[**第二十章 无状态Web应用集成——《跟我学Shiro》**](http://jinnianshilongnian.iteye.com/blog/2041909)

**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)

在一些环境中，可能需要把Web应用做成无状态的，即服务器端无状态，就是说服务器端不会存储像会话这种东西，而是每次请求时带上相应的用户名进行登录。如一些REST风格的API，如果不使用OAuth2协议，就可以使用如REST+HMAC认证进行访问。HMAC（Hash-based Message Authentication Code）：基于散列的消息认证码，使用一个密钥和一个消息作为输入，生成它们的消息摘要。注意该密钥只有客户端和服务端知道，其他第三方是不知道的。访问时使用该消息摘要进行传播，服务端然后对该消息摘要进行验证。如果只传递用户名+密码的消息摘要，一旦被别人捕获可能会重复使用该摘要进行认证。解决办法如：

1、每次客户端申请一个Token，然后使用该Token进行加密，而该Token是一次性的，即只能用一次；有点类似于OAuth2的Token机制，但是简单些；

2、客户端每次生成一个唯一的Token，然后使用该Token加密，这样服务器端记录下这些Token，如果之前用过就认为是非法请求。

为了简单，本文直接对请求的数据（即全部请求的参数）生成消息摘要，即无法篡改数据，但是可能被别人窃取而能多次调用。解决办法如上所示。

**服务器端**

对于服务器端，不生成会话，而是每次请求时带上用户身份进行认证。

**服务控制器**

**Java代码  [收藏代码](javascript:void())**

1. @RestController
2. **public** **class** ServiceController {
3. @RequestMapping("/hello")
4. **public** String hello1(String[] param1, String param2) {
5. **return** "hello" + param1[0] + param1[1] + param2;
6. }
7. }

当访问/hello服务时，需要传入param1、param2两个请求参数。

**加密工具类**

com.github.zhangkaitao.shiro.chapter20.codec.HmacSHA256Utils：

**Java代码  [收藏代码](javascript:void())**

1. //使用指定的密码对内容生成消息摘要（散列值）
2. **public** **static** String digest(String key, String content);
3. //使用指定的密码对整个Map的内容生成消息摘要（散列值）
4. **public** **static** String digest(String key, Map<String, ?> map)

对Map生成消息摘要主要用于对客户端/服务器端来回传递的参数生成消息摘要。

**Subject工厂**

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** StatelessDefaultSubjectFactory **extends** DefaultWebSubjectFactory {
2. **public** Subject createSubject(SubjectContext context) {
3. //不创建session
4. context.setSessionCreationEnabled(**false**);
5. **return** **super**.createSubject(context);
6. }
7. }

通过调用context.setSessionCreationEnabled(false)表示不创建会话；如果之后调用Subject.getSession()将抛出DisabledSessionException异常。

**StatelessAuthcFilter**

类似于FormAuthenticationFilter，但是根据当前请求上下文信息每次请求时都要登录的认证过滤器。

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** StatelessAuthcFilter **extends** AccessControlFilter {
2. **protected** **boolean** isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue) **throws** Exception {
3. **return** **false**;
4. }
5. **protected** **boolean** onAccessDenied(ServletRequest request, ServletResponse response) **throws** Exception {
6. //1、客户端生成的消息摘要
7. String clientDigest = request.getParameter(Constants.PARAM\_DIGEST);
8. //2、客户端传入的用户身份
9. String username = request.getParameter(Constants.PARAM\_USERNAME);
10. //3、客户端请求的参数列表
11. Map<String, String[]> params =
12. **new** HashMap<String, String[]>(request.getParameterMap());
13. params.remove(Constants.PARAM\_DIGEST);
14. //4、生成无状态Token
15. StatelessToken token = **new** StatelessToken(username, params, clientDigest);
16. **try** {
17. //5、委托给Realm进行登录
18. getSubject(request, response).login(token);
19. } **catch** (Exception e) {
20. e.printStackTrace();
21. onLoginFail(response); //6、登录失败
22. **return** **false**;
23. }
24. **return** **true**;
25. }
26. //登录失败时默认返回401状态码
27. **private** **void** onLoginFail(ServletResponse response) **throws** IOException {
28. HttpServletResponse httpResponse = (HttpServletResponse) response;
29. httpResponse.setStatus(HttpServletResponse.SC\_UNAUTHORIZED);
30. httpResponse.getWriter().write("login error");
31. }
32. }

获取客户端传入的用户名、请求参数、消息摘要，生成StatelessToken；然后交给相应的Realm进行认证。

**StatelessToken**

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** StatelessToken **implements** AuthenticationToken {
2. **private** String username;
3. **private** Map<String, ?> params;
4. **private** String clientDigest;
5. //省略部分代码
6. **public** Object getPrincipal() {  **return** username;}
7. **public** Object getCredentials() {  **return** clientDigest;}
8. }

