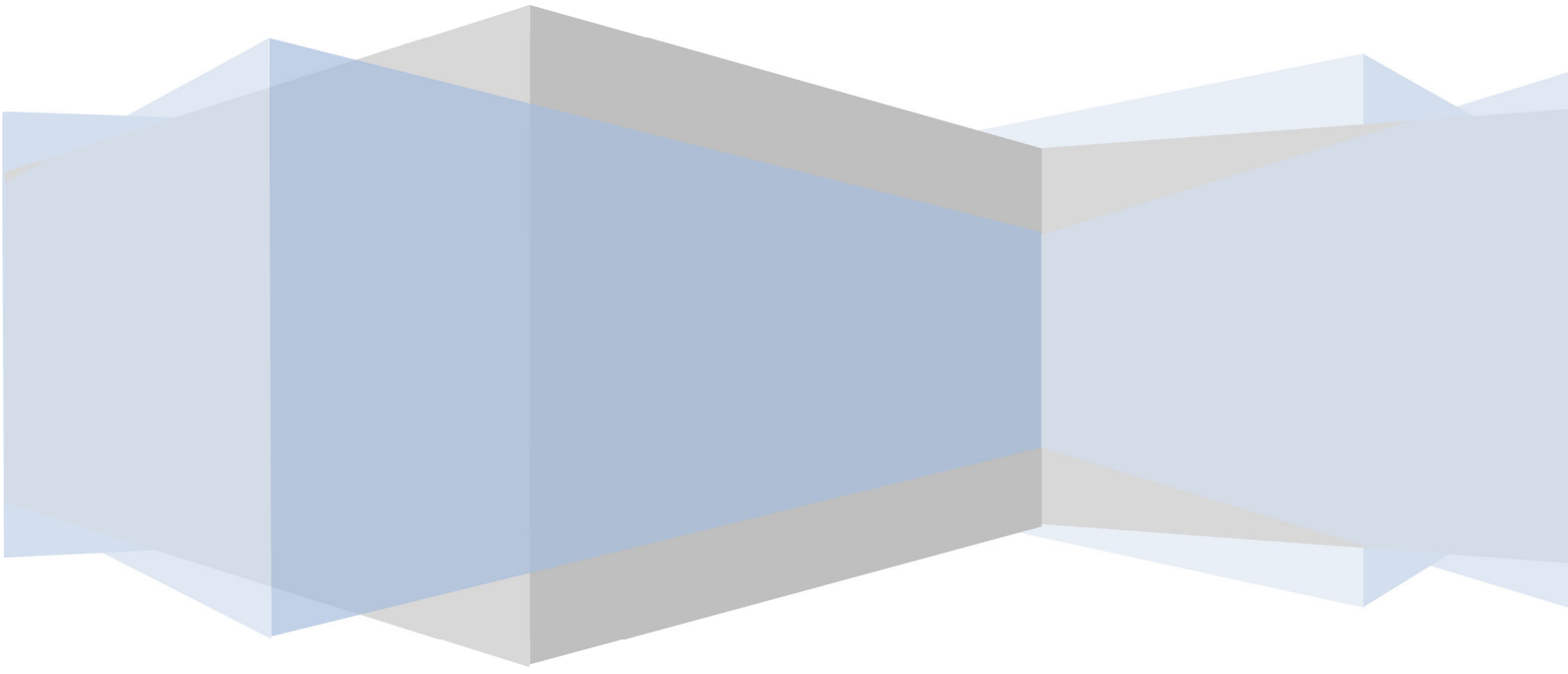
重庆工业职业技术学院
数据接口标准
规范　V1.0



目录

1. 简介

大数据治理平台数据服务开放接口(DATASERVICE OPEN API)是数据门户提供的一种集成网络服务 API。通过该接口，数据门户的用户可以申请使用服务，以编程的方式调用并获取数据服务的开放数据、API 服务数据等。借助于数据服务开放接口，数据门户的用户可以基于不同业务场景和特定需求来定制独特的应用程序。

1.1目的

数据服务开放接口说明书用于说明 API 接口的设计及约束，详细描述 API 的作用和使用方式，从而为应用程序设计者提供参考依据，为制定 API 服务的开发者集成规范提供指导等。

