微服务安全第二课(保护微服务的第一步)

目录

1. 构建你的第一个微服务 1
  1.1. 克隆示例库 2
  1.2. 编译订单处理微服务 2
  1.3. 访问订单处理微服务 3
  1.4 源代码目录中有什么? 4
  1.5 理解微服务的源代码 4
2. 设置 OAuth 2 服务器 5
  2.1 与授权服务器的交互 5
  2.2 运行 OAuth 2 授权服务器 8
  2.3 从 OAuth 2 授权服务器获取访问令牌 9
  2.4 理解访问令牌响应 10
3 使用 OAuth 2 保护微服务 10
  3.1 基于 OAuth 2 的保护 10
  3.2 运行示例 11
4 从客户端应用程序调用安全的微服务 12
5 使用 OAuth 2 scope 执行服务级别授权 14
  5.1 从授权服务器获取作用域访问令牌 14
  5.2 使用 OAuth 2.0 scope 保护对微服务的访问 15
总结 17

1. 构建你的第一个微服务

我们以一个零售商店应用程序为例，我们使用一组微服务来构建它。我们将使用 Spring Boot (https://spring.io/projects/spring-boot) 构建我们的第一个微服务，它接受创建和管理订单的请求。

Spring Boot 是一个基于 Spring 平台的框架，它允许您通过使用一组特殊的注解修饰您的代码，将用 Java 编程语言编写的函数转换为网络可访问的函数，称为服务或 API。代码示例都可以在 GitHub (https://github.com/yyit2022/microservice-security) 上找到。

图 1 显示了一组微服务，它们是我们正在构建的零售商店应用程序的一部分，以及一组消费者应用程序。事实上，消费者应用程序是我们构建的微服务的消费者。

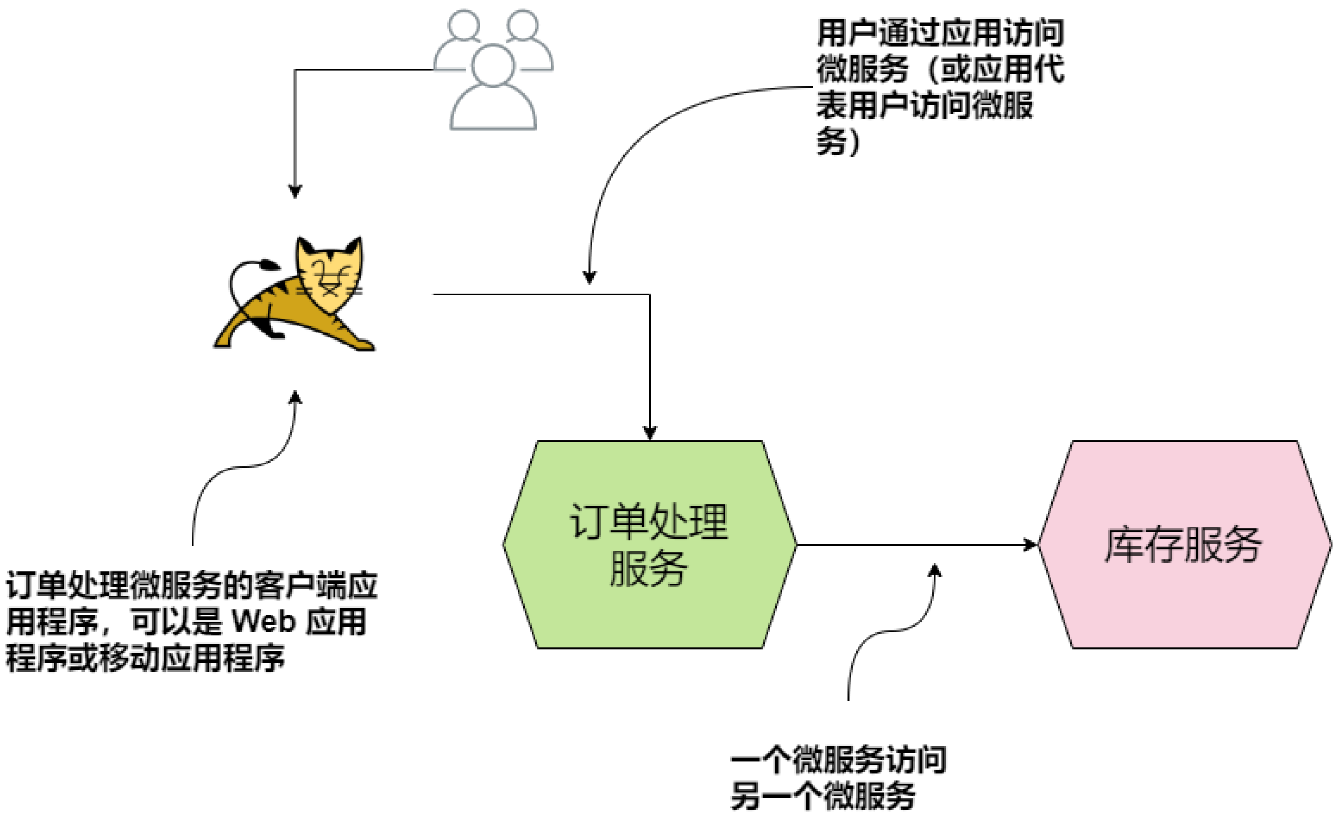


图 . *1* 在这个典型的微服务部署中，消费者应用程序（*Web* 应用程序或移动应用程序）代表其最终用户访问微服务，而微服务之间相互通信。

1.1. 克隆示例库

(1) 克隆示例 Git 源码库，运行以下命令：

```
git clone  https://github.com/yyit2022/microservice-security.git
```

1.2. 编译订单处理微服务

完成克隆步骤后，就该动手运行您的第一个微服务了。首先，在您的操作系统中打开命令行工具，然后导航到文件系统上克隆示例源码库的位置：

```
cd [microservice-security]/lesson02/sample01
```

在 `lesson02/sample01` 目录中，您将找到与订单处理微服务对应的源代码。在该目录中，执行以下命令以构建订单处理微服务：

```
mvn clean install
```

前面的命令指示 Maven 编译您的源代码并生成一个可运行的 artifact，称为 Java 存档 (JAR) 文件。请注意，您需要安装 Java 和 Maven 才能成功执行此步骤。如果您的构建成功，您将看到消息 `BUILD SUCCESS`。

如果构建成功，您应该在当前目录中看到一个名为 target 的目录。目标目录应包含名为 com.yyit.mss.ch02.sample01-1.0.jar 的文件。（其他文件将在 `target` 目录中，但您目前对它们不感兴趣。）然后从 lesson02/sample01/ 目录运行以下命令以启动订单处理微服务。在这里，我们使用了一个名为 spring-boot 的 Maven 插件：

```
mvn spring-boot:run
```

如果微服务启动成功，您应该会在终端上看到一堆消息。在消息堆栈的底部，您应该会看到以下消息：

```
Started OrderApplication in <X> seconds
```

默认情况下，Spring Boot 在 HTTP 8080 端口启动微服务。如果您的本地机器上有其他服务在 8080 端口上运行，请确保停止它们；或者，您可以通过在 `lesson02/sample01/src/main/resources/application.properties` 文件中适当地更改 `server.port` 属性的值来更改订单处理微服务的默认端口。

1.3. 访问订单处理微服务

默认情况下，Spring Boot 运行嵌入式 Apache Tomcat Web 服务器，该服务器监听端口 8080 上的 HTTP 请求。在这里您将使用 curl 作为客户端应用程序访问您的微服务。如果您在自定义端口上运行订单处理微服务，请保将以下命令中的端口（8080）值替换为您使用的值。要调用微服务，请打开命令行客户端并执行以下 curl 命令：

```
curl -v http://localhost:8080/orders \
-H 'Content-Type: application/json' \
--data-binary @- << EOF
{
  "items":[
    {
      "itemCode":"IT0001",
      "quantity":3
    },
    {
      "itemCode":"IT0004",
      "quantity":1
    }
  ],
  "shippingAddress":"福建省厦门市XX区XX街道XX小区XX栋XX室"
}
EOF
```

