7. 第 7 章 微服务安全

本章内容

- 学习在微服务环境安全问题的重要性
- 了解 OAuth2 标准
- 创建和配置一个基于 Spring 的 OAuth2 服务
- 使用 OAuth2 执行用户认证和授权
- 使用 OAuth2 保护你的 Spring 微服务
- 在服务之间传递你的 OAuth2 访问令牌

安全 ,提到这个词常常会引起听到它的开发者的不自觉的呻吟。你会听到他们低声抱怨，并在他们的呼吸下诅咒，“它是迟钝的，难以理解，甚至更难调试。”然而你找不到任何开发人员（除了那些没有经验的开发人员），说他们不担心安全。

一个安全的应用涉及多层次的保护，包括：

- 确保适当的用户控制就位，以便你可以验证用户是他们所说的用户，并允许他们做他们想做的事情。
- 保持服务的基础设施正在运行最新的补丁，以尽量减少漏洞的风险。
- 实现网络访问控制，使服务只能通过定义良好的端口访问，并可访问少量授权服务器。

本章只讨论清单中的第一个要点：如何验证调用你的微服务的用户，他们说他们是谁，并确定它们是否是被授权向你的微服务发起它们的请求。另外两个主题是非常宽泛的安全主题，超出了本书的范围。

实现授权和认证控制，你要用 Spring Cloud security 和 OAuth2（开放认证）标准来

保护你的基于 Spring 的服务。OAuth2 是基于令牌的安全框架，允许用户使用第三方认证服务验证自己。如果用户成功认证 ,他们将提交一个令牌 ,该令牌必须与每个请求一起发送。然后可以将令牌返回给身份验证服务验证。OAuth2 背后的主要目标是当多个服务被调用来满足一个用户的请求 ,用户可以被每个服务认证而无需向每个处理他们请求的服务出示凭证。Spring Boot 和 Spring Cloud 都提供了 OAuth2 服务的开箱即用的实现，使将 OAuth2 security 整合到你的服务极其方便。

注意：在这一章中，我们将告诉你如何使用 OAuth2 保护你的微服务；然而，一个成熟的 OAuth2 实现还需要一个前端 Web 应用程序输入你的用户凭证。我们将不会讨论如何创建前端应用程序，因为超出了一本关于微服务书籍的范围。相反，我们将使用 REST 客户端，像 POSTMAN，来模拟凭证的呈现。关于如何创建前端应用程序的一个很好的教程，我建议你看看下面的 Spring 教程：

https://spring.io/blog/2015/02/03/sso-with-oauth2-angular-js-and-springsecurity-part-v。

OAuth2 背后真正的力量是它使应用程序开发人员能够轻松地与第三方云供应商集成，并使用这些服务进行用户认证和授权而不必向第三方服务不断地传输用户的凭证。云供应商，如 Facebook，GitHub 和 Salesforce 都支持 OAuth2 作为标准。

在我们进入使用 OAuth2 保护我们的服务的技术细节之前，让我们浏览一下 OAuth2 架构。

7.1. 介绍 OAuth2

OAuth2 是基于令牌的安全认证和授权框架，它将安全分成四个构成要素。这四个构成要素分别是：

- 受保护的资源：你想保护的资源（在我们的例子中，即微服务），确保只有经过身份验证的用户拥有可以访问的适当授权。
- 资源所有者：资源所有者定义什么应用程序可以调用他们的服务，允许那个用户访问服务，以及他们可以使用服务做什么。资源所有者注册的每个应用程序都将得到一个应用程序名称，该名称标识应用程序以及应用程序的密钥。应用程序的名称和密钥是凭证的一部分，它们在认证 OAuth2 令牌时被传输。
- 应用程序：这是一个将代表用户调用服务的应用程序。毕竟，用户很少直接调用服务。相反，他们依靠应用程序为他们做工作。
- OAuth2 认证服务器：OAuth2 认证服务器是应用程序和被消费的服务之间的中介。OAuth2 服务器允许用户验证自己而不必传输他们的用户凭证到代表用户调用的应用程序的每个服务。

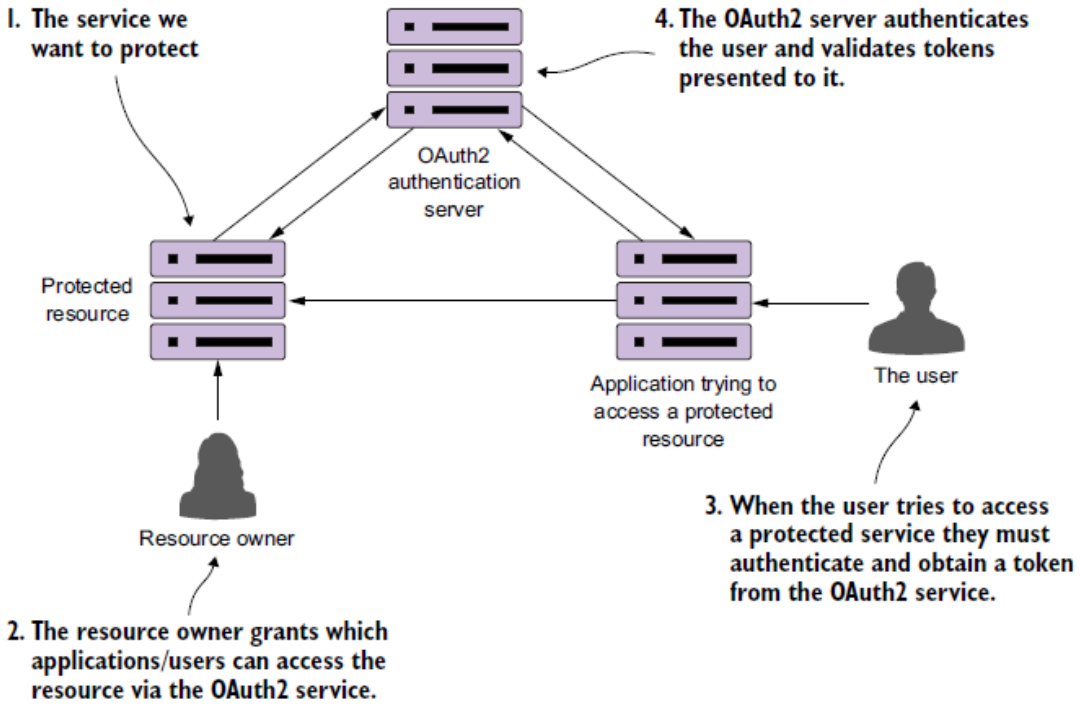

图 7.1 OAuth2 允许用户认证不必出示凭证。

① *The service we want to protect*

我们想要保护的服务

② *The resource owner grants which applications/users can access the resource via the OAuth2 service.*

资源的所有者授权应用程序/用户可以通过 OAuth2 服务访问资源。

③ *When the user tries to access a protected service they must authenticate and obtain a token from the OAuth2 service.*

当用户试图访问受保护的服务时，他们必须身份验证并从 OAuth2 服务获得令牌。

④ *The OAuth2 server authenticates the user and validates tokens presented to it.*

OAuth2 服务器对用户进行身份验证并验证了令牌。

⑤ *Protected resource*

受保护的资源

⑥ *Protected resource*

资源所有者

⑦ *Application trying to access a protected resource*

试图访问受保护资源的应用程序

⑧ *OAuth2 authentication server*

OAuth2 认证服务器

这四个构成要素交互作用来认证用户身份。用户只需出示他们的凭证。如果他们成功认证，则发出一个可从服务传递到服务的身份验证令牌。这如图 7.1 所示。OAuth2 是基于令牌的安全框架。用户通过 OAuth2 服务器提供的应用程序凭证（它们用于访问资源）进行身份认证。如果用户的凭证是有效的，OAuth2 服务器提供一个令牌，该令牌在每一次服务被用户的应用程序试图访问受保护的资源（微服务）时可以被出示。

然后，受保护的资源可以与 OAuth2 服务器通信，来确定令牌的有效性和检索用户已被分配给他们的角色。角色被用于将相关用户分组在一起，并定义用户可以访问的资源。本章的目的，你要使用 OAuth2 和角色来定义服务端点和用户可以调用端点的什么 HTTP 谓词。

Web 服务安全是一个极其复杂的问题。你要明白谁会调用你的服务（企业网络中的内部用户，外部用户），他们是如何去调用你的服务（内部 web 客户端，移动设备，你的企业网络外的 Web 应用程序），他们使用你的代码将会采取什么动作。OAuth2 允许你在这些不同的情况下，通过不同的认证模式（称为授权），保护你的基于 REST 的服务。OAuth2 规范有四种类型的授权：

- 密码
- 客户凭证
- 授权码
- 隐式

我们将不涉及这些权限授予类型中的每一种，也不为每个类型提供代码示例。那样的话，涵盖在一章的内容实在太多了。相反，我会做以下事情：

- 讨论你的微服务如何通过一个简单的 OAuth2 授权类型（密码授权类型）使用 OAuth2。
- 使用 JavaScript web tokens 提供了更强大的 OAuth2 解决方案并建立 OAuth2 令牌信息编码标准。
- 了解其他构建微服务需要考虑的安全注意事项。

我在附录 B 中提供了其它的 OAuth2 授权类型的概述材料，“OAuth2 授权类型”。如果你有兴趣更详细的了解 OAuth2 规范和如何实现所有授权类型，我极力推荐 Justin Richer

和 Antonio Sanso 的书，OAuth2 in Action (Manning, 2017)，这本书对 OAuth2 进行了全面的解释。

7.2. 从小事做起：使用 Spring 和 OAuth2 保护一个单独的端点

了解如何创建 OAuth2 的身份验证和授权部分，你要实现 OAuth2 密码授权类型。要实现此授权，你需要做以下事情：

- 创建一个基于 Spring Cloud 的 OAuth2 认证服务。
- 注册一个仿制的 EagleEyeUI 应用程序作为一个授权的应用程序，使用 OAuth2 服务可以进行身份验证和授权用户身份。
- 使用 OAuth2 密码授权来保护你的 EagleEye 服务。你不会为 EagleEye 构建 UI，所以你会使用 POSTMAN 模拟一个用户登录，用你的 EagleEyeOAuth2 服务来进行身份验证。
- 保护许可和组织服务，以便它们只能由经过身份验证的用户调用。

7.2.1. 配置 EagleEye OAuth2 认证服务

像这本书的章节的所有示例，你的 OAuth2 认证服务将是另一个 Spring Boot 服务。认证服务将对用户凭据进行身份验证并发出令牌。每当用户试图通过认证服务访问受保护的服务，认证服务将验证它发布的 OAuth2 令牌并确认它没有过期。认证服务相当于图 7.1 中的认证服务。

开始之前，你要做两件事：

- 为你的引导类引入需要的适当的 Maven 构建依赖
- 引导类将作为服务的入口点

你可以在 authenticationservice 目录找到认证服务的所有示例代码。为建立一个 OAuth2 认证服务器，你需要在 authentication-service/pom.xml 文件添加以下 Spring Cloud 依赖：

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-security</artifactId>
</dependency>

