应用身份管理服务

开发指南

文档版本 01

发布日期 2024-12-26

华为云计算技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为云计算技术有限公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为云计算技术有限公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为云计算技术有限公司

地址： 贵州省贵安新区黔中大道交兴功路华为云数据中心 邮编：550029

网址： https://www.huaweicloud.com/

目 录

1 通过事件回调方式同步数据至应用

1.1 准备工作

OneAccess提供身份数据同步的功能，数据同步的关系模型可以理解为“上游—中游—下游”。其中“上游”指各种核心身份源，例如钉钉、企业微信、HR系统、也可以是OneAccess本身的身份管理模块，“中游”即OneAccess，“下游”指各类需要和上游保持同步的应用系统。通过该模型，OneAccess可以将上游的身份数据实时传递到下游，从而保证人员的入、离、调、转等行为能够快速准确的传递到下游各个应用系统中，实现用户的全生命周期管理，从而保障身份数据的实时同步与安全。

为了同步OneAccess数据至企业应用，企业应用需提前按照数据格式开发同步事件接口，具体请参考**调用说明**和**API**。

下面将为您介绍通过事件回调方式同步OneAccess数据至应用的操作步骤：

1. **企业管理员在管理门户配置事件回调**。
2. **新增、修改、删除用户**。
3. **新增、修改、删除组织**。
4. **同步事件**。
5. **全量同步**。
6. **同步状态说明**。

事件回调配置

本节主要介绍企业管理员在OneAccess管理门户配置事件回调的方法。

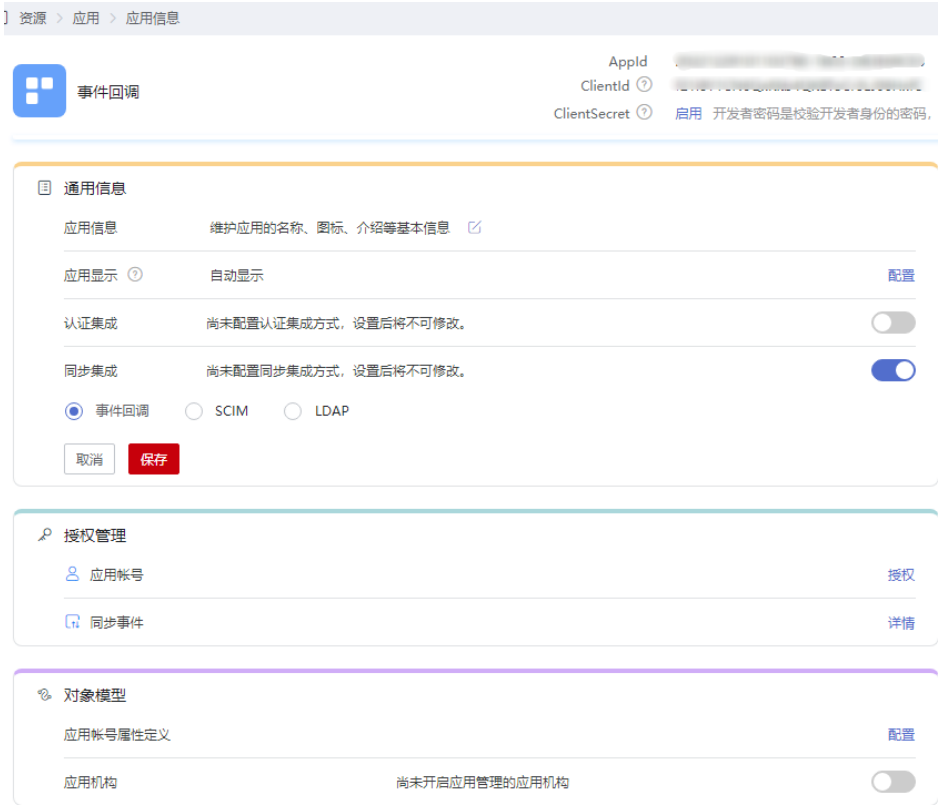
步骤1 登录OneAccess管理门户。

步骤2 在导航栏中，单击“资源 > 应用”。

步骤3 在企业应用页面，单击自建应用下的“添加自建应用”，设置Logo和名称，单击“保存”。

步骤4 单击**步骤3**创建的应用，在应用信息页面，在通用信息模块，单击“同步集成”后的  ，开启同步集成并设置同步集成方式为事件回调。单击“保存”。

图 1-1 开启事件回调



步骤5 在通用信息模块，单击“同步集成”后的“配置”进入“参数配置”页签，根据表1-1说明，配置参数，单击“保存”。

表 1-1 配置参数

参数	说明
* 回调URL	企业应用用来接收OneAccess推送事件的地址。
* 安全令牌	每个事件回调接口中的请求头携带Bear token，企业应用的回调服务需要进行鉴权。
加解密算法	推荐使用AES/GCM/NoPadding（默认），NULL为不加密，有安全风险，请谨慎使用。
加密密钥	用密钥对消息进行加密，默认为空，不为空时长度必须为16位。
签名密钥	使用签名密钥根据消息内容生成数据签名，默认为空，不为空时长度必须为16位。

步骤6 （可选）单击应用图标，在左侧导航栏选择“同步集成 > 常规配置 ”，可修改同步时的映射关系，包括删除账号、删除机构和禁用账号。

步骤7 （可选）单击应用图标，在左侧导航栏选择“对象模型 > 应用机构模型”，单击 开启应用机构。

说明

开启应用机构是为了同步机构数据至企业应用。如果您不需要同步机构数据，可忽略此步骤。

步骤8 （可选）配置机构属性与映射。如果您只需要同步机构的内置属性，可忽略此步骤。

1. 在左侧导航栏选择“对象模型 > 应用机构模型”，在属性定义页签，单击“添加”，可配置同步至企业应用的机构属性。

说明

- 当同步内置属性至企业应用时，需与应用机构模型中内置属性的属性名保持一致。
- 当同步非内置属性至企业应用时，需与企业管理员设置的属性名保持一致。

表 1-2 属性参数

参数	说明
* 属性名	同步至企业应用的字段标识，可自定义，设置成功后不支持修改。
显示标签	机构的属性名称。建议与**属性名**对应。
描述	**属性名**字段的填写说明。
* 属性类型	**属性名**字段的填写类型，可在下拉框选择，设置成功后不支持修改。
格式	**属性类型的格式。**当**属性类型**选择**文本**时，可在下拉框选择。
是否必填	默认不勾选，即非必填。
是否唯一	当“属性类型”选择为“文本”时，可设置是否唯一。勾选后，同步机构数据至应用时，该属性的值具有唯一性，重复时，会提示“{显示标签}”已存在。
是否敏感	当“属性类型”选择文本时才需要设置该参数，勾选后，同步机构数据至应用时，数据隐藏展示，单击可以看到数据内容。

