使用OAuth2安全Spring REST API

**创建于：**2016年7月30日| **最后更新日期：**2017年9月30日  [websystiqueadmin](http://websystique.com/author/blogadminuser/)

现在让我们确保我们的Spring REST API使用，OAuth2这是一个简单的指南，显示了使用REST API来保护所需的内容Spring OAuth2。我们的用例非常符合Resource-owner Password GrantOAUth2规范的流程。我们将使用两个不同的客户端[Postman和一个Spring RestTemplate基于Java的应用程序]来访问我们的受OAuth2保护的REST资源。

如果您已经熟悉OAuth2概念，则可能需要跳过该理论，然后直接跳入代码。一如既往，完整的代码可以在本文末尾的附件中找到。我们走吧。

**您可能会喜欢的其他有趣帖子**

* [Spring Boot + AngularJS + Spring Data + Hibernate + MySQL CRUD App](http://websystique.com/spring-boot/spring-boot-angularjs-spring-data-jpa-crud-app-example/)
* [Spring Boot REST API教程](http://websystique.com/spring-boot/spring-boot-rest-api-example/)
* [Spring Boot WAR部署示例](http://websystique.com/spring-boot/spring-boot-war-deployment-example/)
* [Spring Boot简介+ Hello World示例](http://websystique.com/spring-boot/spring-boot-introduction-hello-world-example/)
* [使用基本身份验证来保护Spring REST API](http://websystique.com/spring-security/secure-spring-rest-api-using-basic-authentication/)
* [使用基本身份验证的AngularJS + Spring安全性](http://websystique.com/spring-security/angularjs-basic-authentication-using-spring-security/)

**什么是OAuth2**

OAuth2是一个标准化的授权协议/框架。根据官方[OAuth2规范](https://tools.ietf.org/html/rfc6749)：

OAuth 2.0授权框架允许第三方应用程序通过编排资源所有者与HTTP服务之间的批准交互，或通过允许第三方应用程序代表资源所有者获得对HTTP服务的有限访问以自己的名义获得访问权限。

谷歌，Facebook等大牌玩家已经在使用他们自己的OAuth2实现了相当长的一段时间。企业也在快速采用OAuth2。

我发现OAuth2规范相当简单。然而，如果你想要快速启动，可以在[这里](http://www.bubblecode.net/en/2016/01/22/understanding-oauth2/)找到关于OAuth2基础知识的优秀文章，从而深入了解OAUth2的理论概念。

[Spring Security OAuth](http://projects.spring.io/spring-security-oauth/)项目提供了我们可能需要的所有必要API，以便使用Spring开发符合OAuth2的实现。官方[Spring Security Oauth](https://github.com/spring-projects/spring-security-oauth/tree/master/samples/oauth2)项目为实现OAuth2提供了一个全面的示例。这篇文章的代码示例受到这些示例本身的启发。这篇文章的目的是为了保护我们的REST API，仅仅使用所需的最低限度的功能，仅此而已。

至少，您应该了解OAuth2中的四个关键概念：

**1. OAuth2角色**

OAuth2定义了四个角色：

* **resource owner：**  
  可能是你。一个能够授权访问受保护资源的实体。当资源所有者是一个人时，它被称为最终用户。
* **resource server：**  
  承载受保护资源的服务器，能够使用访问令牌接受和响应受保护的资源请求。
* **client：**  
  应用程序代表资源所有者及其授权进行受保护资源请求。它可能是一个移动应用程序，要求您允许访问您的Facebook供稿，一个尝试访问REST API的REST客户端，一个使用Facebook帐户提供替代登录选项的网站[Stackoverflow例如]。
* **authorization server：**  
  成功验证资源所有者并获得授权后，服务器向客户端发放访问令牌。

在我们的例子中，我们的REST API只能通过资源服务器访问，这需要访问令牌与请求一起存在

**2. OAuth2授权授权类型**

授权许可是表示客户用来获取访问令牌的资源所有者授权（访问其受保护资源）的凭证。该规范定义了四种授权类型：

* authorization code
* implicit
* resource owner password credentials
* client credentials

我们将使用resource owner password credentials授权类型。原因很简单，我们没有实现将我们重定向到登录页面的视图。只有客户端[基于Postman或RestTemplate的Java客户端]具有资源所有者的凭证，并且将这些凭证（连同客户端凭证）提供给授权服务器以最终接收访问令牌[和可选的刷新令牌]的用法，然后使用该令牌实际访问资源。

一个常见的例子是GMail app智能手机上的[客户端]，它会获取您的凭据并使用它们连接到GMail servers。它还表明，当客户端和服务器都来自同一家公司并且信任在那里时，“密码凭证授权”是最合适的，您不希望将凭据提供给第三方。

**3. OAuth2令牌**

令牌是特定于实现的随机字符串，由授权服务器生成并在客户端请求时发布。

* Access Token ：与每个请求一起发送，通常在很短的生命周期内有效[例如一小时]
* Refresh Token ：主要用于获取一个新的访问令牌，不随每个请求一起发送，通常比访问令牌的寿命更长。

**关于HTTPS的一句话**：对于任何类型的安全实现，从基本认证到完整的OAuth2实现**HTTPS**都是必须的。没有HTTPS，无论您的实施是什么，安全性都容易受到攻击。

**4. OAuth2访问令牌范围**

客户端可以使用范围要求具有特定访问权限的资源[想要访问该用户Facebook帐户的提要和照片]，并且授权服务器反过来返回范围，显示实际授予客户端的访问权限[资源所有者仅允许提供访问权限，没有照片，例如]。

让我们进入代码

为了访问我们的REST资源，我们使用Spring Security来实现必要的构建块以实现OAuth。

**1.资源服务器**

资源服务器托管客户感兴趣的资源[我们的REST API]。资源位于/user/。@EnableResourceServer应用于OAuth2资源服务器的注释使Spring Security过滤器能够使用传入的OAuth2令牌对请求进行身份验证。Class ResourceServerConfigurerAdapter实现ResourceServerConfigurer提供方法来调整受OAuth2安全保护的访问规则和路径。

|  |
| --- |
| package com.websystique.springmvc.security;    import org.springframework.context.annotation.Configuration;  import org.springframework.security.config.annotation.web.builders.HttpSecurity;  import org.springframework.security.oauth2.config.annotation.web.configuration.EnableResourceServer;  import org.springframework.security.oauth2.config.annotation.web.configuration.ResourceServerConfigurerAdapter;  import org.springframework.security.oauth2.config.annotation.web.configurers.ResourceServerSecurityConfigurer;  import org.springframework.security.oauth2.provider.error.OAuth2AccessDeniedHandler;    @Configuration  @EnableResourceServer  public class ResourceServerConfiguration extends ResourceServerConfigurerAdapter {        private static final String RESOURCE\_ID = "my\_rest\_api";        @Override      public void configure(ResourceServerSecurityConfigurer resources) {          resources.resourceId(RESOURCE\_ID).stateless(false);      }        @Override      public void configure(HttpSecurity http) throws Exception {          http.          anonymous().disable()          .requestMatchers().antMatchers("/user/\*\*")          .and().authorizeRequests()          .antMatchers("/user/\*\*").access("hasRole('ADMIN')")          .and().exceptionHandling().accessDeniedHandler(new OAuth2AccessDeniedHandler());      }    } |