<dependency>
   <groupId>org.springframework.security.oauth</groupId>
   <artifactId>spring-security-oauth2</artifactId>
</dependency>
```

第一个依赖是 spring-cloud-security，它引入了传统的 Spring 和 Spring Cloud security 库。第二个依赖是 spring-security-oauth2，从 Spring OAuth2 库拉取。

现在，已经定义了 Maven 依赖，你可以在引导类继续工作。这个类可以在 authentication-service/src/main/java/com/thoughtmechanix/authentication/ Application.java 类中找到。下面的清单显示了 Application.java 类的代码。

清单 7.1 认证服务引导类

```
@SpringBootApplication
@RestController
@EnableResourceServer
@EnableAuthorizationServer
public class Application {
    @RequestMapping(value = { "/user" }, produces = "application/json")
    public Map<String, Object> user(OAuth2Authentication user) {
      Map<String, Object> userInfo = new HashMap<>();
      userInfo.put("user",
                   user.getUserAuthentication().getPrincipal());
      userInfo.put("authorities",
                   AuthorityUtils.authorityListToSet(
                   user.getUserAuthentication(). getAuthorities()));
      return userInfo;
    }

    public static void main(String[] args) {
      SpringApplication.run(Application.class, args);
```

①用来告诉 Spring Cloud，此服务将作为一个 OAuth2 服务

②在本章后面使用，以检索关于用户的信息。

```
    }
}
```

清单中要注意的第一件事是@EnableAuthorizationServer 注解。这个注解告诉 Spring Cloud，这个服务将被作为一个 OAuth2 服务和添加一些基于 REST 的端点，这些端点将被用于 OAuth2 认证和授权处理。

第二件事是你会在清单 7.1 中看到一个称为/user(它映射到/auth/user)端点的加入。在本章的后面，当你试图访问受 OAuth2 保护的服务，你将会使用这个端点。这个端点被受保护的服务调用来验证 OAuth2 访问令牌和检索用户访问受保护服务的指定角色。我将在本章后面更详细地讨论这个端点。

7.2.2. 在 OAuth2 服务注册客户端应用

此时，你拥有一个认证服务，但尚未在身份验证服务器中定义任何的应用程序、用户或角色。你可以开始在你的认证服务注册 EagleEye 应用。为了这样做，你将在你的认证服务创建一个称为 authentication-service/src/main/java/com/thoughtmechanix/authentication/security/OAuth2Config.java 的额外类。

这个类将定义哪些应用程序在你的 OAuth2 认证服务注册。需要注意的是，仅仅因为一个应用程序在你的 OAuth2 服务被注册，但这并不意味着服务可以访问任意受保护的资源。

关于认证与授权

我经常发现开发人员“混合搭配”术语认证和授权的含义。认证是用户通过提供凭证来证明他们是谁的行为。授权决定用户是否被允许做他们想做的事情。例如，用户 Jim 可以通过提供用户 ID 和密码来证明自己的身份，但他可能没有权限查看诸如工资数据之类的敏感数据。为了我们讨论的目的，在授权发生之前必须对用户进行身份验证。

OAuth2Config 类 OAuth2 服务知道的应用程序和用户凭证。在下面的清单你可以看到 OAuth2Config.java 的代码。

清单 7.2 OAuth2Config 服务定义了什么应用程序可以使用你的服务

```
@Configuration
public class OAuth2Config extends AuthorizationServerConfigurerAdapter {

   @Autowired
   private AuthenticationManager authenticationManager;
   @Autowired
   private UserDetailsService userDetailsService;

   @Override
   public void configure(ClientDetailsServiceConfigurer clients) throws
       Exception {
     clients.inMemory()
         .withClient("eagleeye")
         .secret("thisissecret")
         .authorizedGrantTypes(
             ➥ "refresh_token",
             ➥ "password",
             ➥ "client_credentials".scopes("webclient","mobileclient");
   }

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

①重载 configure()方法。这定义了哪些客户端将被你的服务注册。

②该方法定义了在 AuthenticationServerConfigurer 使用的不同组件。这个代码告诉 Spring 使用 Spring 自带的默认身份验证管理器和用户详细信息服务。

在代码中注意到的第一件事是你扩展了 Spring 的 AuthenticationServerConfigurer 类，然后使用@Configuration 注解标记类。这个 AuthenticationServerConfigurer 类是 Spring 安全的核心部分。它提供了实现密钥认证和授权功能的基本机制。对于 OAuth2Config 类你将要重写两个方法。第一个方法，configure()，被用于定义什么客户端应用程序在你的认证服务注册。这个 configure()方法需要一个称为 clients，类型为 ClientDetailsServiceConfigurer 的参数。让我们开始更详细一点了解在 configure()方法

的代码。你在这个方法的第一件事就是注册客户端应用程序，它可以访问受 OAuth2 服务保护的服务。我在这里用最广泛的术语“访问”，因为你可以通过检查被调用服务的用户是否被授权采取他们要采取的行动来控制客户端应用程序的用户稍后可以做什么：

```
clients.inMemory()
    .withClient("eagleeye")
    .secret("thisissecret")
    .authorizedGrantTypes("password",    "client_credentials")
    .scopes("webclient","mobileclient");
```

ClientDetailsServiceConfigurer 类支持应用程序信息的两种不同的存储类型：内存存储和 JDBC 存储。在这个例子中，你将使用 clients.inMemory()存储。

这两个称为 withClient()和 secret()的方法提供应用程序（EagleEye）的名称，该应用程序连同一把密钥（密码，thisissecret）一起注册，当 EagleEye 应用程序调用 OAuth2 服务器来接收一个 OAuth2 访问令牌时密钥将被提交。

下一个方法，authorizedGrantTypes()，传递的是一个逗号分隔的授权批准类型的列表，会在你的 OAuth2 服务支持列表。在你的服务中，你将支持密码和客户凭证授权。scopes()方法用于定义当他们询问你的 OAuth2 服务器访问令牌时，调用的应用程序可以运行的边界。

例如，Thoughtmechanix 可以提供相同的应用程序的两个不同的版本，一个 Web 的应用程序和移动端应用程序。这些应用程序可以使用相同的客户端名称和密钥，通过 OAuth2 服务器请访问受保护的资源。然而，当应用程序需要一个密钥时，他们需要定义它们所运行的特定范围。通过定义作用域，你可以编写特定于客户端应用程序正在工作的范围的授权规则。

例如，你可能有一个用户既可以访问 EagleEye 应用的 Web 客户端应用程序，也可以访问移动端应用程序。应用程序的每个版本都做以下事情：

- 提供相同的功能

- 是一个“受信任的应用程序”，ThoughtMechanix 同时拥有 EagleEye 的前端应用程序和后端用户服务 。

因此，你将使用相同的应用名称和密钥注册 EagleEye 应用，而 Web 应用程序只会在“webclient”范围使用，而移动端应用程序将在“mobileclient”范围使用。通过使用作用域，你可以在受保护的服务中定义授权规则，这些规则可以限制应用程序客户根据他们登录的应用程序可以采取什么操作。这将不考虑用户拥有什么权限。例如，你可能希望根据用户在公司的内部网络中使用浏览器而不是在移动设备上浏览应用程序来限制用户可以看到哪些数据。在处理敏感客户信息（例如健康记录或税务信息）时，基于数据访问机制限制数据的实践是常见的。

此时，你已经注册了一个单独的应用程序，EagleEye，以及 OAuth2 服务器。但是，由于你使用的是密码授权，所以在开始之前，你需要为这些用户设置用户账户和密码。

7.2.3. 配置 EagleEye 用户

你已经定义并存储了应用程序名称和密钥。现在你将创建个人用户凭证和它们所属于的角色。用户角色将用于定义一组用户可以使用一个服务所做的操作。

Spring 可以从内存数据存储、JDBC 支持的关系数据库或 LDAP 服务器中存储和检索用户信息（个人用户的凭证和分配给用户的角色）。

> ***注意***：就定义而言，我在这里要注意一些事项。Spring 的 OAuth2 应用信息可以将它的数据存储在内存或关系数据库中。Spring 用户凭证和安全角色可以存储在内存数据库、关系数据库或 LDAP（活动目录）服务器中。为保持事情简单，因为我们的主要目的是了解 OAuth2，你将使用一个内存数据存储。

在本章的示例代码，你要使用一个内存数据存储定义用户角色。你要定义两个用户账户：

john.carnell 和 william.woodward。john.carnell 将拥有 USER 角色，william.woodward 将拥有 ADMIN 角色。

为了配置你的 OAuth2 服务器进行用户 ID 认证，你需要创建一个新的类：authentication-service/src/main/com/thoughtmechanix/authentication/security/WebSecurityConfigurer.java。下面的清单显示了该类的代码。

清单 7.3 定义应用程序的用户 ID、密码和角色

```
package com.thoughtmechanix.authentication.security;

@Configuration
public class WebSecurityConfigurer
    ➥extends
    WebSecurityConfigurerAdapter {

    @Override
    @Bean
    public AuthenticationManager authenticationManagerBean()
        throws Exception{
      return super.authenticationManagerBean();
    }

    @Override
    @Bean
    public UserDetailsService userDetailsServiceBean() throws Exception {
      return super.userDetailsServiceBean();
    }

    @Override
    protected void configure(
        AuthenticationManagerBuilder auth)
      ➥ throws Exception {
      auth.inMemoryAuthentication()
        .withUser("john.carnell")
        ➥ .password("password1")
        ➥ .roles("USER")
        .and()
        ➥ .withUser("william.woodward")
        .password("password2")
        ➥ .roles("USER", "ADMIN");
    }
```

①扩展了 Spring Security 核心的 WebSecurityConfigurerAdapter

②Spring Security 使用 AuthenticationManagerBean 来处理身份验证。

③Spring Security 使用 UserDetailsService 来处理 Spring Security 返回的用户信息。

④configure() 方法将定义用户、密码和角色。

```
}
```

与 Spring 安全框架的其他部分一样，要创建用户（以及他们的角色），首先扩展 WebSecurityConfigurerAdapter 类并用@Configuration 注解标记它。Spring 安全的实现方式类似于如何将乐高积木组合起来构建一个玩具汽车或模型。因此，你需要为 OAuth2 服务器提供对用户进行身份验证和返回认证用户的用户信息的机制。这是通过在 Spring WebSecurityConfigurerAdapter 实现中定义两个 bean 完成的：authenticationManagerBean() 和 userDetailsServiceBean()。通过使用父类 WebSecurityConfigurerAdapter 默认的验证方法 authenticationManagerBean()和 userDetailsServiceBean()来暴露这两个 bean。

从清单 7.2 中你会记得，这些 bean 被注入到在 OAuth2Config 类显示的 configure(AuthorizationServerEndpointsConfigurer endpoints)方法：

```
public void configure(
   AuthorizationServerEndpointsConfigurer endpoints)
      ➥ throws Exception {
        endpoints.authenticationManager(authenticationManager)
                 .userDetailsService(userDetailsService);
}
```

这两个 bean 用于配置我们不久将看到的/auth/oauth/token 和/auth/user 端点。

7.2.4. 用户认证

此刻，你有足够的基于 OAuth2 服务器的功能来执行密码授权流程的应用和用户认证。现在，你将模拟用户通过使用 POSTMAN 提交 http://localhost:8901/auth/oauth/token 端点，并提供应用名称、密钥、用户 ID 和密码，获得一个 OAuth2 令牌。