2. 在映射定义页签，单击“编辑”，设置机构属性的映射方式。

说明

- 当映射定义的转换方式选择脚本转换时，脚本编写请参考**如何开发映射脚本**。
- 为了避免同步异常，建议添加的机构属性与需要映射的组织属性类型保持一致。

表 1-3 映射参数

参数	说明
组织	映射至**应用机构**的组织属性，可在下拉框选择。
转换方式	**组织**属性与**应用机构**之间的映射方式，可在下拉框选择。

参数	说明
脚本表达式	**转换方式**为**脚本转换**时，填写具体的映射脚本。
执行方式	**组织**属性与**应用机构**之间映射的同步方式，可在下拉框选择。
应用机构	1中定义的机构属性。

3. 在“常规配置”页签，“删除系统组织”默认配置为删除应用机构，“禁用系统组织”默认配置为禁用应用机构。单击“编辑”可以修改配置，其中“删除系统组织”可以配置为禁用应用机构或不影响；“禁用系统组织”可配置为不影响。修改完成后，单击“保存”即可生效。

步骤9 （可选）配置账号属性和映射。如果您只需要同步账号的内置属性，可忽略此步骤。

1. 选择“对象模型 > 应用账号模型”，在“属性定义”页签单击“添加”，可配置同步至企业应用的账号属性。

   说明

   - 当同步内置属性至企业应用时，需与应用账号模型中内置属性的属性名保持一致。
   - 当同步非内置属性至企业应用时，需与企业管理员设置的属性名保持一致。

   表 1-4 属性参数

参数	说明
* 属性名	同步至企业应用的字段标识，可自定义，设置成功后不支持修改。
* 显示标签	账号的属性名称。建议与**属性名**对应。
描述	**属性名**字段的填写说明。
* 属性类型	**属性名**字段的填写类型，可在下拉框选择，设置成功后不支持修改。
格式	**属性类型的格式。**当**属性类型**选择**文本**时，可在下拉框选择。
是否必填	默认不勾选，即非必填。
是否唯一	当“属性类型”选为“文本”时，可设置是否唯一。勾选后，同步用户数据至应用时，该属性的值具有唯一性，重复时，会提示“{显示标签}”已存在。
是否敏感	当“属性类型”选择文本时才需要设置该参数，勾选后，同步用户数据至应用时，数据隐藏展示，单击可以看到数据内容。

2. 在映射定义页签，单击“编辑”，设置机构属性的映射方式。

说明

- 当映射定义的转换方式选择脚本转换时，脚本编写请参考**如何开发映射脚本**。
- 为了避免同步异常，建议添加的账号属性与需要映射的用户属性类型保持一致。

表 1-5 映射参数

参数	说明
系统用户	映射至**应用账号**的用户属性，可在下拉框选择。
转换方式	**用户**属性与**应用账号**之间的映射方式，可在下拉框选择。
脚本表达式	**转换方式**为**脚本转换**时，填写具体的映射脚本。
执行方式	**用户**属性与**应用账号**之间映射的同步方式，可在下拉框选择。
应用账号	1中定义的账号属性。

3. 在“常规配置”页签，“删除系统用户”默认配置为删除应用账号，“禁用系统用户”默认配置为禁用应用账号。单击“编辑”，可以修改配置，其中“删除系统用户”如果选择禁用或保留应用账号，由于用户已删除，账号会自动变更为孤儿账号；“禁用系统用户”可以改成保留应用账号。修改完成后，单击“保存”即可生效。

----结束

新增、修改、删除用户

本节主要介绍企业管理员在OneAccess管理门户同步用户的方法。

- 新增用户

  选择“授权管理 > 应用账号”，单击“添加账号”，勾选需要同步的账号。如需根据**授权策略**同步账号，请参考**应用账号授权策略**。

- 修改用户

  选择“用户 > 组织与用户”，在用户列表页面，鼠标放置在目标用户名右侧状态栏下方单击，弹出“编辑用户”弹框，更新用户信息。

图 1-2 编辑用户

- 删除用户

  选择“用户 > 组织与用户”，在用户列表页面，单击某一用户后的，单击“删除”删除用户。

  图 1-3 删除用户

新增、修改、删除组织

本节主要介绍企业管理员在OneAccess管理门户同步组织的方法。

说明

同步机构的前提是开启应用机构，请参考**步骤7**。

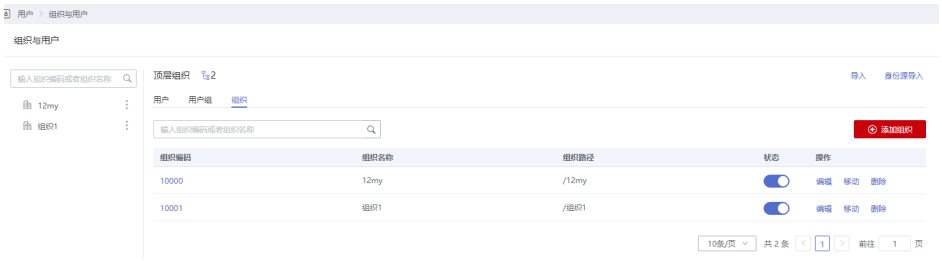
- 新增组织

  选择“授权管理 > 应用机构”，单击“授权策略”，单击开启机构自动授权，按照页面提示勾选需要同步的机构，单击“保存”，单击“执行新增”。

- 修改组织

  选择“用户 > 组织与用户”，在组织列表页面，单击目标组织操作列的“编辑”，更新组织信息。

  图 1-4 编辑组织

- 删除组织

  选择“用户 > 组织与用户”，在组织列表页面，单击目标组织操作列的“删除”，删除组织。

图 1-5 删除组织

同步事件

当OneAccess向下游企业应用同步数据时，同步事件会记录同步的所有操作，方便您进行查看。

1. 登录OneAccess管理门户。
2. 在导航栏中，选择“资源 > 应用”。
3. 在应用页面，单击某应用进入应用信息页面。
4. 单击应用图标，默认进入该应用的通用信息页面。
5. 在左侧导航栏选择“授权管理 > 同步事件”进入同步事件页面。可根据时间、操作类型、对象类型以及同步状态进行筛选查看同步记录。

   说明

   对于同步失败的事件的处理：

   - 可以查看响应信息并在解决问题后，单击操作列的“重试”再次同步。
   - 可以通过“一键重试”快速执行同步。当父机构的同步事件重试成功后，会同时触发执行其下的子机构和账号的同步事件。