**2.授权服务器**

授权服务器负责验证凭证，如果凭证正常，则提供令牌[refresh-token以及access-token]。它还包含有关注册客户端和可能的访问范围和授予类型的信息。令牌存储用于存储令牌。我们将使用内存令牌存储。@EnableAuthorizationServer启用当前应用程序上下文中的授权服务器（即AuthorizationEndpoint和TokenEndpoint）。它提供了配置授权服务器的所有必要方法的类AuthorizationServerConfigurerAdapter实现AuthorizationServerConfigurer。

|  |
| --- |
| package com.websystique.springmvc.security;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.beans.factory.annotation.Qualifier;  import org.springframework.context.annotation.Configuration;  import org.springframework.security.authentication.AuthenticationManager;  import org.springframework.security.oauth2.config.annotation.configurers.ClientDetailsServiceConfigurer;  import org.springframework.security.oauth2.config.annotation.web.configuration.AuthorizationServerConfigurerAdapter;  import org.springframework.security.oauth2.config.annotation.web.configuration.EnableAuthorizationServer;  import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerEndpointsConfigurer;  import org.springframework.security.oauth2.config.annotation.web.configurers.AuthorizationServerSecurityConfigurer;  import org.springframework.security.oauth2.provider.approval.UserApprovalHandler;  import org.springframework.security.oauth2.provider.token.TokenStore;    @Configuration  @EnableAuthorizationServer  public class AuthorizationServerConfiguration extends AuthorizationServerConfigurerAdapter {        private static String REALM="MY\_OAUTH\_REALM";        @Autowired      private TokenStore tokenStore;        @Autowired      private UserApprovalHandler userApprovalHandler;        @Autowired      @Qualifier("authenticationManagerBean")      private AuthenticationManager authenticationManager;        @Override      public void configure(ClientDetailsServiceConfigurer clients) throws Exception {            clients.inMemory()              .withClient("my-trusted-client")              .authorizedGrantTypes("password", "authorization\_code", "refresh\_token", "implicit")              .authorities("ROLE\_CLIENT", "ROLE\_TRUSTED\_CLIENT")              .scopes("read", "write", "trust")              .secret("secret")              .accessTokenValiditySeconds(120).//Access token is only valid for 2 minutes.              refreshTokenValiditySeconds(600);//Refresh token is only valid for 10 minutes.      }        @Override      public void configure(AuthorizationServerEndpointsConfigurer endpoints) throws Exception {          endpoints.tokenStore(tokenStore).userApprovalHandler(userApprovalHandler)                  .authenticationManager(authenticationManager);      }        @Override      public void configure(AuthorizationServerSecurityConfigurer oauthServer) throws Exception {          oauthServer.realm(REALM+"/client");      }    } |

以上配置

* 注册一个客户端，客户端ID为'my-trusted-client'，密码为'secret'，角色和范围为允许。
* 指定任何生成的访问令牌仅在120秒内有效
* 指定任何生成的刷新令牌仅在600秒内有效

**3.安全配置**

把所有东西粘在一起。端点/oauth/token用于请求令牌[访问或刷新]。资源所有者[bill，bob]在这里被配置。

|  |
| --- |
| package com.websystique.springmvc.security;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Bean;  import org.springframework.context.annotation.Configuration;  import org.springframework.security.authentication.AuthenticationManager;  import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;  import org.springframework.security.config.annotation.web.builders.HttpSecurity;  import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  import org.springframework.security.oauth2.provider.ClientDetailsService;  import org.springframework.security.oauth2.provider.approval.ApprovalStore;  import org.springframework.security.oauth2.provider.approval.TokenApprovalStore;  import org.springframework.security.oauth2.provider.approval.TokenStoreUserApprovalHandler;  import org.springframework.security.oauth2.provider.request.DefaultOAuth2RequestFactory;  import org.springframework.security.oauth2.provider.token.TokenStore;  import org.springframework.security.oauth2.provider.token.store.InMemoryTokenStore;    @Configuration  @EnableWebSecurity  public class OAuth2SecurityConfiguration extends WebSecurityConfigurerAdapter {        @Autowired      private ClientDetailsService clientDetailsService;        @Autowired      public void globalUserDetails(AuthenticationManagerBuilder auth) throws Exception {          auth.inMemoryAuthentication()          .withUser("bill").password("abc123").roles("ADMIN").and()          .withUser("bob").password("abc123").roles("USER");      }        @Override      protected void configure(HttpSecurity http) throws Exception {          http          .csrf().disable()          .anonymous().disable()          .authorizeRequests()          .antMatchers("/oauth/token").permitAll();      }        @Override      @Bean      public AuthenticationManager authenticationManagerBean() throws Exception {          return super.authenticationManagerBean();      }          @Bean      public TokenStore tokenStore() {          return new InMemoryTokenStore();      }        @Bean      @Autowired      public TokenStoreUserApprovalHandler userApprovalHandler(TokenStore tokenStore){          TokenStoreUserApprovalHandler handler = new TokenStoreUserApprovalHandler();          handler.setTokenStore(tokenStore);          handler.setRequestFactory(new DefaultOAuth2RequestFactory(clientDetailsService));          handler.setClientDetailsService(clientDetailsService);          return handler;      }        @Bean      @Autowired      public ApprovalStore approvalStore(TokenStore tokenStore) throws Exception {          TokenApprovalStore store = new TokenApprovalStore();          store.setTokenStore(tokenStore);          return store;      }    } |

此外，启用全局方法安全性，如果我们要使用它们，将激活@PreFilter，@PostFilter，@PreAuthorize @PostAuthorize注释。

|  |
| --- |
| package com.websystique.springmvc.security;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.context.annotation.Configuration;  import org.springframework.security.access.expression.method.MethodSecurityExpressionHandler;  import org.springframework.security.config.annotation.method.configuration.EnableGlobalMethodSecurity;  import org.springframework.security.config.annotation.method.configuration.GlobalMethodSecurityConfiguration;  import org.springframework.security.oauth2.provider.expression.OAuth2MethodSecurityExpressionHandler;    @Configuration  @EnableGlobalMethodSecurity(prePostEnabled = true, proxyTargetClass = true)  public class MethodSecurityConfig extends GlobalMethodSecurityConfiguration {      @Autowired      private OAuth2SecurityConfiguration securityConfig;        @Override      protected MethodSecurityExpressionHandler createExpressionHandler() {          return new OAuth2MethodSecurityExpressionHandler();      }  } |

