一.前言

今天，我们转战用友系列的第一个产品---T+/Tplus。前两篇文章讲解分享的都是金蝶的产品，因为本身公司牵涉的业务有限，后续有金蝶其他产品的API对接业务时，会继续来分享经验。

T+的API接口，哎，想起来都是心酸泪。关于该接口的对接开发经验，我之前也简单记录了一些，公众号里记录的在这里就不放链接了。今天我们就来详细解析下这令人头大的财务API接口。

二.API接口详解

2.1接口定义和入参

根据开发者社区API文档的描述我们可以看到，T+版本为12.3以上的API对接，都必须使用V2版本，那v2与v1版本的区别有哪些呢？主要有两点：1.
请求认证方式，增加云企业ID认证方式 ；2.v2版本支持异步请求。OK，因为我们对接的客户财务环境为12.3，那么我们就来处理v2版本的OpenAPI，该版本的API引入了鉴权机制，简单来说就是在请求头增加了授权 Authorization 参数.

2.1.1 Authorization参数以及签名处理

那么 Authorization 参数如何才能生成呢？可以看官网首页的描述是 需要appkey、appsecert、私钥的文件全路径。那这三个参数又如何才能获取呢？必须申请ISV认证，即注册ISV，提交开发申请通过审核后，总部会将这三个参数一并发到注册时预留的邮箱里。

2.1.2 OrgId方式的签名算法处理

这里，我们在上一步已经拿到签名所需的三个必要参数了，官网给了两种请求Head的处理方式，一种使用OrgId，一种使用用户名、密码、账套号，这两种方式我们都会讲到。先看第一种方式OrgId访问。

OrgId的获取方式，官网描述的也有

这样，我们第一种使用OrgId认证方式的所需参数就已全部准备完毕了，接着往下看，首先需要对appKey、orgid、appsecret、私钥全路径做一个签名1的算法加密，这个算法官网给我们提供的也有，这里仅提供C#版本的签名算法1.接着做一个Base64位的加密即可得到Authorization参数。

```
 if (!APIConfig.AuthorizeParameters.ContainsKey("appkey")
                || !APIConfig.AuthorizeParameters.ContainsKey(
"orgid")                    || !APIConfig.AuthorizeParameters.C
ontainsKey("appsecret")                    || !APIConfig.Author
izeParameters.ContainsKey("secerturl"))                {
          throw new Exception("???????");                }
        var request = new AccessTokenRequest();
Dictionary<string, object> parm = new Dictionary<string, obj
ect>();              string appkey = APIConfig.AuthorizeParam
eters["appkey"];              string orgid = APIConfig.Author
izeParameters["orgid"];              string appsecret = APICo
nfig.AuthorizeParameters["appsecret"];              string se
cetrurl = APIConfig.AuthorizeParameters["secerturl"];
       parm.Add("appkey", appkey);              parm.Add("org
id", orgid);              parm.Add("appsecret", appsecret);
```

```
            JsonSerializer jsonSerializer = new JsonSerializ
er();             string datas = jsonSerializer.Serialize(pa
rm);             try             {                 var signC
lass = new TokenManage();                 string signvalue =
 signClass.CreateSignedToken(datas, secetrurl);
    string authStr = @"{""appKey"":""" + appkey + @""",""aut
hInfo"":""" + signvalue + @""",""orgId"":" + orgid + @"}";
               string encode = Convert.ToBase64String(UTF8En
coding.UTF8.GetBytes(authStr));                 Dictionary<s
tring, string> parms = new Dictionary<string, string>();
             parms.Add("Authorization", encode);
     request.SetHeaderParameters(parms);                 var
 response = Excute(request);                 return response
;             }             catch (Exception ex)
 {                 throw new Exception(ex.Message);
    }
```

当你处理完第一步，调试接口正常返回

```
 {"result":true,"access_token":"03e74889-1457-48cd-970a-
ba3742ffcdea","sid":""}
```