全量同步

当下游应用系统需与OneAccess的数据保持全量一致时，可通过全量同步实现。全量同步遵循以下规则：

- 全量同步会忽视之前所有的同步事件，并生成新的同步事件，如果应用系统未返回id，则生成新增事件，如果应用系统已返回id，则生成更新事件。id说明可参考**表1-12**。

  说明

  全量同步触发的更新事件会同步映射定义配置的所有属性，普通的更新事件只同步变更的属性。

- 当依赖的机构同步失败后，其子集包括子机构和账号的所有同步事件状态为挂起，此时，单击应用机构的“开始同步”后，会触发应用账号的全量同步，即应用账号的“开始同步”也处于运行状态。

本节主要介绍企业管理员在管理门户全量同步数据至应用系统的方法。

选择“同步集成 > 全量同步”，单击应用账号或者应用机构的“开始同步”即可。

同步状态说明

本节主要介绍同步完成后，各种状态的含义。方便您对异常状态进行排查定位。

以应用账号App_acc01、应用机构App_org01、原同步事件E0、新同步事件E1为例。

- 待处理

  英文为“PENDING”，当应用账号App_acc01在生成新的同步事件E1时，如果该应用账号App_acc01之前的同步事件E0未执行完成，则生成的新事件E1的状态为待处理。
- 排队

  英文为“QUEUING”，当应用账号App_acc01的同步事件E1可以执行时，会将该事件发送至Kafka执行消息队列，发送成功后，修改ES中存储的同步事件E1的状态为排队，表示该事件已成功加入执行队列。
- 发送

  英文为“RUNNING”，当同步事件E1可执行时，在开始执行前将存储在ES中的同步事件E1的状态修改为RUNNING，修改成功后，调用下游应用系统的接口开始同步，表示该事件已成功处于运行状态。
- 成功

  英文为“SUCCESS”，同步事件执行成功后，会将该事件的状态SUCCESS写入ES，表示该事件已执行成功。
- 失败

  英文为“FAILURE”，同步事件执行失败后，会将该事件的状态FAILURE写入ES，表示该事件已执行失败。
- 忽视

  英文为“IGNORED”，该状态涉及2种场景：
  - 当应用账号 App_acc01 在执行一个更新的同步事件时，如果发现后续有新的更新事件，则直接忽视该事件，并执行新的更新事件。
  - 执行全量同步时，会忽视之前所有的同步事件。
- 略过

  英文为“SKIPPED”，该状态暂时未使用。
- 挂起

英文为“WAITING”，当应用账号App_acc01对应用机构App_org01存在依赖关系时，应用机构App_org01未同步成功，则该机构下应用账号App_acc01的所有同步事件状态为挂起。

1.2 调用说明

1.2.1 接口调用说明

接口格式

OneAccess同步事件回调接口的请求方式为POST，数据编码为UTF-8，数据格式为JSON。 如果应用系统接收事件回调的URL为 https://{app_domain}/callback，当OneAccess的企业发生组织机构或用户变动时，OneAccess将发生变动的业务数据推送至该回调地址。

- URL

  POST https://{app_domain}/callback

- 请求头

  Authorization: Bearer {access_token}，可参考**表1-1**中的**安全令牌**。

- 请求参数

  表 1-6 请求参数

参数	参数类型	描述
nonce	String	随机数，与timestamp结合使用，用于防止请求重放攻击。
timestamp	Integer	时间戳，与nonce结合使用，用于防止请求重放攻击。
eventType	String	事件类型，参见事件类型列表。
data	String	消息体，当未开启加密时，发送明文的消息体；当开启加密时，发送加密的消息体，需要解密得到消息内容，解密后有random、msg二个字段，其中msg即为明文消息内容。
signature	String	消息签名，当未开启签名时，签名信息为空字符串；当开启签名时，生成签名信息，signature计算结合企业应用填写的签名密钥(signatureSaltValue)、请求中的timestamp、nonce、加密的消息体。

- 响应参数

表 1-7 响应参数

参数	参数类型	描述
code	String	返回code码，200表示成功，失败错误码请参考**公共返回码**。
message	String	处理出错时的错误原因说明。
data	String	返回的消息体。不同的业务回调要求返回不同内容，比如回复空串，或者要求的业务数据。 • 当未开启加密时，返回明文的消息体。 • 当开启加密时，返回加密的消息体，需要解密得到消息内容，解密后有random、msg二个字段，其中msg即为明文消息内容。

- 请求示例
  - 未启用消息签名和加密的请求示例：

```
{
    "nonce": "123456",
    "timestamp": 1783610513,
    "eventType": "eventType",
    "data": "明文的消息",
    "signature": ""
}
```

  - 启用消息签名和加密的请求示例：

```
{
    "nonce": "123456",
    "timestamp": 1783610513,
    "eventType": "eventType",
    "data": "1ojvw2WPvW7LijxS8UvISr8pdDP+rXpPbcLGOmIBNbWetRg7IP0vdhkl",
    "signature": "111108bb8e6dbce3c9671d6fdb69d15066227608"
}
```

- 响应示例

  状态码：200

  请求成功

  - 未启用消息签名和加密的响应示例：

```
{
    "code": "200",
    "message": "success",
    "data": "明文的消息"
}
```

  - 启用消息签名和加密的响应示例：

```
{
    "code": "200",
    "message": "success",
    "data": "P+rXpWetRg7IP0vdhVgkVwSoZBJeQwY2zhROsJq/HJ+q6tp1qhl9L1+c"
}
```

1.2.2 签名验签说明

当OneAccess同步数据至企业应用时，需要企业应用对该同步事件进行识别并确认，确保事件来源的安全性和可靠性，从而保证数据在一个安全的环境中进行交互。

签名校验/加解密术语

表 1-8 术语

术语	说明
signature	消息签名，用于验证请求是否来自OneAccess，以防攻击者伪造。签名算法为HMAC-SHA256 + Base64。
AESKey	AES算法的密钥，加密算法为AES/GCM/NoPadding + Base64。
msg	明文消息体，格式为JSON。
encrypt_msg	明文消息msg加密处理并进行Base64编码后的密文。

签名校验

为了让企业应用确认事件推送来自OneAccess，OneAccess将事件推送给企业应用回调服务时，请求body体中包含请求签名并以参数signature标识，企业应用需要验证此参数的正确性后再解密，验证步骤如下：

1. 计算签名。

   计算签名由签名密钥、Nonce值（nonce）、时间戳（timestamp）、事件类型（eventType）、消息体（data）5部分组成，中间使用&进行连接。采用HMAC-SHA256 + Base64算法进行加密。以下为Java语言签名示例：

