原 Secure REST API with oauth2 （翻译）

QiHaiYan 发表于 6个月前 阅读 173 收藏 10 点赞 0 评论 6

HOT

1.概述

demo: https://github.com/qihaiyan/ng-boot-oauth

在这个教程中，我们将用oauth2对REST API进行安全控制，并在一个简单的AngularJS客户端程序中使用。 我们将要构建的应用包含四个独立的模块：

- Authorization Server
- Resource Server
- UI implicit – 使用 Implicit Flow 的前端应用
- UI password – 使用 Password Flow 的前端应用

2.认证服务

我们开始用Spring Boot构建一个认证服务。

2.1 Maven配置

Maven依赖配置如下：

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-jdbc</artifactId>
</dependency>
<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<scope>runtime</scope>
</dependency>
<dependency>
<groupId>org.springframework.security.oauth</groupId>
<artifactId>spring-security-oauth2</artifactId>
<version>${oauth.version}</version>
</dependency>
```

注意我们采用了spring-jdbc和MySQL，因为我们会使用jdbc来实现token store。

2.2. @EnableAuthorizationServer

配置用于管理access tokens的认证服务：

```
@Configuration
@EnableAuthorizationServer
public class AuthServerOAuth2Config extends AuthorizationServerConfigurerAdapter {
    @Autowired
    @Qualifier("authenticationManagerBean")
    private AuthenticationManager authenticationManager;

    @Override
    public void configure(
      AuthorizationServerSecurityConfigurer oauthServer)
      throws Exception {
        oauthServer
          .tokenKeyAccess("permitAll()")
          .checkTokenAccess("isAuthenticated()");
    }

    @Override
```

```
    public void configure(ClientDetailsServiceConfigurer clients)
      throws Exception {
        clients.jdbc(dataSource())
               .withClient("sampleClientId")
               .authorizedGrantTypes("implicit")
               .scopes("read")
               .autoApprove(true)
               .and()
               .withClient("clientIdPassword")
               .secret("secret")
               .authorizedGrantTypes(
                 "password","authorization_code", "refresh_token")
               .scopes("read");
    }

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints)
      throws Exception {
        endpoints
          .tokenStore(tokenStore())
          .authenticationManager(authenticationManager);
    }

    @Bean
    public TokenStore tokenStore() {
        return new JdbcTokenStore(dataSource());
    }
}
```

解释：

- 用JdbcTokenStore来存储tokens
- 注册一个采用“implicit”授权方式的客户端
- 注册另一个采用 “password “，“authorization_code” 和 “refresh_token”授权方式的客户端
- 为了使用“password”授权方式，我们需要通过spring的@Autowired注解来注入和使用AuthenticationManager bean

2.3. 数据源配置

配置JdbcTokenStore用到的数据源

```
@Value("classpath:schema.sql")
private Resource schemaScript;

@Bean
public DataSourceInitializer dataSourceInitializer(DataSource dataSource) {
    DataSourceInitializer initializer = new DataSourceInitializer();
    initializer.setDataSource(dataSource);
    initializer.setDatabasePopulator(databasePopulator());
    return initializer;
}

private DatabasePopulator databasePopulator() {
    ResourceDatabasePopulator populator = new ResourceDatabasePopulator();
    populator.addScript(schemaScript);
    return populator;
}

@Bean
public DataSource dataSource() {
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName(env.getProperty("jdbc.driverClassName"));
    dataSource.setUrl(env.getProperty("jdbc.url"));
    dataSource.setUsername(env.getProperty("jdbc.user"));
    dataSource.setPassword(env.getProperty("jdbc.pass"));
    return dataSource;
}
```

注意：使用JdbcTokenStore 时，我们需要初始化数据库并创建相关的表来存储token数据，通过使用DataSourceInitializer 和下面的语句来实现：

```
drop table if exists oauth_client_details;
create table oauth_client_details (
  client_id VARCHAR(255) PRIMARY KEY,
  resource_ids VARCHAR(255),
  client_secret VARCHAR(255),
  scope VARCHAR(255),
  authorized_grant_types VARCHAR(255),
```