1.2对象

使用大数据治理平台数据服务的业务人员、集成和使用大数据治理平台数据服务 API 接口的开发人员及测试人员等。

1.3范围

大数据治理平台数据服务模块，包含数据服务、标签服务和 API 服务。

1.4输入资料

无。

1.5参考资料

无。

2 指南

2.1集成前准备

要集成使用数据服务开放接口客户端程序，您需要一个适合 JAVA 的开发环境，然后在智能数据运营平台中向资源提供方申请需要的数据资源、标签资源或 API 资源。在您的申请通过审核后，可在智能数据运营平台“个人中心→我的数据”中查看服务，并可在接口调试中调试已审批的服务。在接口调用界面中，可获取到集成数据服务开放接口所必需的请求参数，根据数据服务开放接口说明书和接口指定的参数可以自由构建自己的应用。

2.2API 身份认证

在使用数据服务开放接口时，必须要通过身份认证方可正常访问数据服务和 API 服务提供的数据。数据服务开放接口提供了数据服务和 API 服务的客户端 sdk。数据服务开放接口客户端 sdk 可以帮助您节省时间，并确认所发送的请求格式正确。服务开放接口客户端 sdk 包随接口文档一起下载，您可以在 sdk 目录中获取（服务开放接口客户端 sdk 包的使用方式可参考最佳实践章节，或参考接口文档中提供的 demo 程序）。您也可以通过自编码的方式实现服务开放接口客户端 sdk 包的认证逻辑。 以下介绍了服务开放接口的身份认证流程，供开发人员、测试人员等了解：

数据服务开放接口的身份认证采用请求身份签名认证方式。客户端用身份认证标示字符串（参见构建身份认证表示字符串）和 secret key 通过 HMAC-SHA 算法计算 Signature；客户端提交请求时，将计算好的 Signature 和身份认证参数（参见必须的身份认证参数）放入请求 Header 中一起提交到服务端；服务端用 Header 身份认证参数中的用户标示找到对应的 secret key；并采用相同的算法计算 Signature 和客户端提交的 Signature 比较，若一致即认证成功。客户端在身份认证标示字符串中放入时间戳，采用一次请求一次认证的方式最大限度的保证了数据服务开放接口身份认证的安全性。

构建身份认证标示字符串

要计算数据服务开放接口的请求身份签名，需要先构建一个身份认证标示字符串。身份认证标示字符串是一个有顺序的 JSON 字符串，其顺序按照字段属性字母（a~z）排序。身份认证标示字符串由以下元素组成：

userId – 用户 Id，在大数据治理平台数据服务中的用户标示，可从订阅服务中获取该信息。

deptId – 部门 Id，使用大数据治理平台数据服务的部门标示，可从订阅服务中获取该信息。

timestamp – 当前系统时间戳。每个请求都必须包含请求的时间戳。

以下为数据服务开放接口的身份认证标示字符串示例：

```
{"deptId":"部门 id","timeStamp":时间戳,"userId":"用户 id"}
```

计算 Signature

签名是数据服务开放接口身份验证过程的一部分，目的是识别并验证发送请求的主体。签名作为构建的请求中 signature 标头的值，在数据服务开发接口中用于验证请求者的身份。其具体计算过程如下：

1、根据上一节“构建身份认证标示字符串”中的说明，构建参与身份签名 JSON 字符串。注意字段顺序，如下示例：

```
{"deptId":"67f3cd734d094e719f1900a72f296b0f","timeStamp":1617955673663,"userId":"731da71fdd6d4040b294a471d9fd29fc"}
```

2、根据构建好的身份认证标示 JSON 字符串计算 HMAC 值，同时将您的 secret key 用作密钥。HmacSHA256 和 HmacSHA1 都是受支持的哈希算法，但数据开放服务接口建议使用 HmacSHA256。