   ```
   String message = nonce + "&" + timestamp + "&" + eventType + "&" + data;
   Mac mac = Mac.getInstance("HmacSHA256");
   SecretKeySpec secretKey = new SecretKeySpec(签名密钥.getBytes(StandardCharsets.UTF_8),
   "HmacSHA256");
   mac.init(secretKey);
   String newSignature =
   Base64.getEncoder().encodeToString(mac.doFinal(message.getBytes(StandardCharsets.UTF_8)));
   ```

2. 比较计算的签名cal_signature与请求参数signature是否相等，相等则表示验证通过。
3. 企业应用按照要求返回响应消息格式。

明文加密过程

1. 拼接明文字符串。明文字符串由16个字节的随机字符串、明文msg拼接组成，中间使用&进行连接。以下为Java语言示例：

   ```
   String dataStr = RandomStringUtils.random(16, true, false) + "&" + data;
   ```

2. 对拼接后的明文字符串使用AESkey加密后，再进行Base64编码，获得密文encrypt_msg。以下为Java语言示例：

   ```
   Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
   SecretKeySpec secretKey = new SecretKeySpec(加密密钥.getBytes(StandardCharsets.UTF_8), "AES");
   cipher.init(1, secretKey);
   byte[] bytes = dataStr.getBytes(StandardCharsets.UTF_8);
   String ecnryptStr = Base64.getEncoder().encodeToString(cipher.doFinal(bytes));
   ```

密文解密过程

1. 对密文进行BASE64解码。

   ```
   byte[] encryptStr = Base64.getDecoder().decode(data);
   ```

2. 使用AESKey进行解密。

```
Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
SecretKeySpec secretKey = new SecretKeySpec(加密密钥.getBytes(StandardCharsets.UTF_8), "AES");
cipher.init(2, secretKey);
byte[] bytes = cipher.doFinal(encryptStr);
```

3. 去掉rand_msg头部的16个随机字节，剩余的部分即为明文内容msg。

```
String dataStr = StringUtils.split(new String(bytes, StandardCharsets.UTF_8), "&")[1];
```

数据签名&加密解密示例

- 以下为数据签名& AES/GCM/NoPadding算法加密解密的Java语言示例：

```
package com.example.demo.controller;
import com.alibaba.fastjson.JSONObject;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import javax.crypto.Cipher;
import javax.crypto.Mac;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import javax.servlet.http.HttpServletRequest;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.util.Assert;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

@RequestMapping
@Controller
public class GcmController {
    private static final Logger log =
LoggerFactory.getLogger(com.example.demo.controller.GcmController.class);
    private static final String SEPARATOR = "&";

    //region 以下三个参数均由第三方随机生成的32位字符串(英文字母大小写和数字)，正式环境不要使用以下示例
    /**
     * 安全令牌
     */
    private static final String TOKEN = "4JV**********NCE";

    /**
     * 签名密钥
     */
    private static final String SIGN_KEY = "wGt**********Rrs";

    /**
     * 加密密钥
     */
    private static final String ENCRYPTION_KEY = "ZJI**********FQo";
    //endregion

    /**
     * 加密算法
     */
    private static final String HMAC_SHA256 = "HmacSHA256";
    private static final String ENCRYPT_ALGORITHM = "AES/GCM/NoPadding";
    private static final String ENCRYPT_KEY_ALGORITHM = "AES";
    private static final int AES_KEY_SIZE = 128;
    private static final Charset CHARSET = StandardCharsets.UTF_8;

    @ResponseBody
    @RequestMapping({"/gcm/callback"})
    public JSONObject demo(HttpServletRequest httpRequest, @RequestBody String body) {
```



```
        log.info("接受事件回调信息数据，原始报文：" + body);
        JSONObject result = new JSONObject();
        result.put("code", "200");
        result.put("message", "success");
        String authorization = httpRequest.getHeader("Authorization");
        if (!StringUtils.join((Object[]) new String[]{"Bearer ", TOKEN}).equals(authorization)) {
            result.put("code", "401");
            result.put("message", "不合法的请求！");
            printResult(result);
            return result;
        }
        JSONObject request = JSONObject.parseObject(body);
        String signature = request.getString("signature");
        String eventType = request.getString("eventType");
        log.info("事件类型" + eventType);
        String data = request.getString("data");
        if (StringUtils.isNotEmpty(SIGN_KEY))
            try {
                log.info("开始校验签名");
                String message = request.getString("nonce") + "&" + request.getLong("timestamp") + "&"
+ eventType + "&" + data;
                Mac mac = Mac.getInstance(HMAC_SHA256);
                SecretKeySpec secretKey = new SecretKeySpec(SIGN_KEY.getBytes(CHARSET),
HMAC_SHA256);
                mac.init(secretKey);
                String newSignature =
Base64.getEncoder().encodeToString(mac.doFinal(message.getBytes(CHARSET)));
                Assert.isTrue(newSignature.equals(signature), "签名不一致");
                log.info("签名校验成功");
            } catch (Exception e) {
                log.error("签名校验失败", e);
                result.put("code", "401");
                result.put("message", "签名校验失败");
                printResult(result);
                return result;
            }
        if (StringUtils.isNotEmpty(ENCRYPTION_KEY)) {
            log.info("开始解密数据" + data);
            try {
                Cipher cipher = Cipher.getInstance(ENCRYPT_ALGORITHM);
                SecretKeySpec secretKey = new SecretKeySpec(ENCRYPTION_KEY.getBytes(CHARSET),
ENCRYPT_KEY_ALGORITHM);
                byte[] iv = decodeFromBase64(data.substring(0, 24));
                data = data.substring(24);
                cipher.init(2, secretKey, new GCMParameterSpec(AES_KEY_SIZE, iv));
                byte[] bytes = cipher.doFinal(Base64.getDecoder().decode(data));
                data = new String(bytes, CHARSET);
                log.info("解密数据成功，解密后数据：" + data);
            } catch (Exception e) {
                log.error("解密数据失败", e);
                result.put("code", "401");
                result.put("message", "解密数据失败");
                printResult(result);
                return result;
            }
        }
        JSONObject eventData = JSONObject.parseObject(data);
        JSONObject returnData = new JSONObject(1);
        String dataStr = null;
        switch (eventType) {
            case "CREATE_USER":
                // 根据 username  唯一创建下游数据，如果下游数据已存在做更新处理。并返回下游id
                returnData.put("id", eventData.getString("username"));
                break;
            case "CREATE_ORGANIZATION":
                // 根据 code 唯一创建下游数据，如果下游数据已存在做更新处理。并返回下游id
                returnData.put("id", eventData.getString("code"));
                break;
            case "UPDATE_USER":
```

```
            case "UPDATE_ORGANIZATION":
                // 更新数据时，没有修改的字段传值为空
                returnData.put("id", eventData.getString("id"));
                break;
            case "DELETE_ORGANIZATION":
            case "DELETE_USER":
                // 根据 下游业务逻辑进行删除操作。不需要返回字段
                break;
            default:
                result.put("code", "400");
                result.put("message", "不支持的事件类型");
                printResult(result);
                return result;
        }
        if (dataStr == null && returnData.size() > 0)
            dataStr = returnData.toJSONString();
        if (StringUtils.isNotEmpty(ENCRYPTION_KEY) && dataStr != null) {
            String random = RandomStringUtils.random(24, true, true);
            log.info("开始加密数据" + dataStr);
            try {
                Cipher cipher = Cipher.getInstance(ENCRYPT_ALGORITHM);
                SecretKeySpec secretKey = new SecretKeySpec(ENCRYPTION_KEY.getBytes(CHARSET),
ENCRYPT_KEY_ALGORITHM);
                byte[] iv = decodeFromBase64(random);
                cipher.init(1, secretKey, new GCMParameterSpec(AES_KEY_SIZE, iv));
                byte[] bytes = dataStr.getBytes(CHARSET);
                dataStr = Base64.getEncoder().encodeToString(cipher.doFinal(bytes));
                dataStr = random + dataStr;
                log.info("加密数据成功，加密后数据：" + dataStr);
            } catch (Exception e) {
                log.error("加密数据失败", e);
                result.put("code", "500");
                result.put("message", "加密数据失败");
                printResult(result);
                return result;
            }
        }
        result.put("data", dataStr);
        printResult(result);
        return result;
    }

    private static byte[] decodeFromBase64(String data) {
        return Base64.getDecoder().decode(data);
    }

    private void printResult(JSONObject result) {
        log.info("" + result.toJSONString());
    }
}
```

