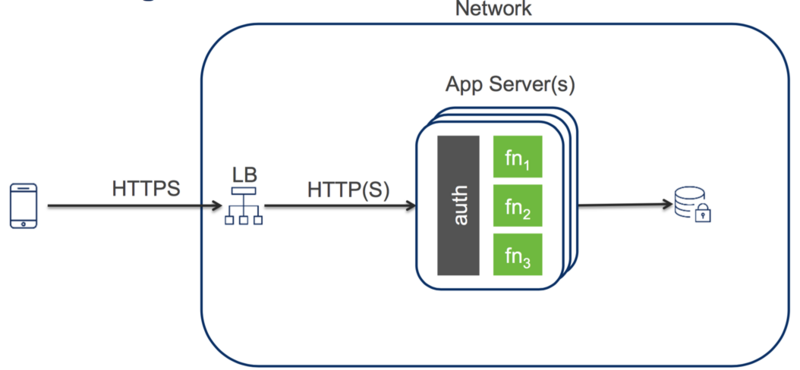
首先从传统单体应用架构下的访问安全设计说起，然后分析现代微服务架构下，访问安全涉及的原则，接着讨论目前常用的几种微服务架构下的访问安全设计方案。最后，详析Spring Cloud微服务架构下如何解决访问安全的问题。

一、传统单体应用的访问安全设计

  
上面的示意图展示了单体应用的访问逻辑。用户通过客户端发出http或者https请求，经过负载均衡后，单体应用收到请求。接着经过auth层，进行身份验证和权限批准，这里，一般会有跟后端数据库的交互。通过后，将请求分发到对应的功能逻辑层中去。完成相关操作后，返回结果给客户端。

传统单体应用的访问安全设计——原则



从以上分析可以看到，传统单体应用的访问安全设计原则为：

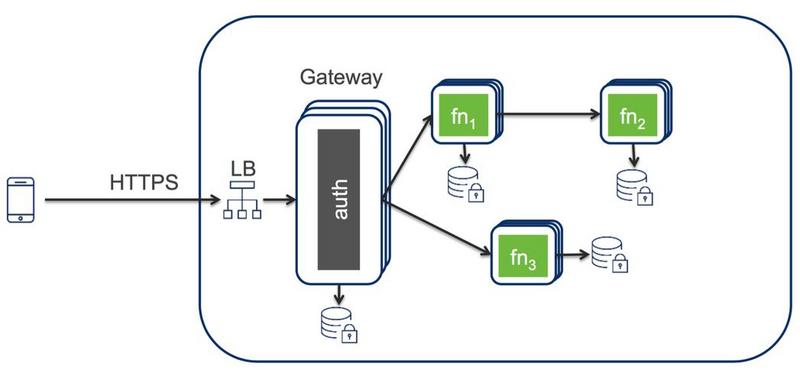
第一，每次的用户请求都需要验证是否安全，这里可以分两种情况：

一种是没有session的请求，需要经过几个步骤完成session化。一般为验证当前用户的credential，获取当前用户的identity，这两步都需要访问数据库等持久化对象来完成，最后一步是为当前可用创建session，返回给客户端后，启用该session。