① Endpoint and verb to Spring OAuth2 service

Spring OAuth2 服务的端点和谓词

② Application name

应用程序名称

③ Application secret key

应用密钥

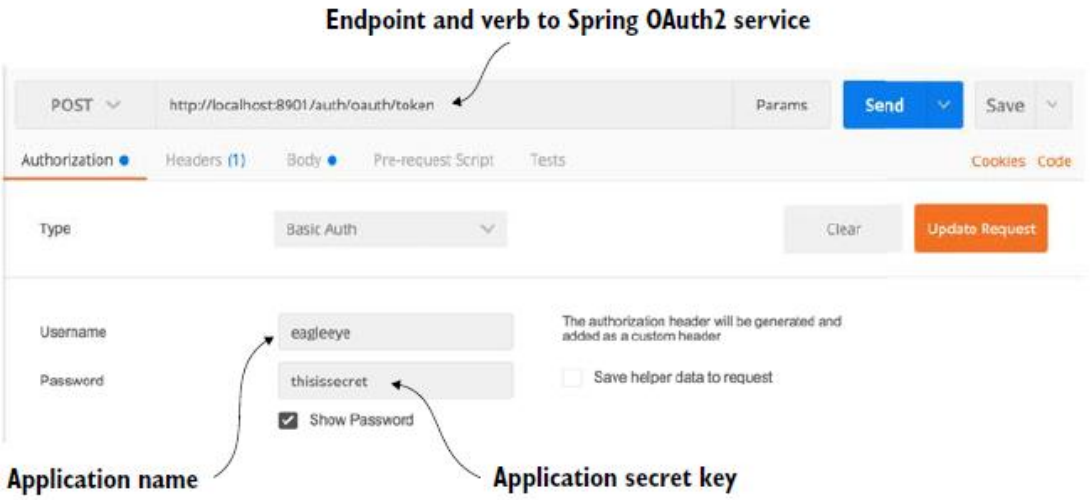

图 7.2 使用应用程序名称和密钥设置基本身份验证

首先，你需要在 POSTMAN 设置应用程序名称和密钥。你将传递这些用于基本身份验证的 OAuth2 服务器端点的元素。图 7.2 显示了如何设置 POSTMAN 来执行基本的身份验证调用。

但是，你还没有准备发起调用来获取令牌。一旦配置了应用程序名称和密钥，就需要将服务中的下列信息作为 HTTP 表单参数传递：

- grant_type：OAuth2 授权你执行的类型。在本例中，你将使用密码授权。
- Scope：应用范围。因为在注册应用程序时只定义了两个合法作用域（webclient 和 mobileclient），传入的值必须是这两个范围中的一个。
- Username：登录用户的名称。
- Password：用户登录密码。

与本书中的其他 REST 调用不同，该列表中的参数不会作为 JavaScript 报文体传入。OAuth2 标准希望所有的参数通过 HTTP 表单参数传递到令牌生成的端点。图 7.3 显示了

如何为 OAuth2 调用配置 HTTP 表单参数。

① HTTP form parameters

HTTP 表单参数

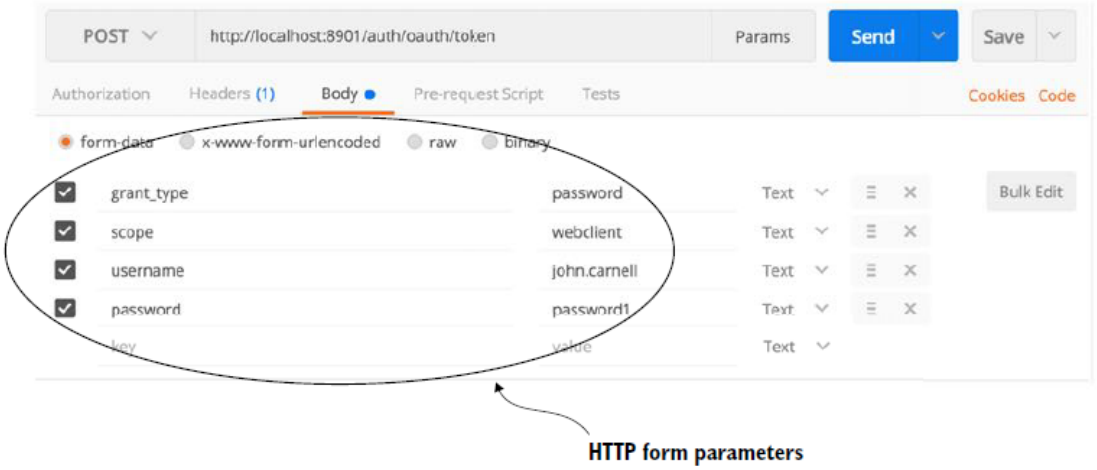

图 7.3 当请求一个 OAuth2 令牌，用户的凭证作为/auth/oauth/token 端点的 HTTP 表单参数传递。

图 7.4 显示了从/auth/oauth/令牌调用返回的 JavaScript 有效负载。

返回的有效负载包含五个属性：

- access_token：OAuth2 令牌随用户向一个受保护资源发出的每个服务调用一起出现。
- token_type：令牌类型。OAuth2 规范允许你定义多个令牌类型。最常用的令牌类型是 bearer 令牌。在本章中，我们不涉及任何其他令牌类型。
- refresh_token：包含一个令牌，在令牌过期后它可以返回到 OAuth2 服务器来重新发布一个令牌。
- expires_in：这是 OAuth2 访问令牌过期前的秒数。Spring 中授权令牌过期的默认值是 12 小时。
- Scope：OAuth2 令牌的有效范围。

① This is the key field. The access_token is the authentication token presented with each call.

这是键的字段域。access_token 是每次调用时提供的身份验证令牌。

② The type of OAuth2 access token being generated

生成 OAuth2 访问令牌的类型

③ The token that is presented when the OAuth2 access token expires and needs to be refreshed

当 OAuth2 访问令到期并需要刷新时该令牌出现

④ The number of seconds before the access token expires

访问令牌到期前的秒数

⑤ The defined scope for which the token is valid

令牌有效的已定义的作用域

图 7.4 客户凭证验证成功后返回的有效载荷

现在，你有一个有效的 OAuth2 访问令牌，我们可以使用你在你的认证服务中创建的/auth/user 端点，来检索与令牌相关的用户信息。在本章后面，将要被保护资源的任何服务都将调用身份验证服务的/auth/user 端点来验证令牌并检索用户信息。

图 7.5 显示了调用/auth/user 端点的结果。当你查看图 7.5 时，请注意 OAuth2 访问令牌是如何作为 HTTP 头传递的。

① OAuth2 access token passed as an HTTP header

作为一个 HTTP 报文头传递的 OAuth2 访问令牌

② The user information looked up based on the OAuth2 token

基于 OAuth2 令牌查找用户信息

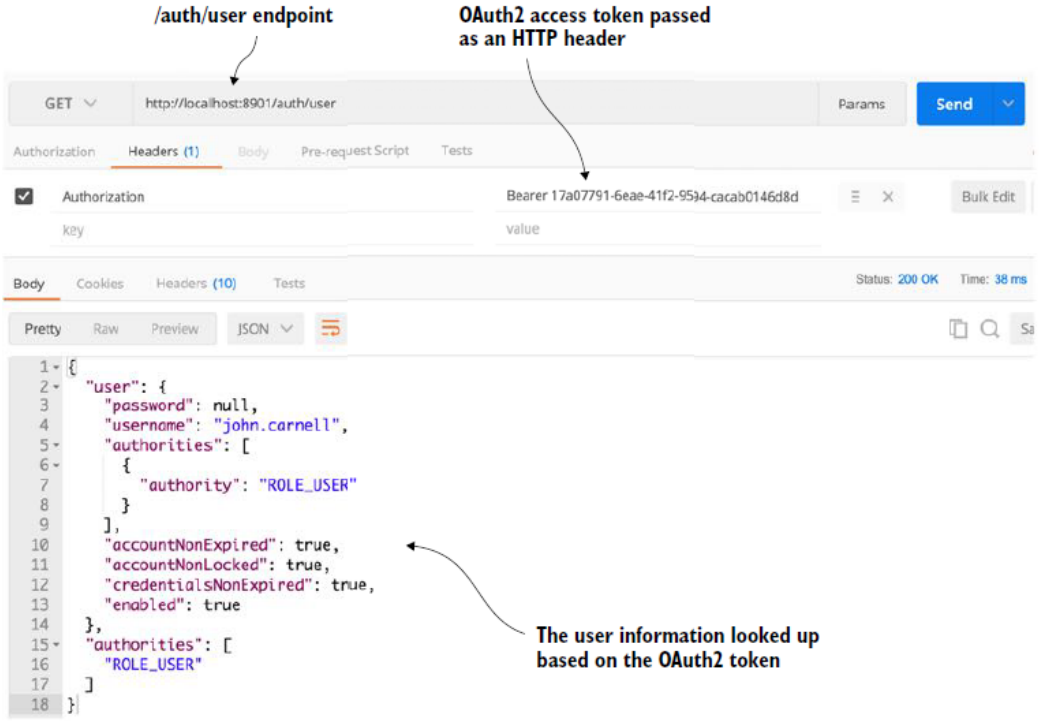

图 7.5 基于发布的 OAuth2 令牌查找用户信息

在图 7.5 中，你以 GET 方式发出了对/auth/user 端点的 HTTP 请求。然而，任何时候你需要通过 OAuth2 访问令牌，调用一个受 OAuth2 保护的端点（包括 OAuth2 端点/auth/user）。要做到这一点，通常需要创建一个称为 Authorization 的 HTTP 头，其值为 Bearer XXXXX。在图 7.5 中调用的情况下，HTTP 头的值是 Bearer e9decabc-165b-4677-9190-2e0bf8341e0b。在图 7.4 中，传入的访问令牌是在你调用/auth/oauth/token 端点时返回。

如果 OAuth2 访问令牌是有效的，/auth/user 端点将返回关于用户的信息，包括分配给他们的角色。例如，在图 7.10 中，你可以看到用户 john.carnell 具有 USER 角色。

注意：Spring 将前缀 ROLE_分配给用户的角色，所以 ROLE_USER 意味着 john.carnell 具有 USER 角色。

7.3. 使用 OAuth2 保护组织服务

一旦你在你的 OAuth2 认证服务已注册一个应用，并创建单独的用户账户和角色，你可以开始探索如何使用 OAuth2 保护资源。虽然 OAuth2 访问令牌的创建和管理是 OAuth2 服务器的职责，但是在 Spring 中，用户角色的定义有权在单个服务级别上执行操作。

要建立受保护的资源，你需要采取以下操作：

- 向你保护的服务添加合适的 Spring Security 和 OAuth2 JAR 包
- 配置服务指向你的 OAuth2 认证服务
- 定义那些资源可以访问服务

让我们先举一个最简单的例子：通过组织服务来设置受保护的资源，并确保它只能由经过身份验证的用户调用。

7.3.1. 为单独的服务添加 Spring Security 和 OAuth2 JAR 包

像往常使用 Spring 微服务一样，你需要向组织服务的 Maven organization-service/pom.xml 文件添加一对依赖。两个依赖被添加：Spring Cloud Security 和 Spring Security OAuth2。Spring Cloud Security JAR 包是核心安全 JAR 包。它们包含在 Spring Cloud 中实现安全的框架代码、注解定义和接口。Spring Security OAuth2 依赖包含实现认证服务所需的所有类。这两个 Maven 依赖项是：

```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-security</artifactId>
</dependency>
<dependency>
```

```
    <groupId>org.springframework.security.oauth</groupId>
    <artifactId>spring-security-oauth2</artifactId>
</dependency>
```