**4.终点及其目的**

* 试图在没有任何授权的情况下访问资源[REST API] [当然会失败]。  
  GET http://localhost:8080/SpringSecurityOAuth2Example/user/
* 问令牌[接入+刷新]使用**HTTP POST**上/oauth/token，与grant\_type =密码和资源所有者凭据REQ-PARAMS。此外，在授权标头中发送客户端凭证。

POST http://localhost:8080/SpringSecurityOAuth2Example/oauth/token?grant\_type=password&username=bill&password=abc123

* 要求通过有效的刷新令牌新的访问令牌，使用**HTTP POST**上/oauth/token，与grant\_type = refresh\_token，并发送实际刷新令牌。此外，在授权标头中发送客户端凭证。

POST http://localhost:8080/SpringSecurityOAuth2Example/oauth/token?grant\_type=refresh\_token&refresh\_token=094b7d23-973f-4cc1-83ad-8ffd43de1845

* 通过使用access\_token带请求的查询参数提供访问令牌来访问资源。  
  GET http://localhost:8080/SpringSecurityOAuth2Example/user/?access\_token=3525d0e4-d881-49e7-9f91-bcfd18259109

**5. Rest API**

我在大多数帖子中使用的简单Spring REST API。

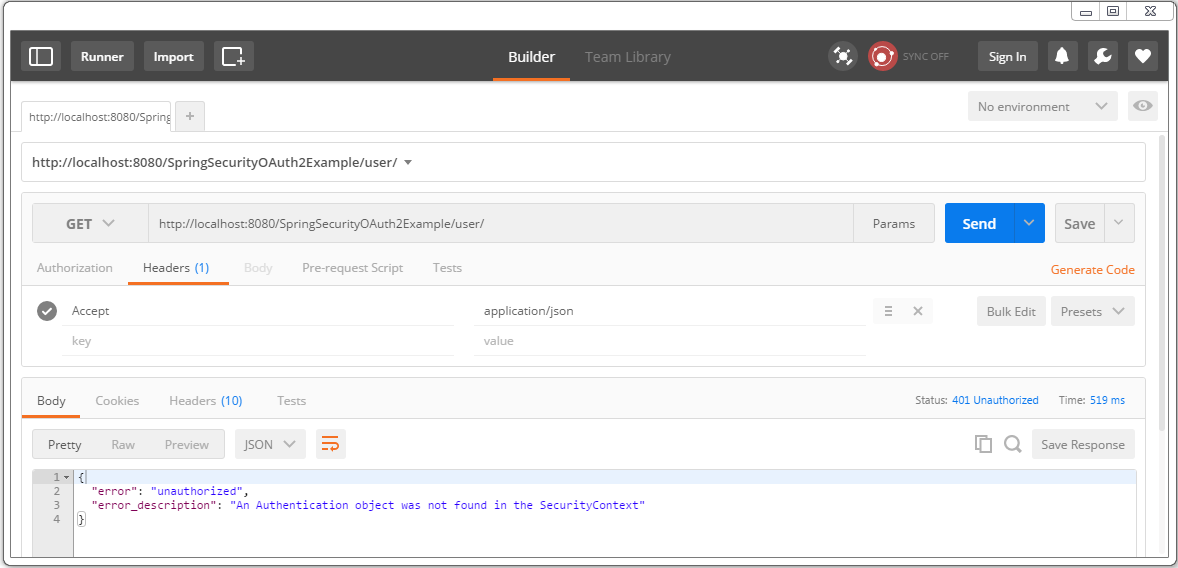
|  |
| --- |
| package com.websystique.springmvc.controller;    import java.util.List;    import org.springframework.beans.factory.annotation.Autowired;  import org.springframework.http.HttpHeaders;  import org.springframework.http.HttpStatus;  import org.springframework.http.MediaType;  import org.springframework.http.ResponseEntity;  import org.springframework.web.bind.annotation.PathVariable;  import org.springframework.web.bind.annotation.RequestBody;  import org.springframework.web.bind.annotation.RequestMapping;  import org.springframework.web.bind.annotation.RequestMethod;  import org.springframework.web.bind.annotation.RestController;  import org.springframework.web.util.UriComponentsBuilder;    import com.websystique.springmvc.model.User;  import com.websystique.springmvc.service.UserService;    @RestController  public class HelloWorldRestController {        @Autowired      UserService userService;  //Service which will do all data retrieval/manipulation work          //-------------------Retrieve All Users--------------------------------------------------------        @RequestMapping(value = "/user/", method = RequestMethod.GET)      public ResponseEntity<List<User>> listAllUsers() {          List<User> users = userService.findAllUsers();          if(users.isEmpty()){              return new ResponseEntity<List<User>>(HttpStatus.NO\_CONTENT);//You many decide to return HttpStatus.NOT\_FOUND          }          return new ResponseEntity<List<User>>(users, HttpStatus.OK);      }          //-------------------Retrieve Single User--------------------------------------------------------        @RequestMapping(value = "/user/{id}", method = RequestMethod.GET, produces = {MediaType.APPLICATION\_JSON\_VALUE,MediaType.APPLICATION\_XML\_VALUE})      public ResponseEntity<User> getUser(@PathVariable("id") long id) {          System.out.println("Fetching User with id " + id);          User user = userService.findById(id);          if (user == null) {              System.out.println("User with id " + id + " not found");              return new ResponseEntity<User>(HttpStatus.NOT\_FOUND);          }          return new ResponseEntity<User>(user, HttpStatus.OK);      }            //-------------------Create a User--------------------------------------------------------        @RequestMapping(value = "/user/", method = RequestMethod.POST)      public ResponseEntity<Void> createUser(@RequestBody User user, UriComponentsBuilder ucBuilder) {          System.out.println("Creating User " + user.getName());            if (userService.isUserExist(user)) {              System.out.println("A User with name " + user.getName() + " already exist");              return new ResponseEntity<Void>(HttpStatus.CONFLICT);          }            userService.saveUser(user);            HttpHeaders headers = new HttpHeaders();          headers.setLocation(ucBuilder.path("/user/{id}").buildAndExpand(user.getId()).toUri());          return new ResponseEntity<Void>(headers, HttpStatus.CREATED);      }          //------------------- Update a User --------------------------------------------------------        @RequestMapping(value = "/user/{id}", method = RequestMethod.PUT)      public ResponseEntity<User> updateUser(@PathVariable("id") long id, @RequestBody User user) {          System.out.println("Updating User " + id);            User currentUser = userService.findById(id);            if (currentUser==null) {              System.out.println("User with id " + id + " not found");              return new ResponseEntity<User>(HttpStatus.NOT\_FOUND);          }            currentUser.setName(user.getName());          currentUser.setAge(user.getAge());          currentUser.setSalary(user.getSalary());            userService.updateUser(currentUser);          return new ResponseEntity<User>(currentUser, HttpStatus.OK);      }        //------------------- Delete a User --------------------------------------------------------        @RequestMapping(value = "/user/{id}", method = RequestMethod.DELETE)      public ResponseEntity<User> deleteUser(@PathVariable("id") long id) {          System.out.println("Fetching & Deleting User with id " + id);            User user = userService.findById(id);          if (user == null) {              System.out.println("Unable to delete. User with id " + id + " not found");              return new ResponseEntity<User>(HttpStatus.NOT\_FOUND);          }            userService.deleteUserById(id);          return new ResponseEntity<User>(HttpStatus.NO\_CONTENT);      }          //------------------- Delete All Users --------------------------------------------------------        @RequestMapping(value = "/user/", method = RequestMethod.DELETE)      public ResponseEntity<User> deleteAllUsers() {          System.out.println("Deleting All Users");            userService.deleteAllUsers();          return new ResponseEntity<User>(HttpStatus.NO\_CONTENT);      }    } |

