如何在微服务架构中实现安全性

应用程序开发人员主要负责实现安全性的四个不同方面：

1. **身份验证：**验证尝试访问应用程序的应用程序或人员（安全的术语叫主体）的身份。例如，应用程序通常会验证访问的凭据，例如用户的 ID 和密码，或应用程序的 API 密钥。
2. **访问授权：**验证是否允许访问主体对指定数据完成请求的操作。应用程序通常使用基于角色的安全性和访问控制列表（ACL）的组合。基于角色的安全性为每个用户分配一个或多个角色，授予他们调用特定操作的权限。ACL 授予用户或角色对特定业务对象或聚合执行操作的权限。
3. **审计：**跟踪用户在应用中执行的所有操作，以便检测安全问题，帮助客户实现并强制执行合规性。
4. **安全的进程间通信：**理想情况下，所有进出服务的通信都应该采用传输层安全性（TLS）加密。服务间通信甚至可能需要使用身份验证。

下面将重点介绍如何实现身份验证和访问授权。审计和安全的进程间通信的更多详细介绍请参阅 Chris Richardson 的《微服务架构设计模式》。

我首先描述如何在 FTGO 单体应用程序中实现安全性。然后介绍在微服务架构中实现安全性所面临的挑战，以及为何在单体架构中运行良好的技术不能在微服务架构中使用。之后，我将介绍如何在微服务架构中实现安全性。

让我们首先回顾一下 FTGO 单体应用程序如何处理安全性。

传统单体应用程序的安全性

FTGO 应用程序有多种用户，包括消费者、送餐员和餐馆员工。他们使用基于浏览器的 Web 应用程序和移动应用程序访问 FTGO。所有 FTGO 用户都必须登录才能访问该应用程序。图 1 显示了单体 FTGO 应用程序的客户端如何验证和发出请求。

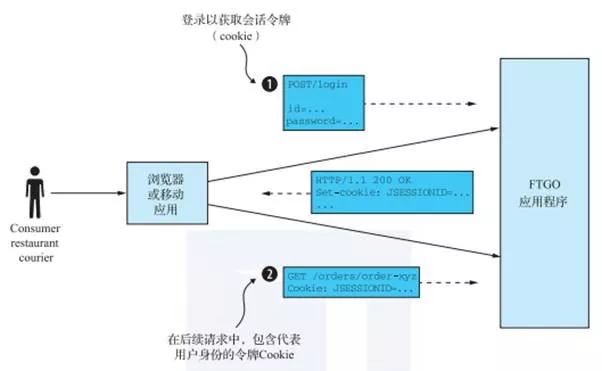


图 1 FTGO 应用程序的客户首先登录以获取会话令牌，该令牌通常是 cookie。客户在向 FTGO 应用程序发出的每个后续请求中都会包括会话令牌

当用户使用其用户 ID 和密码登录时，客户端会向 FTGO 应用程序发出包含用户凭据的 POST 请求。FTGO 应用程序验证凭据并将会话令牌返回给客户端。客户端在 FTGO 应用程序的每个后续请求中包含会话令牌。

图 2 显示了 FTGO 应用程序如何实现安全性。FTGO 应用程序是用 Java 编写的，并使用 Spring Security 框架，但我将使用同样也适用于其他框架（例如 Passport for Node.js）的一般性术语来描述这个设计。

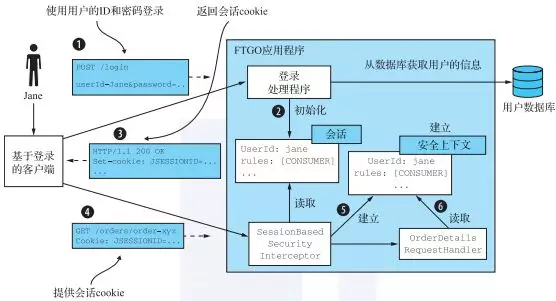


图 2　当 FTGO 应用程序的客户端发出登录请求时，登录处理程序会对用户进行身份验证，初始化会话用户信息，并返回会话令牌 cookie，以便安全地识别会话。接下来，当客户端发出包含会话令牌的请求时，SessionBasedSecurityInterceptor 从指定的会话中检索用户信息并建立安全上下文。请求处理程序（如 OrderDetailsRequestHandler）从安全上下文中检索用户信息