应该在终端上看到类似以下的响应数据：

```
{
  "orderId":"cd992a9f-6900-4625-b73a-0c526451dc81",
  "items":[{
    "itemCode":"IT0001","quantity":3},
    {"itemCode":"IT0004","quantity":1}],
  "shippingAddress":"福建省厦门市XX区XX街道XX小区XX栋XX室"
}
```

如果看到此消息，则您已成功开发、部署和测试了你的第一个微服务！

这个示例都使用 HTTP（而非 HTTPS）端点，这样您就不必设置正确的证书，并使您可以在需要时检查线路（网络）上传递的消息。在生产系统中，我们不建议对任何端点使用 HTTP。您应该仅通过 HTTPS 公开所有端点。在后面，我们将讨论如何使用 HTTPS 保护微服务。

当您执行上述命令时，curl 向位于服务器 localhost 上端口 8080（本地机器）上的 /orders 资源发起了 HTTP POST 请求。请求的内容（有效负载）表示为将要传送到特定地址的两件商品下的订单。Spring Boot 服务器运行时（嵌入式 Tomcat）将此请求分派给 Order Processing 微服务的 `placeOrder` 方法（在 Java 代码中），后者响应消息。

1.4 源代码目录中有什么?

让我们在 sample01 目录中导航并检查其内容。您应该会看到一个名为 `pom.xml` 的文件和一个名为 `src` 的目录。导航到 src/main/java/com/yyit/mss/ch02/sample01/service/ 目录。您将看到两个文件：`OrderApplication.java` 和 `OrderProcesingService.java`。

在深入研究这些文件的内容之前，让我们解释一下在此处尝试构建的内容。微服务是网络可访问功能的集合。在这种情况下，网络可访问意味着这些功能可以通过 Web 浏览器和移动应用程序等应用程序或 curl (https:// curl.haxx.se/) 能够通过 HTTP 进行通信。通常，微服务中的函数作为对 REST 资源 (https://spring.io/guides/tutorials/rest/) 的操作公开。通常，资源表示您打算检查或操作的对象或实体。映射到 HTTP 时，资源通常由请求 URI 标识，verb 由 HTTP method(GET、POST、PUT、DELETE、OPTION、PATCH) 表示。

一个电子商务应用程序使用微服务来检索订单详细信息的场景。映射到微服务中该特定功能的 HTTP 请求模板类似于以下内容：

```
GET /orders/{orderid}
```

`GET` 是本例中使用的 HTTP 方法，因为您正在执行数据检索操作。`/orders/{orderid}` 是托管相应微服务的服务器上的资源路径。此路径可用于唯一标识订单资源。`{orderid}` 是一个变量，需要在实际 HTTP 请求中替换为适当的值。像 `GET /orders/d59dbd56-6e8b-4e06-906f-59990ce2e330` 这样会要求微服务检索 `ID` 为 `d59dbd56-6e8b-4e06-906f-59990ce2e330` 的订单的详细信息。

1.5 理解微服务的源代码

让我们看一下代码示例，看看如何用 Java 开发一个方法并使用 Spring Boot 将其公开为 HTTP 资源。使用操作系统中的文件浏览器打开位于 sample01/src/main/java/com/yyit/mss/ch02/sample01/service 的目录，然后在文本编辑器中打开 `OrderProcessingService.java` 文件。如果您熟悉 Eclipse、NetBeans、IntelliJ IDEA 或任何类似的 Java 集成开发环境 (IDE)，您可以将示例作为 Maven 项目导入到 IDE。以下清单显示了 `OrderProcessingService.java` 文件的内容。

清单 *2.1* `OrderProcessingService.java` 文件的内容

```
@RestController // 通知 Spring Boot 运行时将此类公开为微服务的 Rest 资源
@RequestMapping("/orders") // 指定服务所有资源所在的路径
public class OrderProcessingService {
    private Map<String, Order> orders = new HashMap<>();

    @PostMapping // 通知 Spring Boot 运行时将此方法公开为 POST HTTP 方法
    public ResponseEntity<Order> placeOrder(@RequestBody Order order) {
        System.out.println("接收到订单 "
                + order.getItems().size() + " 项");
        order.getItems().forEach((lineItem) ->
                System.out.println("订单项: " + lineItem.getItemCode() +
```

```
                        " 数量: " + lineItem.getQuantity()));
        String orderId = UUID.randomUUID().toString();
        order.setOrderId(orderId);
        orders.put(orderId, order);
        return new ResponseEntity<Order>(order, HttpStatus.CREATED);
    }
}
```

这段代码是一个简单的 Java 类，带有一个名为 `placeOrder` 的函数。您可能会注意到，我们使用 `@RestController` 注解装饰了该类，以通知 Spring Boot 运行时您有兴趣将此类公开为微服务。 `@RequestMapping` 注解指定了服务的所有资源所在的路径。我们还使用 `@PostMapping` 注解修饰了 `placeOrder` 函数，它通知 Spring Boot 运行时将此函数公开为 `/orders` 上下文中的 `POST` HTTP 方法（操作）。 `@RequestBody` 注解表示 HTTP 请求中的有效负载将分配给 `Order` 类型的对象。

同一目录中的另一个文件名为 `OrderApplication.java`。使用文本编辑器打开此文件并检查其内容，如下所示：

```
@SpringBootApplication
public class OrderApplication {
    public static void main(String args[]) {
        SpringApplication.run(OrderApplication.class, args);
    }
}
```

这个简单的 Java 类只有 `main` 函数。 `@SpringBootApplication` 注解通知 Spring Boot 运行时这个应用程序是一个 Spring Boot 应用程序。它还对 `OrderApplication` 类的同一个包中的 `Controller` 类（例如您之前看到的 `OrderProcessingService` 类）进行运行时检查。 `main` 函数是当您命令 JVM 运行特定 Java 程序时由 JVM 调用的函数。在 `main` 函数中，通过 `SpringApplication` 类的 `run` 实用程序函数启动 Spring Boot 应用程序，该类位于 Spring 框架中。