7.3.2. 配置服务指向 OAuth2 认证服务

请记住，一旦你将组织服务设置为受保护的资源，每次对服务发出调用时，调用方必须将服务的 OAuth2 访问令牌包含在 HTTP 头 Authentication 字段中。然后，你的受保护资源必须回调 OAuth2 服务，以确定令牌是否有效。

你在你的组织服务的 application.yml 文件的 security.oauth2.resource.userInfoUri 属性定义回调的 URL。这是用在组织服务的 application.yml 文件的回调配置。

```
security:
  oauth2:
    resource:
      userInfoUri: http://localhost:8901/auth/user
```

从 security.oauth2.resource.userInfoUri 属性中可以看到，回调 URL 是指向/auth/user 端点的。这个端点早些时候在 7.2.4 “用户认证”章节已讨论。

最后，你还需要告诉组织服务它是受保护的资源。同样，通过向组织服务的引导类添加一个 Spring Cloud 注解来实现这一点。组织服务的引导代码在下一个清单中显示，可以在 organization-service/src/main/java/com/thoughtmechanix/organization/ Application.java 类中找到。

清单 7.4 将引导类配置为受保护的资源

```
package com.thoughtmechanix.organization;

import org.springframework.security.oauth2.
    ➥config.annotation.web.configuration.EnableResourceServer;

@SpringBootApplication
@EnableEurekaClient
@EnableCircuitBreaker
@EnableResourceServer
public class Application {
    @Bean
```

①@EnableResourceServer 注解被用来告诉你的微服务是受保护的资源。

```
    public Filter userContextFilter() {
        UserContextFilter userContextFilter = new UserContextFilter();
        return userContextFilter;
    }
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

@EnableResourceServer 注解告诉 Spring Cloud 和 Spring Security，该服务是受保护的资源。@EnableResourceServer 强制执行一个过滤器，拦截向服务发出的所有调用，检查在传入调用的 HTTP 头中是否存在 OAuth2 访问令牌，然后回调到在 security.oauth2.resource.userInfoUri 中定义的回调 URL，看看令牌是否有效。一旦它知道令牌是有效的，@EnableResourceServer 注解也适用于任何访问控制规则，即谁可以访问服务。

7.3.3. 定义那些资源可以访问服务

现在可以开始定义围绕服务的访问控制规则了。定义访问控制规则，你需要扩展一个 Spring ResourceServerConfigurerAdapter 类并重写类的 configure()方法。在组织服务中，你的 ResourceServerConfiguration 类位于 organization-service/src/main/java/com/thoughtmechanix/organization/security/ResourceServerConfiguration.java 中。访问规则可以从极其粗粒度（任何经过身份验证的用户可以访问整个服务）到细粒度（只有具有此角色的应用程序，才允许通过 DELETE 访问此 URL）。

我们讨论了 Spring Security 的访问控制规则的每一个排列，但是我们可以看看几个比较常见的例子。这些示例包括保护资源以便：

- 只有经过身份验证的用户才能访问服务 URL
- 只有具有特定角色的用户才能访问服务 URL

通过已身份验证的用户保护服务

你要做的第一件事是保护组织服务，以便它只能由经过身份验证的用户访问。下面的清单展示了如何将此规则构建到 ResourceServerConfiguration.java 类中。

清单 7.5 限制仅有经过身份验证的用户有权访问

```
package com.thoughtmechanix.organization.security;

@Configuration
public class ResourceServerConfiguration extends
    ResourceServerConfigurerAdapter {

  @Override
  public void configure(HttpSecurity http) throws Exception{
    http.authorizeRequests().anyRequest().authenticated();
  }
}
```

①必须用@Configuration 注解标记类。

②ResourceServiceConfiguration 类需要扩展自 ResourceServerConfigurerAdapter。

③所有的访问规则被定义在重写的 configure()方法里面。

④所有访问规则都是从传递给方法的 HttpSecurity 对象中配置的。

所有的访问规则将在 configure()方法定义。你将使用 Spring 传入的 HttpSecurity 类来定义规则。在本例中，你将组织服务中对任何 URL 的访问限制为仅允许通过身份验证的用户访问。

如果你不使用 HTTP 标头中的 OAuth2 访问令牌访问组织服务，那么你将获得 HTTP 401 响应代码，以及一条消息，表明需要对服务进行完整的身份验证。

图 7.6 显示了没有 OAuth2 HTTP 头对组织服务调用的输出。

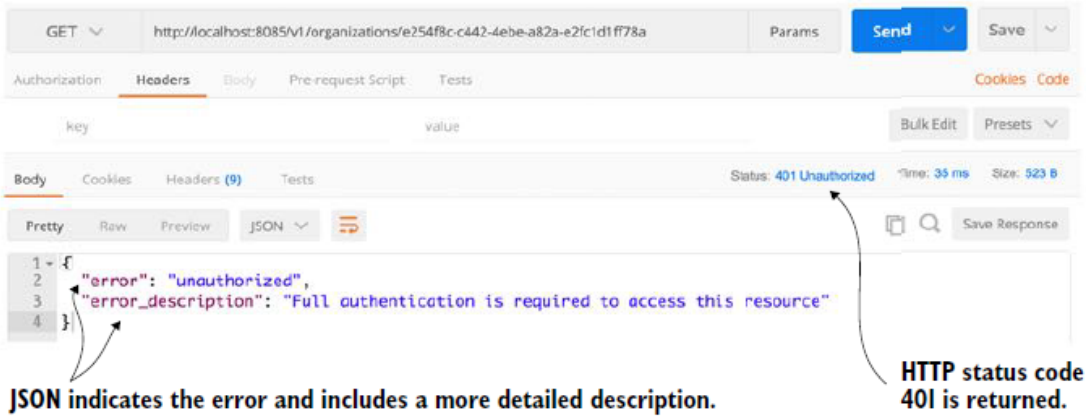

图 7.6 试图调用组织服务会导致一个失败的调用。

① JSON indicates the error and includes a more detailed description.

JSON 表明错误并包含更详细的描述。

② HTTP status code 401 is returned.

返回 HTTP 状态码 401。

接下来，你将调用带有 OAuth2 访问令牌的组织服务。为了得到一个访问令牌，见关于如何生成令牌的 7.2.4 “用户认证” 章节。你想要剪贴从 JavaScript 到/auth/oauth/token 端点的调用返回的 access_token 字段的值，并在组织服务调用中使用它。记住，当调用组织服务时，需要添加一个名为 Authorization 的 HTTP 头，并带有持有者的 access_token 值。

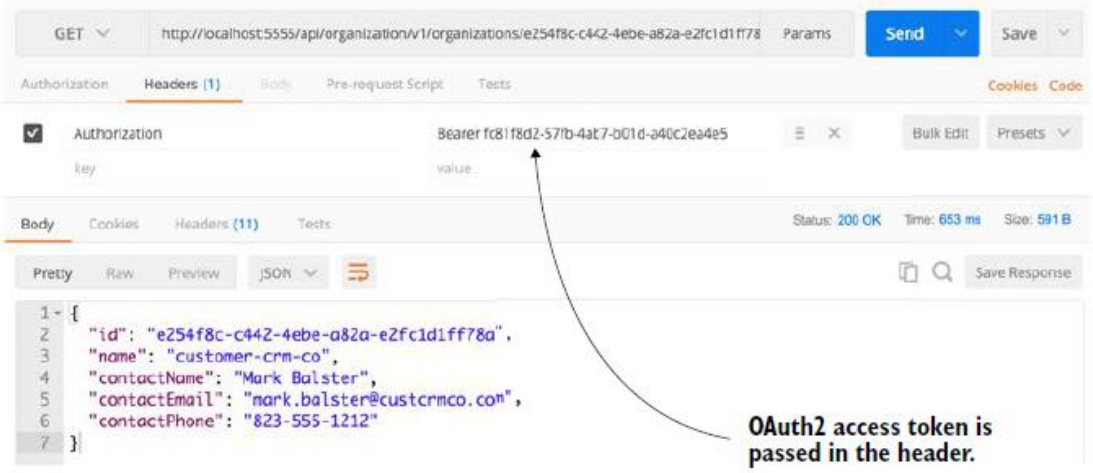

图 7.7 在对组织服务的调用中传递 OAuth2 访问令牌

① OAuth2 access token is passed in the header.

OAuth2 访问令牌在报头中传递。

图 7.7 显示了标注的组织服务，但这次带有传递给它的 OAuth2 访问令牌。

这可能是使用 OAuth2 保护端点的最简单用例之一。接下来，你将在此基础上，将特定端点的访问限制为特定的角色。

通过特定角色保护服务

在下一个示例中 ,你将对组织服务上的 DELETE 调用锁定为只有具有 ADMIN 访问权限

的用户。你会记得 7.2.3 章节，“配置 EagleEye 用户”，你创建了两个用户账户可以访问 EagleEye 服务：john.carnell 和 william.woodward。john.carnell 账户具有分配给它的 USER 角色。william.woodward 账户具有分配给它的 USER 角色和 ADMIN 角色。

下面的清单显示了如何设置 configure()方法来对通过身份验证的、有管理员角色的用户限制访问 DELETE 端点。

清单 7.6 仅将删除限制到管理员角色

```
package com.thoughtmechanix.organization.security;

@Configuration
public class ResourceServerConfiguration extends
      ResourceServerConfigurerAdapter {
   @Override
   public void configure(HttpSecurity http) throws Exception{
      http
      .authorizeRequests()
       .antMatchers(HttpMethod.DELETE, "/v1/organizations/**")
       .hasRole("ADMIN")
       .anyRequest()
       .authenticated();
   }
}
```

①antMatchers()方法允许你限制受保护的 URL 和 HTTP 提交。

②hasRole()方法是一个逗号分隔的可访问角色列表。

在清单 7.6 中，你对 ADMIN 角色限制了以/v1/organizations 开始的任何服务端点的 DELETE 调用。

```
.authorizeRequests()
   .antMatchers(HttpMethod.DELETE, "/v1/organizations/**")
   .hasRole("ADMIN")
```

antMatcher()方法可以使用逗号分隔的端点列表。这些端点可以使用通配符来定义要访问的端点。例如，如果你想限制任何 DELETE 调用，不管 URL 名称中的版本是什么，你可以在 URL 定义中使用*代替版本号：

```
.authorizeRequests()
   .antMatchers(HttpMethod.DELETE, "/*/organizations/**")
   .hasRole("ADMIN")
```

授权规则定义的最后一部分仍然定义服务中的任何其他端点都需要由经过身份验证的用户

访问：

```
.anyRequest()
.authenticated();
```

现在，如果你要获得用户 john.carnell 的一个 OAuth2 令牌（密码：password1）和尝试调用组织服务的 DELETE 端点（http://localhost:8085/v1/organizations/e254f8c-c442-4ebe-a82ae2fc1d1ff78a），你将在调用时获得 401 HTTP 状态代码，并显示拒绝访问的错误消息。你的调用返回的 JavaScript 文本是