**使用安全框架**

正确实现身份验证和访问授权具有挑战性。最好使用经过验证的安全框架。使用哪个框架取决于你的应用程序的技术栈。流行的框架包括以下几个：

1. SpringSecurity：适用于 Java 应用程序的流行框架。它是一个复杂的框架，可以处理身份验证和访问授权。
2. ApacheShiro：另一个 Java 安全框架。
3. Passport：在 Node.js 应用程序流行的一个专注于身份验证的安全框架。

安全架构的一个关键部分是会话，它存储主体的 ID 和角色。FTGO 应用程序是传统的 Java EE 应用程序，因此会话是 HttpSession 内存中会话。会话令牌代表着每一个具体的会话，客户端在每个请求中包含会话令牌。它通常是一串无法读懂的数字标记，例如经过加密的强随机数。FTGO 应用程序的会话令牌是一个名为 JSESSIONID 的 HTTP cookie。

实现安全性的另一个关键是安全上下文，它存储有关发出当前请求的用户的信息。Spring Security 框架使用标准的 Java EE 方法将安全上下文存储在静态的线程局部变量中，任何被调用以处理请求的代码都可以访问该变量。请求处理程序可以调用 SecurityContextHolder. getContext().getAuthentication() 获取有关当前用户的信息，例如他们的身份和角色。相反，Passport 框架将安全上下文存储为 request 对象的 user 属性。

**图 2 中显示的事件序列如下：**

1. 客户端向 FTGO 应用程序发出登录请求。
2. 登录请求由 LoginHandler 处理，LoginHandler 验证凭据，创建会话，并在会话中存储有关主体的信息。
3. Login Handler 将会话令牌返回给客户端。
4. 客户端在后续每次调用请求中都包含会话令牌。
5. 这些请求首先由 SessionBasedSecurityInterceptor 处理。拦截器通过验证会话令牌来验证每个请求并建立安全上下文。安全上下文描述了主体及其角色。
6. 请求处理程序使用安全上下文来获取其身份，并借此确定是否允许用户执行请求的操作。

FTGO 应用程序使用基于角色的授权。它定义了与不同类型用户相对应的几个角色，包括 CONSUMER、RESTAURANT、COURIER 和 ADMIN。它使用 Spring Security 的声明性安全机制来限制对特定角色的 URL 和服务方法的访问。角色也与业务逻辑交织在一起。例如，消费者只能访问自己的订单，而管理员可以访问所有订单。

单体 FTGO 应用程序使用的安全设计只是实现安全性的一种可能方式。例如，使用内存中会话的一个缺点是，它必须把特定会话的所有请求路由到同一个应用程序实例。这个要求使负载均衡和操作变复杂了。例如，你必须实现会话耗尽机制，该机制在关闭应用程序实例之前等待所有会话到期（以免丢失内存中已有的会话）。避免这些问题的另一种方法是将会话存储在数据库中。

开发者可以完全不保存服务器端会话。例如，许多应用程序都有 API 客户端，可以在每个请求中提供其凭据，例如 API 密钥和私钥。因此，无须维护服务器端会话。或者，应用程序可以将会话状态存储在会话令牌中。在本文的后面，我将介绍一种使用会话令牌存储会话状态的方法。但让我们首先看一下在微服务架构中实现安全性的挑战。

**在微服务架构中实现安全性**

微服务架构是分布式架构。每个外部请求都由 API Gateway 和至少一个服务处理。例如，考虑 getOrderDetails() 查询。API Gateway 通过调用多个服务来处理此查询，包括 Order Service、Kitchen Service 和 Accounting Service。每项服务都必须实现安全性的某些方面。例如，Order Service 必须只允许消费者查看他们自己的订单，这需要结合身份验证和访问授权。为了在微服务架构中实现安全性，我们需要确定谁负责验证用户身份以及谁负责访问授权。

在微服务应用程序中实现安全性的一个挑战是我们不能仅仅从单体应用程序借鉴设计思路。这是因为单体应用程序的安全架构的一些方面对微服务架构来说是不可用的，例如：

1. **内存中的安全上下文：**使用内存中的安全上下文（如 ThreadLocal）来传递用户身份。服务无法共享内存，因此它们无法使用内存中的安全上下文（如 ThreadLocal）来传递用户身份。在微服务架构中，我们需要一种不同的机制来将用户身份从一个服务传递到另一个服务。
2. **集中会话：**因为内存中的安全上下文没有意义，内存会话也没有意义。从理论上讲，多种服务可以访问基于数据库的会话，但它会违反松耦合的原则。我们需要在微服务架构中使用不同的会话机制。