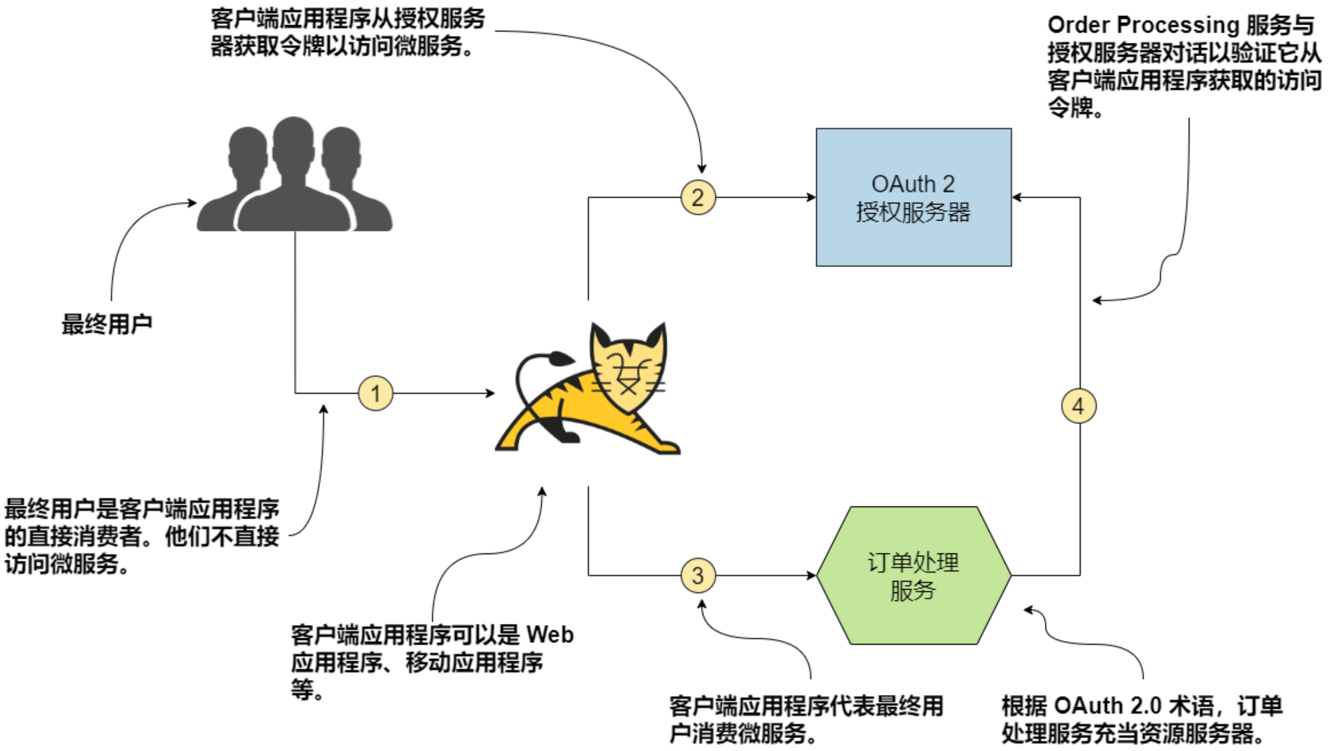
2. 设置 OAuth 2 服务器

现在已经启动并运行了你的第一个微服务，我们可以开始进入主要聚焦点：保护微服务。您将使用 OAuth 2 来保护您的边缘微服务。

当与 JWT 结合使用时，OAuth 2 可以成为一种高度可扩展的身份验证和授权机制，这对于保护微服务至关重要。

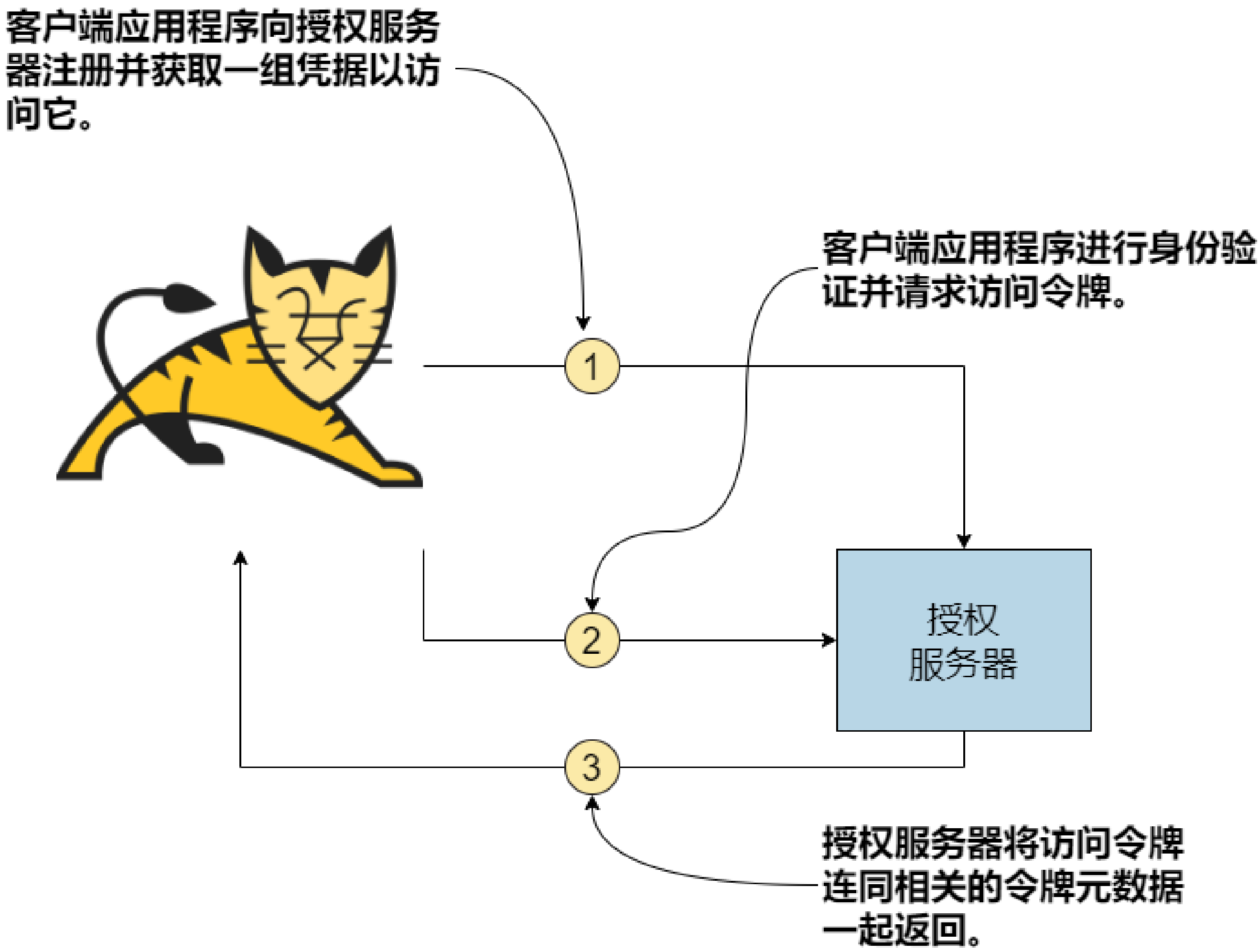
2.1 与授权服务器的交互

在 OAuth 2 流程中，客户 端应用程序、最终用户和资源服务器都在不同阶段直接与授权服务器交互（见图 2）。在从授权服务器请求令牌之前，客户端应用程序必须向它注册自己。



