

首页 资讯 精华 论坛 问答 博客 专栏 群组 更多 ▼
您还未登录！ 登录 注册

开涛的博客

- 博客
- 微博
- 相册
- 收藏
- 留言
- 关于我

第十七章 OAuth2集成——《跟我学Shiro》

博客分类：

- [跟我学Shiro](#)

[跟我学Shiro](#)

目录贴：[跟我学Shiro目录贴](#)

目前很多开放平台如新浪微博开放平台都在使用提供开放API接口供开发者使用，随之带来了第三方应用要到开放平台进行授权的问题，OAuth就是干这个的，OAuth2是OAuth协议的下一个版本，相比OAuth1，OAuth2整个授权流程更简单安全了，但不兼容OAuth1，具体可以到OAuth2官网<http://oauth.net/2/>查看，OAuth2协议规范可以参考<http://tools.ietf.org/html/rfc6749>。目前有好多参考实现供选择，可以到其官网查看下载。

本文使用[Apache Oltu](#)，其之前的名字叫Apache Amber，是Java版的参考实现。使用文档可参考<https://cwiki.apache.org/confluence/display/OLTU/Documentation>。

OAuth角色

资源所有者 (resource owner)：能授权访问受保护资源的一个实体，可以是个人，那我们称之为最终用户；如新浪微博用户zhangsan；

资源服务器 (resource server)：存储受保护资源，客户端通过access token请求资源，资源服务器响应受保护资源给客户端；存储着用户zhangsan的微博等信息。

授权服务器 (authorization server)：成功验证资源所有者并获取授权之后，授权服务器颁发授权令牌 (Access Token) 给客户端。

客户端 (client)：如新浪微博客户端weico、微格等第三方应用，也可以是它自己的官方应用；其本身不存储资源，而是资源所有者授权通过后，使用它的授权 (授权令牌) 访问受保护资源，然后客户端把相应的数据展示出来/提交到服务器。“客户端”术语不代表任何特定实现 (如应用运行在一台服务器、桌面、手机或其他设备)。

OAuth2协议流程



1、客户端从资源所有者那请求授权。授权请求可以直接发给资源所有者，或间接的通过授权服务器这种中介，后者更可取。

2、客户端收到一个授权许可，代表资源服务器提供的授权。

3、客户端使用它自己的私有证书及授权许可到授权服务器验证。

4、如果验证成功，则下发一个访问令牌。

5、客户端使用访问令牌向资源服务器请求受保护资源。

6、资源服务器会验证访问令牌的有效性，如果成功则下发受保护资源。

更多流程的解释请参考OAuth2的协议规范<http://tools.ietf.org/html/rfc6749>。

服务器端

本文把授权服务器和资源服务器整合在一起实现。

POM依赖

此处我们使用apache oltu oauth2服务端实现，需要引入authzserver（授权服务器依赖）和resourceserver（资源服务器依赖）。

Java代码

```
1. <dependency>
2.   <groupId>org.apache.oltu.oauth2</groupId>
3.   <artifactId>org.apache.oltu.oauth2.authzserver</artifactId>
4.   <version>0.31</version>
5. </dependency>
6. <dependency>
7.   <groupId>org.apache.oltu.oauth2</groupId>
8.   <artifactId>org.apache.oltu.oauth2.resourceserver</artifactId>
9.   <version>0.31</version>
10.</dependency>
```

其他的请参考pom.xml。

数据字典

用户(oauth2_user)

名称	类型	长度	描述
id	bigint	10	编号 主键
username	varchar	100	用户名
password	varchar	100	密码
salt	varchar	50	盐

客户端(oauth2_client)

名称	类型	长度	描述
id	bigint	10	编号 主键
client_name	varchar	100	客户端名称
client_id	varchar	100	客户端id
client_secret	varchar	100	客户端安全key

用户表存储着认证/资源服务器的用户信息，即资源拥有者；比如用户名/密码；客户端表存储客户端的的客户端id及客户端安全key；在进行授权时使用。

表及数据SQL

具体请参考

sql/ shiro-schema.sql （表结构）

sql/ shiro-data.sql （初始数据）

默认用户名/密码是admin/123456。

实体

具体请参考com.github.zhangkaitao.shiro.chapter17.entity包下的实体，此处就不列举了。

DAO

具体请参考com.github.zhangkaitao.shiro.chapter17.dao包下的DAO接口及实现。

Service

具体请参考com.github.zhangkaitao.shiro.chapter17.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 ☆

```
1. public interface UserService {
2.     public User createUser(User user); // 创建用户
3.     public User updateUser(User user); // 更新用户
4.     public void deleteUser(Long userId); // 删除用户
5.     public void changePassword(Long userId, String newPassword); // 修改密码
6.     User findOne(Long userId); // 根据id查找用户
7.     List<User> findAll(); // 得到所有用户
8.     public User findByUsername(String username); // 根据用户名查找用户
9. }
```

Java代码 ☆

```
1. public interface ClientService {
2.     public Client createClient(Client client); // 创建客户端
3.     public Client updateClient(Client client); // 更新客户端
4.     public void deleteClient(Long clientId); // 删除客户端
5.     Client findOne(Long clientId); // 根据id查找客户端
6.     List<Client> findAll(); // 查找所有
7.     Client findByClientId(String clientId); // 根据客户端id查找客户端
8.     Client findByClientSecret(String clientSecret); // 根据客户端安全KEY查找客户端
9. }
```