```
{
   "error": "access_denied",
   "error_description": "Access is denied"
}
```

如果你使用用户账户 william.woodward（密码：password2）和它的 OAuth2 令牌尝试完全相同的调用，你将看到一个成功的调用将返回（HTTP 状态代码 204—而不是内容），该组织将被组织服务删除。

此刻，我们已经看到了使用 OAuth2 调用和保护一个单独的服务（组织服务）的两个简单示例。然而，往往在微服务环境，你会有多个服务调用执行一个交易。在这些情况下，你需要确保 OAuth2 访问令牌在服务调用之间传递。

7.3.4. 传递 OAuth2 访问令牌

为展示服务之间传递一个 OAuth2 令牌，现在我们来看如何使用 OAuth2 保护你的许可服务。记住，许可服务调用组织服务来查找信息。问题是，如何将 OAuth2 令牌从一个服务传递到另一个服务？

你将创建一个简单的示例，在这里你将使用许可服务调用组织服务。在第 6 章例子的基础上构建，服务是在 Zuul 网关运行。

图 7.8 显示了如何认证流过 Zuul 网关，到许可服务，再到组织服务的用户的 OAuth2

令牌的基本流程。

① The EagleEye web app calls the licensing service (behind the Zuul gateway) and adds the user's OAuth2 token to the HTTP header "Authorization."

EagleEyeWeb 应用程序调用许可服务（在 Zuul 网关后面）和将用户的 OAuth2 令牌添加到 HTTP 头的""Authorization"。

② The Zuul gateway locates the licensing service and forwards the call with the "Authorization" header.

Zuul 网关定位许可服务并转发带有"Authorization"头的调用。

③ The licensing service validates the user's token with the authentication service and also propagates the token to the organization service.

许可服务使用认证服务验证用户的令牌，并将令牌传递到组织服务。

④ The organization service also validates the user's token with the authentication service.

组织服务也使用认证服务验证用户的令牌。

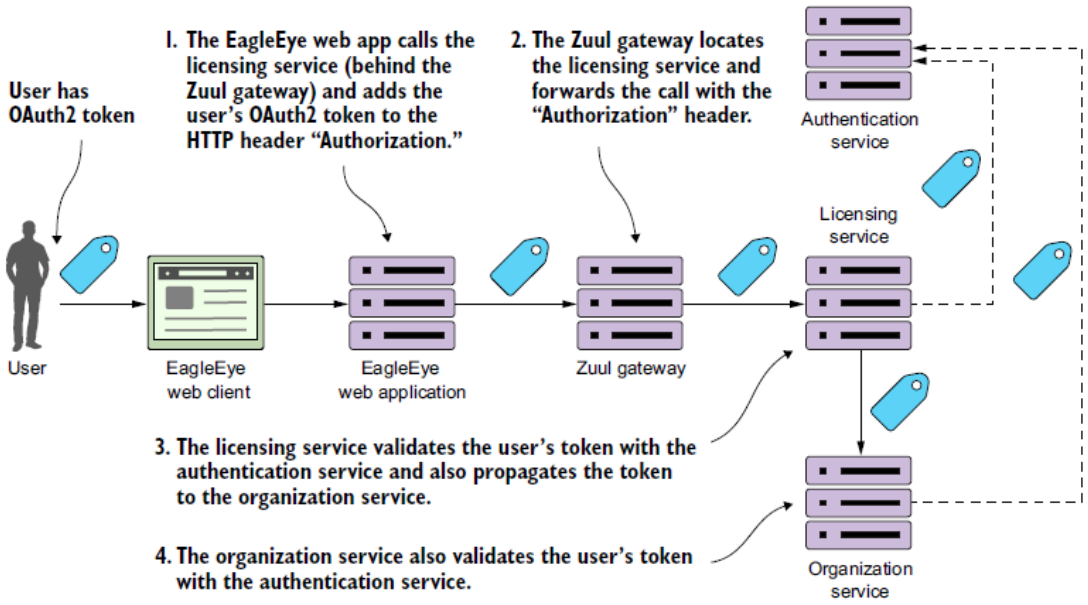

图 7.8 OAuth2 令牌必须贯穿整个调用链。

下面的活动发生在图 7.8 中：

- 用户已经在 OAuth2 服务器完成了验证并向 EagleEye Web 应用发起调用。用户的 OAuth2 访问令牌存储在用户会话中。EagleEye Web 应用需要获取一些许可数据，并向许可服务 REST 端点发起调用。作为调用许可 REST 端点的一部分，EagleEye Web 应用将通过 HTTP 头的“Authorization”添加 OAuth2 访问令牌。
- Zuul 查找许可服务端点，然后转发调用到一个许可服务的服务器。服务网关需要从传入的调用复制 HTTP 头的“Authorization”，并确保将 HTTP 头的“Authorization”转发到新端点。
- 许可服务将收到传入的调用。因为许可服务是一个受保护的资源，许可服务将使用 EagleEye 的 OAuth2 服务验证令牌，并检查用户的角色是否有适当的权限。

  作为其工作的一部分，许可服务调用组织服务。在执行此调用时，许可服务需要将用户的 OAuth2 访问令牌传递到组织服务。
- 当组织服务收到调用，它将再次取得 HTTP 头的“Authorization”，并由 EagleEye 的 OAuth2 服务器验证该令牌。

要实现这些流程，你需要做两件事。首先，你需要修改你的 Zuul 服务网关来传递 OAuth2 令牌到许可服务。默认情况下，Zuul 不会转发敏感的 HTTP 头（如：Cookie，Set-Cookie 和 Authorization）到下游许可服务。为允许 Zuul 传递 HTTP 头的“Authorization”，你需要在 Zuul 服务网关的 application.yml 文件或 Spring Cloud Config 数据存储设置以下配置：

```
zuul.sensitiveHeaders: Cookie,Set-Cookie
```

这个配置是敏感的头信息黑名单列表，Zuul 将阻止它们被传递到下游服务。在前面的列表缺少 Authorization 值，意味着 Zuul 将允许它通过。如果你根部不设置

zuul.sensitiveHeaders 属性，Zuul 将自动阻止所有者三个值(Cookie， Set-Cookie 和 Authorization)被传递。

关于 Zuul 的其它 OAuth2 能力？

Zuul 可以自动向下游传递 OAuth2 访问令牌和通过使用@EnableOAuth2Sso 注解授权对 OAuth2 服务传入的请求。我故意没有使用这种方法，因为我在本章中的目标是在不增加另一个复杂度（或调试）的情况下显示 OAuth2 的工作原理。而 Zuul 服务网关配置不能过于复杂，这将为已经很大的章节增加更多的内容。如果你对 Zuul 服务网关参与单一登录（SSO）感兴趣，Spring Cloud Security 文档有一个很短但综合的教程，涵盖了 Spring 服务器的安装(http://cloud.spring.io/spring-cloudsecurity/spring-cloud-security.html)。

你需要做的下一件事是将许可服务配置为 OAuth2 资源服务，并设置服务所需的任何授权规则。我们不打算详细讨论许可服务配置，因为我们已经在 7.3.3 章节“定义那些资源可以访问服务”，讨论了授权规则。

最后，你需要做的只是修改许可服务中的代码如何调用组织服务。你需要确保 HTTP 头“Authorization”被注入到调用了组织服务的应用程序。如果没有 Spring Security，你必须编写一个 servlet 过滤器来捕获传入的许可服务调用中的 HTTP 头，然后手动将其添加到许可服务中的每个出站服务调用中。Spring OAuth2 提供了一个支持 OAuth2 调用的新 REST 模板类。这类被称为 OAuth2RestTemplate。要使用 OAuth2RestTemplate 类，你首先需要将它暴露为 bean，它可以自动连接到另一个受 OAuth2 保护的服务的服务调用中。你在 licensing-service/src/main/java/com/thoughtmechanix/licenses/ Application.java 类中这样做：

```
@Bean
public OAuth2RestTemplate oauth2RestTemplate(
        OAuth2ClientContext oauth2ClientContext,
        OAuth2ProtectedResourceDetails details) {
```

```
        return new OAuth2RestTemplate(details, oauth2ClientContext);
}
```

要查看实战中的 OAuth2RestTemplate 类 ,可以在 licensingservice/src/main/java/com/thoughtmechanix/licenses/clients/OrganizationRestTemplate.java 中查看。下面的清单显示了 OAuth2 RestTemplate 如何自动连接到这个类中。

清单 7.7 使用 OAuth2RestTemplate 来传递 OAuth2 访问令牌

```
package com.thoughtmechanix.licenses.clients;

@Component
public class OrganizationRestTemplateClient {

   @Autowired
   OAuth2RestTemplate restTemplate;

   private static final Logger logger = LoggerFactory.getLogger(OrganizationRestTemplateClient.class);

   public Organization getOrganization(String organizationId){
      logger.debug("In Licensing Service.getOrganization: {}",
        ➥UserContext.getCorrelationId());

      ResponseEntity<Organization> restExchange =
         restTemplate.exchange("http://zuulserver:5555/api/organization
            ➥/v1/organizations/{organizationId}",
         HttpMethod.GET,
         null, Organization.class, organizationId);
         /*从缓存中保存记录*/
         return restExchange.getBody();
   }
}
```

①OAuth2RestTemplate 是一个标准 RestTemplate 的替代品，处理 OAuth2 访问令牌的传递。

②组织服务的调用是作为一个标准的 RestTemplate 在完全相同的方式完成的。

7.4. JavaScript Web Tokens 和 OAuth2

OAuth2 一个基于令牌的认证框架，但具有讽刺意味的是，它没有提供任何标准来定义其规范中的令牌。为了改进 OAuth2 令牌的不足，一个称为 JavaScript Web Tokens (JWT) 的新标准出现了。JWT 是一个由互联网工程任务组(IETF)提出的开放标准（RFC-7519），试图为 OAuth2 令牌提供标准的结构。JWT 令牌是：

- 小 :JWT 令牌以 Base64 方式编码 ,可以很容易地通过一个 URL ,HTTP 头或 HTTP POST 参数传递。
- 加密的签名：JWT 令牌由发出它的认证服务器签名。这意味着可以保证令牌没有被篡改。
- 自包含的：因为一个 JWT 令牌被加密签名，接收服务的微服务可以保证令牌的内容是有效的。没有必要回调认证服务来验证令牌的内容，因为令牌的签名可以验证和内容（如令牌到期时间和用户信息）可以通过接收的微服务检验。
- 可扩展 :当认证服务生成令牌时 ,它可以在令牌被密封之前在令牌中放置附加信息。接收服务可以解密令牌有效负载并从中检索附加的上下文。

Spring Cloud Security 支持 JWT 开箱即用。但是，要使用和消费 JWT 令牌，必须使用不同的方式配置认证服务和受认证服务保护的服务。配置并不难，所以让我们来看看更改。

> ***注意***：我选择将 JWT 配置保存在本章 GitHub 库(https://github.com/carnellj/spmia-chapter7)的一个独立分支(称为 JWT_Example)上，因为标准的 Spring Cloud Security OAuth2 配置和基于 JWT 的 OAuth2 配置要求不同的配置类。

7.4.1. 修改认证服务来发布 JavaScript Web Tokens

对于认证服务和将受 OAuth2 保护的两个微服务（许可和组织服务），你将需要向它们的 Maven pom.xml 文件添加 Spring Security 依赖来包括 JWT OAuth2 库。这个新依赖是：