用户身份即用户名；凭证即客户端传入的消息摘要。

**StatelessRealm**

用于认证的Realm。

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** StatelessRealm **extends** AuthorizingRealm {
2. **public** **boolean** supports(AuthenticationToken token) {
3. //仅支持StatelessToken类型的Token
4. **return** token **instanceof** StatelessToken;
5. }
6. **protected** AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
7. //根据用户名查找角色，请根据需求实现
8. String username = (String) principals.getPrimaryPrincipal();
9. SimpleAuthorizationInfo authorizationInfo =  **new** SimpleAuthorizationInfo();
10. authorizationInfo.addRole("admin");
11. **return** authorizationInfo;
12. }
13. **protected** AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) **throws** AuthenticationException {
14. StatelessToken statelessToken = (StatelessToken) token;
15. String username = statelessToken.getUsername();
16. String key = getKey(username);//根据用户名获取密钥（和客户端的一样）
17. //在服务器端生成客户端参数消息摘要
18. String serverDigest = HmacSHA256Utils.digest(key, statelessToken.getParams());
19. //然后进行客户端消息摘要和服务器端消息摘要的匹配
20. **return** **new** SimpleAuthenticationInfo(
21. username,
22. serverDigest,
23. getName());
24. }
26. **private** String getKey(String username) {//得到密钥，此处硬编码一个
27. **if**("admin".equals(username)) {
28. **return** "dadadswdewq2ewdwqdwadsadasd";
29. }
30. **return** **null**;
31. }
32. }

此处首先根据客户端传入的用户名获取相应的密钥，然后使用密钥对请求参数生成服务器端的消息摘要；然后与客户端的消息摘要进行匹配；如果匹配说明是合法客户端传入的；否则是非法的。这种方式是有漏洞的，一旦别人获取到该请求，可以重复请求；可以考虑之前介绍的解决方案。

**Spring配置——spring-config-shiro.xml**

**Java代码  [收藏代码](javascript:void())**

1. <!-- Realm实现 -->
2. <bean id="statelessRealm"
3. **class**="com.github.zhangkaitao.shiro.chapter20.realm.StatelessRealm">
4. <property name="cachingEnabled" value="false"/>
5. </bean>
6. <!-- Subject工厂 -->
7. <bean id="subjectFactory"
8. **class**="com.github.zhangkaitao.shiro.chapter20.mgt.StatelessDefaultSubjectFactory"/>
9. <!-- 会话管理器 -->
10. <bean id="sessionManager" **class**="org.apache.shiro.session.mgt.DefaultSessionManager">
11. <property name="sessionValidationSchedulerEnabled" value="false"/>
12. </bean>
13. <!-- 安全管理器 -->
14. <bean id="securityManager" **class**="org.apache.shiro.web.mgt.DefaultWebSecurityManager">
15. <property name="realm" ref="statelessRealm"/>
16. <property name="subjectDAO.sessionStorageEvaluator.sessionStorageEnabled"
17. value="false"/>
18. <property name="subjectFactory" ref="subjectFactory"/>
19. <property name="sessionManager" ref="sessionManager"/>
20. </bean>
21. <!-- 相当于调用SecurityUtils.setSecurityManager(securityManager) -->
22. <bean **class**="org.springframework.beans.factory.config.MethodInvokingFactoryBean">
23. <property name="staticMethod"
24. value="org.apache.shiro.SecurityUtils.setSecurityManager"/>
25. <property name="arguments" ref="securityManager"/>
26. </bean>

sessionManager通过sessionValidationSchedulerEnabled禁用掉会话调度器，因为我们禁用掉了会话，所以没必要再定期过期会话了。

**Java代码  [收藏代码](javascript:void())**

1. <bean id="statelessAuthcFilter"
2. **class**="com.github.zhangkaitao.shiro.chapter20.filter.StatelessAuthcFilter"/>

每次请求进行认证的拦截器。

**Java代码  [收藏代码](javascript:void())**

1. <!-- Shiro的Web过滤器 -->
2. <bean id="shiroFilter" **class**="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
3. <property name="securityManager" ref="securityManager"/>
4. <property name="filters">
5. <util:map>
6. <entry key="statelessAuthc" value-ref="statelessAuthcFilter"/>
7. </util:map>
8. </property>
9. <property name="filterChainDefinitions">
10. <value>
11. /\*\*=statelessAuthc
12. </value>
13. </property>
14. </bean>