Java代码 ☆

```
1. public interface OAuthService {
2.     public void addAuthCode(String authCode, String username); // 添加 auth code
3.     public void addAccessToken(String accessToken, String username); // 添加 access token
4.     boolean checkAuthCode(String authCode); // 验证auth code是否有效
5.     boolean checkAccessToken(String accessToken); // 验证access token是否有效
6.     String getUsernameByAuthCode(String authCode); // 根据auth code获取用户名
7.     String getUsernameByAccessToken(String accessToken); // 根据access token获取用户名
8.     long getExpiresIn(); // auth code / access token 过期时间
9.     public boolean checkClientId(String clientId); // 检查客户端id是否存在
10.    public boolean checkClientSecret(String clientSecret); // 坚持客户端安全KEY是否存在
11. }
```

此处通过OAuthService实现进行auth code和access token的维护。

后端数据维护控制器

具体请参考com.github.zhangkaitao.shiro.chapter17.web.controller包下的IndexController、LoginController、UserController和ClientController，其用于维护后端的数据，如用户及客户端数据；即相当于后台管理。

授权控制器AuthorizeController

Java代码 ☆

```
1. @Controller
2. public class AuthorizeController {
3.     @Autowired
4.     private OAuthService oAuthService;
5.     @Autowired
6.     private ClientService clientService;
7.     @RequestMapping("/authorize")
8.     public Object authorize(Model model, HttpServletRequest request)
9.         throws URISyntaxException, OAuthSystemException {
10.    try {
11.        //构建OAuth 授权请求
12.        OAuthAuthzRequest oauthRequest = new OAuthAuthzRequest(request);
13.        //检查传入的客户端id是否正确
14.        if (!oAuthService.checkClientId(oauthRequest.getClientId())) {
15.            OAuthResponse response = OAuthASResponse
16.                .errorResponse(HttpServletResponse.SC_BAD_REQUEST)
17.                .setError(OAuthError.TokenResponse.INVALID_CLIENT)
18.                .setErrorDescription(Constants.INVALID_CLIENT_DESCRIPTION)
19.                .buildJSONMessage();
20.            return new ResponseBody(
21.                response.getBody(), HttpStatus.valueOf(response.getResponseStatus()));
22.        }
23.
24.        Subject subject = SecurityUtils.getSubject();
25.        //如果用户没有登录，跳转到登陆页面
26.        if (!subject.isAuthenticated()) {
```

```
27.     if(!login(subject, request)) { //登录失败时跳转到登陆页面
28.         model.addAttribute("client",
29.             clientService.findByClientId(oauthRequest.getClientId()));
30.         return "oauth2login";
31.     }
32. }
33.
34. String username = (String)subject.getPrincipal();
35. //生成授权码
36. String authorizationCode = null;
37. //responseType目前仅支持CODE，另外还有TOKEN
38. String responseType = oauthRequest.getParam(OAuth.OAUTH_RESPONSE_TYPE);
39. if (responseType.equals(ResponseType.CODE.toString())) {
40.     OAuthIssuerImpl oauthIssuerImpl = new OAuthIssuerImpl(new MD5Generator());
41.     authorizationCode = oauthIssuerImpl.authorizationCode();
42.     oauthService.addAuthCode(authorizationCode, username);
43. }
44. //进行OAuth响应构建
45. OAuthASResponse.OAuthAuthorizationResponseBuilder builder =
46.     OAuthASResponse.authorizationResponse(request,
47.         HttpServletResponse.SC_FOUND);
48. //设置授权码
49. builder.setCode(authorizationCode);
50. //得到到客户端重定向地址
51. String redirectURI = oauthRequest.getParam(OAuth.OAUTH_REDIRECT_URI);
52.
53. //构建响应
54. final OAuthResponse response = builder.location(redirectURI).buildQueryMessage();
55. //根据OAuthResponse返回ResponseEntity响应
56. HttpHeaders headers = new HttpHeaders();
57. headers.setLocation(new URI(response.getLocationUri()));
58. return new ResponseEntity(headers, HttpStatus.valueOf(response.getResponseStatus()));
59. } catch (OAuthProblemException e) {
60.     //出错处理
61.     String redirectUri = e.getRedirectUri();
62.     if (OAuthUtils.isEmpty(redirectUri)) {
63.         //告诉客户端没有传入redirectUri直接报错
64.         return new ResponseEntity(
65.             "OAuth callback url needs to be provided by client!!!", HttpStatus.NOT_FOUND);
66.     }
67.     //返回错误消息（如?error=）
68.     final OAuthResponse response =
69.         OAuthASResponse.errorResponse(HttpServletResponse.SC_FOUND)
70.             .error(e).location(redirectUri).buildQueryMessage();
71.     HttpHeaders headers = new HttpHeaders();
72.     headers.setLocation(new URI(response.getLocationUri()));
73.     return new ResponseEntity(headers, HttpStatus.valueOf(response.getResponseStatus()));
74. }
75. }
76.
77. private boolean login(Subject subject, HttpServletRequest request) {
78.     if("get".equalsIgnoreCase(request.getMethod())) {
79.         return false;
80.     }
81.     String username = request.getParameter("username");
82.     String password = request.getParameter("password");
83.
84.     if(StringUtils.isEmpty(username) || StringUtils.isEmpty(password)) {
85.         return false;
86.     }
87.
88.     UsernamePasswordToken token = new UsernamePasswordToken(username, password);
89.     try {
90.         subject.login(token);
91.         return true;
92.     } catch (Exception e) {
93.         request.setAttribute("error", "登录失败:" + e.getClass().getName());
94.         return false;
95.     }
96. }
97. }
```

