# 数字身份管控平台（员工版）

# 应用对接指南

腾讯云

版权所有：腾讯云计算（北京）有限责任公司

【版权声明】

©2013-2024 腾讯云版权所有

本文档（含所有文字、数据、图片等内容）完整的著作权归腾讯云计算（北京）有限责任公司单独所有，未经腾讯云事先明确书面许可，任何主体不得以任何形式复制、修改、使用、抄袭、传播本文档全部或部分内容。前述行为构成对腾讯云著作权的侵犯，腾讯云将依法采取措施追究法律责任。

【商标声明】

腾讯云

及其它腾讯云服务相关的商标均为腾讯云计算（北京）有限责任公司及其关联公司所有。本文档涉及的第三方主体的商标，依法由权利人所有。未经腾讯云及有关权利人书面许可，任何主体不得以任何方式对前述商标进行使用、复制、修改、传播、抄录等行为，否则将构成对腾讯云及有关权利人商标权的侵犯，腾讯云将依法采取措施追究法律责任。

【服务声明】

本文档意在向您介绍腾讯云全部或部分产品、服务的当时的相关概况，部分产品、服务的内容可能不时有所调整。
您所购买的腾讯云产品、服务的种类、服务标准等应由您与腾讯云之间的商业合同约定，除非双方另有约定，否则，腾讯云对本文档内容不做任何明示或默示的承诺或保证。

【联系我们】

我们致力于为您提供个性化的售前购买咨询服务，及相应的技术售后服务，任何问题请联系 4009100100或95716。

# 文档目录

应用对接指南
- 概述
- OAuth2 应用对接
  - 概述
  - 创建应用并授权访问
  - API 场景介绍
  - 调用方式
    - 使用认证门户登录
      - 授权码模式
      - 简化模式
    - 获取 Token
      - 授权码模式
      - 密码模式
      - 客户端模式
      - 短信验证码模式
      - 令牌刷新
    - 获取用户信息
      - 获取用户基本信息
      - 获取用户授权应用列表
      - 获取当前用户指定应用的账号列表
    - 注销刷新令牌
    - MFA 认证因子质询
    - 注销
    - 数据结构
- JWT 应用对接
  - 概述
  - 对接操作
- CAS 应用对接
  - 概述
  - 对接操作
- 表单代填应用对接
  - 概述
  - 对接操作
- SAML 应用对接
  - 概述
  - 对接操作
- OIDC 应用对接
  - 概述
  - 创建 OIDC 应用
  - API 场景介绍
  - 调用方式
    - 获取授权码
    - 获取 Token
      - 授权码模式
      - 密码模式
      - 客户端模式
      - 短信验证码模式
      - 令牌刷新
      - 错误码
    - 获取 JWT 公钥
    - 获取 OpenID Provider 配置信息