```
  web_server_redirect_uri VARCHAR(255),
  authorities VARCHAR(255),
  access_token_validity INTEGER,
  refresh_token_validity INTEGER,
  additional_information VARCHAR(4096),
  autoapprove VARCHAR(255)
);

drop table if exists oauth_client_token;
create table oauth_client_token (
  token_id VARCHAR(255),
  token LONG VARBINARY,
  authentication_id VARCHAR(255) PRIMARY KEY,
  user_name VARCHAR(255),
  client_id VARCHAR(255)
);

drop table if exists oauth_access_token;
create table oauth_access_token (
  token_id VARCHAR(255),
  token LONG VARBINARY,
  authentication_id VARCHAR(255) PRIMARY KEY,
  user_name VARCHAR(255),
  client_id VARCHAR(255),
  authentication LONG VARBINARY,
  refresh_token VARCHAR(255)
);

drop table if exists oauth_refresh_token;
create table oauth_refresh_token (
  token_id VARCHAR(255),
  token LONG VARBINARY,
  authentication LONG VARBINARY
);

drop table if exists oauth_code;
create table oauth_code (
  code VARCHAR(255), authentication LONG VARBINARY
);

drop table if exists oauth_approvals;
create table oauth_approvals (
    userId VARCHAR(255),
    clientId VARCHAR(255),
    scope VARCHAR(255),
    status VARCHAR(10),
    expiresAt TIMESTAMP,
    lastModifiedAt TIMESTAMP
);

drop table if exists ClientDetails;
create table ClientDetails (
  appId VARCHAR(255) PRIMARY KEY,
  resourceIds VARCHAR(255),
  appSecret VARCHAR(255),
  scope VARCHAR(255),
  grantTypes VARCHAR(255),
  redirectUrl VARCHAR(255),
  authorities VARCHAR(255),
  access_token_validity INTEGER,
  refresh_token_validity INTEGER,
  additionalInformation VARCHAR(4096),
  autoApproveScopes VARCHAR(255)
);
```

2.4. 安全权限配置

最后，为认证服务增加安全权限控制功能。 当客户端程序需要获取Access Token时，会执行下面一个简单的from-login驱动的认证过程：

```
@Configuration
public class ServerSecurityConfig extends WebSecurityConfigurerAdapter {

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
      throws Exception {
        auth.inMemoryAuthentication()
          .withUser("john").password("123").roles("USER");
    }

    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean()
      throws Exception {
```

```
        return super.authenticationManagerBean();
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.authorizeRequests()
            .antMatchers("/login").permitAll()
            .anyRequest().authenticated()
            .and()
            .formLogin().permitAll();
    }
}
```

需要注意对于oauth2的Password flow模式，from-login配置不是必须的，只对Implicit flow是必须的。

3. Resource 服务

Resource 服务用于提供REST API。

3.1. Maven 配置

Resource 服务的Maven配置与前面的认证服务的Maven配置相同。

3.2. Token Store 配置

TokenStore 采用与前面的认证服务相同的数据源：

```
@Autowired
private Environment env;

@Bean
public DataSource dataSource() {
    DriverManagerDataSource dataSource = new DriverManagerDataSource();
    dataSource.setDriverClassName(env.getProperty("jdbc.driverClassName"));
    dataSource.setUrl(env.getProperty("jdbc.url"));
    dataSource.setUsername(env.getProperty("jdbc.user"));
    dataSource.setPassword(env.getProperty("jdbc.pass"));
    return dataSource;
}

@Bean
public TokenStore tokenStore() {
    return new JdbcTokenStore(dataSource());
}
```

为了简化起见，虽然认证服务和Resource 服务是两个独立的应用程序，但是用了同一个数据库，原因是Resource 服务需要验证认证服务中生成的access token。

3.3. Remote Token Service

除了在Resource服务中使用TokenStore 之外，还可以使用RemoteTokeServices：