如上代码的作用：

1、首先通过如<http://localhost:8080/chapter17-server/authorize>

?client_id=c1ebe466-1cdc-4bd3-ab69-77c3561b9dee&response_type=code&redirect_uri=http://localhost:9080/chapter17-client/oauth2-login访问授权页面；

2、该控制器首先检查clientId是否正确；如果错误将返回相应的错误信息；

3、然后判断用户是否登录了，如果没有登录首先到登录页面登录；

4、登录成功后生成相应的auth code即授权码，然后重定向到客户端地址，如<http://localhost:9080/chapter17-client/oauth2-login?code=52b1832f5dff68122f4f00ae995da0ed>；在重定向到的地址中会带上code参数（授权码），接着客户端可以根据授权码去换取access token。

访问令牌控制器AccessTokenController

Java代码 ☆

```
1. @RestController
2. public class AccessTokenController {
3.     @Autowired
4.     private OAuthService oAuthService;
5.     @Autowired
6.     private UserService userService;
7.     @RequestMapping("/accessToken")
8.     public HttpEntity token(HttpServletRequest request)
9.         throws URISyntaxException, OAuthSystemException {
10.    try {
11.        //构建OAuth请求
12.        OAuthTokenRequest oauthRequest = new OAuthTokenRequest(request);
13.
14.        //检查提交的客户端id是否正确
15.        if (!oAuthService.checkClientId(oauthRequest.getClientId())) {
16.            OAuthResponse response = OAuthASResponse
17.                .errorResponse(HttpServletResponse.SC_BAD_REQUEST)
18.                .setError(OAuthError.TokenResponse.INVALID_CLIENT)
19.                .setErrorDescription(Constants.INVALID_CLIENT_DESCRIPTION)
20.                .buildJSONMessage();
21.            return new ResponseEntity(
22.                response.getBody(), HttpStatus.valueOf(response.getResponseStatus()));
23.        }
24.
25.        // 检查客户端安全KEY是否正确
26.        if (!oAuthService.checkClientSecret(oauthRequest.getClientSecret())) {
27.            OAuthResponse response = OAuthASResponse
28.                .errorResponse(HttpServletResponse.SC_UNAUTHORIZED)
29.                .setError(OAuthError.TokenResponse.UNAUTHORIZED_CLIENT)
30.                .setErrorDescription(Constants.INVALID_CLIENT_DESCRIPTION)
31.                .buildJSONMessage();
32.            return new ResponseEntity(
33.                response.getBody(), HttpStatus.valueOf(response.getResponseStatus()));
34.        }
35.
36.        String authCode = oauthRequest.getParam(OAuth.OAUTH_CODE);
37.        // 检查验证类型，此处只检查AUTHORIZATION_CODE类型，其他的还有PASSWORD或REFRESH_TOKEN
38.        if (oauthRequest.getParam(OAuth.OAUTH_GRANT_TYPE).equals(
39.            GrantType.AUTHORIZATION_CODE.toString())) {
40.            if (!oAuthService.checkAuthCode(authCode)) {
41.                OAuthResponse response = OAuthASResponse
42.                    .errorResponse(HttpServletResponse.SC_BAD_REQUEST)
43.                    .setError(OAuthError.TokenResponse.INVALID_GRANT)
44.                    .setErrorDescription("错误的授权码")
45.                    .buildJSONMessage();
46.                return new ResponseEntity(
47.                    response.getBody(), HttpStatus.valueOf(response.getResponseStatus()));
48.            }
49.        }
50.
51.        //生成Access Token
52.        OAuthIssuer oauthIssuerImpl = new OAuthIssuerImpl(new MD5Generator());
53.        final String accessToken = oauthIssuerImpl.accessToken();
54.        oAuthService.addAccessToken(accessToken,
55.            oAuthService.getUsernameByAuthCode(authCode));
56.
57.        //生成OAuth响应
58.        OAuthResponse response = OAuthASResponse
59.            .tokenResponse(HttpServletResponse.SC_OK)
60.            .setAccessToken(accessToken)
61.            .setExpiresIn(String.valueOf(oAuthService.getExpireIn()))
62.            .buildJSONMessage();
63.
64.        //根据OAuthResponse生成ResponseEntity
65.        return new ResponseEntity(
66.            response.getBody(), HttpStatus.valueOf(response.getResponseStatus()));
67.    } catch (OAuthProblemException e) {
68.        //构建错误响应
69.        OAuthResponse res = OAuthASResponse
70.            .errorResponse(HttpServletResponse.SC_BAD_REQUEST).error(e)
71.            .buildJSONMessage();
72.        return new ResponseEntity(res.getBody(), HttpStatus.valueOf(res.getResponseStatus()));
73.    }
74. }
75. }
```