让我们通过研究如何处理身份验证来开始探索微服务架构中的安全性。

**由 API Gateway 处理身份验证**

处理身份验证有两种不同的方法。一种选择是让各个服务分别对用户进行身份验证。这种方法的问题在于它允许未经身份验证的请求进入内部网络。它依赖于每个开发团队在所有服务中正确实现安全性。因此，出现安全漏洞的风险和概率都很大。

在服务中实现身份验证的另一个问题是不同的客户端以不同的方式进行身份验证。纯 API 客户端使用基本身份验证为每个请求提供凭据。其他客户端可能首先登录，然后为每个请求提供会话令牌。但我们要避免在服务中处理多种不同的身份验证机制。

更好的方法是让 API Gateway 在将请求转发给服务之前对其进行身份验证。在 API Gateway 中进行集中 API 身份验证的优势在于只需要确保这里的验证是正确的。因此，出现安全漏洞的可能性要小得多。另一个好处是只有 API Gateway 需要处理各种不同的身份验证机制。这使得其他服务的实现变得简单了。

图 3 显示了这种方法的工作原理。客户端使用 API Gateway 进行身份验证。API 客户端在每个请求中包含凭据。基于登录的客户端将用户的凭据发送到 API Gateway 进行身份验证，并接收会话令牌。一旦 API Gateway 验证了请求，它就会调用一个或多个服务。

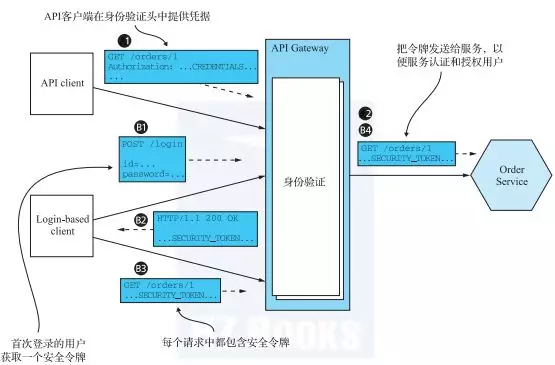


图 3 API Gateway 对来自客户端的请求进行身份验证，并在其对服务的请求中包含安全令牌。服务使用令牌获取有关主体的信息。API Gateway 还可以将安全令牌用作会话令牌

**模式：访问令牌**

API Gateway 将包含用户信息（例如其身份和角色）的令牌传递给它调用的服务。请参阅：

http://microservices.io/patterns/security/access-token.html

API Gateway 调用的服务需要知道发出请求的主体（用户的身份）。它还必须验证请求是否已经过通过身份验证。解决方案是让 API Gateway 在每个服务请求中包含一个令牌。服务使用令牌验证请求，并获取有关主体的信息。API Gateway 还可以为面向会话的客户端提供相同的令牌，以用作会话令牌。

**客户端的事件序列如下：**

1. 客户端发出包含凭据的请求给 API Gateway。
2. API Gateway 对凭据进行身份验证，创建安全令牌，并将其传递给服务。

**基于登录的客户端的事件序列如下：**

1. 客户端发出包含凭据的登录请求。
2. API Gateway 返回安全令牌。
3. 客户端在调用操作的请求中包含安全令牌。
4. API Gateway 验证安全令牌并将其转发给服务。

让我们首先看一下安全性的另一个主要方面：访问授权。

**处理访问授权**

验证客户端的凭据很重要，但这还不够。应用程序还必须实现访问授权机制，以验证是否允许客户端执行所请求的操作。例如，在 FTGO 应用程序中，getOrderDetails() 查询只能由下此 Order 的消费者（基于实例的安全性的一个示例）和为所有消费者提供服务的客户服务代表调用。

实现访问授权的一个位置是 API Gateway。例如，它可以将对 GET/orders/{orderId}的访问限制为消费者和客户服务代表。如果不允许用户访问特定路径，则 API Gateway 可以在将请求转发到服务之前拒绝该请求。与身份验证一样，在 API Gateway 中集中实现访问授权可降低安全漏洞的风险。你可以使用安全框架（如 Spring Security）在 API Gateway 中实现访问授权。

在 API Gateway 中实现访问授权的一个弊端是，它有可能产生 API Gateway 与服务之间的耦合，要求它们以同步的方式进行代码更新。而且，API Gateway 通常只能实现对 URL 路径的基于角色的访问。由 API Gateway 实现对单个领域对象的访问授权通常是不实际的，因为这需要详细了解服务的领域逻辑。