- 以下为数据签名& AES/ECB/PKCS5Padding算法加密解密的Java语言示例：

```
package com.example.demo.controller;
import com.alibaba.fastjson.JSONObject;
import java.nio.charset.Charset;
import java.nio.charset.StandardCharsets;
import java.util.Base64;
import java.util.UUID;
import javax.crypto.Cipher;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.servlet.http.HttpServletRequest;
import org.apache.commons.lang3.RandomStringUtils;
import org.apache.commons.lang3.StringUtils;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.stereotype.Controller;
import org.springframework.util.Assert;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
```



```
import org.springframework.web.bind.annotation.ResponseBody;

@RequestMapping
@Controller
public class DemoController {
  private static final Logger log =
LoggerFactory.getLogger(com.example.demo.controller.DemoController.class);

  /** 签名拼接符 **/
  private static final String SEPARATOR = "&";

  /** 签名密钥 **/
  private static final String SIGN_KEY = "wGt**********Rrs";

  /** 加密密钥 **/
  private static final String ENCRYPTION_KEY = "ZJI**********FQo";

  /** 安全令牌 **/
  private static final String TOKEN = "4JV**********NCE";

  /** 签名算法 **/
  private static final String HMAC_SHA256 = "HmacSHA256";

  /** 加密算法 **/
  private static final String ENCRYPT_ALGORITHM = "AES/ECB/PKCS5Padding";

  /** **/
  private static final String ENCRYPT_KEY_ALGORITHM = "AES";

  /** 字符编码 **/
  private static final Charset CHARSET = StandardCharsets.UTF_8;

  @ResponseBody
  @RequestMapping({"/callback"})
  public JSONObject demo(HttpServletRequest httpRequest, @RequestBody String body) {
    JSONObject result = new JSONObject();
    result.put("code", "200");
    result.put("message", "success");
    log.info("接收事件回调消息数据,原始报文: " + body);
    String authorization = httpRequest.getHeader("Authorization");
    if (!StringUtils.join("Bearer ", TOKEN).equals(authorization)) {
      result.put("code", "401");
      result.put("message", "不合法的请求!");
      printResult(result);
      return result;
    }
    JSONObject request = JSONObject.parseObject(body);
    String signature = request.getString("signature");
    String eventType = request.getString("eventType");
    log.info("事件类型： " + eventType);
    String data = request.getString("data");
    if (StringUtils.isNotEmpty(SIGN_KEY)) {
      try {
        log.info("开始校验签名");
        String message = request.getString("nonce") + SEPARATOR + request.getLong("timestamp") +
SEPARATOR + eventType + SEPARATOR + data;
        Mac mac = Mac.getInstance(HMAC_SHA256);
        SecretKeySpec secretKey = new SecretKeySpec(SIGN_KEY.getBytes(CHARSET), HMAC_SHA256);
        mac.init(secretKey);
        String newSignature =
Base64.getEncoder().encodeToString(mac.doFinal(message.getBytes(CHARSET)));
        Assert.isTrue(newSignature.equals(signature), "签名不一致");
        log.info("签名校验成功");
      } catch (Exception e) {
        log.info("签名校验失败");
        log.error("签名校验失败", e);
        result.put("code", "401");
        result.put("message", "签名校验失败");
        printResult(result);
```



```
      return result;
    }
  }
  if (StringUtils.isNotEmpty(ENCRYPTION_KEY)) {
    log.info("开始解密数据:  " + data);
    try {
      Cipher cipher = Cipher.getInstance(ENCRYPT_ALGORITHM);
      SecretKeySpec secretKey = new SecretKeySpec(ENCRYPTION_KEY.getBytes(CHARSET),
ENCRYPT_KEY_ALGORITHM);
      cipher.init(2, secretKey);
      byte[] bytes = cipher.doFinal(Base64.getDecoder().decode(data));
      data = StringUtils.split(new String(bytes, CHARSET), SEPARATOR)[1];
      log.info("解密数据成功,解密后数据: " + data);
    } catch (Exception e) {
      log.info("解密数据失败");
      log.error("解密数据异常", e);
      result.put("code", "401");
      result.put("message", "解密数据失败");
      printResult(result);
      return result;
    }
  }
  JSONObject eventData = JSONObject.parseObject(data);
  JSONObject returnData = new JSONObject(1);
  String dataStr = null;
  switch (eventType) {
    case "CREATE_USER":
      returnData.put("id", eventData.getString("username"));
      break;
    case "CREATE_ORGANIZATION":
      returnData.put("id", eventData.getString("code"));
      break;
    case "UPDATE_USER":
    case "UPDATE_ORGANIZATION":
      returnData.put("id", eventData.getString("id"));
      break;
    case "DELETE_ORGANIZATION":
    case "DELETE_USER":
      break;
    case "CHECK_URL":
      dataStr = UUID.randomUUID().toString().replaceAll("-", "");
      break;
    default:
      result.put("code", "400");
      result.put("message", "不支持的事件类型");
      printResult(result);
      return result;
  }
  if (dataStr == null && returnData.size() > 0) {
    dataStr = returnData.toJSONString();
  }
  if (StringUtils.isNotEmpty(ENCRYPTION_KEY) && dataStr != null) {
    dataStr = RandomStringUtils.random(16, true, false) + SEPARATOR + dataStr;
    log.info("开始加密数据: " + dataStr);
    try {
      Cipher cipher = Cipher.getInstance(ENCRYPT_ALGORITHM);
      SecretKeySpec secretKey = new SecretKeySpec(ENCRYPTION_KEY.getBytes(CHARSET),
ENCRYPT_KEY_ALGORITHM);
      cipher.init(1, secretKey);
      byte[] bytes = dataStr.getBytes(CHARSET);
      dataStr = Base64.getEncoder().encodeToString(cipher.doFinal(bytes));
      log.info("加密数据成功,加密后数据: " + dataStr);
    } catch (Exception e) {
      log.info("加密数据失败");
      log.error("加密数据异常:", e);
      result.put("code", "500");
      result.put("message", "加密数据失败");
      printResult(result);
      return result;
```



```
    }
  }
  result.put("data", dataStr);
  printResult(result);
  return result;
 }

 private void printResult(JSONObject result) {
  log.info("返回数据为: " + result.toJSONString());
 }
}
```