如上代码的作用：

- 1、首先通过如<http://localhost:8080/chapter17-server/accessToken>, POST提交如下数据: client_id= c1ebe466-1cdc-4bd3-ab69-77c3561b9dee& client_secret= d8346ea2-6017-43ed-ad68-19c0f971738b&grant_type=authorization_code&code=828beda907066d058584f37bcfd597b6&redirect_uri=http://localhost:9080/chapter17-client/oauth2-login访问;
- 2、该控制器会验证client_id、client_secret、auth code的正确性, 如果错误会返回相应的错误;
- 3、如果验证通过会生成并返回相应的访问令牌access token。

资源控制器UserInfoController

Java代码 ☆

```
1. @RestController
2. public class UserInfoController {
3.     @Autowired
4.     private OAuthService oAuthService;
5.
6.     @RequestMapping("/userInfo")
7.     public HttpEntity userInfo(HttpServletRequest request) throws OAuthSystemException {
8.         try {
9.             //构建OAuth资源请求
10.            OAuthAccessResourceRequest oauthRequest =
11.                new OAuthAccessResourceRequest(request, ParameterStyle.QUERY);
12.            //获取Access Token
13.            String accessToken = oauthRequest.getAccessToken();
14.
15.            //验证Access Token
16.            if (!oAuthService.checkAccessToken(accessToken)) {
17.                // 如果不存在/过期了, 返回未验证错误, 需重新验证
18.                OAuthResponse oauthResponse = OAuthRSResponse
19.                    .errorResponse(HttpServletResponse.SC_UNAUTHORIZED)
20.                    .setRealm(Constants.RESOURCE_SERVER_NAME)
21.                    .setError(OAuthError.ResourceResponse.INVALID_TOKEN)
22.                    .buildHeaderMessage();
23.
24.                HttpHeaders headers = new HttpHeaders();
25.                headers.add(OAuth.HeaderType.WWW_AUTHENTICATE,
26.                    oauthResponse.getHeader(OAuth.HeaderType.WWW_AUTHENTICATE));
27.                return new ResponseEntity(headers, HttpStatus.UNAUTHORIZED);
28.            }
29.            //返回用户名
30.            String username = oAuthService.getUsernameByAccessToken(accessToken);
31.            return new ResponseEntity(username, HttpStatus.OK);
32.        } catch (OAuthProblemException e) {
33.            //检查是否设置了错误码
34.            String errorCode = e.getError();
35.            if (OAuthUtils.isEmpty(errorCode)) {
36.                OAuthResponse oauthResponse = OAuthRSResponse
37.                    .errorResponse(HttpServletResponse.SC_UNAUTHORIZED)
38.                    .setRealm(Constants.RESOURCE_SERVER_NAME)
39.                    .buildHeaderMessage();
40.
41.                HttpHeaders headers = new HttpHeaders();
42.                headers.add(OAuth.HeaderType.WWW_AUTHENTICATE,
43.                    oauthResponse.getHeader(OAuth.HeaderType.WWW_AUTHENTICATE));
44.                return new ResponseEntity(headers, HttpStatus.UNAUTHORIZED);
45.            }
46.
47.            OAuthResponse oauthResponse = OAuthRSResponse
48.                .errorResponse(HttpServletResponse.SC_UNAUTHORIZED)
49.                .setRealm(Constants.RESOURCE_SERVER_NAME)
50.                .setError(e.getError())
51.                .setErrorDescription(e.getDescription())
52.                .setErrorUri(e.getUri())
53.                .buildHeaderMessage();
54.
55.            HttpHeaders headers = new HttpHeaders();
56.            headers.add(OAuth.HeaderType.WWW_AUTHENTICATE,
57.                oauthResponse.getHeader(OAuth.HeaderType.WWW_AUTHENTICATE));
58.            return new ResponseEntity(HttpStatus.BAD_REQUEST);
59.        }
60.    }
61. }
```

如上代码的作用:

- 1、首先通过如http://localhost:8080/chapter17-server/userInfo? access_token=828beda907066d058584f37bcfd597b6进行访问;
- 2、该控制器会验证access token的有效性: 如果无效了将返回相应的错误, 客户端再重新进行授权;
- 3、如果有效, 则返回当前登录用户的用户名。

Spring配置文件

具体请参考resources/spring*.xml，此处只列举spring-config-shiro.xml中的shiroFilter的filterChainDefinitions属性：

Java代码 ☆

```
1. <property name="filterChainDefinitions">
2.   <value>
3.     / = anon
4.     /login = authc
5.     /logout = logout
6.
7.     /authorize=anon
8.     /accessToken=anon
9.     /userInfo=anon
10.
11.    /** = user
12.   </value>
13. </property>
```