□ . *2 OAuth 2* □□□

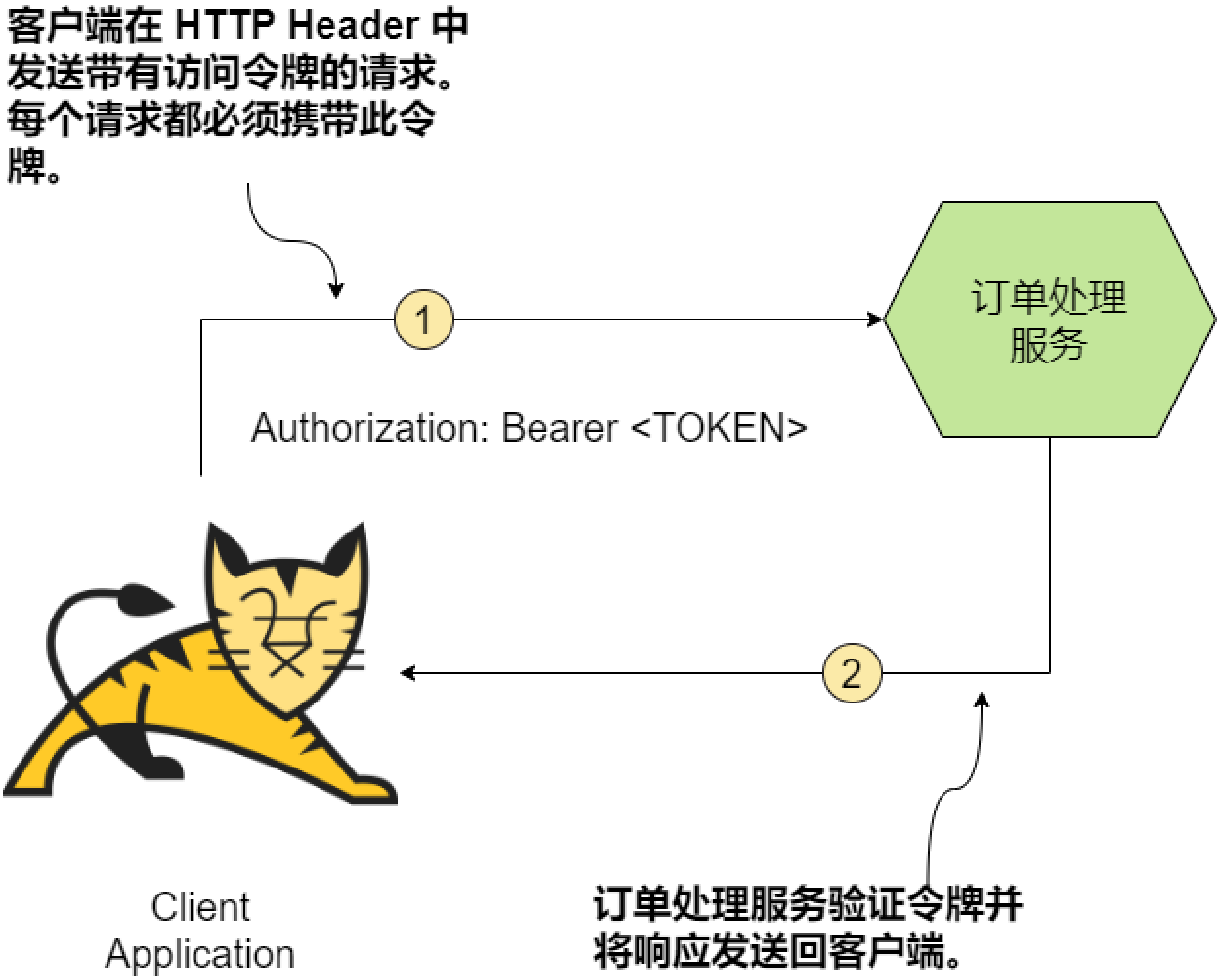
□□□ (https://tools.ietf.org/html/rfc7591)□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ 3□□



□ . *3* □□

Order Processing 微服务在这里扮演了资源服务器的角色，它会从客户端接收授权服务器发出的令牌，通常作为 HTTP 请求头或客户端发出 HTTP 请求时的查询参数（参见第 1 步）图 4)。建议客户端通过 *HTTPS* 与微服务通信，并在 HTTP header 而不是查询参数中发送令牌。因为查询参数是在 URL 中发送的，所以这些参数可以记录在服务器日志中。因此，任何有权访问日志的人都可以看到此信息。

使用 TLS 来保护 OAuth 2 流中所有实体之间的通信（或换句话说，使用 HTTPS）非常重要。授权服务器为访问微服务（或资源）而颁发的令牌（访问令牌）必须像密码一样受到保护。我们不会通过纯 HTTP 发送密码，而是始终使用 HTTPS。因此，我们在通过网络发送访问令牌时遵循相同的过程。


客户端在 HTTP Header 中发送带有访问令牌的请求。每个请求都必须携带此令牌。

订单处理服务验证令牌并将响应发送回客户端。

图 . *4* 客户端应用程序在 *HTTP* 授权请求头中传递 *OAuth* 访问令牌以从资源服务器访问资源。

收到访问令牌后，订单处理微服务应在授予对其资源的访问权限之前根据授权服务器对其进行验证。 OAuth 2 授权服务器通常支持 OAuth 2 令牌自省配置文件 (https://tools.ietf.org/html/rfc7662) 或资源服务器的类似替代方案，以检查访问令牌的有效性（见图 5）。如果访问令牌是自包含的 JWT，资源服务器可以自行验证它，而无需与授权服务器交谈。我们将在后面详细讨论自包含的 JWT。

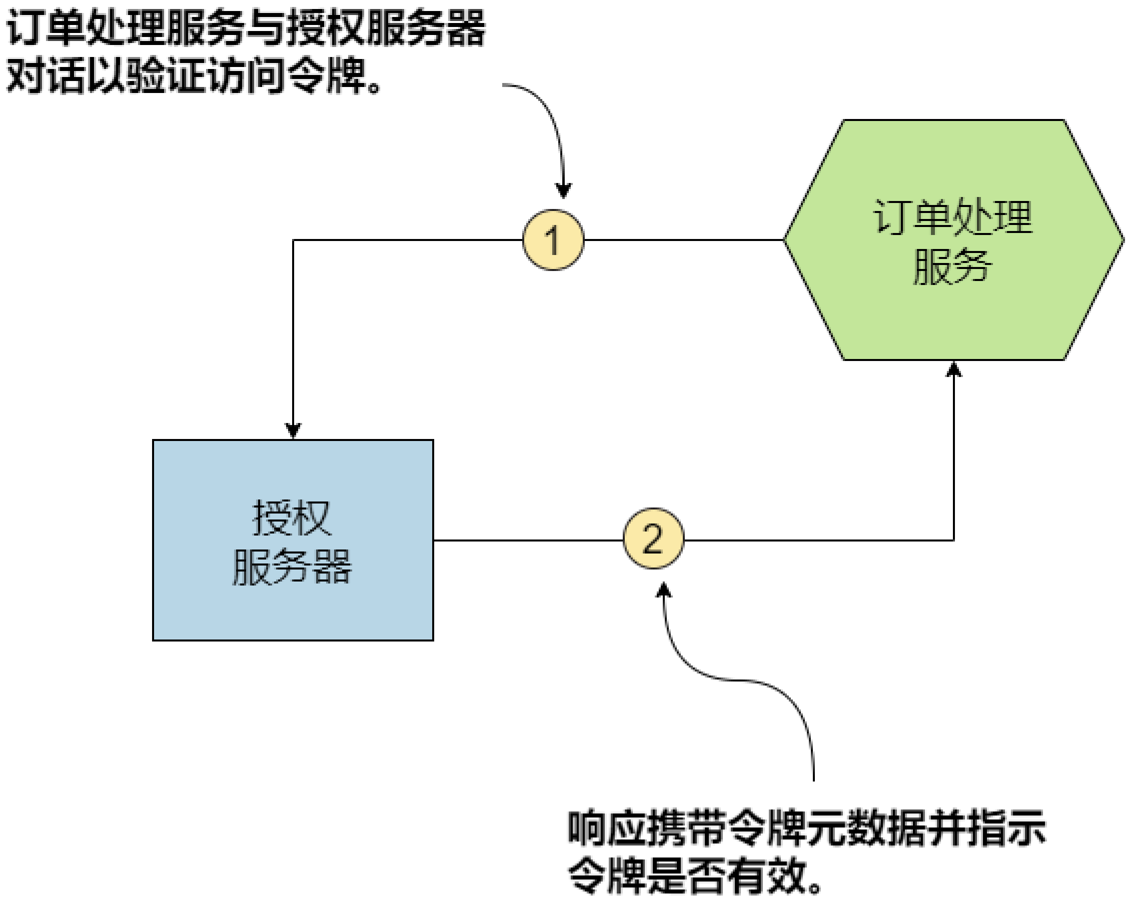


图 . 5 订单处理微服务（资源服务器）通过与授权服务器对话来内省访问令牌。

2.2 运行 OAuth 2 授权服务器

许多生产级 OAuth 2.0 授权服务器都在那里，既有专有的，也有开源的。 然而，在本章中，我们使用一个能够发布访问令牌的简单授权服务器。 它是使用 Spring Boot 构建的。在之前克隆的 Git 源码库中，您应该会在目录 lesson02 下找到一个名为 sample02 的目录。 您将在那里找到简单 OAuth 2 授权服务器的源代码。首先，编译并运行它；然后查看代码以了解它的作用。