```
@Primary
@Bean
public RemoteTokenServices tokenService() {
    RemoteTokenServices tokenService = new RemoteTokenServices();
    tokenService.setCheckTokenEndpointUrl(
      "http://localhost:8080/spring-security-oauth-server/oauth/check_token");
    tokenService.setClientId("fooClientIdPassword");
    tokenService.setClientSecret("secret");
    return tokenService;
}
```

注意：

- RemoteTokenService会使用认证服务中的CheckTokenEndPoint去验证AccessToken并获取Authentication对象.
- 访问地址为：认证服务器的URL +”/oauth/check_token“
- 认证服务可以使用任意的TokenStore类型，包括 [JdbcTokenStore, JwtTokenStore, ...] ，不会影响到RemoteTokenService 或 Resource 服务

3.4. 一个简单的 Controller

下面用一个简单的Controller来提供Foo 接口

```
@Controller
public class FooController {

    @PreAuthorize("#oauth2.hasScope('read')")
    @RequestMapping(method = RequestMethod.GET, value = "/foos/{id}")
    @ResponseBody
    public Foo findById(@PathVariable long id) {
        return
          new Foo(Long.parseLong(randomNumeric(2)), randomAlphabetic(4));
    }
}
```

使用这个接口的客户端需要具有“read” scope 。 同时需要启用全局安全权限控制，并且需要配置MethodSecurityExpressionHandler：

```
@Configuration
@EnableResourceServer
@EnableGlobalMethodSecurity(prePostEnabled = true)
public class OAuth2ResourceServerConfig
  extends GlobalMethodSecurityConfiguration {

    @Override
    protected MethodSecurityExpressionHandler createExpressionHandler() {
        return new OAuth2MethodSecurityExpressionHandler();
    }
}
```

Foo接口的实现如下：

```
public class Foo {
    private long id;
    private String name;
}
```

3.5. Web 配置

为API提供一个基础的web配置：

```
@Configuration
@EnableWebMvc
@ComponentScan({ "org.baeldung.web.controller" })
public class ResourceWebConfig extends WebMvcConfigurerAdapter {}
```

4. 前端程序 – Password Flow

下面来看一下用AngularJS实现的简单的前端客户端程序。 我们将采用OAuth2 Password flow，客户端的凭证信息将会暴露在前端（**这是不安全的**）。 首先创建两个简单的页面 - “index” 和 “login”，用户在页面上录入凭证信息，前端的JS程序用这些凭证信息去认证服务上获取Access Token。

4.1. 登录页面

```
<body ng-app="myApp" ng-controller="mainCtrl">
<h1>Login</h1>
<label>Username</label><input ng-model="data.username"/>
<label>Password</label><input type="password" ng-model="data.password"/>
<a href="#" ng-click="login()">Login</a>
</body>
```

4.2. 获取 Access Token

下面来看一下怎么获取 access token：

```
var app = angular.module('myApp', ["ngResource","ngRoute","ngCookies"]);
app.controller('mainCtrl',
  function($scope, $resource, $http, $httpParamSerializer, $cookies) {

    $scope.data = {
        grant_type:"password",
        username: "",
        password: "",
        client_id: "clientIdPassword"
```

```
    };
    $scope.encoded = btoa("clientIdPassword:secret");

    $scope.login = function() {
        var req = {
            method: 'POST',
            url: "http://localhost:8080/spring-security-oauth-server/oauth/token",
            headers: {
                "Authorization": "Basic " + $scope.encoded,
                "Content-type": "application/x-www-form-urlencoded; charset=utf-8"
            },
            data: $httpParamSerializer($scope.data)
        }
        $http(req).then(function(data){
            $http.defaults.headers.common.Authorization =
              'Bearer ' + data.data.access_token;
            $cookies.put("access_token", data.data.access_token);
            window.location.href="index";
        });
   }
});
```

解释：

- 通过提交一个 POST 请求到 “/oauth/token” 来获取Access Token
- 使用客户端凭证和 Basic Auth
- 通过 url encode 对用户凭证、客户端 id 和 grant type进行编码
- 得到Access Token后将其存放到cookie中

4.3. Index 页面