**6.运行应用程序**

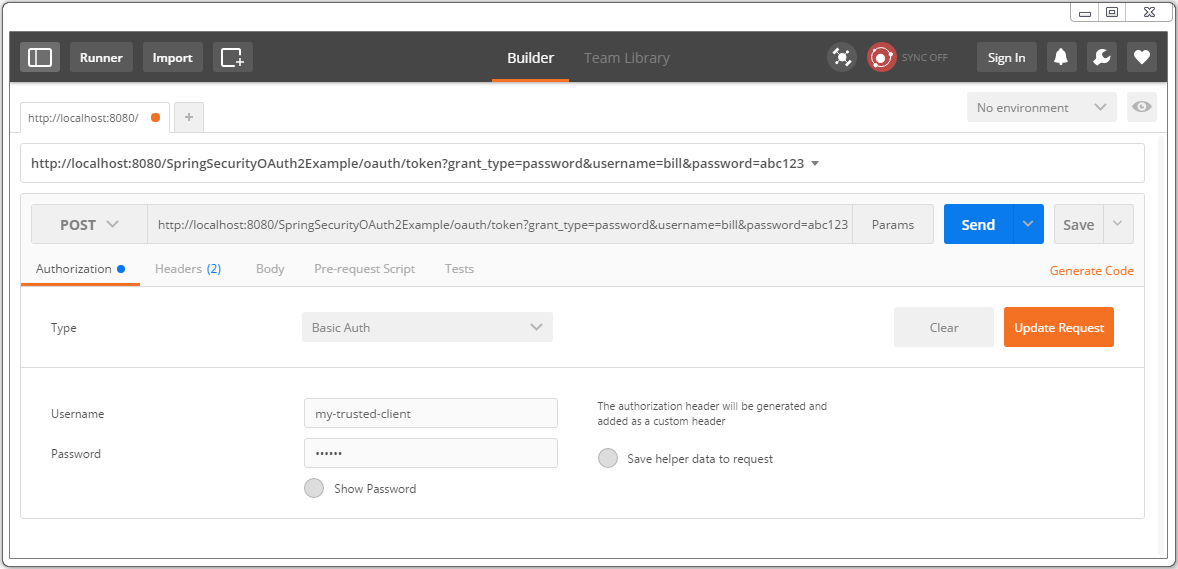
运行它并使用两个不同的客户端进行测试。

**客户1：邮递员**

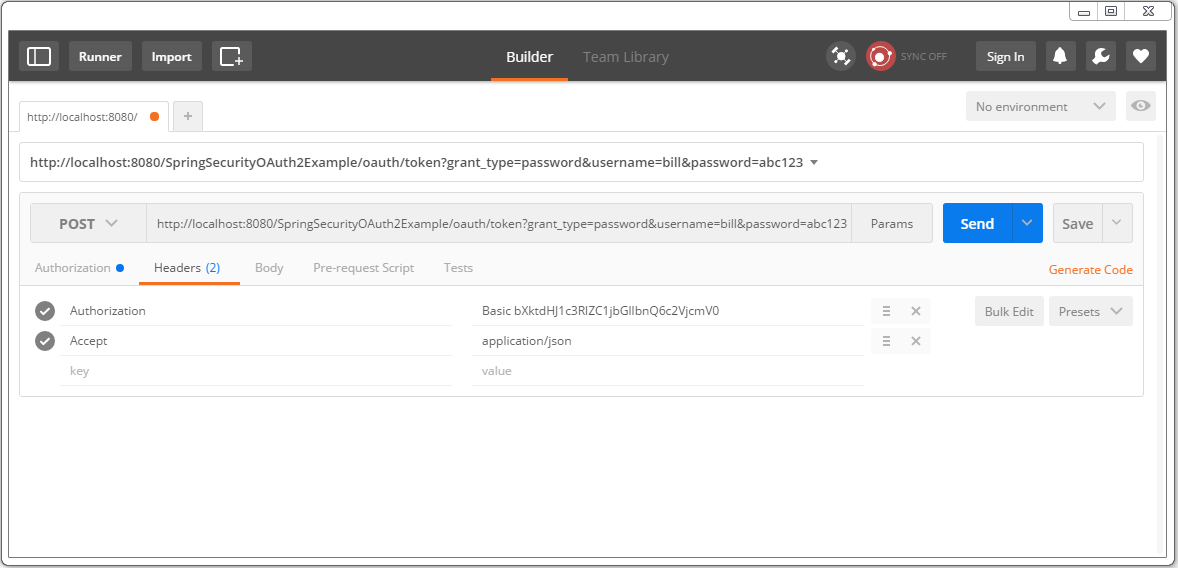
尝试访问没有任何身份验证信息的资源，将获得401。



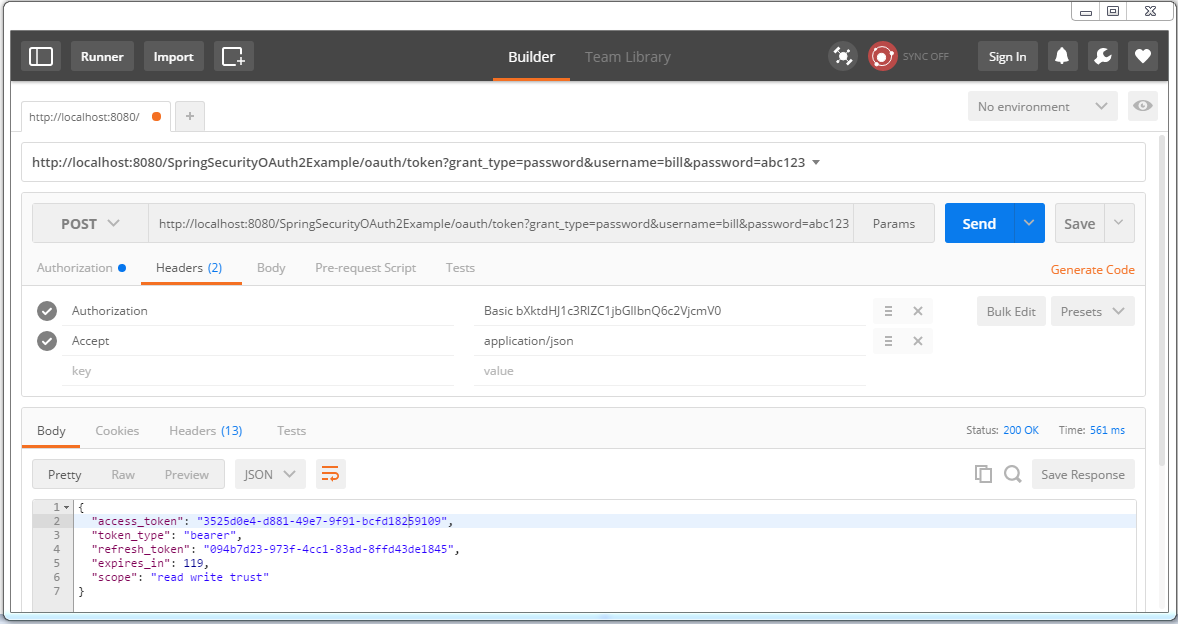
让我们得到令牌。首先添加一个带有**客户端凭证**的授权头[my-trusted-client / secret]。