要编译，使用命令行客户端导航到 lesson02/sample02 目录。从该目录中，执行以下 Maven 命令以编译和构建可运行 artifact:

```
mvn clean install
```

如果您的构建成功，您将看到消息 BUILD SUCCESS。您应该在名为 target 的目录中找到名为 com.yyit.mss.ch02.sample02-1.0.jar 的文件。从lesson02/sample02 目录中使用命令行客户端执行以下命令，以运行OAuth 2.0 授权服务器:

```
mvn spring-boot:run
```

如果成功运行服务器，应该看到以下消息:

```
Started OAuthServerApplication in <X> seconds
```

此消息表明您已成功启动授权服务器。默认情况下，OAuth 2.0 授权服务器在 HTTP 端口 8085 上运行。如果您的本地机器上有任何其他服务在 8085

端口上运行，请确保停止它们；或者，您可以通过在 lesson02/sample02/src/main/resources/application.properties 文件中适当地更改 server.port 属性的值来更改授权服务器的默认端口。

2.3 从 OAuth 2 授权服务器获取访问令牌

要从授权服务器获取访问令牌，请使用 HTTP 客户端向服务器发出 HTTP 请求。在现实世界中，访问微服务的客户端应用程序会发出此请求。为此，您将使用 curl 作为 HTTP 客户端。要从授权服务器（在端口 8085 上运行）请求访问令牌，请使用命令行客户端运行以下命令：

```
curl  -u "orderprocessingapp:orderprocessingsecret" \
-H "Content-Type: application/x-www-form-urlencoded" \
-d 'grant_type=client_credentials'  \
-X POST http://localhost:8085/oauth2/token
```

如果您的请求成功，您应该会看到类似于以下内容的响应：

```
{
  "access_token":
"eyJraWQiOiJhOTQ5ODkyNC1kODE1LTRlZmItODlmYS1lYmFkMmFkOTU4OGMiLCJhbGciOiJSUzI1NiJ9.eyJz
dWIiOiJ5eWl0IiwiYXVkIjoieXlpdCIsIm5iZiI6MTY1MTExNzE2MCwic2NvcGUiOlsib3BlbmlkIiwib3JkZX
JzIl0sImlzcyI6Imh0dHA6XC9cL2xvY2FsaG9zdDo5MDAwIiwiZXhwIjoxNjUxMTE3NDYwLCJpYXQiOjE2NTEx
MTcxNjB9.SboppQgJ57rKCiq2sIivOeOKxNJoYjEZ-
YXFjeLAE1x80cPLBAwf106YuUsZJNyxW_3uEn0K7JNIT7DWg3mVdhIHe5XOAD7W6nRR3DP_e3WXlXwugaNxDII
nXBCFqTPleVldeSXEjMMpSGrsDnaIClAV1D9c0vrfZCtrUT0CvUO_tgMWtpQyCXGHLeeDsDYtasxXvgsocnozf
oNpQyxvBdARtsqZEmBIErDPP-gr7FN_KETqiUfQ_CZMjVYV-992SNW-
l7kOxdI03LhLygQ6CBiQWkCehQu1YMGgroiMVS6x1-yeT1A2LpClnnr2HsUbZAa9BDEMUzLqjDVZXCopqQ",
  "scope":"openid orders",
  "token_type":"Bearer",
  "expires_in":300
}
```

快速浏览一下这个请求并尝试理解它。您可以将 `orderprocessingapp:orderprocessingsecret` 视为客户端应用程序的用户名 (orderprocessingapp) 和密码 (orderprocessingsecret)。唯一的区别是这些凭证属于应用程序，而不是用户。用于请求令牌的应用程序需要携带唯一标识符和授权服务器已知的密钥。提供给 `curl` 的 `-u` 标志指示它创建一个基本的身份验证请求头并将其作为 HTTP 请求的一部分发送到授权服务器。然后 curl base64 编码 `orderprocessingapp:orderprocessingsecret` 字符串并创建基本身份验证 HTTP 请求头，如下所示：

```
Authorization: Basic b3JkZXJwcm9jZXNzaW5nYXBwOm9yZGVycHJvY2Vzc2luZ3NlY3JldA==
```

Basic 关键字后面的字符串是 `orderprocessingapp:orderprocessingsecret` 的 base64 编码值。您可能已经注意到，您正在向 OAuth 2 授权服务器的令牌端点发送基本身份验证请求头，因为令牌端点受基本身份验证 (https://tools.ietf.org/html/rfc2617) 保护。由于客户端应用程序在此处请求令牌，因此基本身份验证请求头应包含客户端应用程序的凭据，而不是用户的凭据。请注意，这里的基本身份验证不用于保护资源服务器（或微服务）；您为此目的使用 OAuth 2。此时的基本身份验证仅用于从授权服务器获取访问微服务所需的 OAuth 令牌。

```
curl  -H "Authorization: Basic
b3JkZXJwcm9jZXNzaW5nYXBwOm9yZGVycHJvY2Vzc2luZ3NlY3JldA==" \
-H "Content-Type: application/x-www-form-urlencoded" \
```

```
-d 'grant_type=client_credentials'  \
-X POST http://localhost:8085/oauth2/token
```

2.4 理解访问令牌响应

以下列表提供了有关来自授权服务器的上述 JSON 响应的详细信息。

- *access_token* — 授权服务器向客户端应用程序颁发的令牌值（在本例中为 curl）。
- *token_type* — 令牌类型（当我们在附录 A 中讨论 OAuth 2 时将详细介绍此主题）。我们今天看到的大多数 OAuth 部署都使用不记名令牌。
- *expires_in* — 令牌的有效期，以秒为单位。在此期间之后，令牌将被视为无效（过期）。
- *scope* — 允许令牌在资源服务器（微服务）上执行的操作。

3 使用 OAuth 2 保护微服务

到目前为止，您已经学习了如何开发您的第一个微服务以及如何设置 OAuth 2 授权服务器以获取访问令牌。在本节中，您将看到如何保护您开发的微服务。到目前为止，您已经在没有任何安全措施的情况下访问了它。

3.1 基于 OAuth 2 的保护

使用 OAuth 2 进行保护后，订单处理微服务现在需要来自调用客户端应用程序的有效安全令牌（访问令牌）。然后，它会在授权服务器的帮助下验证此访问令牌，然后再授予对其资源的访问权限。图 6 说明了这种情况。