```
<body ng-app="myApp" ng-controller="mainCtrl">
<h1>Foo Details</h1>
<label>ID</label><span>{{foo.id}}</span>
<label>Name</label><span>{{foo.name}}</span>
<a href="#" ng-click="getFoo()">New Foo</a>
</body>
```

4.4. 对客户端请求进行授权

因为Resource服务需要使用access token对客户端请求进行授权验证，我们用access token在http头中增加一个简单的authorization header：

```
var isLoginPage = window.location.href.indexOf("login") != -1;
if(isLoginPage){
    if($cookies.get("access_token")){
        window.location.href = "index";
    }
} else{
    if($cookies.get("access_token")){
        $http.defaults.headers.common.Authorization =
          'Bearer ' + $cookies.get("access_token");
    } else{
        window.location.href = "login";
    }
}
```

如果没找到cookie，将重定向到login页面。

5. 前端程序 – Implicit Grant

下面来看一下采用implicit grant的客户端程序。 这个程序是一个单独的模块，采用oauth2的implicit grant flow，从认证服务中获取access token，然后用这个access token去访问Resource服务。

5.1. Maven 配置

这是pom.xml：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
```

```
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
```

这儿不需要spring的oauth模块，我们将使用AngularJS的OAuth-ng directive，以implicit grant flow方式去访问oauth2 认证服务。

5.2. Web 配置

```
@Configuration
@EnableWebMvc
public class UiWebConfig extends WebMvcConfigurerAdapter {
    @Bean
    public static PropertySourcesPlaceholderConfigurer
      propertySourcesPlaceholderConfigurer() {
        return new PropertySourcesPlaceholderConfigurer();
    }

    @Override
    public void configureDefaultServletHandling(
      DefaultServletHandlerConfigurer configurer) {
        configurer.enable();
    }

    @Override
    public void addViewControllers(ViewControllerRegistry registry) {
        super.addViewControllers(registry);
        registry.addViewController("/index");
        registry.addViewController("/oauthTemplate");
    }

    @Override
    public void addResourceHandlers(ResourceHandlerRegistry registry) {
        registry.addResourceHandler("/resources/**")
          .addResourceLocations("/resources/");
    }
}
```

5.3. Home 页面

OAuth-ng directive需要以下参数:

- site: 认证服务的URL
- client-id: 客户端应用的 client id
- redirect-uri: 从认证服务获取到access token后，重定向到此URI
- scope: 从认证服务获取到的权限
- template: AngularJS的页面模板

```
<body ng-app="myApp" ng-controller="mainCtrl">
<oauth
  site="http://localhost:8080/spring-security-oauth-server"
  client-id="clientId"
  redirect-uri="http://localhost:8080/spring-security-oauth-ui-implicit/index"
  scope="read"
  template="oauthTemplate">
</oauth>

<h1>Foo Details</h1>
<label >ID</label><span>{{foo.id}}</span>
<label>Name</label><span>{{foo.name}}</span>
</div>
<a href="#" ng-click="getFoo()">New Foo</a>

<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular.min.js">
</script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular-resource.min.js">
</script>
<script src="http://ajax.googleapis.com/ajax/libs/angularjs/1.3.14/angular-route.min.js">
</script>
<script src="https://cdnjs.cloudflare.com/ajax/libs/ngStorage/0.3.9/ngStorage.min.js">
</script>
<script th:src="@{/resources/oauth-ng.js}"></script>
</body>
```

现在说明如何用 OAuth-ng directive来获取Access Token，这是一个简单oauthTemplate.html:

```
<div>
  <a href="#" ng-show="show=='logged-out'" ng-click="login()">Login</a>
  <a href="#" ng-show="show=='denied'" ng-click="login()">Access denied. Try again.</a>
</div>
```