对于oauth2的几个地址/authorize、/accessToken、/userInfo都是匿名可访问的。

其他源码请直接下载文档查看。

服务器维护

访问localhost:8080/chapter17-server/，登录后进行客户端管理和用户管理。

客户端管理就是进行客户端的注册，如新浪微博的第三方应用就需要到新浪微博开发平台进行注册；用户管理就是进行如新浪微博用户的管理。

对于授权服务和资源服务的实现可以参考新浪微博开发平台的实现：

<http://open.weibo.com/wiki/授权机制说明>

<http://open.weibo.com/wiki/微博API>

客户端

客户端流程：如果需要登录首先跳到oauth2服务端进行登录授权，成功后服务端返回auth code，然后客户端使用auth code去服务器端换取access token，最好根据access token获取用户信息进行客户端的登录绑定。这个可以参照如很多网站的新浪微博登录功能，或其他的第三方帐号登录功能。

POM依赖

此处我们使用apache oltu oauth2客户端实现。

Java代码 ☆

```
1. <dependency>
2.   <groupId>org.apache.oltu.oauth2</groupId>
3.   <artifactId>org.apache.oltu.oauth2.client</artifactId>
4.   <version>0.31</version>
5. </dependency>
```

其他的请参考pom.xml。

OAuth2Token

类似于UsernamePasswordToken和CasToken；用于存储oauth2服务端返回的auth code。

Java代码 ☆

```
1. public class OAuth2Token implements AuthenticationToken {
2.   private String authCode;
3.   private String principal;
4.   public OAuth2Token(String authCode) {
5.     this.authCode = authCode;
6.   }
7.   //省略getter/setter
8. }
```

OAuth2AuthenticationFilter

该filter的作用类似于FormAuthenticationFilter用于oauth2客户端的身份验证控制；如果当前用户还没有身份验证，首先会判断url中是否有code（服务端返回的auth code），如果没有则定向到服务端进行登录并授权，然后返回auth code；接着OAuth2AuthenticationFilter会用auth code创建OAuth2Token，然后提交给Subject.login进行登录；接着OAuth2Realm会根据OAuth2Token进行相应的登录逻辑。

Java代码 ☆

```
1. public class OAuth2AuthenticationFilter extends AuthenticatingFilter {
2.     //oauth2 authc code参数名
3.     private String authcCodeParam = "code";
4.     //客户端id
5.     private String clientId;
6.     //服务器端登录成功/失败后重定向到的客户端地址
7.     private String redirectUrl;
8.     //oauth2服务器响应类型
9.     private String responseType = "code";
10.    private String failureUrl;
11.    //省略setter
12.    protected AuthenticationToken createToken(ServletRequest request, ServletResponse response) throws Exception {
13.        HttpServletRequest httpRequest = (HttpServletRequest) request;
14.        String code = httpRequest.getParameter(authcCodeParam);
15.        return new OAuth2Token(code);
16.    }
17.    protected boolean isAccessAllowed(ServletRequest request, ServletResponse response, Object mappedValue) {
18.        return false;
19.    }
20.    protected boolean onAccessDenied(ServletRequest request, ServletResponse response) throws Exception {
21.        String error = request.getParameter("error");
22.        String errorDescription = request.getParameter("error_description");
23.        if(!StringUtils.isEmpty(error)) { //如果服务端返回了错误
24.            WebUtils.issueRedirect(request, response, failureUrl + "?error=" + error + "error_description=" + errorDescription);
25.            return false;
26.        }
27.        Subject subject = getSubject(request, response);
28.        if(!subject.isAuthenticated()) {
29.            if(StringUtils.isEmpty(request.getParameter(authcCodeParam))) {
30.                //如果用户没有身份验证，且没有auth code，则重定向到服务端授权
31.                saveRequestAndRedirectToLogin(request, response);
32.                return false;
33.            }
34.        }
35.        //执行父类里的登录逻辑，调用Subject.login登录
36.        return executeLogin(request, response);
37.    }
38.
39.    //登录成功后的回调方法 重定向到成功页面
40.    protected boolean onLoginSuccess(AuthenticationToken token, Subject subject, ServletRequest request, ServletResponse response) throws Exception {
41.        issueSuccessRedirect(request, response);
42.        return false;
43.    }
44.
45.    //登录失败后的回调
46.    protected boolean onLoginFailure(AuthenticationToken token, AuthenticationException ae, ServletRequest request,
47.        ServletResponse response) {
48.        Subject subject = getSubject(request, response);
49.        if (subject.isAuthenticated() || subject.isRemembered()) {
50.            try { //如果身份验证成功了 则也重定向到成功页面
51.                issueSuccessRedirect(request, response);
52.            } catch (Exception e) {
53.                e.printStackTrace();
54.            }
55.        } else {
56.            try { //登录失败时重定向到失败页面
57.                WebUtils.issueRedirect(request, response, failureUrl);
58.            } catch (IOException e) {
59.                e.printStackTrace();
60.            }
61.        }
62.        return false;
63.    }
64.}
```

该拦截器的作用：

- 1、首先判断有没有服务端返回的error参数，如果有则直接重定向到失败页面；
- 2、接着如果用户还没有身份验证，判断是否有auth code参数（即是不是服务端授权之后返回的），如果没有则重定向到服务端进行授权；
- 3、否则调用executeLogin进行登录，通过auth code创建OAuth2Token提交给Subject进行登录；
- 4、登录成功将回调onLoginSuccess方法重定向到成功页面；
- 5、登录失败则回调onLoginFailure重定向到失败页面。