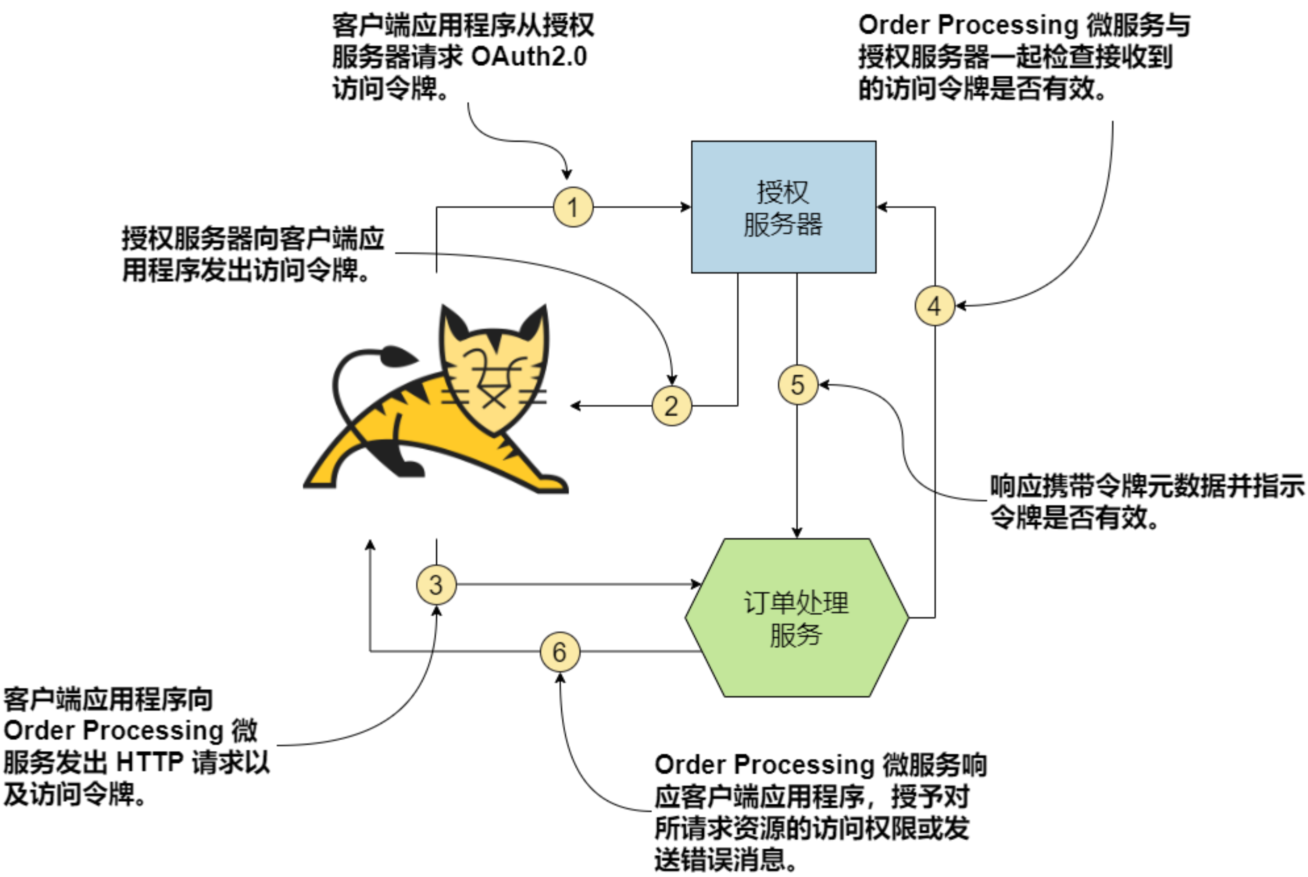


图 . *6* 客户端应用程序使用从授权服务器获取的访问令牌访问安全微服务。 *Order Processing* 微服务在授予对其资源的访问权限之前与授权服务器对话以验证令牌。

以下是图 6 所示的每个步骤中发生的情况：

1. 客户端应用程序从授权服务器请求 OAuth 2 访问令牌。
2. 响应步骤 1 中的请求，授权服务器向客户端应用程序发出访问令牌。
3. 客户端应用程序向订单处理微服务发出 HTTP 请求。此请求携带在步骤 2 中获取的访问令牌作为 HTTP 请求头。
4. Order Processing 微服务与授权服务器一起检查接收到的访问令牌是否有效。
5. 响应步骤 4 中的请求，授权服务器检查提供的访问令牌是否是系统中的活动令牌（其状态为活动）以及该令牌在该特定时刻是否有效（未过期）.然后它响应 Order Processing 微服务，指示访问令牌是否有效。
6. 响应第 3 步中的请求，并根据第 5 步中的结果，订单处理微服务响应客户端应用程序，要么授予对所请求资源的访问权限，要么发送错误消息。

已经使用 client_credentials 授权类型从授权服务器获取访问令牌。在这种特殊情况下，授权服务器的令牌端点通过使用 client ID 和应用程序的 client secret 进行基本身份验证来保护。当客户端应用程序不需要担心最终用户时，client_credentials 授予类型很好。如果必须，它应该选择合适的授权类型。 ***client_credentials*** 授权类型主要用于系统到系统的身份验证。

3.2 运行示例

导航到你从命令行应用程序从 Git 仓库克隆示例的目录，然后转到 lesson02/sample03 目录。从该目录中，执行以下 Maven 命令以构建示例：

```
mvn clean install
```

如果构建成功，您应该在当前目录中看到一个名为 target 的目录。target 目录应包含名为 com.yyit.mss.ch02.sample03-1.0.jar 的文件。然后从 lesson02/sample03/ 目录运行以下命令以启动安全的订单处理微服务。在这里，我们使用了一个名为 spring-boot 的 Maven 插件：

```
mvn spring-boot:run
```

如果您成功运行服务器，您应该看到如下消息：

```
Started OrderApplication in <X> seconds
```

现在运行前面使用的相同 curl 命令来访问订单处理微服务：

```
curl -v http://localhost:8080/orders \
-H 'Content-Type: application/json' \
--data-binary @- << EOF
{
  "items":[
    {
      "itemCode":"IT0001",
      "quantity":3
    },
    {
      "itemCode":"IT0004",
      "quantity":1
    }
```

```
  ],
  "shippingAddress":"福建省厦门市XX区XX街道XX小区XX栋XX室"
}
EOF
```

您应该会看到一条错误消息，指出请求不成功。预期的响应消息如下：

```
HTTP/1.1 401
```

您的订单处理微服务现已受到保护，如果没有从授权服务器获取的有效访问令牌，则无法再访问。要了解这是如何发生的，请查看订单处理微服务的修改后的源代码。使用您喜欢的文本编辑器或 IDE，打开位于 src/main/java/com/yyit/mss/ch02/sample03/configuration 目录中的 SpringSecurityConfiguration.java 文件。这个类的一个补充是注解 @EnableWebSecurity。此注解通知您的 Spring Boot 运行时将保护应用于此微服务的资源。

4 从客户端应用程序调用安全的微服务

在客户端应用程序可以访问您的安全订单处理微服务之前，它应该从授权服务器获取 OAuth2.0 访问令牌。如前面所述，客户端应用程序至少需要有效的 Client ID 和 Client Secret 才能获取此令牌。目前在您的 OAuth 2.0 授权服务器上注册的 Client ID 和 Client Secret 分别是 `orderprocessingservice` 和 `orderprocessingservicesecret`。和前面一样，您可以使用以下 curl 命令来获取访问令牌：

```
curl  -H "Authorization: Basic
b3JkZXJwcm9jZXNzaW5nYXBwOm9yZGVycHJvY2Vzc2luZ3NlY3JldA==" \
-H "Content-Type: application/x-www-form-urlencoded" \
-d 'grant_type=client_credentials'  \
-X POST http://localhost:8085/oauth2/token
```

如果请求成功，您应该得到一个访问令牌作为响应，如下所示：

```
{
  "access_token":
"eyJraWQiOiI0MTU1MTQ2OC0zNjUyLTQ4ZTktYjNlMy04OTYyYTRiYTAzZDUiLCJhbGciOiJSUzI1NiJ9.eyJz
dWIiOiJvcmRlcnByb2Nlc3NpbmdzZXJ2aWNlIiwiYXVkIjoib3JkZXJwcm9jZXNzaW5nc2VydmljZSIsIm5iZi
I6MTY1MTEyOTI5MCwic2NvcGUiOlsicmVhZCIsIm9wZW5pZCIsIndyaXRlIl0sImlzcyI6Imh0dHA6XC9cL2xv
Y2FsaG9zdDo4MDg1IiwiZXhwIjoxNjUxMTI5NTkwLCJpYXQiOjE2NTExMjkyOTB9.eZ2Jkwzo10qTVeMPCaJHc
Qy329bXky1rRcVpQlY6mzqp0EnlGac86CGp2Fz_P1GxYCrvzpqUgO2B26IMd4n2y9D1j-
c3wqyndDAw7DdK673YBCpS-7_9Ovgb1q9kON0uvlLGPT4cO91_F0W2Tj6Ar6XyhiZKRmfblEYAKbNhh14-
LC8l4dSwEwWtr7JzTTSmEjuFUoD955QBN3uDz3fHXW4TyRgs9zc59MEEXBCqyRETNJIOxRaRPlU-
XGryHX8ntKp7UkrwbNRPLo93GUACZ9MMGTaltstbjsCnFmsULXXwam7g3cXm6HY0bqH9bWdJfYSfqkfj4diaHb
_XZML7BQ",
  "scope":"read openid write",
  "token_type":"Bearer",
  "expires_in":299
}
```