1.2.3 验证回调地址

如果企业应用接收事件推送的URL为https://{app_domain}/callback。当企业管理员在保存回调配置信息时，OneAccess会发送一条验证事件到填写的URL回调服务。

URL

POST https://{app_domain}/callback

请求头

Authorization: Bearer {access_token}

请求参数

表 1-9 请求参数

参数	参数类型	描述
nonce	String	随机数，与timestamp结合使用，用于防止请求重放攻击。
timestamp	Integer	时间戳，与nonce结合使用，用于防止请求重放攻击。
eventType	String	事件类型，此处事件类型为CHECK_URL。
data	String	消息体，当未开启加密时，发送明文的随机字符串； 当开启加密时，发送加密的随机字符串，需要解密得到消息内容，解密后有random、msg二个字段，其中msg即为明文消息内容。
signature	String	消息签名，signature计算结合企业应用填写的签名密钥(signatureSaltValue)、请求中的timestamp、nonce、加密的消息体。

响应参数

表 1-10 响应参数

参数	参数类型	描述
code	String	返回code码，200表示成功，失败错误码请参考**公共返回码**。
message	String	响应结果说明。
data	String	• 当未开启加密时，返回请求包体中的明文随机字符串。 • 当开启加密时，对请求包体中加密的随机字符串进行解密，返回重新加密的该随机字符串的值，需要解密得到消息内容，解密后有random、msg二个字段，其中msg即为明文消息内容。

请求示例

- 未启用消息签名和加密的请求示例：

```
{
    "nonce": "bqVHvThFGooCRjSf",
    "timestamp": 1573784783795,
    "eventType": "CHECK_URL",
    "data": "random string",
    "signature": ""
}
```

- 启用消息签名和加密的请求示例：

```
{
    "nonce": "jmgjjEAJbrMzWmUw",
    "timestamp": 15093849585,
    "eventType": "CHECK_URL",
    "data": "jRqGWO08Tyuxq+ChqGFk7SiPCt6MgcUDvzP5CBYnD30=",
    "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

响应示例

状态码：200

请求成功

- 未启用消息签名和加密的响应示例：

```
{
    "code": "200",
    "data": " 2852325935078140700",
    "message": "success"
}
```

- 启用消息签名和加密的响应示例：

```
{
    "code": "200",
    "message": "success",
    "data": "u5GkfEdZC0EDvDldLWBK/w=="
}
```

1.3 API

1.3.1 概述

- 同步组织数据的前提是开启应用机构，可参考**步骤7**。
- 本章节的业务事件的消息示例前提是企业应用开启签名和数据加密。

1.3.2 新增组织事件

通过该接口将新增的组织同步至应用系统。

URL

POST https://{app_domain}/callback

请求头

Authorization: Bearer {access_token}

请求参数

以下请求参数以企业实际配置的机构属性为准，企业管理员可以参考**步骤8**设置同步至目标应用的属性。

表 1-11 请求参数

参数	固定参数	参数类型	描述
code	是	String(100)	组织编号，全局唯一。
name	是	String(40)	组织名称，所在层级下名称不能重复。
parentId	否	String(50)	父组织ID。

响应参数

表 1-12 响应参数

参数	参数类型	描述
id	String(50)	• 下游企业应用创建机构成功后，生成的机构ID，并回传至OneAccess，作为该机构的唯一标识。 • 在修改、删除机构时，传递id至下游应用，需下游应用保持一致，如果不一致，后面接口返回的id会覆盖之前的id。

请求示例

- 启用消息签名和加密的请求示例：

```
{
    "nonce": "AmgjjEAJbrMzWmUw",
    "timestamp": 15093849585,
    "eventType": "CREATE_ORGANIZATION",
    "data": "6lu6gxrHydJIXEWxQhUa3UqsWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
    "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

- 请求包体data解密后的JSON字符串格式：

```
{
    "code": "1000003",
    "name": "武汉分公司",
    "parentId": "5b183439-36a8-4d08-94ba-61b3c8d40b66"
}
```

响应示例

状态码：200

请求成功

- 启用消息签名和加密的响应示例：

```
{
    "code": "200",
    "message": "success",
    "data": "j3rRBbc1Q1z1lZM0DDcUGFyaazO3NgnMbgK6UeWT35Druf5zyXg="
}
```

- 响应包体data解密后的JSON字符串格式：

```
{
    "id": "6c5bb468-14b2-4183-baf2-06d523e03bd3"
}
```

1.3.3 修改组织事件

通过该接口将更新的组织同步至应用系统。

URL

POST https://{app_domain}/callback

请求头

Authorization: Bearer {access_token}

请求参数

表 1-13 请求参数

参数	固定参数	参数类型	描述
id	是	String(50)	• 下游企业应用的机构ID。 • 当机构通过**新增组织事件**同步成功后，下游企业应用创建机构并生成机构ID，回传至OneAccess，作为该机构的唯一标识。

