# 映射模式调用开发指南

在我们公司的一些服务，因为要求对每个环境的数据都是统一的，类似的这些服务都只在生产环境进行了部署，那开发环境，测试环境如何跨环境调通生产环境的机器叻? 映射模式调用就解决了这个问题。  
 在映射模式开发之前，sso-service自行有一套跨环境调用的方案，即通过在请求头中附带X-SSO-MR 和 X-SSO-SIGN 从而能够完成跨环境调用的效果。但sso原来的方案存在两个问题：

1. sso服务本身不再维护各个应用的签名秘钥，现存的都是一些老服务原来遗留下来的秘钥，所以新应用在用原来的老方案对接sso，sso没有新应用的秘钥，所以无法完成新应用的签名校验。
2. 原来的方案安全性并不好，怎么不好，就不说了，秘密。

基于sso的跨环境调用需求，我们将原来的调用方案做了应用维度的推广，做成了一个更加广泛的方案----映射模式调用。

以sso为例，我们在实现单点登录时需要依赖sso的RemoteSsoService远程接口，在

没有映射模式的时候，调用远程接口的方法是:从eureka中获取目标服务的ip列表，运用负载均衡策略选择其中的一个ip,拼凑远程接口的链接，然后实现调用。而映射模式调用的过程是：如果发现apollo配置中存在某个应用的映射模式调用的公网域名，则忽略eureka和负载均衡选择的过程，直接把目标服务的域名拼在远程接口的前面，此时还要按映射调用的要求添加一个验证请求头，即可完成映射的调用。

## 映射模式的配置

映射模式在apollo配置中的配置格式为:

application.boot.mapping-mode.[目标服务名称]=目标服务调用地址

然后还有一个oauth 第三方服务器的地址

application.boot.oauth.host

最后解析 application.boot. 前缀的配置可以得到一个json结构的对象

{

mapping-mode:{

应用名称:映射调用地址

},

oauth.host:验证服务器地址

}

## 映射模式的调用过程

每个远程接口都归属于一个服务，所以平时的同环境调用时，会用这个服务的appId去eureka中获取服务实例列表，而加入映射模式的逻辑后，过程为：

首先判断appId 在配置对象 mapping-mode 中是否存在，如果存在，取出该应用的映射调用地址，拼在接口名称前面，如sso-service的一个接口例子 http:// sso.duiba.com.cn:443/remoteSsoService/findOneAdmin?\_p0=1。sso.duiba.com.cn:443这部分就是从映射配置中取出来的。

然后添加请求头：X-MAPPING-MODE-AUTH 和 X-MAPPING-MODE-STATE  
 X-MAPPING-MODE-STATE：是oauth验证的存根，在应用启动的时候，使用UUID随机生成一个字符串。在后续的过程永恒不变。

X-MAPPING-MODE-AUTH：是向oauth服务器请求来的AccessToken。

## AccessToken的获取与管理

accessToken的获取是通过配置对象中的 oauth.host 指定的验证服务器地址来获取的。

获取方法代码参考为  
  
OauthTokenRequest request = new OauthTokenRequest();  
request.setAppAlias(applicationProperties.getApplicationName());  
request.setAppSecret(applicationProperties.getSecret());  
request.setEnvironment(currentEnvironment.getEnvId());  
request.setState(getState());  
  
OauthToken token = oauthRestTemplate.postForObject(oauth.host + "/oauth/getAccessToken",request,OauthToken.class);  
if(Objects.*isNull*(token)){  
 throw new RuntimeException("获取accessToken失败:OauthToken为空");  
}  
if(StringUtils.*isBlank*(token.getAccessToken())){  
 throw new RuntimeException("获取accessToken失败:"+token.getMessage());  
}  
if(!StringUtils.*equals*(getState(),token.getState())){  
 throw new RuntimeException("获取accessToken验证失败");  
}  
*log*.info("获取AccessToken:"+token.getAccessToken());  
return token;

获取过程是一个post请求，参数{appAlias, secret, envId, state}

appAlias是当前服务的名称，也就是自己服务的名称，secret 取自apollo配置 spring.application.secret ，envId 为环境变量，开发环境：dev,测试环境：test,生产和预发环境：prod, state:就是应用在启动的时候生成的存根

每个accessToken的有效时间为两个小时,可自行选择方案把accessToken缓存在内存中，然后定时去失效。在接口调用过程中，如果发现对方服务放回的异常code 为 mapping\_mode\_error 也可以主动去失效accessToken。下次调用重新去获取。