如前所述，`eyJraWQiOiI0MTU1MTQ2OC0zNjUyLTQ4ZTktYjNlMy04OTYyYTRiYTAzZDUiLCJhbGciOiJSUzI1NiJ9…` 是您获得的访问令牌的值，其有效期为将近 5 分钟（299 秒）。需要将此访问令牌提供给您将向订单处理微服务发出的 HTTP 请求。您需要将令牌作为名为 Authorization 的

HTTP 请求头发送，并且请求头值需要以字符串 Bearer 为前缀，如下所示：

```
Authorization: Bearer
eyJraWQiOiI0MTU1MTQ2OC0zNjUyLTQ4ZTktYjNlMy04OTYyYTRiYTAzZDUiLCJhbGciOiJSUzI1NiJ9...
```

访问订单处理微服务的新 curl 命令如下：

```
curl -v http://localhost:8080/orders \
-H 'Authorization: Bearer
eyJraWQiOiI0MTU1MTQ2OC0zNjUyLTQ4ZTktYjNlMy04OTYyYTRiYTAzZDUiLCJhbGciOiJSUzI1NiJ9.eyJzd
WIiOiJvcmRlcnByb2Nlc3NpbmdzZXJ2aWNlIiwiYXVkIjoib3JkZXJwcm9jZXNzaW5nc2VydmljZSIsIm5iZiI
6MTY1MTEyOTI5MCwic2NvcGUiOlsicmVhZCIsIm9wZW5pZCIsIndyaXRlIl0sImlzcyI6Imh0dHA6XC9cL2xvY
2FsaG9zdDo4MDg1IiwiZXhwIjoxNjUxMTI5NTkwLCJpYXQiOjE2NTExMjkyOTB9.eZ2Jkwzo10qTVeMPCaJHcQ
y329bXky1rRcVpQlY6mzqp0EnlGac86CGp2Fz_P1GxYCrvzpqUgO2B26IMd4n2y9D1j-
c3wqyndDAw7DdK673YBCpS-7_9Ovgb1q9kON0uvlLGPT4cO91_F0W2Tj6Ar6XyhiZKRmfblEYAKbNhh14-
LC8l4dSwEwWtr7JzTTSmEjuFUoD955QBN3uDz3fHXW4TyRgs9zc59MEEXBCqyRETNJIOxRaRPlU-
XGryHX8ntKp7UkrwbNRPLo93GUACZ9MMGTaltstbjsCnFmsULXXwam7g3cXm6HY0bqH9bWdJfYSfqkfj4diaHb
_XZML7BQ'
-H 'Content-Type: application/json' \
--data-binary @- << EOF
{
  "items":[
    {
      "itemCode":"IT0001",
      "quantity":3
    },
    {
      "itemCode":"IT0004",
      "quantity":1
    }
  ],
  "shippingAddress":"福建省厦门市XX区XX街道XX小区XX栋XX室"
}
EOF
```

请注意，`-H` 参数用于将访问令牌作为名为 Authorization 的 HTTP 请求头传递。这一次，您应该看到订单处理微服务响应一条正确的消息，说明订单成功：

```
{
  "orderId":"481cd245-df48-4088-ba6c-e2daeac69b6c",
  "items":[
    {"itemCode":"IT0001","quantity":3},
    {"itemCode":"IT0004","quantity":1}
  ],
  "shippingAddress":"福建省厦门市XX区XX街道XX小区XX栋XX室"
}
```

如果您看到此消息，则您已成功创建、部署和测试了安全的微服务。恭喜！客户端应用程序 (curl) 在 HTTP 请求头中发送到订单处理微服务的访问令牌已根据授权服务器进行验证。

这个过程称为令牌自省。由于内省操作的结果最终成功，订单处理微服务授予对其资源的访问权限。

5 使用 OAuth 2 scope 执行服务级别授权

需要有效的访问令牌才能访问微服务。身份验证是应用于微服务以防止其受到欺骗的第一级防御。

在授予对微服务的访问权限之前发生的身份验证步骤可确保调用实体是系统中的有效客户端（用户、应用程序或两者）。然而，身份验证并未提及客户端在系统中拥有的权限级别。

一个给定的微服务可能有多个操作。例如，订单处理微服务有一个用于创建订单的操作 (`POST /orders`) 和另一个用于检索订单详细信息的操作 (`GET /orders/{id}`)。微服务中的每个操作可能需要不同级别的访问权限。

privilege 描述了您被允许对资源执行的操作。通常，您在组织中的一个或多个角色描述了您在该组织内可以执行的操作以及您不可以执行的操作。privilege 还可以表明地位或可信度。如果您乘坐过商业航空公司，您可能熟悉属于航空公司飞行常客计划的旅客的会员身份。同样，privilege 是用户或应用程序在系统中拥有的访问级别的指示。

在 OAuth 2.0 的世界中，privilege 被映射到一个 scope。scope 是抽象 privilege 的方式。privilege 可以是用户的角色、成员身份、可信度或其他内容。它也可以是几个这样的属性的组合。您使用 scope 来抽象 privilege 的含义。scope 声明调用客户端应用程序授予对资源的访问权限所需的 privilege。例如，placeOrder 操作需要一个称为 write 的 scope，而 getOrder 操作需要一个称为 read 的 scope。

5.1 从授权服务器获取作用域访问令牌

您在本章中构建的授权服务器包含两个应用程序：一个带有 Client ID `orderprocessingapp`，用于访问微服务，另一个带有 Client ID `orderprocessingservice`。您以这样一种方式配置这些应用程序，即具有 Client ID `orderprocessingapp` 的第一个应用程序有权获得读取和写入 scope，而具有 Client ID `orderprocessingservice` 的第二个应用程序仅有权获得读取 scope，如以下列表：

```
 @Bean
    public RegisteredClientRepository registeredClientRepository(PasswordEncoder
encoder) {
        RegisteredClient registeredClient = RegisteredClient.withId(UUID.randomUUID()
.toString())
                .clientId("orderprocessingapp")
                .clientSecret(encoder.encode("orderprocessingsecret"))
                .clientAuthenticationMethod(ClientAuthenticationMethod
.CLIENT_SECRET_POST)
                .authorizationGrantType(AuthorizationGrantType.AUTHORIZATION_CODE)
                .authorizationGrantType(AuthorizationGrantType.CLIENT_CREDENTIALS)
                .authorizationGrantType(AuthorizationGrantType.REFRESH_TOKEN)
                .redirectUri("https://oidcdebugger.com/debug")
                .redirectUri("https://www.baidu.com")
                .scope(OidcScopes.OPENID)
                .scope("read")
                .scope("write")
                .build();

        RegisteredClient registeredClient1 = RegisteredClient.withId(UUID.randomUUID(
).toString())
                .clientId("orderprocessingservice")
                .clientSecret(encoder.encode("orderprocessingservicesecret"))
                .clientAuthenticationMethod(ClientAuthenticationMethod
.CLIENT_SECRET_BASIC)
```

```
                .clientAuthenticationMethod(ClientAuthenticationMethod
.CLIENT_SECRET_POST)
                .authorizationGrantType(AuthorizationGrantType.CLIENT_CREDENTIALS)
                .redirectUri("https://oidcdebugger.com/debug")
                .redirectUri("https://www.baidu.com")
                .scope(OidcScopes.OPENID)
                .scope("read")
                .build();
        return new InMemoryRegisteredClientRepository(registeredClient
,registeredClient1);
    }
```