参数	固定参数	参数类型	描述
code	是	String(100)	组织编号，全局唯一。
name	是	String(40)	组织名称，所在层级下名称不能重复。
parentId	否	String(50)	父组织ID。

响应参数

表 1-14 响应参数

参数	参数类型	描述
id	String(50)	• 与表1-13中的id一致。 • 当下游企业应用更新机构成功后，回传至OneAccess的机构ID。 • 需下游应用保持一致，如果不一致，后面接口返回的id会覆盖之前的id。

请求示例

- 启用消息签名和加密的请求示例：

```
{
    "nonce": "AmgjjEAJbrMzWmUw",
    "timestamp": 15093849585,
    "eventType": "UPDATE_ORGANIZATION ",
    "data": "6xrHydJIXEWxQhUa3UqsXHWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
    "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

- 请求包体data解密后的JSON字符串格式。

  根据请求的组织机构ID参数更新机构信息，将变更的组织机构属性发送给企业应用。

```
{
    "id": "6c5bb468-14b2-4183-baf2-06d523e03bd3",
    "code": "1000003",
    "name": "武汉分公司",
    "parentId": "5b183439-36a8-4d08-94ba-61b3c8d40b66"
}
```

响应示例

状态码：200

请求成功

- 启用消息签名和加密的响应示例：

```
{
    "code": "200",
    "message": "success",
```

```
    "data": "T41FtX1Q1z1lZM0DDcUGFyaazO3NgnMbgK6UeWT35Druf5zyXg="
}
```

- 响应包体data解密后的JSON字符串格式：

```
{
    "id": "6c5bb468-14b2-4183-baf2-06d523e03bd3"
}
```

1.3.4 删除组织事件

通过该接口将删除的组织同步至应用系统。

URL

POST https://{app_domain}/callback

请求头

Authorization: Bearer {access_token}

请求参数

表 1-15 请求参数

参数	参数类型	描述
id	String	• 下游企业应用的机构ID。 • 当机构通过**新增组织事件**同步成功后，下游企业应用创建机构并生成机构ID，回传至OneAccess，作为该机构的唯一标识。

响应参数

表 1-16 响应参数

参数	参数类型	描述
code	String	返回code码，200表示成功。
message	Integer	处理出错时的错误原因说明。

请求示例

- 启用消息签名和加密的请求示例：

```
{
    "nonce": "AmgjjEAJbrMzWmUw",
    "timestamp": 15093849585,
    "eventType": "DELETE_ORGANIZATION ",
    "data": "6lrHydJIXEWxQhUa3UqsXHWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
    "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

- 请求包体data解密后的JSON字符串格式：

```
{
   "id": "6c5bb468-14b2-4183-baf2-06d523e03bd3"
}
```

响应示例

状态码：200

请求成功

```
{
   "code": "200",
   "message": "success"
}
```

1.3.5 新增用户事件

通过该接口将新增的用户同步至应用系统。

URL

POST https://{app_domain}/callback

请求头

Authorization: Bearer {access_token}

请求参数

以下请求参数以企业实际配置的身份同步参数为准，企业管理员可以参考**步骤9**设置同步至目标应用的属性。

表 1-17 请求参数

参数	固定参数	参数类型	描述
username	是	String(100)	用户名。
name	是	String(40)	用户姓名。
organizationId	是	String	组织机构ID。
password	是	String	密码。
disabled	是	Boolean	是否禁用，true表示禁用状态，fasle表示启用（未禁用）。
firstName	否	String(20)	名字。
middleName	否	String(20)	中间名。
lastName	否	String(20)	姓氏。

参数	固定参数	参数类型	描述
mobile	否	String	手机号。
email	否	String	邮箱地址。
extAttr1	否	--	扩展属性1，企业扩展的用户属性，以实际情况为准。
extAttr2	否	--	扩展属性2，企业扩展的用户属性，以实际情况为准。

响应参数

表 1-18 响应参数

参数	参数类型	描述
id	String(50)	• 下游企业应用创建用户成功后，生成的用户ID，并回传至OneAccess，作为该用户的唯一标识。 • 在修改、删除用户时，传递id至下游应用，需下游应用保持一致，如果不一致，后面接口返回的id会覆盖之前的id。

请求示例

- 启用消息签名和加密的请求示例：

```
{
    "nonce": "AmgjjEAJbrMzWmUw",
    "timestamp": 1509384....,
    "eventType": "CREATE_USER",
    "data": "6lu6gxrdJIXEWxQhUa3UqsXHWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
    "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

- 请求包体data解密后的JSON字符串格式：

```
{
    "username": "zhangsan",
    "name": "张三",
    "mobile": "1899876....",
    "email": "zhangsan@test.com",
    "organizationId": "391551e8-160f-4993-8177-e7b9c5f6....",
    "extAttr1": "value",
    "extAttr2": "value"
}
```

响应示例

状态码：200

请求成功

- 启用消息签名和加密的响应示例：

```
{
    "code": "200",
    "message": "success",
```

```
    "data": "P+rXpWetRg7IP0vdhVgkVwSoZBJeQwY2zhROsJq/HJ+q6tp1qhl9L1+c"
}
```

- 响应包体data解密后的JSON字符串格式：

```
{
    "id": "c3a26dd3-27a0-4dec-a2ac-ce211e10...."
}
```

1.3.6 修改用户事件

通过该接口将更新的用户同步至应用系统。

URL

POST https://{app_domain}/callback

请求头

Authorization: Bearer {access_token}

请求参数

表 1-19 请求参数

参数	固定参数	参数类型	描述
id	是	String(50)	企业应用的用户ID。
username	是	String(100)	用户名。
name	否	String(40)	用户姓名
firstName	否	String(20)	名字。
middleName	否	String(20)	中间名。
lastName	否	String(20)	姓氏。
organizationId	否	String	组织机构ID。
mobile	否	String	手机号。
email	否	String	邮箱地址。
disabled	是	Boolean	是否禁用，true表示禁用状态，fasle表示启用（未禁用）。
extAttr1	否	--	扩展属性1，企业扩展的用户属性，以实际情况为准。

参数	固定参数	参数类型	描述
extAttr2	否	--	扩展属性2，企业扩展的用户属性，以实际情况为准。

响应参数

表 1-20 响应参数

参数	参数类型	描述
id	String(50)	• 与表1-19中的id一致。 • 当下游企业应用更新用户成功后，回传至OneAccess的用户ID。 • 需下游应用保持一致，如果不一致，后面接口返回的id会覆盖之前的id。

请求示例

- 启用消息签名和加密的请求示例：