另一种是有session的请求，只需验证请求中当前session的有效性，即可继续请求。

第二，用户的操作请求都在后端单个进程中执行完成，完全依赖后端调用方法的可靠性。一旦出错，应用是无法再次重复请求。

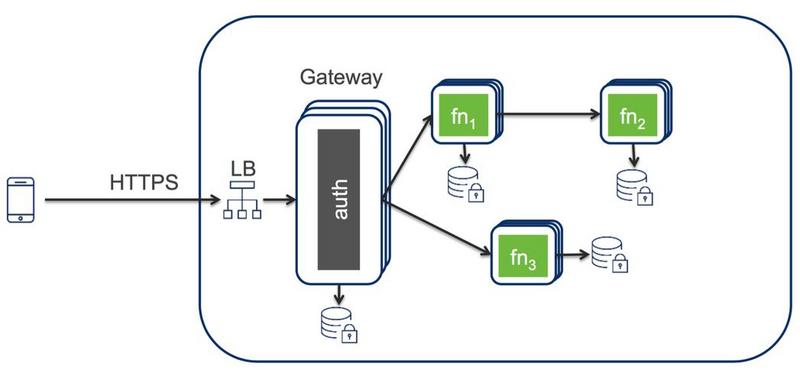
传统单体应用的访问安全设计——优势和注意点

  
小结，传统单体应用由于设计相对简单单一，暴露给外界的入口相对较少，从而具有被攻击并造成危害性的可能小的优势。

也正是由于单体应用简单单一的特点，需要注意相关问题：

* 应用后端保存了所有的credential等敏感信息
* 一旦入侵了对这个应用的请求，就有可能拿到所有的保存在后端的信息
* 应用的每次操作一般都需要和数据库进行交互，造成数据库负载变高

二、微服务架构下，访问安全设计原则

  
先来看下这张典型的微服务设计架构图，如图所示，有以下几点特征：

* 每个服务只有权限去操作自己负责的那部分功能。
* 用户请求的身份验证和权限批准都由独立的gateway服务来保障
* 对外服务的LB层无法直接与提供业务服务的应用层进行访问  
  

从上面的特征分析来看，想要给出一份访问安全设计的原则说明，就要看看微服务架构下，访问安全有哪些痛点，以下罗列了几点：

* 单点登录，即在微服务这种多独立服务的架构下，实现用户只需要登录一次就能访问所有相互信任的应用系统
* 微服务架构下的应用一般都是无状态的，导致用户的请求每次都需要鉴权，可能引发Auth服务的性能瓶颈
* 微服务架构下，每个组件都管理着各自的功能权限，这种细粒度的鉴权机制需要事先良好的规划
* 微服务架构下，需要考虑到那些非浏览器端的客户请求，是否具有良好的可操作性

根据实际情况，还有一些其他痛点，这里不再一一赘述，而这些痛点，就形成了我们在为微服务架构设计访问安全的原则。

三、微服务架构下，常用的访问安全设计方案

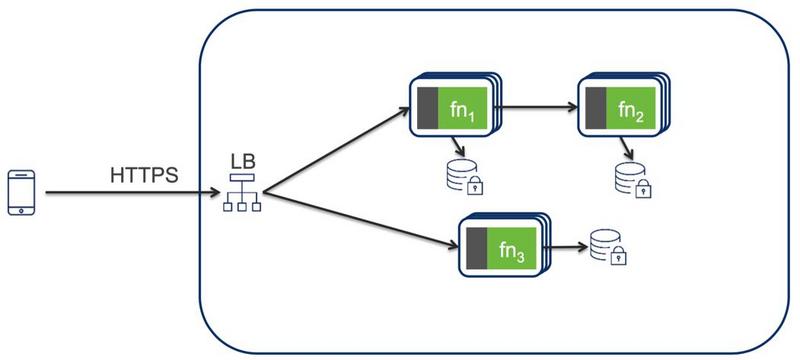
* HTTP Basic Authentication + Independent Auth DB
* HTTP Basic Authentication + Central Auth DB
* API Tokens
* SAML

这里列出4种，首先简单介绍下，然后一一叙述。

第一种，使用HTTP Basic Auth协议，加上独立的Auth数据库。  
第二种，也是使用HTTP Basic Auth协议，跟第一种不同的是，使用集中式的Auth数据库  
第三种，API Tokens协议，这种大家应该比较熟悉，很多公有服务（比如Github、Twitter等）的API都是用这种方式。  
第四种，SAML，即Security Assertion Markup Language，翻译过来，是『安全声明标记语言』，它是基于XML的一种协议，企业内使用得较多。

下面一一做介绍。

微服务常用访问安全设计方案——Basic Auth + Independent Auth DB

  
第一种，如上示意图所示，使用Basic Auth协议，配合每个服务自己都拥有存储Credential敏感数据的数据库（或者其他持久化仓库）。

简单介绍下Basic Auth协议，它是在用户的请求中添加一个Authorization消息头，这个消息头的值是一个固定格式：  
Basic base64encode(username+“:”+password)