3、将得出的值转换为 base64 字符串，即可得到 Signature 字符串。

以下示例介绍如何使用 Java 计算签名：

```
public static String signParameters(String paramStr, String key) throws SignatureException {
    String algorithm = SignatureMethod.HmacSHA256.getCode();
    return sign(paramStr, key, algorithm);
}
public static String sign(String data, String key, String algorithm) throws SignatureException{
    byte[] signature;
    try {
        Mac mac = Mac.getInstance(algorithm);
        mac.init(new SecretKeySpec(key.getBytes(), algorithm));
        signature = Base64.encodeBase64(mac.doFinal(data.getBytes("utf-8")));
    } catch (Exception e) {
        throw new SignatureException("Failed to generate signature: " + e.getMessage(), e);
    }
    return new String(signature);
}
```

必须的身份认证参数

身份认证参数是数据开放服务接口请求必不可少的参数，要求在提交请求时将用于身份认证的参数放入请求的表头中（请求 Header）。数据开放服务平台只会从请求的标头中获取 Signature 和身份认证参数来识别请求者身份和请求的合法性。身份认证参数包括以下内容：

- **Signature**：请求身份签名，在计算 Signature 章节的计算结果。每个请求都必须包含有效的签名，否则请求将被拒绝。
- **Sign-User**：请求数据开放服务接口的用户标示，和身份认证标示字符串中的 userId 一致。
- **Sign-Timestamp**：提交数据开放服务接口请求的当前系统时间戳。必须和身份认证标示字符串中的 timestamp 一致。
- **Sign-Encoding**：请求数据开放服务接口的编码格式。

2.3 API 请求参数安全

API 请求参数安全是防止客户端提交的数据服务开放接口请求参数被恶意篡改的保护策略，是提高数据开放服务接口安全性的第二道保证。构建 API 请求参数安全不是访问数据开放服务接口的必须步骤，因为数据开放服务接口已经足够保障请求数据的安全性。

API 请求参数安全是在客户端提交请求时，将客户端提交的业务参数组织成有序的 JSON 字符串，通过 MD5“摘要”（或哈希）算法计算出固定长度的 MD5 校验和，并放入请求的表头中；服

务端接收到请求后用统一的方式将业务参数组织成有序的 JSON 字符串，通过 MD5 算法计算出固定长度的 MD5 校验码和请求表头中的 Content-MD5 值比较，确保接收的请求数据在传输过程中没有出现错误以及被篡改。具体实现步骤如下：

1、根据提交请求的业务参数构建有顺序的 JSON 字符串，其顺序按照字段属性字母（a~z）排序。业务参数请参考具体接口的输入参数说明章节，如下示例展示数据服务的业务参数字符串：

```
{"applicationId":"应用系统编号","subServiceId":"订阅服务 ID","body":"body 字符串
","headers":{"headerKey":"headerValue"},"pathParams":{"pathKey":"pathValue"},"queryString":{
"queryKey":"queryValue"}}
```

2、将您的 secret key 作为 Slot 追加到已经构建好的请求业务参数 JSON 字符串后面。如下所示：

```
{"applicationId":"应用系统编号","subServiceId":"订阅服务 ID","body":"body 字符串
","headers":{"headerKey":"headerValue"},"pathParams":{"pathKey":"pathValue"},"queryString":{
"queryKey":"queryValue"}}731da71fdd6d4040b294a471d9fd2fadsfdc
```

3、用构建好的请求业务参数 JSON 字符串计算 MD5 值。

4、将得出的 MD5 字符串作为 Content-MD5 表头放入请求表头中。

以下示例介绍如何使用 Java 计算 MD5 值：

```
public static String encryptMD5Content(String plaintext, String salt,String encoding) throws Exception {
        if (Tools.isNull(plaintext)) { return null; }
        //获取加盐明文
        byte[] saltText = addSalt2End(plaintext, salt, encoding);
        byte[] digest = DigestHelper.digest(saltText, EncryptionEnum.MD5.getCode());
        return Hex.toHexString(digest);
    }
    public static String toHexString(byte[] var0) {
        return toHexString(var0, 0, var0.length);
    }
    public static String toHexString(byte[] var0, int var1, int var2) {
        byte[] var3 = encode(var0, var1, var2);
        return Strings.fromByteArray(var3);
    }
    public static byte[] encode(byte[] var0, int var1, int var2) {
        ByteArrayOutputStream var3 = new ByteArrayOutputStream();
        try {
            encoder.encode(var0, var1, var2, var3);
        } catch (Exception var5) {
            throw new EncoderException("exception encoding Hex string: " + var5.getMessage(), var5);
        }
        return var3.toByteArray();
    }
```