OAuth2Realm

Java代码 ☆

```
1. public class OAuth2Realm extends AuthorizingRealm {
2.     private String clientId;
```



```

3. private String clientSecret;
4. private String accessTokenUrl;
5. private String userInfoUrl;
6. private String redirectUrl;
7. //省略setter
8. public boolean supports(AuthenticationToken token) {
9.     return token instanceof OAuth2Token; //表示此Realm只支持OAuth2Token类型
10. }
11. protected AuthorizationInfo doGetAuthorizationInfo(PrincipalCollection principals) {
12.     SimpleAuthorizationInfo authorizationInfo = new SimpleAuthorizationInfo();
13.     return authorizationInfo;
14. }
15. protected AuthenticationInfo doGetAuthenticationInfo(AuthenticationToken token) throws AuthenticationException {
16.     OAuth2Token oAuth2Token = (OAuth2Token) token;
17.     String code = oAuth2Token.getAuthCode(); //获取 auth code
18.     String username = extractUsername(code); //提取用户名
19.     SimpleAuthenticationInfo authenticationInfo =
20.         new SimpleAuthenticationInfo(username, code, getName());
21.     return authenticationInfo;
22. }
23. private String extractUsername(String code) {
24.     try {
25.         OAuthClient oAuthClient = new OAuthClient(new URLConnectionClient());
26.         OAuthClientRequest accessTokenRequest = OAuthClientRequest
27.             .tokenLocation(accessTokenUrl)
28.             .setGrantType(GrantType.AUTHORIZATION_CODE)
29.             .setClientId(clientId).setClientSecret(clientSecret)
30.             .setCode(code).setRedirectURI(redirectUrl)
31.             .buildQueryMessage();
32.         //获取access token
33.         OAuthAccessTokenResponse oAuthResponse =
34.             oAuthClient.accessToken(accessTokenRequest, OAuth.HttpMethod.POST);
35.         String accessToken = oAuthResponse.getAccessToken();
36.         Long expiresIn = oAuthResponse.getExpiresIn();
37.         //获取user info
38.         OAuthClientRequest userInfoRequest =
39.             new OAuthBearerClientRequest(userInfoUrl)
40.                 .setAccessToken(accessToken).buildQueryMessage();
41.         OAuthResourceResponse resourceResponse = oAuthClient.resource(
42.             userInfoRequest, OAuth.HttpMethod.GET, OAuthResourceResponse.class);
43.         String username = resourceResponse.getBody();
44.         return username;
45.     } catch (Exception e) {
46.         throw new OAuth2AuthenticationException(e);
47.     }
48. }
49. }

```

此Realm首先只支持OAuth2Token类型的Token；然后通过传入的auth code去换取access token；再根据access token去获取用户信息（用户名），然后根据此信息创建AuthenticationInfo；如果需要AuthorizationInfo信息，可以根据此处获取的用户名再根据自己的业务规则去获取。

Spring shiro配置 (spring-config-shiro.xml)

Java代码 ☆

```

1. <bean id="oAuth2Realm"
2.     class="com.github.zhangkaitao.shiro.chapter18.oauth2.OAuth2Realm">
3.     <property name="cachingEnabled" value="true"/>
4.     <property name="authenticationCachingEnabled" value="true"/>
5.     <property name="authenticationCacheName" value="authenticationCache"/>
6.     <property name="authorizationCachingEnabled" value="true"/>
7.     <property name="authorizationCacheName" value="authorizationCache"/>
8.     <property name="clientId" value="c1ebe466-1cdc-4bd3-ab69-77c3561b9dee"/>
9.     <property name="clientSecret" value="d8346ea2-6017-43ed-ad68-19c0f971738b"/>
10.    <property name="accessTokenUrl"
11.        value="http://localhost:8080/chapter17-server/accessToken"/>
12.    <property name="userInfoUrl" value="http://localhost:8080/chapter17-server/userInfo"/>
13.    <property name="redirectUrl" value="http://localhost:9080/chapter17-client/oauth2-login"/>
14. </bean>

```

此OAuth2Realm需要配置在服务端申请的clientId和clientSecret；及用于根据auth code换取access token的accessTokenUrl地址；及用于根据access token换取用户信息（受保护资源）的userInfoUrl地址。

Java代码 ☆

```

1. <bean id="oAuth2AuthenticationFilter"
2.     class="com.github.zhangkaitao.shiro.chapter18.oauth2.OAuth2AuthenticationFilter">
3.     <property name="authcCodeParam" value="code"/>
4.     <property name="failureUrl" value="/oauth2Failure.jsp"/>
5. </bean>