完整的消息头列子为：  
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==

Basic Auth协议基本上被所有流行的网页浏览器都支持。

这种方案的特点：

* 每个提供功能的服务都拥有自己独立的鉴权和授权机制
* 每个提供功能的服务都拥有自己独立的数据库，来保存敏感信息
* 每次用户请求都需要携带用户的credential来完成操作

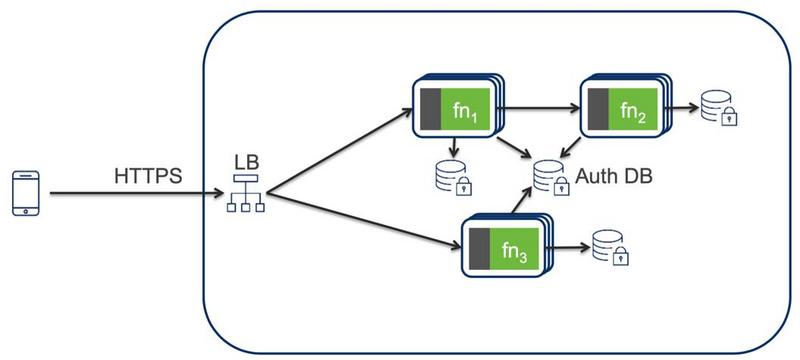
小结下使用这种方案的好处：

* 微服务的应用可以实现100%无状态化
* 基于Basic Auth开发简单

同时，小结下使用这种方案需要注意的地方：

由于每个服务都有自己存储credential的机制，需要事先为每个服务设计好如何存储和查找用户的Credential  
由于每次用户请求都会携带用户的Credential，需要事先设计好如何管理鉴权机制

微服务常用访问安全设计方案——Basic Auth + Central Auth DB

  
第二种，如上示意图所示，使用Basic Auth协议，与第一种方案相比，每个服务共用有同一个Auth DB。

第二种方案的特点和第一种很相似：

* 每个提供功能的服务都拥有自己独立的鉴权和授权机制
* 每个提供功能的服务共用同一个DB，来保存Credential等敏感信息
* 每次用户请求都需要携带用户的credential来完成操作

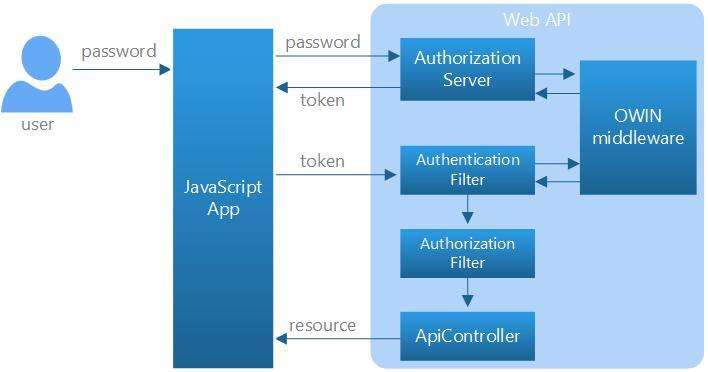
小结下使用第二种方案的好处：

除了拥有第一种方案相似的好处外，由于共用了同一个持久化仓库来管理用户信息，简化了原来独立管理的机制

同时，小结下使用这种方案需要注意的地方：

* 中心化Auth DB会被每次用户请求来访问连接，可能引发AuthDB性能瓶颈
* 需要在每个服务中实现对共有Auth DB查找用户信息的逻辑

微服务常用访问安全设计方案——API Tokens



第三种，如上示意图所示，使用Token Based协议来对用户请求进行操作鉴权。

简单介绍下最基本的Token Based的交互方式：

* 用户使用包含用户名和密码的credential从客户端发起资源请求
* 后端接受请求，通过授权中心，生产有效token字符串，返回给客户端
* 客户端获得token后，再次发出资源请求
* 后端接受带token的请求，通过授权中心，获取相关资源，返回给客户端