2.4 API 节流限制

要正常使用数据服务开放接口，需要了解相关限制，即特定时间内提交的请求数量不得超过设

定的范围。限制功能可以保护网络服务免受过量请求的不利影响，确保所有授权的开发者均可访问网络服务。

数据服务开放接口采用漏桶算法来衡量网络服务状态并执行相关限制。该算法的依据是漏桶假设（漏桶中的水以一定的速率从桶底的小孔中漏出），在此场景下，可以间歇性地向桶中注水，但如果一次性注水量太大，或注水的频率过高，水就会溢出桶外。

在数据服务开放接口中采用漏桶算法时，设想漏桶的容积代表最大请求限额，即单次提交的最大请求数量。桶底的小孔代表恢复速率，即提出新的请求前需要的恢复时间。因此，如果您一次提交了太多请求，漏桶中的水就会溢出，而在数据服务开放接口中，则会触发限制。在将漏桶注满后，由于漏水速率是一定的，为了防止漏桶的水溢出，因此需要等待一段时间后再继续注水。同理，达到最大请求限额后，恢复速率决定了提出新的请求前需要等待的时间。

控制数据服务开放接口限制功能的三个值定义如下：

- **请求限额**：每次允许提交而不会触发限制的请求数量。请求限额会随着提交请求的增加而减少，并以还原速率增加。请求按各订阅服务（在数据门户中订阅服务并审核通过后）进行统计计算。
- **还原速率（也称恢复速率）**：随着时间流逝，请求限额增加到最大请求限额的速率。（目前还在完善中，等待更新）
- **最大请求限额（也称溢出速率）**：请求限额可以达到的最大数值。

案例分析

假设您需要使用“数据服务 A”操作来请求 50 页数据（50 次请求）。“数据服务 A”操作的最大请求限额为 30，恢复速率为每分钟 10 个请求。如果您提交了 50 个上传数据请求，而在达到 30 个请求之后，数据服务开放接口会触发限制，那剩余的 20 个上传数据请求则必须等到请求限额还原后再重新提交。由于还原速率是每分钟 10 个请求，因此要过 2 分钟才能提交剩余的 20 个上传数据请求。（目前的算法是：最大请求限额=恢复速率，例如每分钟 20 次，即最大请求限额为 20 次，恢复速率为每分钟 20 个请求)

最佳应用

由于现有部分算法暂未开放，现在的最佳算法是避免瞬时到达最大请求限额的操作。以“数据服务 A”的限额配置为每分钟 30 个请求为例，则可以每 10 秒提交 5 个“数据服务 A”操作请求，而下一分钟又恢复了 30 个请求。采用此种平滑的请求提交模式来保证请求永远不会达到最大限额。

同时，应该考虑自动化提交请求，并做好应急准备，以应对达到最大请求限额而触发限制，或网络服务流量过高的情况。此时应减少提交请求的数量，并重新提交失败的请求。

2.5必需的请求参数

下表列出了数据开放服务接口操作需要设置的所有请求参数：

名称	描述	必填项	传参方式
Signature	请求身份签名，在计算 Signature 章节的计算结果。 类型：string	是	Header
Sign-User	请求数据开放服务接口的用户标示。 类型：string	是	Header

名称	描述	必填项	传参方式
Sign-Timestamp	提交数据开放服务接口请求的当前系统时间戳。 类型：long	是	Header
Sign-Encoding	请求数据开放服务接口的编码格式。 类型：string	是	Header
Content-MD5	API 请求参数防篡改头。 类型：string	否	Header
业务参数	具体接口支持的业务参数 类型：Json	是	Body

2.6响应格式

在操作请求的响应中，数据开放服务接口会返回一个 JSON 文件，其中包含请求结果。如果请求成功完成，则返回的响应中将附带所请求的数据。以下示例显示了一个成功响应。

```
{
    "status": false,  //请求成功或失败
    "code": 401,  //相应码
    "data": null,  // 返回数据

    "message": "认证失败"  //相应消息

}
```