# 应用对接指南
# 概述

最近更新时间：2023-12-28 11:36:22

## 单点登录方式

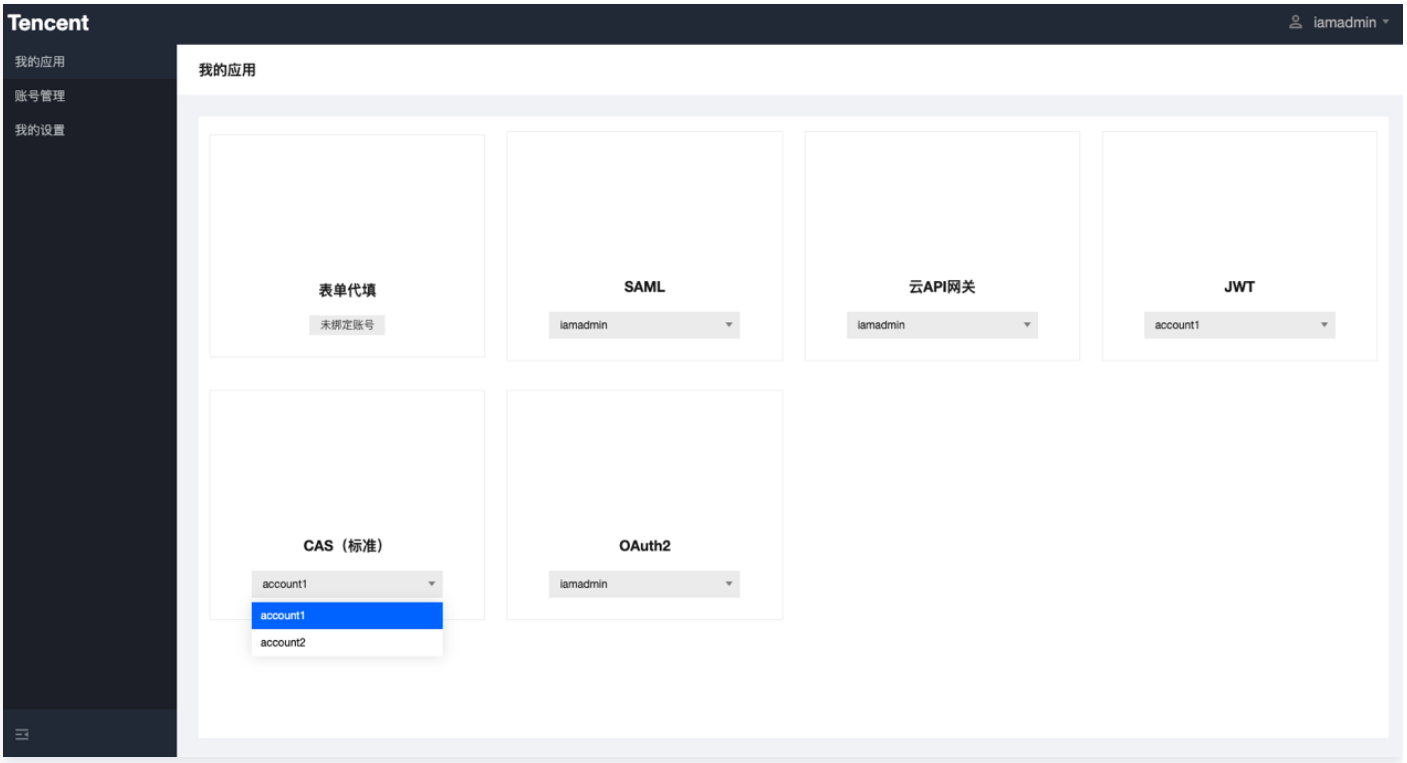
### 从门户应用列表发起登录

#### 运行流程

1. 直接登录认证门户应用列表页面，单击指定应用的图标。
2. 数字身份管控平台（员工版）根据应用类型，生成指定单点登录协议的认证要素（例如 OAuth2 协议的 code、CAS 协议的 ticket），并重定向回应用系统。
3. 应用系统验证认证要素，并完成后续登录流程。

#### 关于应用账号

- 除“表单代填”外的应用系统，当登录用户没有绑定应用账号（映射关系）时，则以用户名（userName）作为应用账号，单点登录到应用系统。
- 对于“表单代填”应用系统，当登录用户没有绑定应用账号（映射关系）时，则不允许进行单点登录，应根据提示绑定应用账号。
- 当登录用户绑定了唯一一个应用账号（映射关系）时，则以唯一的应用账号，单点登录到应用系统。
- 当登录用户绑定了多个应用账号（映射关系）时，用户在应用图标下方的下拉框中，选择指定的登录账号，再以该应用账号单点登录到应用系统。




### 从应用系统发起登录

#### 运行流程

1. 用户访问应用系统。
2. 此时，应用系统不存在认证信息，则带上关键参数重定向到 数字身份管控平台（员工版）。
3. 数字身份管控平台（员工版）对用户进行身份认证。
4. 认证通过后，数字身份管控平台（员工版）根据应用类型，生成指定单点登录协议的认证要素（例如 OAuth2 协议的 code、CAS 协议的 ticket），并重定向回应用系统。
5. 应用系统验证认证要素，并完成后续登录流程。