```
{
    "nonce": "AmgjjEAJbrMzWmUw",
    "timestamp": 15093849585,
    "eventType": "UPDATE_USER",
    "data": "6lu6gxrHydJIXEQhUa3UqsXHWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
    "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

- 请求包体data解密后的JSON字符串格式。

  根据请求的用户ID参数更新用户信息，将变更的用户属性发送给企业应用。

```
{
    "id": "c3a26dd3-27a0-4dec-a2ac-ce211e10....",
    "username": "zhangs",
    "name": "张三2",
    "mobile": "1867237....",
    "email": "454205....@qq.com",
    "extAttr1": "value",
    "extAttr2": "value"
}
```

响应示例

状态码：200

请求成功

- 启用消息签名和加密的响应示例：

```
{
    "code": "200",
    "message": "success",
    "data": "P+rXpWetRg7IP0vdhVgkVwSoZBJeQwY2zhROsJq/HJ+q6tp1qhl9L1+c"
}
```

- 响应包体data解密后的JSON字符串格式：

```
{
    "id": "c3a26dd3-27a0-4dec-a2ac-ce211e105f97"
}
```

1.3.7 删除用户事件

通过该接口将删除的用户同步至应用系统。

URL

POST https://{app_domain}/callback

请求头

Authorization: Bearer {access_token}

请求参数

表 1-21 请求参数

参数	参数类型	描述
id	String	• 下游企业应用的用户ID。 • 当用户通过**新增用户事件**同步成功后，下游企业应用创建用户并生成用户ID，回传至OneAccess，作为该用户的唯一标识。

响应参数

表 1-22 请求参数

参数	参数类型	描述
code	String	返回code码，200表示成功。
message	Integer	处理出错时的错误原因说明。

请求示例

- 启用消息签名和加密的请求示例：

```
{
    "nonce": "AmgjjEAJbrMzWmUw",
    "timestamp": "15093849585",
    "eventType": "DELETE_USER",
    "data": "6IXEWxQhUa3UqsXHWsDZ5LTAo/xU3zhjq9H3syCuFYDYKg==",
    "signature": "K08yDiTEc094KoccOY+VYLQFxxQ="
}
```

- 请求包体data解密后的JSON字符串格式：

```
{
    "id": "c3a26dd3-27a0-4dec-a2ac-ce211e10...."
}
```

响应示例

状态码：200

请求成功

```
{
    "code": "200",
    "message": "success"
}
```

1.4 公共返回码

事件回调接口采用HTTP状态码表示操作结果成功或者失败。企业应用的回调服务参考以下错误码及错误提示进行返回，OneAccess将返回的错误码和错误信息保存到应用同步记录中。

表 1-23 公共返回码

返回码 code	说明
200	success。
400	● 参数XX已存在，如参数userName已存在。 ● 参数XX不能为空，如参数id不能为空。 ● 参数XX超过指定长度，如参数name超过指定的长度。 ● 参数XX格式不符合，如参数email格式不符合。
401	接口鉴权失败。
404	XX记录不存在，如用户不存在。
500	系统繁忙，请稍后重试。

2 如何开发映射脚本

OneAccess支持将企业的组织与用户属性映射至应用系统。可通过映射脚本自动生成应用的属性值，同时，可限制映射至应用的属性值。

下面将为您介绍如何开发映射定义的脚本。

代码规则

在编写映射脚本时，OneAccess对脚本做了一些限制。包括禁止使用Java class、限制使用CPU的时间、限制内存的使用量、限定脚本格式、限制使用部分函数。

- 禁止使用Java class。

  如果使用下面的代码：

  ```
  var File = Java.type('java.io.File'); File;
  ```

  会抛出下面的异常：

  ```
  java.lang.ClassNotFoundException: java.io.File
  ```

- 限制使用CPU的时间。

  默认限制执行时间为1秒，超过时间将抛出异常。

  如果使用下面的代码：

  ```
  do{}while(true);
  ```

  会抛出下面的异常：

  ```
  ScriptCPUAbuseException
  ```

- 限制内存的使用量。

  默认大小为10M，超过将抛出异常。

  如果使用下面的代码：

  ```
  var o={},i=0; while (true) {o[i++] = 'abc'}
  ```

  会抛出下面的异常：

  ```
  ScriptMemoryAbuseException
  ```

- 限定脚本格式。

  为了方便重写脚本，脚本中的if、while、for语句必须使用大括号，否则将出现格式错误。

  如果使用下面的代码：

  ```
  var o={},i=0; while (true) o[i++] = 'abc';
  ```

  会抛出下面的异常：


```
BracesException
```

- 限制使用部分函数。

  如下的函数不允许在代码中使用，如果使用，将没有任何效果。

```
print
echo
quit
exit
readFully
readLine
load
loadWithNewGlobal
```

脚本示例

- 用户属性

  在脚本中可以使用用户(user)这个对象，这个对象中包含了用户的所有属性，具体的属性以属性定义中的属性代码为准。用户属性定义可参考**用户属性定义**，账号属性定义可参考**步骤9**。


用户 > 用户属性定义

用户属性定义

基本信息

字段名称	属性代码
用户名	userName
机构 必填	organizationId
姓名	name
手机号	mobile
邮箱	email
名字	firstName
中间名	middleName
姓氏	lastName
昵称	attrNickName
生日	attrBirthday
性别	attrGender
证件类型	attrIdentityType
证件号码	attrIdentityNumber
国家或地区	attrArea
城市	attrCity

工作信息

字段名称	属性代码
员工id	employeeid
直属上级	attrManagerId
人员类型	attrUserType
入职时间	attrHireDate
工作所在地	attrWorkPlace

- 示例1：映射用户注册时间：

```
var createdAt = user.createdAt;
var date =new Date(createdAt);
date.toISOString();
```

- 示例2：映射用户的手机号，并隐藏中间4位：

```
var mobile = user.mobile;
var result = "";
if(mobile.length == 15) {
  result = mobile.slice(0,7) + "****" + mobile.slice(-4);
}
result;
```

- 示例3：根据用户名生成用户邮箱：

```
var username = user.userName;
username.toLowerCase()+"@huaweicloud.com";
```

- 组织属性

  在脚本中可以使用组织（organization）这个对象，这个对象中包含了组织的所有属性。

  - 示例1：映射组织名称：

```
var orgName = organization.name;
orgName.toString();
```

  - 示例2：映射组织编码：

```
var orgCode = organization.code;
orgCode.toString();
```

  - 示例3：映射组织id：

```
var id= organization.id;
id.toString();
```

- 系统属性

  获取系统属性，如日期。

  示例：映射明天的当天时间：

```
var date =new Date();
date.setDate(date.getDate()+1);
date.toISOString();
```