2.7处理错误

下表列出了数据开放服务接口的相关错误代码及描述，其他错误请参考具体接口对错误信息的描述。

错误代码	描述
400	错误请求（接口内部验证错误）。
401	认证失败。
403	服务器拒绝请求，不支持非 POST 请求。
411	请求参数不能为空。
412	认证参数不能为空。
413	请求参数不匹配或参数已被篡改
416	接口已达到请求上限，请稍后再试。
417	服务器未满足"期望"请求标头字段的要求，签名参数不能为空。
500	服务器内部错误。

3 API 服务 API 参考指南

3.1API 请求转发接口

3.1.1 功能描述

通过智能数据治理平台的 API 服务封装功能将 API 接口资源封装为 API 接口服务，智能数据治理平台对接口资源访问、安全、流控统一管理。API 请求转发接口将用户提交的合法请求转发到接口资源提供的 API 接口，并响应数据到客户端。

3.1.2 接口格式

接口名称	API 请求转发接口
URL	{{serviceUrl}}
通信协议	HTTP/HTTPS
数据格式	JSON
HTTP 请求方式	POST

输入参数结构

输入参数数据结构描述了 API 服务接口的入参数据格式，业务参数必须按照此数据结构传递方能正常获取数据。输入参数结构并不代表业务含义，需要结合具体 API 服务接口的业务参数组合使用。API 服务接口的业务参数请参见业务参数章节。以下为 API 服务接口输入参数数据结构：

输入参数数据结构 JSON 示例：

```
{
    "applicationId": "应用系统编号",
    "subServiceId": "订阅服务 ID",
    "body": {
        "jsonBody": "body 字符串",
        "params": {
            "bodyKey": "bodyValue"
        }
    },
    "headers": {
        "headerKey": "headerValue"
    },
    "pathParams": {
        "pathKey": "pathValue"
    },
    "queryString": {
```

```
        "queryKey": "queryValue"
    }
}
```

返回值数据结构

返回值数据结构描述了 API 服务接口的响应格式，需要结合具体 API 服务接口返回的业务数据结构处理业务。API 服务接口的业务数据结构请参见返回值列表章节。

API 服务返回值数据结构 JSON 示例：

```
{
    "status": true,
    "code": 200,
    "data": {}//响应数据
}
```

3.1.3 输入参数

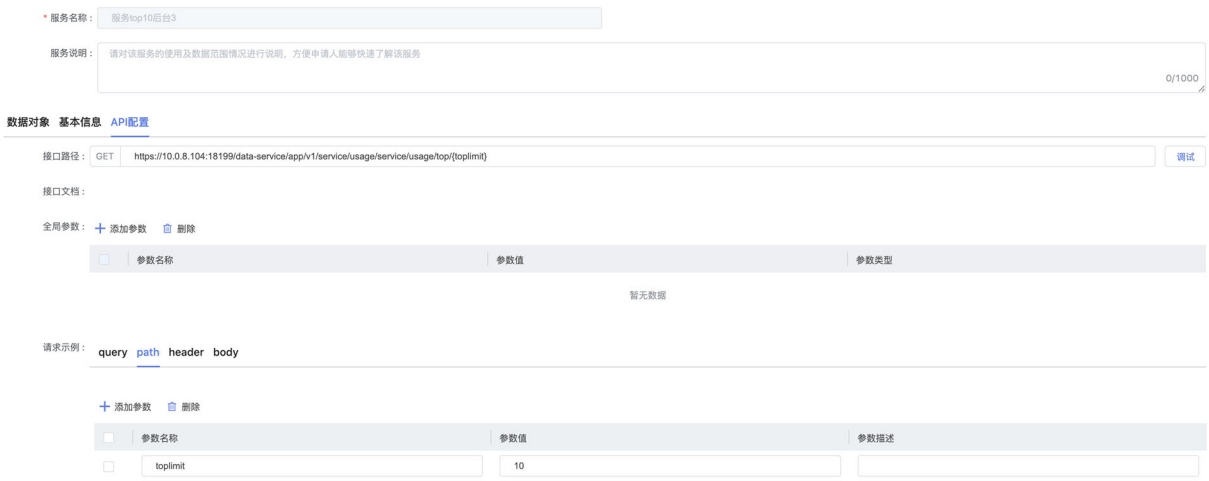
API 服务的输入参数取决于具体接口资源的 API 接口提供方。通常接口资源的 API 接口提供方会提供 API 接口文档，伴随 API 服务文档一起下载。API 接口提供方的详细的接口文档说明可在 document 目录中的其他文档中获取，也可参考订阅服务的请求示例。