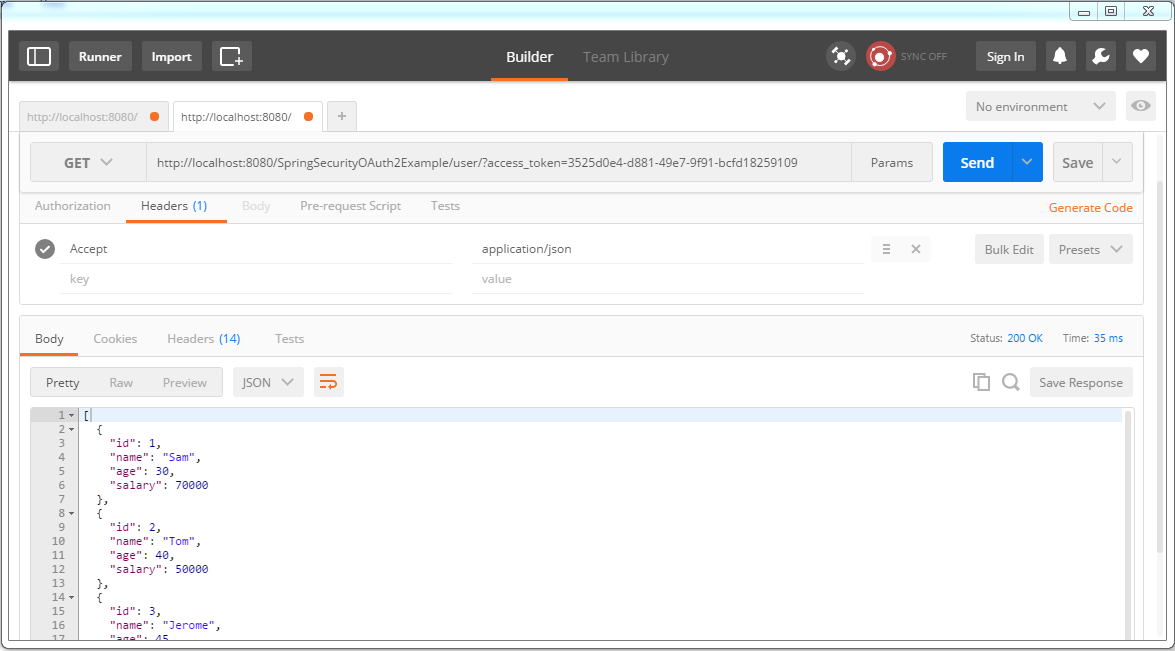
点击更新请求，验证标题标签中的标题。



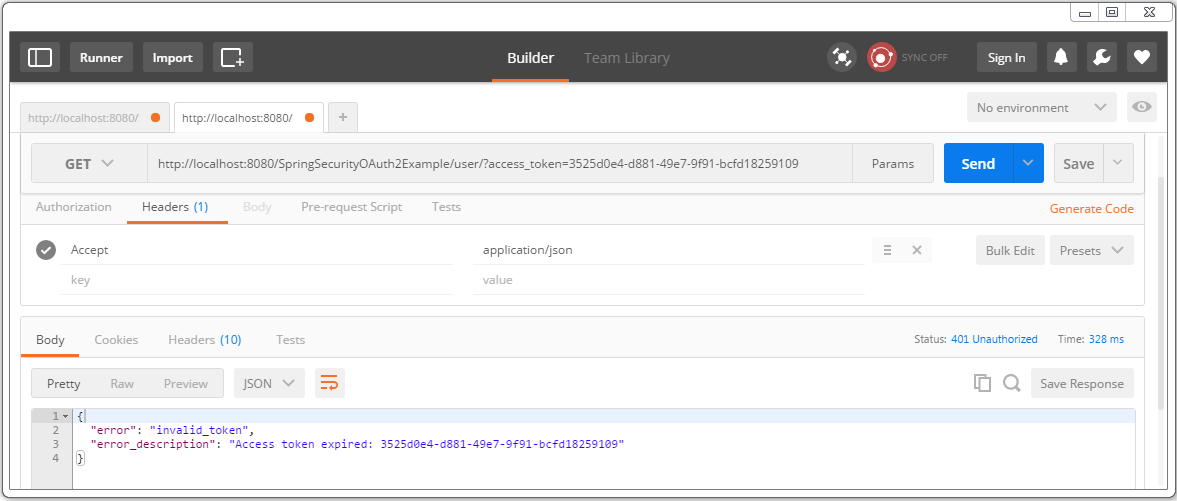
发送POST请求时，您将收到包含响应access-token以及refresh-token。