业界常用的OAuth就是基于Token Based这套逻辑，实现的互联网级的鉴权机制。

第三种方案的特点明显：

使用token来进行鉴权，替换用户本身的用户名和密码，提高了交互安全性  
每次用户请求需要携带有效token，与Auth服务进行交互验证

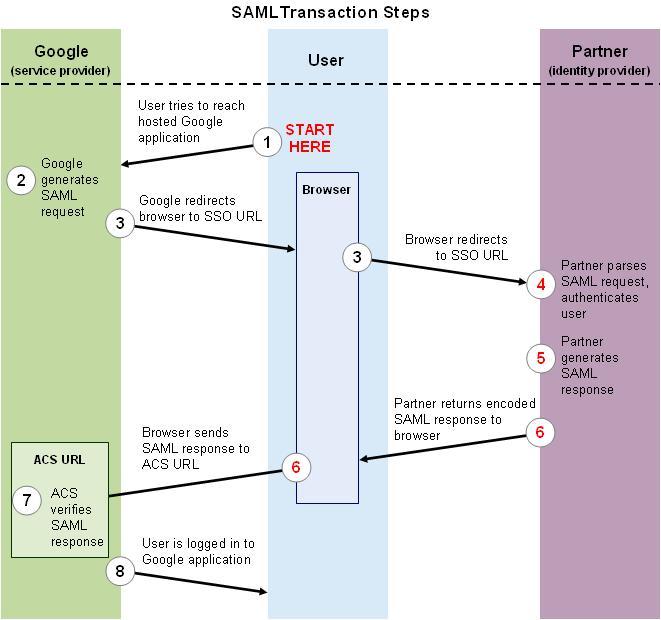
小结下使用第三种方案的好处：

由于使用了token来鉴权，业务服务不会看到用户的敏感信息

同时，小结下使用这种方案需要注意的地方：

Auth服务可能需要处理大量的生产token的操作

微服务常用访问安全设计方案——SAML

  
第四种，如上示意图所示，使用SAML协议来对用户请求进行操作鉴权。它是一个基于XML的标准，用于在不同的安全域(security domain)之间交换认证和授权数据。在SAML标准定义了身份提供者(identity provider)和服务提供者(service provider)，这两者构成了前面所说的不同的安全域。

以上图Google提供的Apps SSO的机制，简单介绍下SAML鉴权的交互方式：

* 用户请求访问自建的google application
* 当前application 生成一个 SAML 身份验证请求。SAML 请求将进行编码并嵌入到SSO 服务的网址中。
* 当前application将重定向发送到用户的浏览器。重定向网址包含应向SSO 服务提交的编码 SAML 身份验证请求。
* SSO（统一认证中心或叫Identity Provider）解码 SAML 请求，并提取当前application的 ACS（声明客户服务）网址以及用户的目标网址（RelayState 参数）。然后，统一认证中心对用户进行身份验证。
* 统一认证中心生成一个 SAML 响应，其中包含经过验证的用户的用户名。按照 SAML 2.0 规范，此响应将使用统一认证中心的 DSA/RSA 公钥和私钥进行数字签名。
* 统一认证中心对 SAML 响应和 RelayState 参数进行编码，并将该信息返回到用户的浏览器。统一认证中心提供了一种机制，以便浏览器可以将该信息转发到当前application ACS。
* 当前application使用统一认证中心的公钥验证 SAML 响应。如果成功验证该响应，ACS 则会将用户重定向到目标网址。
* 用户将重定向到目标网址并登录到当前application。