```
<dependency>
   <groupId>org.springframework.security</groupId>
   <artifactId>spring-security-jwt</artifactId>
</dependency>
```

在加入 Maven 依赖后，首先你需要告诉你的认证服务如何生成和解析 JWT 令牌。要做到

这一点，你要为认证服务创建一个称为 authenticationservice/src/java/com/thoughtmechanix/authentication/security/JWTTokenStoreConfig.java 的新配置类。下面的清单显示了这个类的代码。

清单 7.8 创建 JWT 令牌存储

```
@Configuration
public class JWTTokenStoreConfig {

    @Autowired
    private ServiceConfig serviceConfig;

    @Bean
    public TokenStore tokenStore() {
       return new JwtTokenStore(jwtAccessTokenConverter());
    }

    @Bean
    @Primary
    public DefaultTokenServices tokenServices() {
       DefaultTokenServices defaultTokenServices
             = new DefaultTokenServices();
       defaultTokenServices.setTokenStore(tokenStore());
       defaultTokenServices.setSupportRefreshToken(true);
       return defaultTokenServices;
    }

    @Bean
    public JwtAccessTokenConverter jwtAccessTokenConverter() {
        JwtAccessTokenConverter converter =
            new JwtAccessTokenConverter();
        converter
            .setSigningKey(serviceConfig.getJwtSigningKey());
        return converter;
    }

    @Bean
    public TokenEnhancer jwtTokenEnhancer() {
        return new JWTTokenEnhancer();
    }
}
```

①@Primary 注解被用来告诉 Spring，如果有一个以上特定类型(像 DefaultTokenService)的 bean，使用标记为 @Primary 的 bean 类型进行自动注入。

②用于读取提交给服务的令牌的数据。

③充当 JWT 到 OAuth2 服务器之间的转换器。

④定义将用于签名令牌的签名密钥。

JWTTokenStoreConfig 类用于定义 Spring 将如何管理 JWT 令牌的创建、签名和转换。

tokenServices()方法将使用 Spring Security 的默认令牌服务实现，所以这里的工作是死记硬背。jwtAccessTokenConverter()方法是我们想要关注的。它定义了令牌将如何被转换。关于这个方法要注意的最重要的一点是，你将设置用于签名的签名密钥。

在这个例子中，你要使用一个对称密钥，这意味着认证服务和受认证服务保护的服务必须在所有服务之间共享相同的密钥。密钥只不过是存储在认证服务 Spring Cloud Config 条目(https://github.com/carnellj/config-repo/blob/master/authenticationservice/authenticationservice.yml)中的一个随机字符串值。签名密钥的实际值是：

signing.key: “345345fsdgsf5345”

注意：Spring Cloud Security 支持使用公钥/私钥的对称密钥加密和非对称加密。我们不打算通过使用公钥/私钥来设置 JWT。不幸的是，JWT、Spring Security 和公钥/私钥只存在少量的正式文档。如果你对如何做到这一点有兴趣，我强烈推荐你看看 baeldung.com(http://www.baeldung.com/spring-security-oauth-jwt)。他们出色地解释了 JWT 和公钥/私钥设置。

在清单 7.8 中的 JWTTokenStoreConfig 类中，你定义了 JWT 令牌是如何被签名和创建的。现在需要将此挂钩到整个 OAuth2 服务中。在清单 7.2 中，你使用 OAuth2Config 类定义 OAuth2 服务的配置。你设置了将由你的服务使用的认证管理器，以及应用程序名称和密钥。你将用一个称为 authenticationservice/src/main/java/com/thoughtmechanix/authentication/security/JWTOAuth2Config.java 的新类取代 OAuth2Config 类。

下面的清单显示了 JWTOAuth2Config 类的代码。

清单 7.9 通过 JWTOAuth2Config 类将 JWT 令牌连接到你的认证服务

```
package com.thoughtmechanix.authentication.security;

@Configuration
public class JWTOAuth2Config extends AuthorizationServerConfigurerAdapter {
```

```
    @Autowired
    private AuthenticationManager authenticationManager;

    @Autowired
    private UserDetailsService userDetailsService;

    @Autowired
    private TokenStore tokenStore;

    @Autowired
    private DefaultTokenServices tokenServices;

    @Autowired
    private JwtAccessTokenConverter jwtAccessTokenConverter;

    @Override
    public void configure(AuthorizationServerEndpointsConfigurer endpoints)
      ➥ throws Exception {
      TokenEnhancerChain tokenEnhancerChain =new TokenEnhancerChain();
      tokenEnhancerChain.setTokenEnhancers(
      ➥ Arrays.asList(jwtTokenEnhancer ,jwtAccessTokenConverter));

      endpoints.tokenStore(tokenStore)
              .accessTokenConverter(jwtAccessTokenConverter)
              .authenticationManager(authenticationManager)
              .userDetailsService(userDetailsService);
    }
    ......
}
```

①清单 7.8 中定义的令牌存储将在这里注入。

② 这 是 告 诉 Spring Security OAuth2 代码使用 JWT 的钩子。

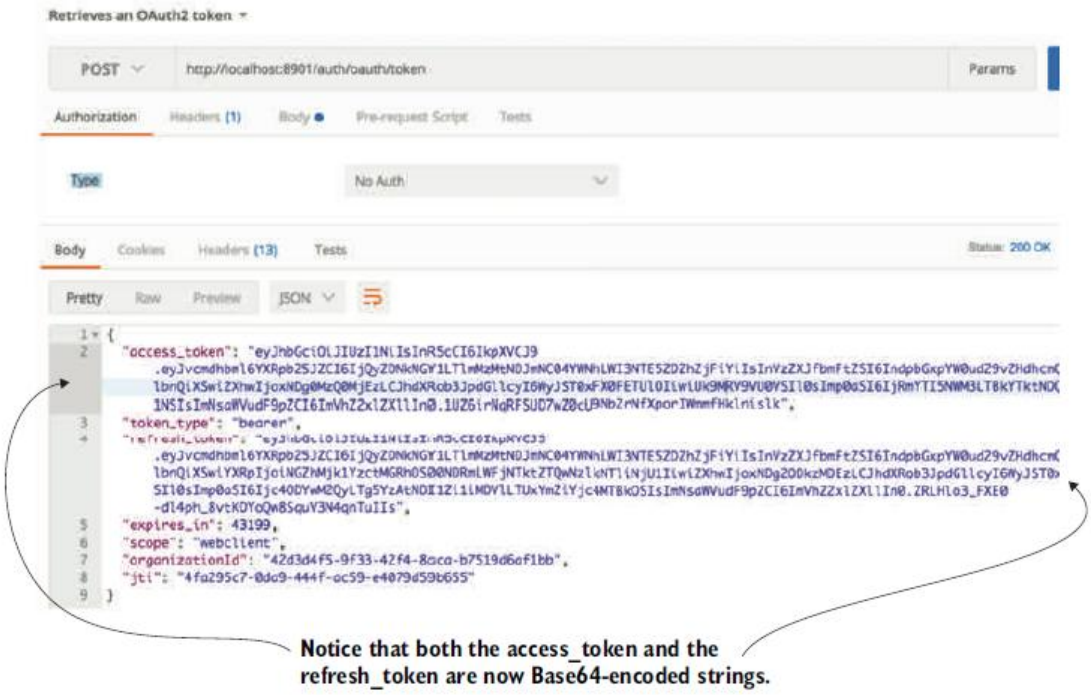

图 7.9 从你的认证调用的访问令牌和刷新令牌现在都是 JWT 令牌。

① Notice that both the access_token and the refresh_token are now Base64-encoded strings.

注意，access_token 和 refresh_token 现在都是 Base64 编码的字符串。

现在，如果你重建你的认证服务并重新启动它，你会看到一个返回的 JWT 令牌。图 7.9 显示了调用认证服务的结果，现在它使用了 JWT。

实际的令牌本身不会直接返回为 JavaScript。相反，JavaScript 的报文体使用 Base64 编码进行编码。如果你有兴趣看到 JWT 令牌的内容，可以使用在线的工具解码令牌。我喜欢使用一家名为 Stormpath 的公司的在线工具。我喜欢使用一个在线工具从一家名为 stormpath。他们的工具，http://jsonwebtoken.io，是一个在线的解码器。图 7.10 显示了解码令牌的输出。

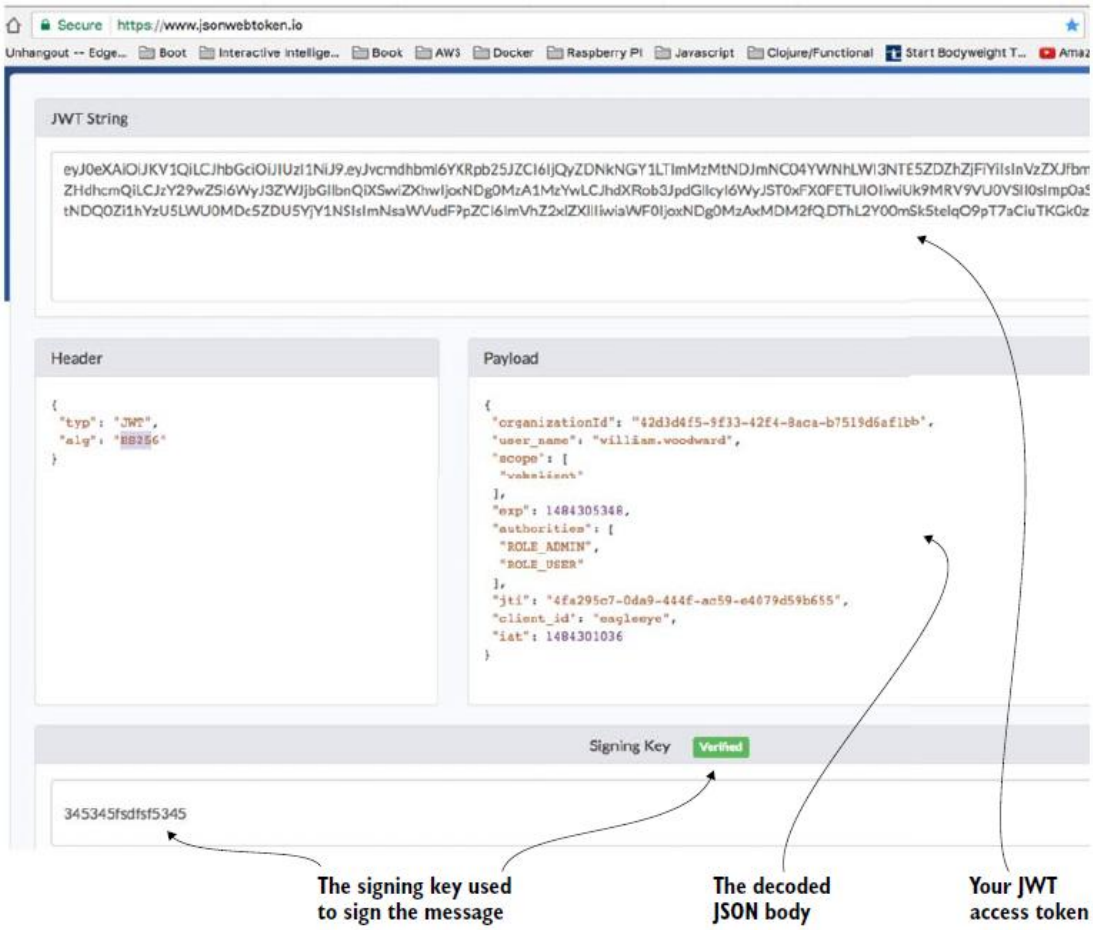

图 7.10 使用 http://jswebtoken.io 允许你解码内容。

① *Your JWT access token*

你的 JWT 访问令牌

② *The decoded JSON body*

解码的 JSON 报文体

③ *The signing key used to sign the message*

用于签名消息的签名密钥

注意：理解 JWT 令牌被签名，但没有加密是非常重要的。任何在线 JWT 工具都可以解码 JWT 令牌并公开其内容。我之所以提出这一点，是因为 JWT 规范允许扩展令牌并向令牌添加附加信息。在你的 JWT 令牌里，不要暴露敏感或个人可识别信息（PII）。

7.4.2. 在微服务中消费 JavaScript Web Tokens

现在你的 OAuth2 认证服务创建 JWT 令牌。下一步是配置你的许可服务和组织服务以使用 JWT。这是一个需要你做两件事的小事：

- 向许可服务和组织服务的 pom.xml 文件添加 spring-security-jwt 依赖。（看到 7.4.1 章节的开始部分，“修改认证服务来发布 JavaScript Web Tokens”，添加确切需要的 Maven 依赖。）
- 在许可和组织服务中创建一个 JWTTokenStoreConfig 类。这个类与使用的认证服务几乎是完全相同的类（参见清单 7.8）。我不想再重复一遍同样的内容，但您可以在 licensingservice/src/main/com/thoughtmechanix/licensing-service/security/JWTTokenStoreConfig.java 和 organization-service/src/main/com/thoughtmechanix/organization-service/security/JWTTokenStoreConfig.java 类中看到 JWTTokenStoreConfig 类的示例。

你需要做最后一件工作。由于许可服务调用组织服务，所以需要确保 OAuth2 令牌被传递。这通常是通过 OAuth2RestTemplate 类完成的；但是，OAuth2RestTemplate 类不传递 JWT 令牌。为了确保许可服务做到这一点，你需要添加一个自定义的 RestTemplate bean，它将为你执行注入。这个自定义 RestTemplate 类可以在 licensing-service/src/main/java/com/thoughtmechanix/licenses/Application.java 类中找到。

清单 7.10 创建一个自定义 RestTemplate 类来注入 JWT 令牌