所有请求都将走statelessAuthc拦截器进行认证。

其他配置请参考源代码。

SpringMVC学习请参考：

5分钟构建spring web mvc REST风格HelloWorld

<http://jinnianshilongnian.iteye.com/blog/1996071>

跟我学SpringMVC

<http://www.iteye.com/blogs/subjects/kaitao-springmvc>

**客户端**

此处使用SpringMVC提供的RestTemplate进行测试。请参考如下文章进行学习：

Spring MVC测试框架详解——客户端测试

<http://jinnianshilongnian.iteye.com/blog/2007180>

Spring MVC测试框架详解——服务端测试

<http://jinnianshilongnian.iteye.com/blog/2004660>

此处为了方便，使用内嵌jetty服务器启动服务端：

**Java代码  [收藏代码](javascript:void())**

1. **public** **class** ClientTest {
2. **private** **static** Server server;
3. **private** RestTemplate restTemplate = **new** RestTemplate();
4. @BeforeClass
5. **public** **static** **void** beforeClass() **throws** Exception {
6. //创建一个server
7. server = **new** Server(8080);
8. WebAppContext context = **new** WebAppContext();
9. String webapp = "shiro-example-chapter20/src/main/webapp";
10. context.setDescriptor(webapp + "/WEB-INF/web.xml");  //指定web.xml配置文件
11. context.setResourceBase(webapp);  //指定webapp目录
12. context.setContextPath("/");
13. context.setParentLoaderPriority(**true**);
14. server.setHandler(context);
15. server.start();
16. }
17. @AfterClass
18. **public** **static** **void** afterClass() **throws** Exception {
19. server.stop(); //当测试结束时停止服务器
20. }
21. }

在整个测试开始之前开启服务器，整个测试结束时关闭服务器。

**测试成功情况**

**Java代码  [收藏代码](javascript:void())**

1. @Test
2. **public** **void** testServiceHelloSuccess() {
3. String username = "admin";
4. String param11 = "param11";
5. String param12 = "param12";
6. String param2 = "param2";
7. String key = "dadadswdewq2ewdwqdwadsadasd";
8. MultiValueMap<String, String> params = **new** LinkedMultiValueMap<String, String>();
9. params.add(Constants.PARAM\_USERNAME, username);
10. params.add("param1", param11);
11. params.add("param1", param12);
12. params.add("param2", param2);
13. params.add(Constants.PARAM\_DIGEST, HmacSHA256Utils.digest(key, params));
14. String url = UriComponentsBuilder
15. .fromHttpUrl("http://localhost:8080/hello")
16. .queryParams(params).build().toUriString();
17. ResponseEntity responseEntity = restTemplate.getForEntity(url, String.**class**);
18. Assert.assertEquals("hello" + param11 + param12 + param2, responseEntity.getBody());
19. }

对请求参数生成消息摘要后带到参数中传递给服务器端，服务器端验证通过后访问相应服务，然后返回数据。

**测试失败情况**

**Java代码  [收藏代码](javascript:void())**

1. @Test
2. **public** **void** testServiceHelloFail() {
3. String username = "admin";
4. String param11 = "param11";
5. String param12 = "param12";
6. String param2 = "param2";
7. String key = "dadadswdewq2ewdwqdwadsadasd";
8. MultiValueMap<String, String> params = **new** LinkedMultiValueMap<String, String>();
9. params.add(Constants.PARAM\_USERNAME, username);
10. params.add("param1", param11);
11. params.add("param1", param12);
12. params.add("param2", param2);
13. params.add(Constants.PARAM\_DIGEST, HmacSHA256Utils.digest(key, params));
14. params.set("param2", param2 + "1");
16. String url = UriComponentsBuilder
17. .fromHttpUrl("http://localhost:8080/hello")
18. .queryParams(params).build().toUriString();
19. **try** {
20. ResponseEntity responseEntity = restTemplate.getForEntity(url, String.**class**);
21. } **catch** (HttpClientErrorException e) {
22. Assert.assertEquals(HttpStatus.UNAUTHORIZED, e.getStatusCode());
23. Assert.assertEquals("login error", e.getResponseBodyAsString());
24. }
25. }

在生成请求参数消息摘要后，篡改了参数内容，服务器端接收后进行重新生成消息摘要发现不一样，报401错误状态码。

到此，整个测试完成了，需要注意的是，为了安全性，请考虑本文开始介绍的相应解决方案。

**SpringMVC相关知识请参考**

5分钟构建spring web mvc REST风格HelloWorld

<http://jinnianshilongnian.iteye.com/blog/1996071>

跟我学SpringMVC

<http://www.iteye.com/blogs/subjects/kaitao-springmvc>

Spring MVC测试框架详解——客户端测试

<http://jinnianshilongnian.iteye.com/blog/2007180>

Spring MVC测试框架详解——服务端测试

<http://jinnianshilongnian.iteye.com/blog/2004660>

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