localhost:10080/chapter23-app2/>即可访问，因为启动了nginx，那么可以直接访问<http://localhost/chapter23-app1/>和<http://localhost/chapter23-app2/>。

**5、服务器端维护**

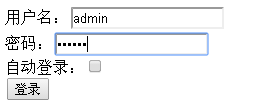
1、访问<http://localhost/chapter23-server/>；

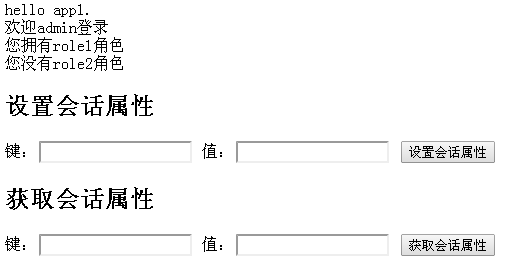
2、输入默认的用户名密码：admin/123456登录

3、应用管理，进行应用的CRUD，主要维护应用KEY（必须唯一）及应用安全码；客户端就可以使用应用KEY获取用户对应应用的权限了。  


4、授权管理，维护在哪个应用中用户的角色列表。这样客户端就可以根据应用KEY及用户名获取到对应的角色/权限字符串列表了。  
  


**6、App\*模块身份认证及授权**

1、在未登录情况下访问<http://localhost/chapter23-app1/hello>，看到下图：  
http://dl2.iteye.com/upload/attachment/0096/0340/092139cf-934c-3da9-af50-ab44313f54de.png  
 2、登录地址是<http://localhost/chapter23-app1/login?backUrl=/chapter23-app1>，即登录成功后重定向回<http://localhost/chapter23-app1>（这是个错误地址，为了测试登录成功后重定向地址），点击登录按钮后重定向到Server模块的登录界面：   


3、登录成功后，会重定向到相应的登录成功地址；接着访问<http://localhost/chapter23-app1/hello>，看到如下图：  


4、可以看到admin登录，及其是否拥有role1/role2角色；可以在server模块移除role1角色或添加role2角色看看页面变化；

5、可以在<http://localhost/chapter23-app1/hello>页面设置属性，如key=123；接着访问<http://localhost/chapter23-app2/attr?key=key>就可以看到刚才设置的属性，如下图：  


另外在app2，用户默认拥有role2角色，而没有role1角色。

到此整个测试就完成了，可以看出本示例实现了：会话的分布式及权限的集中管理。

**本示例缺点**

1、没有加缓存；

2、客户端每次获取会话/权限都需要通过客户端访问服务端；造成服务端单点和请求压力大；单点可以考虑使用集群来解决；请求压力大需要考虑配合缓存服务器（如Redis）来解决；即每次会话/权限获取时首先查询缓存中是否存在，如果有直接获取即可；否则再查服务端；降低请求压力；

3、会话的每次更新（比如设置属性/更新最后访问时间戳）都需要同步到服务端；也造成了请求压力过大；可以考虑在请求的最后只同步一次会话（需要对Shiro会话进行改造，通过如拦截器在执行完请求后完成同步，这样每次请求只同步一次）；

4、只能同域名才能使用，即会话ID是从同一个域名下获取，如果跨域请考虑使用CAS/OAuth2之实现。

所以实际应用时可能还是需要改造的，但大体思路是差不多的。

示例源代码：<https://github.com/zhangkaitao/shiro-example>；可加群 231889722 探讨Spring/Shiro技术。