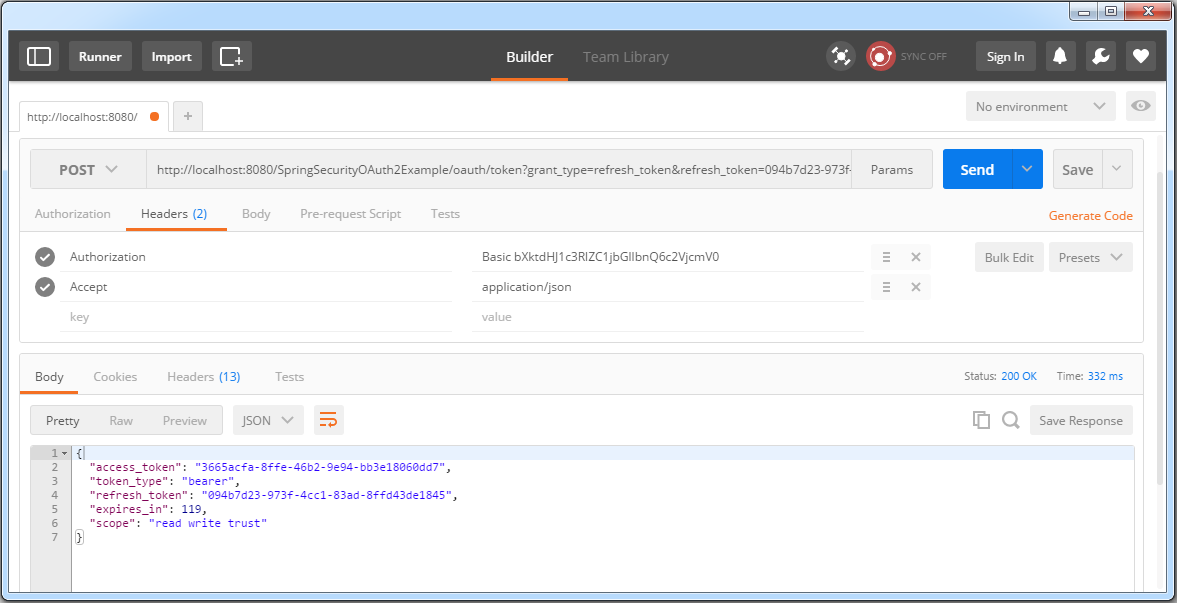
将这些令牌保存在某处，您将需要它们。现在您可以使用此访问令牌[有效期为2分钟]来访问资源。



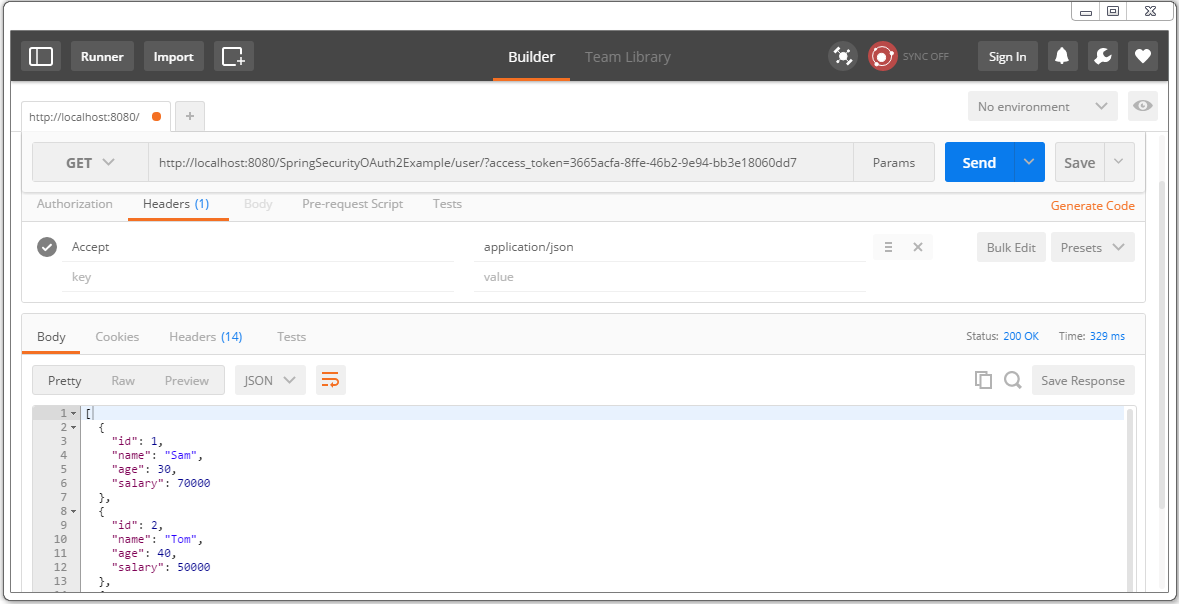
2分钟后，访问令牌会过期，您的更多资源请求将失败。



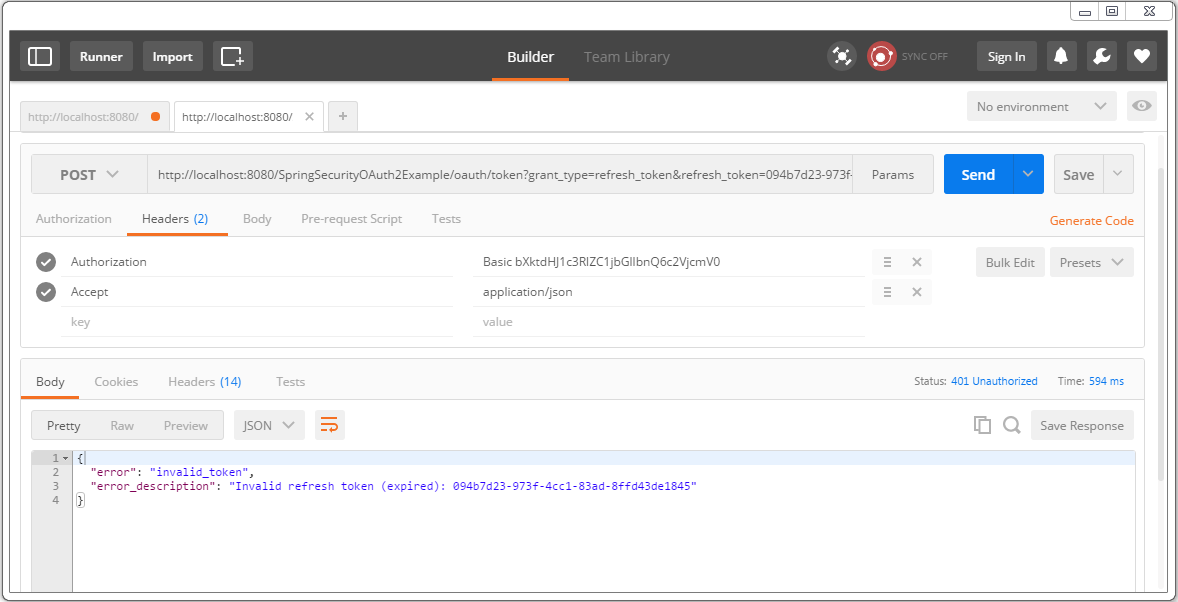
我们需要一个新的访问令牌。用刷新标记发布帖子以获取全新的访问令牌。



使用这个新的访问令牌来访问资源。



刷新令牌也会过期[10分钟]。之后，你应该看到你的刷新请求失败。



这意味着您需要请求一个新的刷新+访问令牌，如第2步中所述。

**客户端2：基于RestTemplate的Java应用程序**

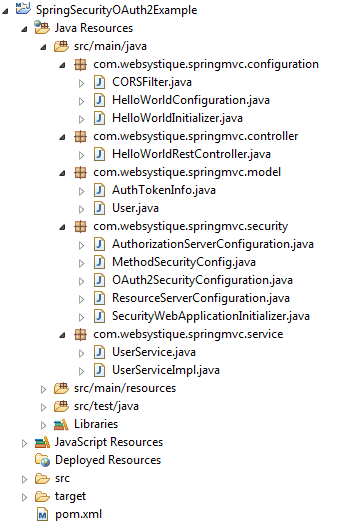
方法**sendTokenRequest**用于实际获取令牌。我们得到的访问令牌随后用于每个请求。如果需要，您可以在下面的示例中轻松实现刷新令牌流。