先不要高兴的太早了，我们还要根据这一步获取到的Token做业务调用。如图所示

这是第一步得到Token的方法，可以看到签名方式和加密方式不变，变得是签名方式的参数不同，orgId为空，在PostBody里要传入用户名、密码和账套号。

增加了上一步获取到的Token，详细代码如下

```
            var signClass = new TokenManage();
var request = new GetTokenByPwdRequest();            string
 appkey = APIConfig.AuthorizeParameters["appkey"];
   string appsecret = APIConfig.AuthorizeParameters["appsecr
et"];            string secetrurl = APIConfig.AuthorizePara
meters["secerturl"];            string userName = APIConfig
.AuthorizeParameters["userName"];            string passwor
d = APIConfig.AuthorizeParameters["password"];            s
tring EncodePassword = signClass.GetMd5(password);
   string accNum = APIConfig.AuthorizeParameters["accNum"];
            Dictionary<string, object> parm = new Dictionar
```

```
y<string, object>();              parm.Add("appkey", appkey);
             parm.Add("orgid", "");             parm.Add("ap
psecret", appsecret);              JsonSerializer jsonSerial
izer = new JsonSerializer();              string datas = json
Serializer.Serialize(parm);             try             {
               string signvalue = signClass.CreateSignedToken
(datas, secetrurl);                string authStr = @"{""ap
pKey"":""" + appkey + @""",""authInfo"":""" + signvalue + @"
"",""orgId"":""""}";                string encode = Convert
.ToBase64String(UTF8Encoding.UTF8.GetBytes(authStr));
          Dictionary<string, string> parms = new Dictionary<
string, string>();                parms.Add("Authorization"
, encode);                request.SetHeaderParameters(parms
);                Dictionary<string, object> postParms = n
ew Dictionary<string, object>();                var args =
new PwdEntity() { userName = userName, password = EncodePass
word, accNum = accNum };                var argsJson = json
Serializer.Serialize(args);                postParms.Add("_
args", argsJson);                request.SetPostParameters(
postParms);                var response = Excute(request);
                return response;             }
catch (Exception ex)             {                throw new
 Exception(ex.Message);             }
```

两种方式都处理完毕了，我们就可以愉快的使用API来做业务的处理啦。

2.2 业务接口调用

2.2.1 会计科目查询

接口描述

查询凭证信息,返回类型DataTable。

请求语法

```
Post  TPlus/api/v1/brand/QueryHTTP/1.1
Host:: ${host}
Content-Type: application/x-www-form-urlencoded;charset=utf-8
Authorization:  ${SignatureValue}
```

公共参数

12.3及以上版本，请查看：http://tplusdev.chanjet.com/library/5b2223723eeec22d2787feae

名称	描述
URL	TPlus/api/v1/doc/Query
${host}	T+服务器，域名或IP，支持端口号
${SignatureValue}	

业务参数

```
{
   dtos:[{
     ExternalCode:"2",
     DocType:{Name:"记账凭证"},
     VoucherDate:"2014-10-13"
      },{
     ExternalCode:"3",
     DocType:{Name:"记账凭证",Code:"记"},
     VoucherDate:"2014-10-13"
   }],
   param:{}
}
```

参数描述

名称	是否必须	描述
dtos	是	数组类型，数组的第一元素为其实期间，第二元素为截止期间

字段描述

名称	描述
ExternalCode	外部凭证号
DocType	凭证类别
VoucherDate	制单日期
AccountingYear	会计年度
AccountingPeriod	年度内期间

三.结束语

其实，真正对于T+的业务调用并没有花费很多时间，因为前面的坑已经踩完了，后面基本上也没啥了，就是我很纳闷的是，每个接口的返回值格式是不固定的，这个就很令人费解啊。咱也搞不懂到底为啥这样定义。倒是前面处理签名算法和dll不兼容的问题前前后后大约搞了一个星期才完整解决，这个很是让人头大。

曾经在T+的开发群里看到这样一句话，每个开发都是一个实施。也确实是这样一种情况，你对接的每个API接口，不可能总会有人给你解答问题，这时候你只能靠自己去摸索，去猜，当然了大部分的开发文档还是很规范的。其实做对接的做多了，你会遇到不同形式的API接口，每家厂商都有自己独特的开发标准，我们能做的就是遵循这套标准，不然如何对接，如何正确处理我们的工作。

最后的最后，希望我们做API对接或者说是做开发的，要保持一颗良好的心态去面对问题，要相信问题总是会被解决的，只是时间早晚。而且要找对方法，比如社区，或者对应的交流群等等，会有很多大佬帮你解答疑惑，祝你在开发的道路上勇往直前的。

我是程序猿贝塔，一个分享自己对接过财务系统API经历和生活感悟的程序员。