```
public class Application {

    @Primary
    @Bean
    public RestTemplate getCustomRestTemplate() {
        RestTemplate template = new RestTemplate();
        List interceptors = template.getInterceptors();
```

```
        if (interceptors == null) {
          template.setInterceptors(
            ➥ Collections.singletonList(
                new UserContextInterceptor()));
        } else {
          interceptors.add(new UserContextInterceptor());
          template.setInterceptors(interceptors);
        }

        return template;
    }
}
```

①UserContextInterceptor 类将在每次 Rest 调用中注入 Authorization 头。

在前面的代码中，你定义了一个将使用 ClientHttpRequestInterceptor 类的自定义 RestTemplate bean。回顾第 6 章的 ClientHttpRequestInterceptor，该类是一个 Spring 类，它允许你在执行 REST 调用之前与方法挂钩。这个拦截器类是你在第 6 章中定义的 UserContextInterceptor 类的变体。这个类是在 licensing-service/src/main/java/com/thoughtmechanix/licenses/utils/UserContextInterceptor.java。下面的清单显示了这个类。

清单 7.11 UserContextInterceptor 类将 JWT 令牌注入到 REST 调用中

```
public class UserContextInterceptor
    implements ClientHttpRequestInterceptor {

    @Override
    public ClientHttpResponse intercept(
            HttpRequest request, byte[] body,
            ➥ClientHttpRequestExecution execution)
            throws IOException {
      headers.add(UserContext.CORRELATION_ID,
            ➥UserContextHolder.getContext().getCorrelationId());
      headers.add(UserContext.AUTH_TOKEN,
            ➥UserContextHolder.getContext().getAuthToken());

      return execution.execute(request, body);
    }
}
```

①将授权令牌添加到 HTTP 头

UserContextInterceptor 类正在使用第 6 章中的几个实用工具类。记住，你的每一个服务使用自定义的 Servlet 过滤器（称为 UserContextFilter）从 HTTP 头解析出的认证令牌和

关联 ID。在清单 7.11 中，你使用已经解析过的 UserContext.AUTH_TOKEN 值来填充传出的 HTTP 调用。

有了这些，就足够了。现在你可以调用许可服务（或组织服务），将经过 Base64 编码的 JWT 放置在你的 HTTP 头 Authorization 字段（它的值为 Bearer <<JWT-Token>>），并且你的服务将正确读取和验证 JWT 令牌。

7.4.3. 扩展 JWT Token

如果你仔细看看图 7.10 中的 JWT 令牌，你会注意到 EagleEye 的 organizationId 字段。（图 7.11 显示了一个比前面图 7.10 所示更大的 JWT 令牌）。这不是标准的 JWT 令牌字段。它是通过在 JWT 令牌中注入一个新字段来添加的。

① *This is not a standard JWT field.*

这不是标准的 JWT 字段。

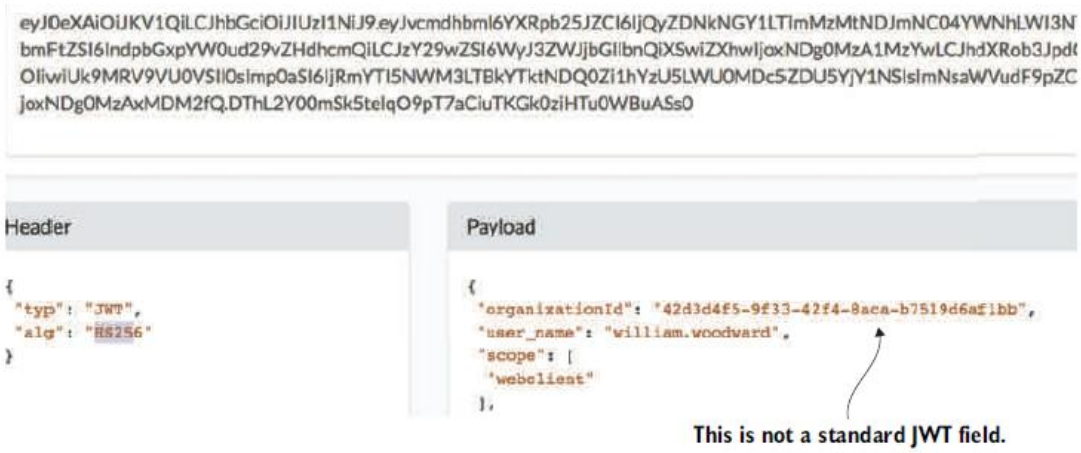

图 7.11 一个使用 organizationId 扩展 JWT 令牌的示例

通过将一个 Spring OAuth2 令牌增强类添加到认证服务中，可以轻松地扩展 JWT 令牌。该类的源代码可以在 authentication-service/src/main/java/com/thoughtmechanix/authentication/security/JWTTokenEnhancer.java 类中找到。下面的

清单显示了此代码。

清单 7.12 使用 JWT 令牌增强类添加自定义字段

```
package com.thoughtmechanix.authentication.security;

import org.springframework.security.oauth2.provider.token.TokenEnhancer;

public class JWTTokenEnhancer implements TokenEnhancer {
    @Autowired
    private OrgUserRepository orgUserRepo;

    private String getOrgId(String userName){
        UserOrganization orgUser =
        orgUserRepo.findByUserName( userName );
        return orgUser.getOrganizationId();
    }

    @Override
    public OAuth2AccessToken enhance(
      OAuth2AccessToken accessToken,
      ➥OAuth2Authentication authentication)
    ➥{
        Map<String, Object> additionalInfo = new HashMap<>();
        String orgId = getOrgId(authentication.getName());
      additionalInfo.put("organizationId", orgId);

      ((DefaultOAuth2AccessToken) accessToken)
          .setAdditionalInformation(additionalInfo);
      return accessToken;
   }
}
```

①你需要扩展 TokenEnhancer 类。

②getOrgId()方法基于用户名来查找用户的 org ID。

③要执行此增强，需要添加覆盖 enhance()方法。

④所有的附加属性被放置在一个 HashMap 并设置在 accessToken 变量传递到方法。

你需要做的最后一件事是告诉你的 OAuth2 服务使用你的 JWTTokenEnhancer 类。首先需要为 JWTTokenEnhancer 类暴露为一个 Spring bean。通过向清单 7.8 中定义的 JWTTokenStoreConfig 类中添加 bean 定义来实现这一点：

```
package com.thoughtmechanix.authentication.security;

@Configuration
public class JWTTokenStoreConfig {

    @Bean
    public TokenEnhancer jwtTokenEnhancer() {
```

```
        return new JWTTokenEnhancer();
    }
}
```

一旦你将 JWTTokenEnhancer 暴露为 bean，就可以将它从清单 7.9 中引入到 JWTOAuth2Config 类中。这是在类的 configure()方法中完成的。下面的清单显示了对 JWTOAuth2Config 类 configure()方法的修改。

清单 7.13 挂入到 TokenEnhancer

```
package com.thoughtmechanix.authentication.security;

@Configuration
public class JWTOAuth2Config extends AuthorizationServerConfigurerAdapter {

    @Autowired
    private TokenEnhancer jwtTokenEnhancer;

    @Override
    public void configure(
      AuthorizationServerEndpointsConfigurer endpoints)
        ➥ throws Exception {
      TokenEnhancerChain tokenEnhancerChain = new TokenEnhancerChain();
      tokenEnhancerChain.setTokenEnhancers(
          Arrays.asList(jwtTokenEnhancer, jwtAccessTokenConverter));
      endpoints.tokenStore(tokenStore)
               .accessTokenConverter(jwtAccessTokenConverter)
               .tokenEnhancer(tokenEnhancerChain)
               .authenticationManager(authenticationManager)
               .userDetailsService(userDetailsService);
    }
}
```

①自动注入 TokenEnhancer 类。

②Spring OAuth 允许你在多个令牌增强程序中挂钩，因此将令牌增强器添加到 TokenEnhancerChain 类中。

③将令牌增强链挂钩到传递到 configure() 调用中的端点参数。

此时，你已经将一个自定义字段添加到 JWT 令牌中了。你的下一个问题应该是："如何从 JWT 令牌中解析自定义的字段？"

7.4.4. 解析来自自定义字段的 JavaScript token

我们将把你的 Zuul 网关作为如何从自定义的字段中解析出 JWT 令牌的示例。具体来说，你将修改我们在第 6 章中引入的 TrackingFilter 类来解码流经网关的 JWT 令牌的

organizationId 字段。

为此你要拉取一个 JWT 解析器库并添加到 Zuul 服务器的 pom.xml 文件。多个令牌解析器是可用的，我选择了 JJWT 库(https://github.com/jwtk/jjwt)进行解析。库的 Maven 依赖是：