|  |
| --- |
| package com.websystique.springmvc;    import java.net.URI;  import java.util.Arrays;  import java.util.LinkedHashMap;  import java.util.List;    import org.apache.commons.codec.binary.Base64;  import org.springframework.http.HttpEntity;  import org.springframework.http.HttpHeaders;  import org.springframework.http.HttpMethod;  import org.springframework.http.MediaType;  import org.springframework.http.ResponseEntity;  import org.springframework.util.Assert;  import org.springframework.web.client.RestTemplate;    import com.websystique.springmvc.model.AuthTokenInfo;  import com.websystique.springmvc.model.User;    public class SpringRestClient {        public static final String REST\_SERVICE\_URI = "<http://localhost:8080/SpringSecurityOAuth2Example>";        public static final String AUTH\_SERVER\_URI = "<http://localhost:8080/SpringSecurityOAuth2Example/oauth/token>";        public static final String QPM\_PASSWORD\_GRANT = "?grant\_type=password&username=bill&password=abc123";        public static final String QPM\_ACCESS\_TOKEN = "?access\_token=";        /\*       \* Prepare HTTP Headers.       \*/      private static HttpHeaders getHeaders(){          HttpHeaders headers = new HttpHeaders();          headers.setAccept(Arrays.asList(MediaType.APPLICATION\_JSON));          return headers;      }        /\*       \* Add HTTP Authorization header, using Basic-Authentication to send client-credentials.       \*/      private static HttpHeaders getHeadersWithClientCredentials(){          String plainClientCredentials="my-trusted-client:secret";          String base64ClientCredentials = new String(Base64.encodeBase64(plainClientCredentials.getBytes()));            HttpHeaders headers = getHeaders();          headers.add("Authorization", "Basic " + base64ClientCredentials);          return headers;      }        /\*       \* Send a POST request [on /oauth/token] to get an access-token, which will then be send with each request.       \*/      @SuppressWarnings({ "unchecked"})      private static AuthTokenInfo sendTokenRequest(){          RestTemplate restTemplate = new RestTemplate();            HttpEntity<String> request = new HttpEntity<String>(getHeadersWithClientCredentials());          ResponseEntity<Object> response = restTemplate.exchange(AUTH\_SERVER\_URI+QPM\_PASSWORD\_GRANT, HttpMethod.POST, request, Object.class);          LinkedHashMap<String, Object> map = (LinkedHashMap<String, Object>)response.getBody();          AuthTokenInfo tokenInfo = null;            if(map!=null){              tokenInfo = new AuthTokenInfo();              tokenInfo.setAccess\_token((String)map.get("access\_token"));              tokenInfo.setToken\_type((String)map.get("token\_type"));              tokenInfo.setRefresh\_token((String)map.get("refresh\_token"));              tokenInfo.setExpires\_in((int)map.get("expires\_in"));              tokenInfo.setScope((String)map.get("scope"));              System.out.println(tokenInfo);              //System.out.println("access\_token ="+map.get("access\_token")+", token\_type="+map.get("token\_type")+", refresh\_token="+map.get("refresh\_token")              //+", expires\_in="+map.get("expires\_in")+", scope="+map.get("scope"));;          }else{              System.out.println("No user exist----------");            }          return tokenInfo;      }        /\*       \* Send a GET request to get list of all users.       \*/      @SuppressWarnings({ "unchecked", "rawtypes" })      private static void listAllUsers(AuthTokenInfo tokenInfo){          Assert.notNull(tokenInfo, "Authenticate first please......");            System.out.println("\nTesting listAllUsers API-----------");          RestTemplate restTemplate = new RestTemplate();            HttpEntity<String> request = new HttpEntity<String>(getHeaders());          ResponseEntity<List> response = restTemplate.exchange(REST\_SERVICE\_URI+"/user/"+QPM\_ACCESS\_TOKEN+tokenInfo.getAccess\_token(),                  HttpMethod.GET, request, List.class);          List<LinkedHashMap<String, Object>> usersMap = (List<LinkedHashMap<String, Object>>)response.getBody();            if(usersMap!=null){              for(LinkedHashMap<String, Object> map : usersMap){                  System.out.println("User : id="+map.get("id")+", Name="+map.get("name")+", Age="+map.get("age")+", Salary="+map.get("salary"));;              }          }else{              System.out.println("No user exist----------");          }      }        /\*       \* Send a GET request to get a specific user.       \*/      private static void getUser(AuthTokenInfo tokenInfo){          Assert.notNull(tokenInfo, "Authenticate first please......");          System.out.println("\nTesting getUser API----------");          RestTemplate restTemplate = new RestTemplate();          HttpEntity<String> request = new HttpEntity<String>(getHeaders());          ResponseEntity<User> response = restTemplate.exchange(REST\_SERVICE\_URI+"/user/1"+QPM\_ACCESS\_TOKEN+tokenInfo.getAccess\_token(),                  HttpMethod.GET, request, User.class);          User user = response.getBody();          System.out.println(user);      }        /\*       \* Send a POST request to create a new user.       \*/      private static void createUser(AuthTokenInfo tokenInfo) {          Assert.notNull(tokenInfo, "Authenticate first please......");          System.out.println("\nTesting create User API----------");          RestTemplate restTemplate = new RestTemplate();          User user = new User(0,"Sarah",51,134);          HttpEntity<Object> request = new HttpEntity<Object>(user, getHeaders());          URI uri = restTemplate.postForLocation(REST\_SERVICE\_URI+"/user/"+QPM\_ACCESS\_TOKEN+tokenInfo.getAccess\_token(),                  request, User.class);          System.out.println("Location : "+uri.toASCIIString());      }        /\*       \* Send a PUT request to update an existing user.       \*/      private static void updateUser(AuthTokenInfo tokenInfo) {          Assert.notNull(tokenInfo, "Authenticate first please......");          System.out.println("\nTesting update User API----------");          RestTemplate restTemplate = new RestTemplate();          User user  = new User(1,"Tomy",33, 70000);          HttpEntity<Object> request = new HttpEntity<Object>(user, getHeaders());          ResponseEntity<User> response = restTemplate.exchange(REST\_SERVICE\_URI+"/user/1"+QPM\_ACCESS\_TOKEN+tokenInfo.getAccess\_token(),                  HttpMethod.PUT, request, User.class);          System.out.println(response.getBody());      }        /\*       \* Send a DELETE request to delete a specific user.       \*/      private static void deleteUser(AuthTokenInfo tokenInfo) {          Assert.notNull(tokenInfo, "Authenticate first please......");          System.out.println("\nTesting delete User API----------");          RestTemplate restTemplate = new RestTemplate();          HttpEntity<String> request = new HttpEntity<String>(getHeaders());          restTemplate.exchange(REST\_SERVICE\_URI+"/user/3"+QPM\_ACCESS\_TOKEN+tokenInfo.getAccess\_token(),                  HttpMethod.DELETE, request, User.class);      }          /\*       \* Send a DELETE request to delete all users.       \*/      private static void deleteAllUsers(AuthTokenInfo tokenInfo) {          Assert.notNull(tokenInfo, "Authenticate first please......");          System.out.println("\nTesting all delete Users API----------");          RestTemplate restTemplate = new RestTemplate();          HttpEntity<String> request = new HttpEntity<String>(getHeaders());          restTemplate.exchange(REST\_SERVICE\_URI+"/user/"+QPM\_ACCESS\_TOKEN+tokenInfo.getAccess\_token(),                  HttpMethod.DELETE, request, User.class);      }        public static void main(String args[]){          AuthTokenInfo tokenInfo = sendTokenRequest();          listAllUsers(tokenInfo);            getUser(tokenInfo);            createUser(tokenInfo);          listAllUsers(tokenInfo);            updateUser(tokenInfo);          listAllUsers(tokenInfo);            deleteUser(tokenInfo);          listAllUsers(tokenInfo);            deleteAllUsers(tokenInfo);          listAllUsers(tokenInfo);      }  } |