```

此OAuth2AuthenticationFilter用于拦截服务端重定向回来的auth code。

Java代码 ☆

```
1. <bean id="shiroFilter" class="org.apache.shiro.spring.web.ShiroFilterFactoryBean">
2.   <property name="securityManager" ref="securityManager"/>
3.   <property name="loginUrl" value="http://localhost:8080/chapter17-server/authorize?client_id=c1ebe466-1cdc-4bd3-ab69-77c3561b9dee&
   &response_type=code&redirect_uri=http://localhost:9080/chapter17-client/oauth2-login"/>
4.   <property name="successUrl" value="/" />
5.   <property name="filters">
6.     <util:map>
7.       <entry key="oauth2Authc" value-ref="oAuth2AuthenticationFilter"/>
8.     </util:map>
9.   </property>
10.  <property name="filterChainDefinitions">
11.    <value>
12.      / = anon
13.      /oauth2Failure.jsp = anon
14.      /oauth2-login = oauth2Authc
15.      /logout = logout
16.      /** = user
17.    </value>
18.  </property>
19.</bean>
```

此处设置loginUrl为<http://localhost:8080/chapter17-server/authorize>

?client_id=c1ebe466-1cdc-4bd3-ab69-77c3561b9dee&response_type=code&redirect_uri=<http://localhost:9080/chapter17-client/oauth2-login>"; 其会自动设置到所有的AccessControlFilter，如oAuth2AuthenticationFilter；另外/oauth2-login = oauth2Authc表示/oauth2-login地址使用oauth2Authc拦截器拦截并进行oauth2客户端授权。

测试

1、首先访问<http://localhost:9080/chapter17-client/>，然后点击登录按钮进行登录，会跳到如下页面：

使用你的Shiro示例Server帐号访问 [chapter17-client]，并同时登录Shiro示例Server

用户名：

密码：

2、输入用户名进行登录并授权：

3、如果登录成功，服务端会重定向到客户端，即之前客户端提供的地址<http://localhost:9080/chapter17-client/oauth2-login?code=473d56015bcf576f2ca03eac1a5bcc11>，并带着auth code过去；

4、客户端的OAuth2AuthenticationFilter会收集此auth code，并创建OAuth2Token提交给Subject进行客户端登录；

5、客户端的Subject会委托给OAuth2Realm进行身份验证；此时OAuth2Realm会根据auth code换取access token，再根据access token获取受保护的用户信息；然后进行客户端登录。

到此OAuth2的集成就完成了，此处的服务端和客户端相对比较简单，没有进行一些异常检测，请参考如新浪微博进行相应API及异常错误码的设计。

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





17

顶

1

踩

分享到： 

[第十八章 并发登录人数控制——《跟我学Sh...》](#) | [第十六章 综合实例——《跟我学Shiro》](#)

- 2014-03-31 07:50
- 浏览 21333
- [评论\(13\)](#)
- 分类:[企业架构](#)
- [相关推荐](#)

评论

13 楼 [D.F.S](#) 2015-06-05

你好,之前说错了,确实我想用你的例子成功登录后,想退出再登录后后台会报异常,OAuthProblemException{error='invalid_request', description='Missing parameters: code', uri='null', state='null', scope='null', redirectUri='null', responseStatus=0, parameters={}},退出后点登录按钮,它没有跳到登录页面而是跳到登录成功的页面,这是怎么回事呢?

12 楼 [D.F.S](#) 2015-06-05

你好,我想用你的例子成功登录后,想退出怎么办呢?我在客户端调用退出方法时后台会报异常😞

11 楼 [worldfather168](#) 2015-05-20

worldfather168 写道

worldfather168 写道

michnus 写道

例子好像有bug,为何server登录成功后跳转到client, client的OAuth2AuthenticationFilter 执行了两次拦截,第二次请求oauth2-login时, code为空

OAuth2AuthenticationFilter#onLoginSuccess中调用了issueSuccessRedirect(request, response);这个方法不一定使用successUrl: 登录成功后如果之前有保存的请求, 则重定向到之前的那个请求, 否则转到默认的成功页面; 这边登录之前访问的是/oauth2-login,so,会拦截两次。

onAccessDenied的时候saveRequestAndRedirectToLogin,onLoginSuccess的时候issueSuccessRedirect, 有SavedRequest

这是把服务端客户端在一个web环境才会这样, 好像。

10 楼 [worldfather168](#) 2015-05-20

worldfather168 写道

michnus 写道

例子好像有bug,为何server登录成功后跳转到client, client的OAuth2AuthenticationFilter 执行了两次拦截,第二次请求oauth2-login时, code为空

OAuth2AuthenticationFilter#onLoginSuccess中调用了issueSuccessRedirect(request, response);这个方法不一定使用successUrl: 登录成功后如果之前有保存的请求, 则重定向到之前的那个请求, 否则转到默认的成功页面; 这边登录之前访问的是/oauth2-login,so,会拦截两次。

onAccessDenied的时候saveRequestAndRedirectToLogin,onLoginSuccess的时候issueSuccessRedirect, 有SavedRequest

9 楼 [worldfather168](#) 2015-05-20

michnus 写道

例子好像有bug,为何server登录成功后跳转到client, client的OAuth2AuthenticationFilter 执行了两次拦截,第二次请求oauth2-login时, code为空

OAuth2AuthenticationFilter#onLoginSuccess中调用了issueSuccessRedirect(request, response);这个方法不一定使用successUrl: 登录成功后如果之前有保存的请求, 则重定向到之前的那个请求, 否则转到默认的成功页面; 这边登录之前访问的是/oauth2-login,so,会拦截两次。

8 楼 [michnus](#) 2015-05-04

例子好像有bug,为何server登录成功后跳转到client, client的OAuth2AuthenticationFilter 执行了两次拦截,第二次请求oauth2-login时, code为空

7 楼 [fengyonglei](#) 2014-11-10

看代码, 没有涉及到数据加签/验签 或加密的操作, 难道 oauth2 不需要验证数据

6 楼 [hiyachen](#) 2014-10-02

用涛g的<https://github.com/zhangkaitao/shiro-example>的代码搭个环境。能给传个代码吗?

hiyachen@163.com谢谢!