```
<dependency>
   <groupId>io.jsonwebtoken</groupId>
   <artifactId>jjwt</artifactId>
   <version>0.7.0</version>
</dependency>
```

一旦添加了 JJWT 库 ,你可以为你的 zuulsvr/src/main/java/com/thoughtmechanix/zuulsvr/filters/TrackingFilter.java 类添加一个称为 getOrganizationId()的新方法。下面的清单显示了这个新方法。

清单 7.14 从 JWT 令牌解析出 organizationId

```
private String getOrganizationId(){
   String result="";
   if (filterUtils.getAuthToken()!=null){
      String authToken = filterUtils.getAuthToken()
                                    .replace("Bearer ","");
      try {
            Claims claims = Jwts.parser()
               ➥ .setSigningKey(serviceConfig.getJwtSigningKey()
               ➥ .getBytes("UTF-8"))
                  .parseClaimsJws(authToken)
                  .getBody();
            result = (String) claims.get("organizationId");
      }
      catch (Exception e){
            e.printStackTrace();
      }
   }
   return result;
}
```

①从 HTTP 头 Authorization 解析出令牌。

②使用 JWTS 类解析令牌，传递用于签名令牌的签名密钥。

③从 JavaScript token 令牌拉取 organizationId

一旦 getOrganizationId() 的功能实现，向 TrackingFilter 类的 run() 方法添加 System.out.println 来打印 organizationId ,它解析自流经 Zuul 网关的 JWT 令牌 ,因此，你可以调用任何启用网关的 REST 端点。我以 GET 方式调用 http://localhost:5555/api/

licensing/v1/organizations/e254f8c-c442-4ebe-a82a-e2fc1d1ff78a/licenses/f3831f8c-c338-4ebe-a82a-e2fc1d1ff78a。记住，在进行此调用时，仍然需要设置所有 HTTP 表单参数和 HTTP authorization 头，以包括 Authorization 头和令牌。

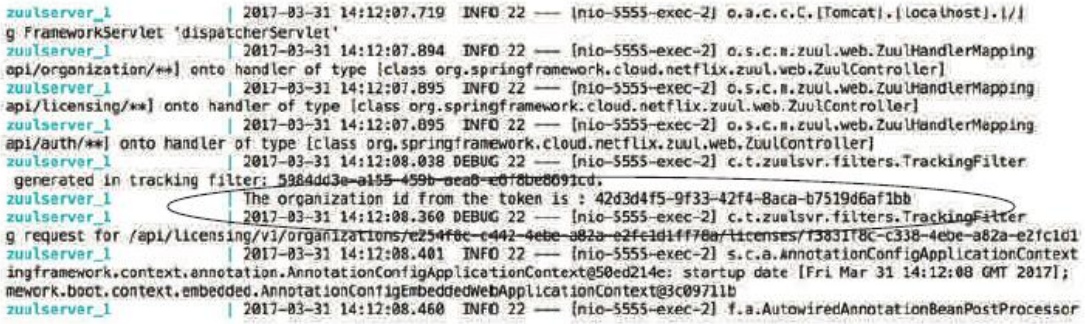

图 7.12 Zuul 服务器从流经它的 JWT 令牌解析出 organization ID。

图 7.12 显示了你解析的 organizationId 输出到命令行控制台。

7.5. 关于微服务安全的思考

本章向你介绍了 OAuth2 规范和如何使用 Spring Cloud security 来实现 OAuth2 认证服务，OAuth2 只是微服务安全难题的一部分。当你为生产使用构建微服务，你应该围绕下列做法建立你的微服务安全：

- 所有服务通信使用 HTTPS 安全套接字层（SSL）。
- 所有的服务调用都要经过 API 网关。
- 将你的服务划分为公共 API 和私有 API。
- 通过锁定不需要的网络端口限制你的微服务攻击面。

图 7.13 显示了如何将这些不同的部分结合在一起。每个列表中的项目符号项映射到图 7.13 中的数字。

让我们更详细的检查一下以前列举的列表和图表里的每个主题领域。

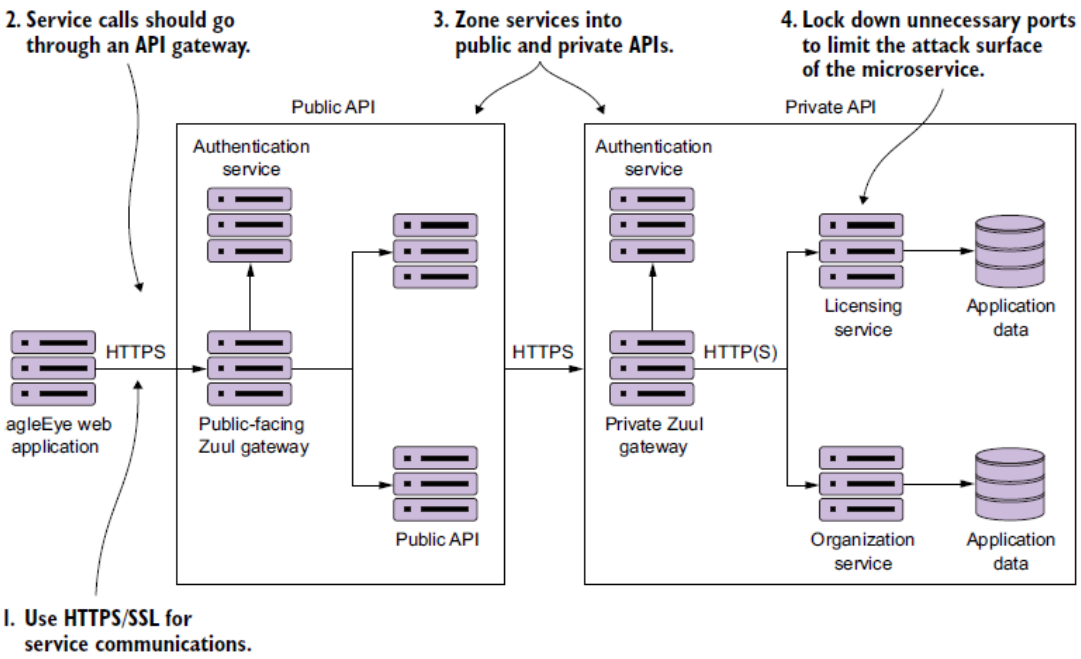

图 7.13 微服务安全体系不仅仅是实现 OAuth2。

所有服务通信使用 HTTPS 安全套接字层（SSL）

在本书中的所有代码示例中，你一直在使用 HTTP，因为 HTTP 是一个简单的协议，在你开始使用服务之前不需要对每个服务进行设置。

在生产环境中，你的微服务只能通过 HTTPS 和 SSL 加密通道进行通信。HTTPS 的配置和设置可以通过你的 DevOps 脚本自动化完成。

> ***注意***：如果你的应用程序需要满足信用卡支付的支付卡行业（PCI）的遵从性，你将需要为所有的服务通信实现 HTTPS。尽早使用 HTTPS 构建你所有的服务，比你的应用和微服务在生产后做项目迁移更容易。

使用服务网关来访问你的微服务

单个服务器、服务端点和服务正在运行的端口不应该被客户端直接访问。相反，使用服务网关作为服务调用的入口点和看门人。配置网络层上的操作系统或运行微服务的容器只接受服务网关的流量。

记住，服务网关可以作为一个策略执行点（PEP），它可以针对所有服务执行。把服务

调用通过服务网关，如 Zuul 允许你如何保护和审计服务是一致的。服务网关还允许你锁定将要暴露给外部世界的端口和端点。

将你的服务划分为公共 API 和私有 API

一般性的安全是所有关于建立访问层和强制执行最小权限的概念。最小权限的概念，用户应该有最少的网络访问和权限来做他们的日常工作。为此，你应该将你的服务分为两个不同的作用域来实现最小权限：public 和 private。

公共区域包含客户端将使用的公共 API（EagleEye 应用程序）。公共 API 微服务应该进行面向工作流的有限任务。公共 API 微服务往往是为集成商提供服务，跨多个服务拉取数据和执行任务。

公共微服务应该在自己的服务网关之后，并有自己的认证服务来进行 OAuth2 身份验证。客户端应用程序访问公共服务时应该经过由服务网关保护的单个路由。此外，公共区域应该有自己的认证服务。

私有区域作为一堵墙来保护你的核心应用程序的功能和数据。只有通过一个众所周知并且应该被锁定的端口才能访问私有区域，只接受来自私有服务正在运行的网络子网的网络通信流量。私有区域应该有自己的服务网关和认证服务。公共 API 服务应该对私有区域认证服务进行身份验证。所有应用程序数据至少应该位于私有区域的网络子网中，只能由驻留在私有区域的微服务访问。

私有 API 网络区域应该如何锁定？

许多组织采取的方法是，他们的安全模型应该有一个硬的外部中心和较软的内表面。这意味着一旦交易是在内部的私有 API 区域，在私有区域服务之间的通信可以不加密（没有 HTTPS）和不需要的认证机制。大多数情况下，这是为了方便和开发速度而做的。你有更多的安全、更难调试的问题，增加管理应用程序的整体复杂性。

我倾向于偏执地看待这个世界。（我在金融服务业工作了八年，所以偏执狂伴随着领域而来。）我宁愿将增加复杂性（可以通过 DevOps 的脚本减轻）作为交换条件，执行所有在我的私有 API 区域使用 SSL 运行的、并被证运行在私有区域的认证服务验证的服务。问题是你必须问问自己，你有多愿意因为网络中断而看到你的组织出现在当地报纸的头版？

通过锁定不需要的网络端口限制你的微服务攻击面

许多开发人员并没有仔细检查他们需要打开的端口的绝对最小数量，以便它们的服务能够正常运行。配置你的服务正在运行的操作系统，只允许对服务或服务所需的基础设施（监视、日志聚合）所需的端口进行入站和出站访问。

不要只关注入站的访问端口。许多开发者忘记锁定他们的出站端口。锁定你的出站端口可以防止在服务本身已被攻击者攻破的事件中泄露了你的服务数据。另外，请确保查看访问公共和私有 API 区域的网络端口。

7.6. 小结

- OAuth2 是一个基于令牌的身份认证框架，它用来验证用户。
- OAuth2 确保每个微服务执行用户请求不需要提供每个调用的用户凭证。
- OAuth2 提供不同的机制来保护 Web 服务调用。这些机制称为授权。
- 在 Spring 中使用 OAuth2，你需要创建一个 OAuth2 认证服务。
- 想要调用服务的每个应用程序都需要在 OAuth2 认证服务中注册。
- 每个应用程序都有自己的应用程序名称和密钥。
- 用户凭证和角色是在内存或数据存储，且通过 Spring security 访问。
- 每个服务必须定义角色可以采取的操作。
- Spring Cloud Security 支持 JavaScript Web Token (JWT) 规范。

- JWT 定义了一个签名，生成 OAuth2 令牌的 JavaScript 标准。
- 使用 JWT，你可以将自定义字段注入到规范中。
- 保护你的微服务涉及的不仅仅是使用 OAuth2。你应该使用 HTTPS 来加密服务间的所有调用。
- 使用服务网关来缩小通过服务可以到达的接入点的数量。
- 通过限制服务正在运行的操作系统上的入站和出站端口的数量来限制服务的攻击面。