以上代码将生成以下输出：

|  |
| --- |
| AuthTokenInfo [access\_token=fceed386-5923-4bf8-b193-1d76f95da4c4, token\_type=bearer, refresh\_token=29d28ee2-9d09-483f-a2d6-7f93e7a31667, expires\_in=71, scope=read write trust]    Testing listAllUsers API-----------  User : id=1, Name=Sam, Age=30, Salary=70000.0  User : id=2, Name=Tom, Age=40, Salary=50000.0  User : id=3, Name=Jerome, Age=45, Salary=30000.0  User : id=4, Name=Silvia, Age=50, Salary=40000.0    Testing getUser API----------  User [id=1, name=Sam, age=30, salary=70000.0]    Testing create User API----------  Location : <http://localhost:8080/SpringSecurityOAuth2Example/user/5>    Testing listAllUsers API-----------  User : id=1, Name=Sam, Age=30, Salary=70000.0  User : id=2, Name=Tom, Age=40, Salary=50000.0  User : id=3, Name=Jerome, Age=45, Salary=30000.0  User : id=4, Name=Silvia, Age=50, Salary=40000.0  User : id=5, Name=Sarah, Age=51, Salary=134.0    Testing update User API----------  User [id=1, name=Tomy, age=33, salary=70000.0]    Testing listAllUsers API-----------  User : id=1, Name=Tomy, Age=33, Salary=70000.0  User : id=2, Name=Tom, Age=40, Salary=50000.0  User : id=3, Name=Jerome, Age=45, Salary=30000.0  User : id=4, Name=Silvia, Age=50, Salary=40000.0  User : id=5, Name=Sarah, Age=51, Salary=134.0    Testing delete User API----------    Testing listAllUsers API-----------  User : id=1, Name=Tomy, Age=33, Salary=70000.0  User : id=2, Name=Tom, Age=40, Salary=50000.0  User : id=4, Name=Silvia, Age=50, Salary=40000.0  User : id=5, Name=Sarah, Age=51, Salary=134.0    Testing all delete Users API----------    Testing listAllUsers API-----------  No user exist---------- |

**项目结构**



**的pom.xml**

|  |
| --- |
| <project xmlns="<http://maven.apache.org/POM/4.0.0>" xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>"      xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0> <http://maven.apache.org/xsd/maven-4.0.0.xsd>">      <modelVersion>4.0.0</modelVersion>        <groupId>com.websystique.springmvc</groupId>      <artifactId>SpringSecurityOAuth2Example</artifactId>      <version>1.0.0</version>      <packaging>war</packaging>        <name>SpringSecurityOAuth2Example</name>        <properties>          <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>          <springframework.version>4.3.1.RELEASE</springframework.version>          <springsecurity.version>4.1.1.RELEASE</springsecurity.version>          <springsecurityoauth2.version>2.0.10.RELEASE</springsecurityoauth2.version>          <jackson.library>2.7.5</jackson.library>      </properties>        <dependencies>          <!-- Spring -->          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-core</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-web</artifactId>              <version>${springframework.version}</version>          </dependency>          <dependency>              <groupId>org.springframework</groupId>              <artifactId>spring-webmvc</artifactId>              <version>${springframework.version}</version>          </dependency>            <!-- Spring Security -->          <dependency>              <groupId>org.springframework.security</groupId>              <artifactId>spring-security-web</artifactId>              <version>${springsecurity.version}</version>          </dependency>          <dependency>              <groupId>org.springframework.security</groupId>              <artifactId>spring-security-config</artifactId>              <version>${springsecurity.version}</version>          </dependency>            <!-- Spring Security OAuth2-->          <dependency>              <groupId>org.springframework.security.oauth</groupId>              <artifactId>spring-security-oauth2</artifactId>              <version>${springsecurityoauth2.version}</version>          </dependency>            <!-- Jackson libraries -->          <dependency>              <groupId>com.fasterxml.jackson.core</groupId>              <artifactId>jackson-databind</artifactId>              <version>${jackson.library}</version>          </dependency>          <dependency>              <groupId>com.fasterxml.jackson.dataformat</groupId>              <artifactId>jackson-dataformat-xml</artifactId>              <version>${jackson.library}</version>          </dependency>            <dependency>              <groupId>javax.servlet</groupId>              <artifactId>javax.servlet-api</artifactId>              <version>3.1.0</version>          </dependency>      </dependencies>        <build>              <plugins>                  <plugin>                      <groupId>org.apache.maven.plugins</groupId>                      <artifactId>maven-compiler-plugin</artifactId>                      <version>3.2</version>                      <configuration>                          <source>1.7</source>                          <target>1.7</target>                      </configuration>                  </plugin>                  <plugin>                      <groupId>org.apache.maven.plugins</groupId>                      <artifactId>maven-war-plugin</artifactId>                      <version>2.4</version>                      <configuration>                          <warSourceDirectory>src/main/webapp</warSourceDirectory>                          <warName>SpringSecurityOAuth2Example</warName>                          <failOnMissingWebXml>false</failOnMissingWebXml>                      </configuration>                  </plugin>              </plugins>          <finalName>SpringSecurityOAuth2Example</finalName>      </build>  </project> |

***下载源代码***

[**现在下载！**](http://websystique.com/?smd_process_download=1&download_id=2926)