此代码表明，任何使用 `orderprocessingapp` 的人都可以在 `read` 和 `write` scope 内获取访问令牌，而任何使用 `orderprocessingservice` 的用户只能在 `read` scope 内获取访问令牌。在迄今为止获取访问令牌的所有请求中，您使用 `orderprocessingapp` 作为 Client ID 并请求了 read 和 write 两个 scope。

现在执行相同的请求以获取一个以 `orderprocessingservice` 作为 Client ID 的访问令牌，以查看令牌响应是什么样的。执行此 curl 命令以发出令牌请求：

```
curl  -u "orderprocessingservice:orderprocessingservicesecret" \
-H "Content-Type: application/x-www-form-urlencoded" \
-d 'grant_type=client_credentials'  \
-X POST http://localhost:8085/oauth2/token
```

如果令牌请求成功，您应该会看到以下响应：

```
{
  "access_token":
"eyJraWQiOiIyNzlkNzM5Zi1mNjIzLTRjMDEtOTFmYi00NTVmZThjMzFiZTIiLCJhbGciOiJSUzI1NiJ9.eyJz
dWIiOiJvcmRlcnByb2Nlc3NpbmdzZXJ2aWNlIiwiYXVkIjoib3JkZXJwcm9jZXNzaW5nc2VydmljZSIsIm5iZi
I6MTY1MTEzMDQ5Nywic2NvcGUiOlsicmVhZCIsIm9wZW5pZCJdLCJpc3MiOiJodHRwOlwvXC9sb2NhbGhvc3Q6
ODA4NSIsImV4cCI6MTY1MTEzMDc5NywiaWF0IjoxNjUxMTMwNDk3fQ.ifh0XrokV_mueOddSq-
3dZpcrAkbrZsgFFE9A7l9oIrYbudEXxgeKt71MyNLe7vkS_TO1HijOusTxoqhQH-
CbsXmz_44rOPinSYLXLQJhlXLG7YGisePchbwr4MdR_j7PbtpVtaKPpGF8_y5WKLDdXbYhwZZ2cSDhVBljGKhj
imakeKRb8FvpAATasdOdxSCFRB_9d2e6phoSKMOSUHRvLEcv__UIWvaBzmaDV1D241Tblziw8SFNWpDdb0HXEI
DCec-aktPEyxd2qPweEW5G-fTtrAb7bbSY5yI8Mlax7z0q1XGPHNuuobG6r1GvtB5x2iaS-MuzpRZSWLa-
t1ajQ",
  "scope":"read openid",
  "token_type":"Bearer",
  "expires_in":299
}
```

5.2 使用 OAuth 2.0 scope 保护对微服务的访问

现在您已经了解授权服务器如何根据 scope 向令牌授予权限。您将看到资源服务器或微服务如何对其想要保护的资源实施这些 scope。以下清单（lesson02/sample03/src/main/java/com/yyit/mss/ch02/sample03/configuration/SpringSecurityConfiguration.java 类文件）解释了资源服务器如何执行这些规则。

```
@EnableWebSecurity
public class SpringSecurityConfiguration extends WebSecurityConfigurerAdapter {

    private static final String SECURED_READ_SCOPE = "SCOPE_read";

    private static final String SECURED_WRITE_SCOPE = "SCOPE_write";

    private static final String SECURED_PATTERN_WRITE = "/orders/**";

    private static final String SECURED_PATTERN_READ = "/orders/{id}";

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        // @formatter:off
        http.authorizeRequests()
                .antMatchers(SECURED_PATTERN_WRITE).hasAuthority(SECURED_WRITE_SCOPE)
                .antMatchers(SECURED_PATTERN_READ).hasAuthority(SECURED_READ_SCOPE)
                .anyRequest().authenticated()
                .and()
                .oauth2ResourceServer().jwt();
        // @formatter:on
    }

}
```

如您所见，该代码指示微服务运行时 (Spring Boot) 检查特定 HTTP 方法和请求路径的相关 scope。这行代码

```
.antMatchers(SECURED_PATTERN_WRITE).hasAuthority(SECURED_WRITE_SCOPE)
```

检查针对与正则表达式 `/orders/**` 匹配的请求路径发出的任何请求的 scope write。类似地，这行代码检查在路径 `/orders/{id}` 上为请求 read 的 scope：

```
.antMatchers(SECURED_PATTERN_READ).hasAuthority(SECURED_READ_SCOPE)
```

现在尝试使用只有 `read` scope 的令牌访问 POST /orders 资源。执行您上次使用的相同 curl 命令来访问此资源，但这次使用不同的令牌（仅具有读取访问权限的令牌）：

```
curl -v http://localhost:8080/orders \
-H 'Authorization: Bearer
eyJraWQiOiIyNzlkNzM5Zi1mNjIzLTRjMDEtOTFmYi00NTVmZThjMzFiZTIiLCJhbGciOiJSUzI1NiJ9.eyJzd
WIiOiJvcmRlcnByb2Nlc3NpbmdzZXJ2aWNlIiwiYXVkIjoib3JkZXJwcm9jZXNzaW5nc2VydmljZSIsIm5iZiI
6MTY1MTEzMDg2NCwic2NvcGUiOlsicmVhZCIsIm9wZW5pZCJdLCJpc3MiOiJodHRwOlwvXC9sb2NhbGhvc3Q6O
DA4NSIsImV4cCI6MTY1MTEzMTE2NCwiaWF0IjoxNjUxMTMwODY0fQ.EjUNRVaXQHiqWNFbi0Vbi13HMVo9zjVN
cA8TlxpVCRwS9it43RiKxrctZeYGozuFqQGTDq9K3ikQGMeZAnj0iPM2G7n8xkBEj-
JPc7ORY8ywkMKta52I_osRK3-_CZVnukaNniv6oD5UKHHQ5iLI0fO-k5sa-
Qj8Mc1vsOBQTgj9fhv86r3BXsL8nXtdKwD9HBNDQJgbIPzZhoF4vmOf4TnolWz3b-
fqnEUqLaj7WHJUkxZW6ZDrsrxIe3T8cNhr76Xf8PpoWu_amw5FJ9acgsuRIAoWqTgabVPonQyUQFuKxBWOA-
KBOcYsfADBVAQ5zu-LzaLJFBSqknylPaqBmw' \
-H 'Content-Type: application/json' \
```

```
--data-binary @- << EOF
{
  "items":[
    {
      "itemCode":"IT0001",
      "quantity":3
    },
    {
      "itemCode":"IT0004",
      "quantity":1
    }
  ],
  "shippingAddress":"福建省厦门市XX区XX街道XX小区XX栋XX室"
}
EOF
```

执行此命令时，应该会看到来自资源服务器的此错误响应：

```
< HTTP/1.1 403
< WWW-Authenticate: Bearer error="insufficient_scope", error_description="The request
requires higher privileges than provided by the access token.", error_uri="
```

此响应表示此特定操作的令牌 scope 不足，所需 scope 为 `write`。

总结

- OAuth 2 是一种授权框架，广泛用于保护边缘微服务部署。
- OAuth 2 支持多种授权类型。我们在本章中使用的客户端凭据授权类型主要用于系统到系统的身份验证。
- 授权服务器发出的每个访问令牌都与一个或多个 scope 耦合。OAuth 2 中使用 scope 来表达附加到访问令牌的权限。
- OAuth 2 scope 用于保护和强制执行微服务中某些操作中的访问控制检查。
- 当前所有示例都使用 HTTP（而不是 HTTPS）端点来使您不必设置正确的证书，并使您可以在需要时检查线路（网络）上传递的消息。在生产系统中，我们不建议对任何端点使用 HTTP。