目前SAML在业界也有相当的使用度，包括IBM Weblogic等产品。

第四种方案的特点有：

由Identity Provider提供可信的签名声明  
服务的访问安全由可信的Identity Provider提供

小结下使用第四种方案的好处：标准的可信访问模型

同时，小结下使用这种方案需要注意的地方：

基于XML协议，传输相对复杂  
对非浏览器客户端适配不方便

四、Spring Cloud Security解决方案

Spring Cloud Security特点有：

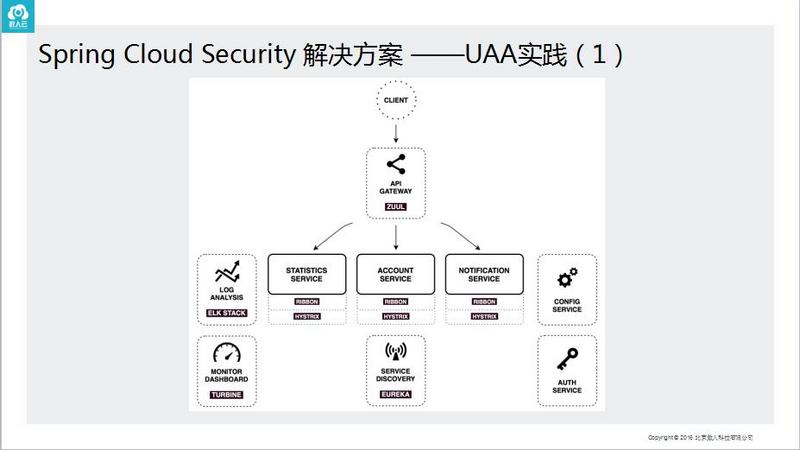
* 基于OAuth2 和OpenID协议的可配置的SSO登录机制
* 基于tokens保障资源访问安全
* 引入UAA鉴权服务，UAA是一个Web服务，用于管理账户、Oauth2客户端和用户用于鉴权的问题令牌(Issue Token)。UAA实现了Oauth2授权框架和基于JWT(JSON web tokens)的问题令牌。   
  

下面简单介绍下UAA，事实上，它是由CloudFoundry发起的，也是CloudFoundry平台的身份管理服务（[https://docs.cloudfoundry.org...](https://docs.cloudfoundry.org/concepts/architecture/uaa.html)）。

主要功能是基于OAuth2，当用户访问客户端应用时，生成并发放token给目标客户端。

UAA认证服务包含如下几个方面的内容：

* 认证对象。如用户、客户端以及目标资源服务器
* 认证类型。主要有授权码模式、密码模式以及客户端模式
* 认证范围，即认证权限，并作为一个命名的参数附加到AccessToken上。

接下来，结合实例，一起来看下UAA在Spring Cloud中的实践。  
  
如图所示，这是一个简单的基于Spring Cloud微服务架构的例子，它的主要组件有：

* Eureka组件提供服务发现功能
* 独立的Config组件提供类似配置中心的服务，持久化层可以是文件系统，也可是git repository
* Auth组件提供基于UAA的鉴权服务
* Account组件保存用户的业务信息  
  其他组件不一一介绍了

这里主要讲Auth组件和Account组件是如何基于UAA服务进行认证和授权。  
  
图一为Auth组件业务代码中定义了不同客户端的认证类型和认证范围，其中：

浏览器端的认证类型是password，认证范围是ui  
account组件端的认证类型是client\_credentials，认证范围是server

图二为config组件（配置中心）定义的请求路由的规则，其中：

使用/uaa/\*\*来转发基于uaa的认证请求至auth组件  
使用/accouts/\*\*来转发请求至account组件，并标记serviceId为account-service，与图一中的withClient对应。  
  
图一为浏览器打开应用入口后，输入用户名和密码后，发出的认证请求：

认证url为/uaa/oauth/token，这是uaa模式下标准的请求获取token的url  
表单中包含了字段scope（认证范围）和字段password（认证类型）

图二为图一发出认证请求的返回结果：

Access\_token为有效认证token，将来被其他请求使用

图三为发出获取当前用户的信息的请求：

在请求里的Authorization的值为图二中获得的access\_token  
  
图一为Account组件在Config组件（配置中心）定义的OAuth2协议下获取token的方式，这里定义了：

clientID和clientSecret  
accessTokenUrl，这里指定了auth组件的uaa获取token的url  
grant-type，即认证类型，这里指定为client\_credentials  
scope，这里指定了server，说明是这个认证请求只适用在各微服务之间的访问。