### 应用类型

支持与标准协议应用对接，包括 OAuth2、JWT、CAS、表单代填、SAML。

# OAuth2 应用对接

# 概述

最近更新时间：2024-11-14 14:50:32

## 协议概述

OAuth（开放授权）是一个开放标准，允许用户授权第三方移动应用访问他们存储在另外的服务提供者上的信息，而不需要将用户名和密码提供给第三方移动应用或分享他们数据的所有内容，OAuth 2.0 是 OAuth 协议的延续版本，但不向后兼容 OAuth 1.0 即完全废止了 OAuth1.0。

## 应用场景

- 第三方应用授权登录：在 App 或者网页接入一些第三方应用时，时常会需要用户登录另一个合作平台，如 QQ，微博，微信的授权登录。
- Native App 授权：App 登录请求后台接口，为了安全认证，所有请求都带 token 信息，如登录验证、请求后台数据。
- 前后端分离单页面应用（SPA）：前后端分离框架，前端请求后台数据，需要进行 OAuth2 安全认证，如使用 vue、react 后者 h5 开发的 App。

## 名词说明

- Third-party application：第三方应用程序，本文中又称“客户端”（client），如打开知乎，使用第三方登录，选择 QQ 登录，这时知乎就是客户端。
- HTTP service：HTTP 服务提供商，本文中简称“服务提供商”，即上例的 QQ。
- Resource Owner：资源所有者，又称“用户”（user），即登录用户。
- User Agent：用户代理，本文中指浏览器。
- Authorization server：认证服务器，即服务提供商专门用来处理认证的服务器。
- Resource server：资源服务器，即服务提供商存放用户生成的资源的服务器。它与认证服务器，可以是同一台服务器，也可以是不同的服务器。

## 运行流程

### 授权码模式（Authorization Code）

功能最完整、流程最严密的授权模式。

```
+--------+                               +---------------+
|        |--(A)- Authorization Request ->|   Resource    |
|        |                               |     Owner     |
|        |<-(B)-- Authorization Grant ---|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(C)-- Authorization Grant -->| Authorization |
| Client |                               |     Server    |
|        |<-(D)----- Access Token -------|               |
|        |                               +---------------+
|        |
|        |                               +---------------+
|        |--(E)----- Access Token ------>|    Resource   |
|        |                               |     Server    |
|        |<-(F)--- Protected Resource ---|               |
+--------+                               +---------------+
```

步骤说明：

1. 用户打开客户端以后，客户端要求用户给予授权。
2. 用户同意给予客户端授权。
3. 客户端使用上一步获得的授权，向认证服务器申请令牌。
4. 认证服务器对客户端进行认证以后，确认无误，同意发放令牌。
5. 客户端使用令牌，向资源服务器申请获取资源。
6. 资源服务器确认令牌无误，同意向客户端开放资源。

### 简化模式（Implicit Grant Type）

不通过第三方应用程序的服务器，直接在浏览器中向认证服务器申请令牌，跳过了“授权码”这个步骤。所有步骤在浏览器中完成，令牌对访问者是可见的，且客户端不需要认证。

```
+----------+
| Resource |
|  Owner   |
|          |
+----------+
     ^
     |
    (B)
+----|-----+          Client Identifier     +---------------+
|         -+----(A)-- & Redirection URI --->|               |
|  User-   |                                | Authorization |
|  Agent  -|----(B)-- User authenticates -->|     Server    |
|          |                                |               |
|          |<---(C)--- Redirection URI ----<|               |
|          |          with Access Token     +---------------+
|          |            in Fragment
|          |                                +---------------+
|          |----(D)--- Redirection URI ---->|   Web-Hosted  |
|          |          without Fragment      |     Client    |
|          |                                |    Resource   |
|     (F)  |<---(E)------- Script ---------<|               |
|          |                                +---------------+
+-|--------+
  |    |
 (A)  (G) Access Token
  |    |
  ^    v
+---------+
|         |
|  Client |
|         |
+---------+
```