3.1.4 返回值

API 服务的返回值取决于具体接口资源的 API 接口提供方。通常接口资源的 API 接口提供方会提供 API 接口文档，伴随 API 服务文档一起下载。API 接口提供方的详细的接口文档说明可在 document 目录中的其他文档中获取，也可参考订阅服务的请求示例。

使用示例：

以 ***测试 API 服务 GET 请求(path 传参)*** 为例，具体服务配置如下图：



请求参数示例：


```
{
    "subServiceId": "2259530762223670",
    "body": {},
    "headers": {
        "X-Request-Auth": "af462b4445d4449895d5c006427f5f1a"
    },
    "pathParams": {
        "toplimit": "10"
    },
    "queryString": {}
}
```


响应示例：


```
{
    "status": true,
    "code": 200,
    "data":"{\"code\":\"0\",\"data\":[{\"serviceType\":\"TABLE\",\"serviceVersion\
":\"v1.0\",\"reqTotalCount\":0,\"serviceStatus\":\"5_ENABLED\",\"totalCountView\":
{\"unit\":\"次\",\"count\":\"0\",\"originCount\":\"0\"},\"serviceName\":\"help
keywords( help_keyword )a\",\"serviceId\":\"2152568896684056\"}]} "
  }
```

4 最佳实践

API 服务使用示列

```
import com.sefonsoft.cloud.govern.service.invoke.ServiceSdk;
import com.sefonsoft.cloud.govern.service.invoke.domain.ThirdServiceRequestDTO;

import java.util.HashMap;

public class ThirdApiServiceInvocationDemo {

    public static void main(String[] args) throws Exception {
        //请求 URL
        String url ="https://govern.sefon.com:18199/data-service/gateway/v3.0/third-service/dispatch/1399678309564423";
        //用户 ID
        String userId="a88b6f172087449e8e79b7fb3db3cbde";
        //部门 ID
        String deptId = "063db71d02fa4baaa05d6a7d6b943ff1";
        //订阅 ID
        String subscribeId = "1399678309564433";
        //应用系统标识，没有应用系统即设置为 null
        String applicationId = null;
        //认证 token
        String accessToken = "2452769930805297";

        //********根据接口服务发布的接口文档构造业务参数**********//
        //构造 Query String 参数。根据接口文档确定是否需要配置
        HashMap<String,String> queryString = new HashMap<String, String>();
        queryString.put("name","彭丽林");
        queryString.put("gender","男");

        //构造 Path 参数。根据接口文档确定是否需要配置
        HashMap<String,String> pathParams = new HashMap<String, String>();
        pathParams.put("Id","123456");

        //构造 Header 参数。根据接口文档确定是否需要配置
        HashMap<String,String> headers = new HashMap<String, String>();
        headers.put("appKey","abc");

        //body 参数。根据接口文档确定是否需要配置
        BodyContent body = new BodyContent();
        body.setJsonBody("{\"key\":\"value\"}");

        //HashMap<String,String> formData = new HashMap<String, String>();
```

```
            //formData.put("key","value");
            //body.setParams(formData);

            // 构造请求
            ApiServiceDispatchRequest req = new ApiServiceDispatchRequest.Builder()
                    //设置 path 参数
                    .pathParams(pathParams)
                    //设置查询参数
                    .queryString(queryString)
                    //设置请求 header
                    .headers(headers)
                    //设置请求 body
                    .body(body)
                    .build();

            String result = ServiceSdk.invoke(url,req,deptId,userId,subscribeId,applicationId,accessToken);
            System.out.println("请求结果" + result);
        }
    }
```

工具包

API 服务调用 sdk 包 service-invoke-sdk-3.1.0-SNAPSHOT.jar