图二为Accout组件业务代码中定义了需要使用Auth组件进行事先鉴权的方法：

使用@PreAuthorize  
annotation中可以指定认证范围的具体条件，这里是限定了server或者是demo账户，才有权限发起认证。  
  
最后小结下Spring Cloud Security的特点：

* 基于UAA，使用OAuth2协议。不会暴露用户的敏感信息
* 基于认证类型和认证范围，实现细粒度的鉴权机制
* 非浏览器客户端下良好的操作性

Q&A

问题：Basic Auth + Central Auth DB这种方式中，每个服务有自己的鉴权DB，这块只是一个缓冲吗？如果中途通过别的方式修改了中心DB的数据，而缓冲又没过期，这个时候有什么解决方案吗？  
答：不是缓冲，这里的Central Auth DB是指各个微服务共用一个数据库。

问题：微服务架构需要服务路由和服务注册么？跟esb的区别在哪里？  
答：服务路由组件和服务注册组件和是相对必要的，他们保证了用户请求能发到正确的微服务中去。ESB企业服务总线是相对比较重的组件，而不是像微服务组件一样只负责单个业务。

问题:在微服务中，对于数据权限的粒度，是可以集中在在gateway中进行还是由每个微服务自己独立配置？  
答：推荐由那个专门负载权限的微服务组件来配置。

问题：您好，辛苦了，请问现在有类似SAML协议，但是不基于XML，而是基于JSON或者其他简化格式的协议吗？  
答：目前据我所知没有基于JSON的SAML协议。有个叫JWT（JSON web token）的协议，它是完全基于JSON的，Spring Cloud架构中也使用了JWT。

问题：对于这个架构，服务划分的粒度有没有什么好的建议？另外登录凭证保存在客户端如何解决报文被拦截的安全漏洞？  
答：服务划分需要按具体业务来说，一般来说，一个业务实体作为一个微服务。使用https可以一定程度上提高安全性。

问题:spring cloud security可以解决token重放攻击么？  
答：token重放攻击不是特别了解，可能是数据弱一致性导致的，建议设计尽可能短的过期时间。

问题：我们公司现在在设计DMP，从行业的状况来看，采用了微服务，但是有一点，首先对于应用本身暴露出来的服务，是和应用一起部署的，也就是并非单独的部署，那么业务组件接口暴露部署是否合理呢？  
答：业务组件的接口一般可以通过统一网关来管理。也可以对业务接口像spring cloud中设置访问scope限制。

问题：所有暴露的微服务是否需要一个统一的服务管控和治理平台？  
答：是的，一般有服务网关和服务发现组件来管理用户请求。

问题：微服务的gateway需要实现底层多个细粒度的API组合的场景，我们现在一部分使用异步，但是遇到了没办法全面的解藕。我想问问，对于此，使用响应式？还是异步回调式？它们的区别点会有哪些呢？  
答：使用哪种API方案，其实要看业务。如果后端业务需要强数据一致性，建议使用响应式的。反之，可以使用异步回调或者消息队列。

问题：uaa和netflix zull集成 可行吗？是否做过这方面的尝试？  
答：可以。Zuul组件提供网关服务，uaa是基于OAuth2协议，提供授权服务的。微服务架构下，他们是独立的，是可以自由组合的。举个例子，可以在zuul组件的配置文件中，为授权服务（auth-service）组件的指定路由表。可参照：

# zuul 配置

zuul:

ignoredServices: '\*'

host:

connect-timeout-millis: 20000

socket-timeout-millis: 20000

routes:

auth-service:

path: /uaa/\*\*

url: http://auth-service:5000

stripPrefix: false

sensitiveHeaders:

account-service:

path: /accounts/\*\*

serviceId: account-service

stripPrefix: false

sensitiveHeaders:

转载URL：https://segmentfault.com/a/1190000006785852