Atitit 负载均衡可能遇到的问题总结

目录

[1. Session分布式共享问题 1](#_Toc8535)

[1.1. session id 集中存储Redis （spring session组件） 1](#_Toc16965)

[1.2. session 的复制 1](#_Toc9339)

[1.3.  session sticky粘性会话 1](#_Toc7307)

[1.4. 基于持续时间的会话粘性 2](#_Toc20279)

[1.5. 应用程序控制的会话粘性 2](#_Toc16842)

[1.6. 改造成基于客户端模式cookie token （推荐） 3](#_Toc30903)

[1.6.1. Token 3](#_Toc23727)

[2. Cookie转发问题 4](#_Toc14457)

[3. 域名导致cookie跨域转发问题 4](#_Toc26682)

[3.1. CORS(跨域资源共享) 5](#_Toc16654)

[3.2. Host重设置 5](#_Toc10747)

# Session分布式共享问题

## session id 集中存储Redis （spring session组件）

后来有个叫Memcached的支了招：把session id 集中存储到一个地方， 所有的机器都来访问这个地方的数据， 这样一来，就不用复制了， 但是增加了单点失败的可能性， 要是那个负责session 的机器挂了， 所有人都得重新登录一遍。

常见语言体系都有对sesseion保存在redis的机制，配置化

如果需要开发，拦截器开发一下即可 保存读取session api

## session 的复制

## ****session sticky****粘性会话

默认情况下，传统负载均衡器会将每项请求单独路由到负载最小的已注册实例。但是，您可以使用粘性会话 功能 (也称为会话关联)，使负载均衡器能够将用户会话绑定到特定的实例。这可确保在会话期间将来自用户的所有请求发送到相同的实例中

****内容****

* [基于持续时间的会话粘性](https://docs.aws.amazon.com/zh_cn/elasticloadbalancing/latest/classic/elb-sticky-sessions.html" \l "enable-sticky-sessions-duration)
* [应用程序控制的会话粘性](https://docs.aws.amazon.com/zh_cn/elasticloadbalancing/latest/classic/elb-sticky-sessions.html" \l "enable-sticky-sessions-application)

## 基于持续时间的会话粘性

负载均衡器会使用一个特殊 Cookie（即 AWSELB）来跟踪发送到每个侦听器的每个请求的实例。在负载均衡器收到请求时，它首先会检查并查看请求中是否存在这个 Cookie。如果是这样的话，该请求会发送到 Cookie 中指定的实例。如果没有 Cookie，负载均衡器会根据现有的负载均衡算法选择一个实例。响应中会插入 Cookie，从而将同一用户发出的后续请求绑定到该实例中。粘性策略配置会定义 Cookie 的过期时间，从而确定每个 Cookie 的有效持续时间。负载均衡器不会刷新 Cookie 的过期时间，并且在使用 Cookie 前不会检查它是否已过期。Cookie 过期后，会话将不再具有粘性。一旦 Cookie 过期，客户端就会从其 Cookie 存储中删除 Cookie。

对于 CORS（跨源资源共享）请求，某些浏览器需要 SameSite=None; Secure 来启用粘性。在这种情况下，Elastic Load Balancing 会创建第二个粘性 Cookie（即 AWSELBCORS），该 Cookie 中包含原始粘性 Cookie 中的信息加上此 SameSite 属性。客户端会同时收到这两个 Cookie。

如果实例失败或者实例运行状况不佳，负载均衡器会停止将请求路由到该实例，并根据现有负载均衡算法来选择新的运行状况良好的实例。此时会将请求路由到新实例，就像没有 Cookie 一样，会话不再具有粘性。

如果客户端切换到带其他后端端口的侦听器，粘性将丢失。

## 应用程序控制的会话粘性

负载均衡器使用一个特殊的 Cookie 将会话和处理初始请求的实例相关联，但会沿用策略配置中指定的应用程序 Cookie 的使用时间限制。负载均衡器只会在应用程序响应中包含新的应用程序信息记录程序时，才会插入新的粘性信息记录程序。负载均衡器粘性信息记录程序不会根据每项请求进行更新。如果应用程序信息记录程序被明确删除或过期，会话便会停止粘着，直至发布新的应用程序信息记录程序为止。

系统会将以下由后端实例设置的属性发送到 Cookie 中的客户端：path、port、domain、secure、httponly、discard、max-age、expires、version、comment、commenturl 和 samesite。

如果实例失败或者实例运行状况不佳，负载均衡器会停止将请求路由到该实例，并根据现有负载均衡算法来选择新的运行状况良好的实例。现在，负载均衡器将会话视为“附加”到新的正常运行实例，并继续将请求路由至该实例，即使之前失败的实例已恢复正常运行。

## 改造成基于客户端模式cookie token （推荐）

### ****Token****

在Web领域基于Token的身份验证随处可见。在大多数使用Web API的互联网公司中，tokens 是多用户下处理认证的最佳方式。

1.用户通过用户名和密码向服务端发送请求

2.服务端通过验证，生成一个token发送给客户端

3.客户端保存token，发送请求时带上token

4.服务器通过验证，返回数据

****token的优势****

1.无状态、可扩展

2.支持移动设备

3.跨程序调用

4.安全

大部分你见到过的API和Web应用都使用tokens。例如Facebook, Twitter, Google+, GitHub等。

比如说， 小F已经登录了系统， 我给他发一个令牌(token)， 里边包含了小F的 user id， 下一次小F 再次通过Http 请求访问我的时候， 把这个token 通过Http header 带过来不就可以了。

带过来之后就可以通过user id 查到他的权限和和他相关各种数据返回了了。

****安全性****

对数据做一个签名， 比如说HMAC-SHA256 算法，加上一个只有我才知道的密钥， 对数据做一个签名， 把这个签名和数据一起作为token ， 由于密钥别人不知道， 就无法伪造token了。

这个token 我不保存， 当用户把这个token 给我发过来的时候，我再用同样的HMAC-SHA256 算法和同样的密钥，对数据再计算一次签名， 和token 中的签名做个比较， 如果相同， 我就知道用户已经登录过了，并且可以直接取到该用户的的user id , 如果不相同， 数据部分肯定被人篡改过，就证明这个人没有登录

这样还是维持了http 无状态情况，机器集群现在可以轻松地做水平扩展， 用户访问量增大， 直接加机器就行。

# Cookie转发问题

# 域名导致cookie跨域转发问题

## CORS(跨域资源共享)

## Host重设置