另一个实现访问授权的位置是服务。服务可以对 URL 和服务方法实现基于角色的访问授权。它还可以实现 ACL 来管理对聚合的访问。例如，在 Order Service 中可以实现基于角色和基于 ACL 的授权机制，以控制对 Order 的访问。FTGO 应用程序中的其他服务也可以实现类似的访问授权逻辑。

**使用 JWT 传递用户身份和角色**

在微服务架构中实现安全性时，你需要确定 API Gateway 应使用哪种类型的令牌来将用户信息传递给服务。有两种类型的令牌可供选择。一种选择是使用不透明（无可读性）的令牌，它们通常是一串 UUID。不透明令牌的缺点是它们会降低性能和可用性，并增加延迟。因为这种令牌的接收方必须对安全服务发起同步 RPC 调用，以验证令牌并检索用户信息。

另一种消除对安全服务调用的方法是使用包含有关用户信息的透明令牌。透明令牌的一个流行的标准是 JSON Web 令牌（JWT）。JWT 是在访问双方之间安全地传递信息（例如用户身份和角色）的标准方式。JWT 的内容包含一个 JSON 对象，其中有用户的信息，例如其身份和角色，以及其他元数据，如到期日期等。它使用仅为 JWT 的创建者所知的数字签名，例如 API Gateway 和 JWT 的接收者（服务）。该签名确保恶意第三方不能伪造或篡改 JWT。

因为不需要再访问安全服务进行验证，JWT 的一个问题是这个令牌是自包含的，也就是说它是不可撤消的。根据设计，服务将在验证 JWT 的签名和到期日期之后执行请求操作。因此，没有切实可行的方法来撤消落入恶意第三方手中的某个 JWT 令牌。解决方案是发布具有较短到期时间的 JWT，这可以限制恶意方。但是，短期 JWT 的一个缺点是应用程序必须以某种方式不断重新发布 JWT 以保持会话活动。幸运的是，这是 OAuth 2.0 安全标准旨在解决的众多问题之一。让我们来看看它是如何工作的。

**在微服务架构中使用 OAuth 2.0**

假设你要为 FTGO 应用程序实现一个 User Service，该应用程序管理包含用户信息（如凭据和角色）的数据库。API Gateway 调用 User Service 来验证客户端请求并获取 JWT。你可以设计 User Service 的 API 并使用你喜欢的 Web 框架实现它。但这不是 FTGO 应用程序特有的通用功能，自己开发此类服务往往是得不偿失的。

幸运的是，你不需要开发这种安全基础设施。你可以使用名为 OAuth 2.0 的标准的现成服务或框架。OAuth 2.0 是一种访问授权协议，最初旨在使公共云服务（如 GitHub 或 Google）的用户能够授予第三方应用程序访问其信息的权限，而不必向第三方应用透露他们的密码。例如，OAuth 2.0 使你能够安全地授予第三方基于云的持续集成（CI）服务，访问你的 GitHub 存储库。

虽然 OAuth 2.0 最初的重点是授权访问公共云服务，但你也可以将其用于应用程序中的身份验证和访问授权。让我们快速了解一下微服务架构如何使用 OAuth 2.0。

**OAuth 2.0 中的关键概念如下：**

1. 授权服务器：提供用于验证用户身份以及获取访问令牌和刷新令牌的 API。Spring OAuth 是一个很好的用来构建 OAuth 2.0 授权服务器的框架。
2. 访问令牌：授予对资源服务器的访问权限的令牌。访问令牌的格式取决于具体的实现技术。Spring OAuth 的实现中采用了 JWT 格式的访问令牌。
3. 刷新令牌：客户端用于获取新的 AccessToken 的长效但同时也可被可撤消的令牌。
4. 资源服务器：使用访问令牌授权访问的服务。在微服务架构中，服务是资源服务器。
5. 客户端：想要访问资源服务器的客户端。在微服务架构中，API Gateway 是 OAuth 2.0 客户端。

首先，我们来谈谈如何验证 API 客户端，然后介绍如何支持基于登录的客户端。

图 4 显示了 API Gateway 如何验证来自 API 客户端的请求。API Gateway 通过向 OAuth 2.0 授权服务器发出请求来验证 API 客户端，该服务器返回访问令牌。然后，API Gateway 将包含访问令牌的一个或多个请求发送到服务。