5 楼 [lyq881209](#) 2014-04-01

二十四章会话管理 不用spring 怎么获取SessionDAO?

4 楼 [jinnianshilongnian](#) 2014-03-31

supercwg 写道

涛ge, 看版本号就知道Oltu还比较嫩, 真不敢用在生产环境哦😞

其之前的名字叫Apache Amber, 已经比较老了; 不过文档很少。

3 楼 [jinnianshilongnian](#) 2014-03-31

lyq881209 写道

客户端登陆也可以自定义credentialsMatcher, 方法doCredentialsMatch判断如果是通过OAuth2登陆的, 校验code直接返回true。这样普通登陆和OAuth2登陆都可以登陆。

嗯

2 楼 [supercwg](#) 2014-03-31

涛ge, 看版本号就知道Oltu还比较嫩, 真不敢用在生产环境哦😞

1 楼 [lyq881209](#) 2014-03-31

客户端登陆也可以自定义credentialsMatcher, 方法doCredentialsMatch判断如果是通过OAuth2登陆的, 校验code直接返回true。这样普通登陆和OAuth2登陆都可以登陆。

发表评论



您还没有登录,请您登录后再发表评论



jinnianshilongnian

• 浏览: 4747236 次

- 性别: 
-  我现在离线

最近访客 [更多访客>>](#)



[He_angel](#)



[liyong](#)



[lyyu1988](#)



[tianshaojie](#)

博客专栏



[跟我学spring3](#)

浏览量：868120



[Spring杂谈](#)

浏览量：858864



[跟开涛学SpringMVC...](#)

浏览量：1536259



[Servlet3.1规范翻...](#)

浏览量：103344



[springmvc杂谈](#)

浏览量：356869



[hibernate杂谈](#)

浏览量：114702



[跟我学Shiro](#)

浏览量：616710



[跟我学Nginx+Lua开...](#)

浏览量：42569

文章分类

- [全部博客 \(306\)](#)
- [跟我学Nginx+Lua开发 \(11\)](#)
- [跟我学spring \(54\)](#)
- [跟开涛学SpringMVC \(34\)](#)
- [spring4 \(16\)](#)
- [spring杂谈 \(50\)](#)
- [springmvc杂谈 \(22\)](#)

- [跟我学Shiro \(26\)](#)
- [shiro杂谈 \(3\)](#)
- [hibernate杂谈 \(10\)](#)
- [java开发常见问题分析 \(36\)](#)
- [加速Java应用开发 \(5\)](#)
- [Servlet 3.1规范\[翻译\] \(21\)](#)
- [servlet3.x \(2\)](#)
- [websocket协议\[翻译\] \(14\)](#)
- [websocket规范\[翻译\] \(1\)](#)
- [java web \(5\)](#)
- [db \(1\)](#)
- [js & jquery & bootstrap \(4\)](#)
- [非技术 \(4\)](#)
- [reminder\[转载\] \(23\)](#)
- [跟叶子学把妹 \(9\)](#)
- [nginx \(2\)](#)

社区版块

- [我的资讯 \(10\)](#)
- [我的论坛 \(1112\)](#)
- [我的问答 \(2428\)](#)

存档分类

- [2015-04 \(2\)](#)
- [2015-03 \(5\)](#)
- [2015-02 \(6\)](#)
- [更多存档...](#)

评论排行榜

- [第七章 Web开发实战2——商品详情页](#)
- [Spring4.1新特性——综述](#)
- [跟叶子学把妹——教程序猿把妹第二集](#)
- [跟叶子学把妹——教程序猿把妹第四集](#)
- [跟叶子学把妹——教程序猿把妹第六集](#)

最新评论

- [dongze1213](#) : @CachePut(value="cacheName ...
[Spring Cache抽象详解](#)
- [wxzlvj0221](#) : 非常好 有没有spring data的博文
[源代码下载 第六章 注解式控制器详解](#)
- [jybzjf](#) : 我用Atomikos开启了事事务，在spring中使用@Pers ...
[【第九章】 Spring的事务之 9.2 事务管理器 ——跟我学spring3](#)
- [鱼崽崽](#) : 可以看看这个，里面有真实的案例<http://43.249.81> ...
[第二章 Spring MVC入门 —— 跟开涛学SpringMVC](#)
- [MorningSunshine](#) : 到ioc扩展没有了，博主加油啊
[跟我学spring3 目录贴及电子书下载](#)

声明：ITeye文章版权属于作者，受法律保护。没有作者书面许可不得转载。若作者同意转载，必须以超链接形式标明文章原始出处和作者。

© 2003-2015 ITeye.com. All rights reserved. [京ICP证110151号 京公网安备110105010620]