5.4. AngularJS 应用

```
var app = angular.module('myApp', ["ngResource","ngRoute","oauth"]);
app.config(function($locationProvider) {
  $locationProvider.html5Mode({
      enabled: true,
      requireBase: false
    }).hashPrefix('!');
});

app.controller('mainCtrl', function($scope,$resource,$http) {
    $scope.$on('oauth:login', function(event, token) {
        $http.defaults.headers.common.Authorization= 'Bearer ' + token.access_token;
    });

    $scope.foo = {id:0 , name:"sample foo"};
    $scope.foos = $resource(
      "http://localhost:8080/spring-security-oauth-resource/foos/:fooId",
      {fooId:'@id'});
    $scope.getFoo = function(){
        $scope.foo = $scope.foos.get({fooId:$scope.foo.id});
    }
});
```

获取到Access Token后，通过http头的Authorization header来访问Resrouce服务中提供的接口服务。

6. 总结

至此我们阐述了如何使用OAuth2来为应用程序提供安全权限控制功能。 本文的所有实例代码在 the github project 中 - 这是一个eclipse项目，可以直接导入并运行。

分类：工作日志 字数：2114

Ctrl+Enter 发表评论

橙子先生

1楼 2017/03/08 13:32

认证和业务服务可以做到一起吗？这样做会不会比较复杂

QiHaiYan

2楼 2017/03/08 13:42

> 引用来自“橙子先生”的评论
> 认证和业务服务可以做到一起吗？这样做会不会比较复杂

oauth除了有对用户的认证，还有针对客户端的认证：grant_type=client_credentials

橙子先生

3楼 2017/03/08 13:49

> 引用来自“QiHaiYan”的评论
> > 引用来自“橙子先生”的评论
> > 认证和业务服务可以做到一起吗？这样做会不会比较复杂
>
> oauth除了有对用户的认证，还有针对客户端的认证：grant_type=client_credentials

为什么要针对客户端?

QiHaiYan

4楼 2017/03/08 14:14

> 引用来自“橙子先生”的评论
> > 引用来自“QiHaiYan”的评论
> > > 引用来自“橙子先生”的评论
> > > 认证和业务服务可以做到一起吗？这样做会不会比较复杂
> >
> > oauth除了有对用户的认证，还有针对客户端的认证：grant_type=client_credentials
>
> 为什么要针对客户端?

回复@橙子先生：因为有些接口是不需要用户登录也可以调用的。这儿的客户端(client_id)是oauth里的一个定义，不是普通意义上的客户端。

橙子先生

5楼 2017/03/08 14:20

> 引用来自“QiHaiYan”的评论
> > 引用来自“橙子先生”的评论
> > > 引用来自“QiHaiYan”的评论
> > > > 引用来自“橙子先生”的评论
> > > > 认证和业务服务可以做到一起吗？这样做会不会比较复杂
> > >
> > > oauth除了有对用户的认证，还有针对客户端的认证：grant_type=client_credentials
> >
> > 为什么要针对客户端?
>
> 回复@橙子先生：因为有些接口是不需要用户登录也可以调用的。这儿的客户端(client_id)是oauth里的一个定义，不是普通意义上的客户端。

好的，多谢指点，请问有什么具体的书或者系列文章吗?

QiHaiYan

6楼 2017/03/08 14:31

> 引用来自“橙子先生”的评论
> > 引用来自“QiHaiYan”的评论
> > > 引用来自“橙子先生”的评论
> > > > 引用来自“QiHaiYan”的评论
> > > > > 引用来自“橙子先生”的评论
> > > > > 认证和业务服务可以做到一起吗？这样做会不会比较复杂
> > > >
> > > > oauth除了有对用户的认证，还有针对客户端的认证：grant_type=client_credentials

> 为什么要针对客户端?
>
> 回复@橙子先生 : 因为有些接口是不需要用户登录也可以调用的。这儿的客户端(client_id)是oauth里的一个定义，不是普通意义上的客户端。

好的，多谢指点，请问有什么具体的书或者系列文章吗?

回复@橙子先生 : https://my.oschina.net/hiease/blog/742156 这是一篇介绍oauth的文章