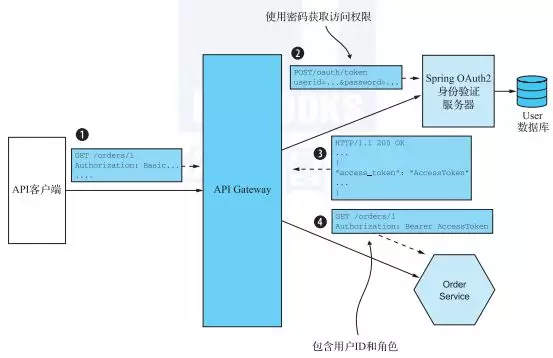


图 4 API Gateway 通过向 OAuth 2.0 身份验证服务器发出请求来验证 API 客户端。身份验证服务器返回访问令牌，API Gateway 将其传递给服务。服务验证令牌的签名，并提取有关用户的信息，包括其身份和角色

**图 4 所示的事件顺序如下：**

1. 客户端发出请求，使用基本身份验证提供它的凭据。
2. API Gateway 向 OAuth 2.0 身份验证服务器发出 OAuth 2.0 密码授予（Password Grant）请求（www.oauth.com/oauth2-servers/access-tokens/password-grant/）。
3. 身份验证服务器验证 API 客户端的凭据，并返回访问令牌和刷新令牌。
4. API Gateway 在其对服务的请求中包含访问令牌。服务验证访问令牌并使用它来授权请求。

基于 OAuth 2.0 的 API Gateway 可以使用 OAuth 2.0 访问令牌作为会话令牌来验证面向会话的客户端。而且，当访问令牌到期时，它可以使用刷新令牌获得新的访问令牌。图 5 显示了 API Gateway 如何使用 OAuth 2.0 来处理面向会话的客户端。API 客户端通过将其凭据（发送 POST）到 API Gateway 的 /login 端点来启动会话。API Gateway 向客户端返回访问令牌和刷新令牌。然后，API 客户端在向 API Gateway 发出请求时提供这两个令牌。

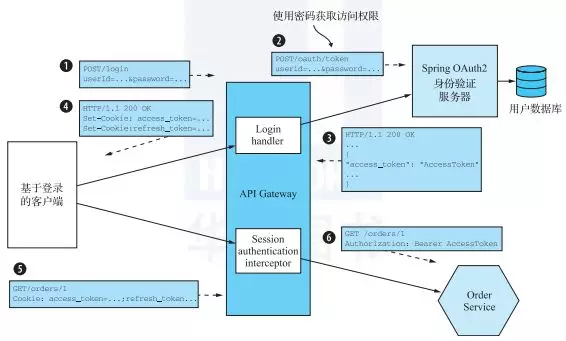


图 5　客户端通过将其凭据发送到 API Gateway 来登录。API Gateway 使用 OAuth 2.0 身份验证服务器对凭据进行身份验证，并将访问令牌和刷新令牌作为 cookie 返回。客户端在其对 API Gateway 的请求中包括这些令牌

**事件顺序如下：**

1. 基于登录的客户端将其凭据发送到 API Gateway。
2. API Gateway 的 LoginHandler 向 OAuth 2.0 身份验证服务器发出密码授予请求（www. oauth.com/oauth2-servers/access-tokens/password-grant/）。
3. 身份验证服务器验证客户端的凭据，并返回访问令牌和刷新令牌。
4. API Gateway 将访问令牌和刷新令牌返回给客户端，通常是采用 cookie 的形式。
5. 客户端在向 API Gateway 发出的请求中包含访问令牌和刷新令牌。
6. API Gateway 的 Session Authentication Interceptor 验证访问令牌，并将其包含在对服务的请求中。

如果访问令牌已经过期或即将过期，API Gateway 将通过发出 OAuth 2.0 刷新授权请求来获取新的访问令牌（www.oauth.com/oauth2-servers/access-tokens/refresh-access-tokens/），刷新授权请求发送给授权服务器，请求中包含刷新令牌。如果刷新令牌尚未过期或未被撤消，则授权服务器将返回新的访问令牌。API Gateway 将新的访问令牌传递给服务并将其返回给客户端。

使用 OAuth 2.0 的一个重要好处是它是经过验证的安全标准。使用现成的 OAuth 2.0 身份验证服务器意味着你不必浪费时间重新发明轮子或者是没有开发不安全的设计的风险。但 OAuth 2.0 不是在微服务架构中实现安全性的唯一方法。无论你使用哪种方法，三个关键思想如下：

1. API Gateway 负责验证客户端的身份。
2. API Gateway 和服务使用透明令牌（如 JWT）来传递有关主体的信息。
3. 服务使用令牌获取主体的身份和角色。