步骤说明：

1. 客户端将用户导向认证服务器。
2. 用户决定是否给予客户端授权。
3. 假设用户给予授权，认证服务器将用户导向客户端指定的“重定向URI”，并在 URI 的 Hash 部分包含了访问令牌。
4. 浏览器向资源服务器发出请求，其中不包括上一步收到的 Hash 值。
5. 资源服务器返回一个网页，其中包含的代码可以获取 Hash 值中的令牌。
6. 浏览器执行上一步获得的脚本，提取出令牌。
7. 浏览器将令牌发送给客户端。

### 密码模式（Resource Owner Password Credentials Grant）

用户向客户端提供自己的用户名和密码。客户端使用这些信息，向“服务商提供商”索要授权。在这种模式中，用户必须把自己的密码给客户端，但是客户端不得储存密码。这通常用在用户对客户端高度信任的情况下。一般不支持 refresh token。

```
+----------+
| Resource |
|  Owner   |
|          |
+----------+
     v
     |    Resource Owner
    (A) Password Credentials
     |
     v
+---------+                                  +---------------+
|         |>--(B)---- Resource Owner -------->|               |
|         |         Password Credentials     | Authorization |
| Client  |                                  |     Server    |
|         |<--(C)---- Access Token ---------<|               |
|         |    (w/ Optional Refresh Token)   |               |
+---------+                                  +---------------+
```

步骤说明：

1. 用户向客户端提供用户名和密码。
2. 客户端将用户名和密码发给认证服务器，向后者请求令牌。
3. 认证服务器确认无误后，向客户端提供访问令牌。

### 客户端模式（Client Credentials Grant）

指客户端以自己的名义，而不是以用户的名义，向“服务提供商”进行认证。严格地说，客户端模式并不属于 OAuth 框架所要解决的问题。在这种模式中，用户直接向客户端注册，客户端以自己的名义要求“服务提供商”提供服务，其实不存在授权问题。

```
+---------+                                  +---------------+
|         |                                  |               |
|         |>--(A)- Client Authentication --->| Authorization |
| Client  |                                  |     Server    |
|         |<--(B)---- Access Token ---------<|               |
|         |                                  |               |
+---------+                                  +---------------+
```

步骤如下：

1. 客户端向认证服务器进行身份认证，并要求一个访问令牌。
2. 认证服务器确认无误后，向客户端提供访问令牌。

## 数字身份管控平台（员工版）功能概述

### 完整地实现 OAuth2 的四种模式

创建 OAuth2 应用时指定的 grantType，只针对从数字身份管控平台（员工版）门户发起登录的场景，grantType 设置为 authorization_code 时，授权成功后返回 code；grantType 设置为 implicit 时，授权成功后返回 access_token。
对于从应用系统发起登录的场景，四种模式都可以支持，应用发起登录重定向或接口调用时，根据实际情况传入各模式对应的 response_type 参数或 grant_type 参数即可。

### 支持令牌刷新

提供令牌刷新接口，应用使用 refresh_token 换取新的 access_token。

### 支持PKCE模式

授权码模式下，支持 PKCE 校验码验证。PKCE 模式适用于功能逻辑主要在客户端完成的 Native App。因为 Native App 没有浏览器那样的 cookie 支持，CP 服务器没有 session 这样的东西来保存 state 参数从而防止 CSRF，所以使用 PKCE 的 code_verifier 和 code_challenge 来防止 CSRF。
步骤如下：
这里主要考虑到的是，Native App 必须经过浏览器才能进行 redirect 操作，在下图的第③步中，由于使用了 App 的 custom schemer，如果有恶意 App

注册了同一个 schemer，则不能保证第③步的重定向会回到您的 App。

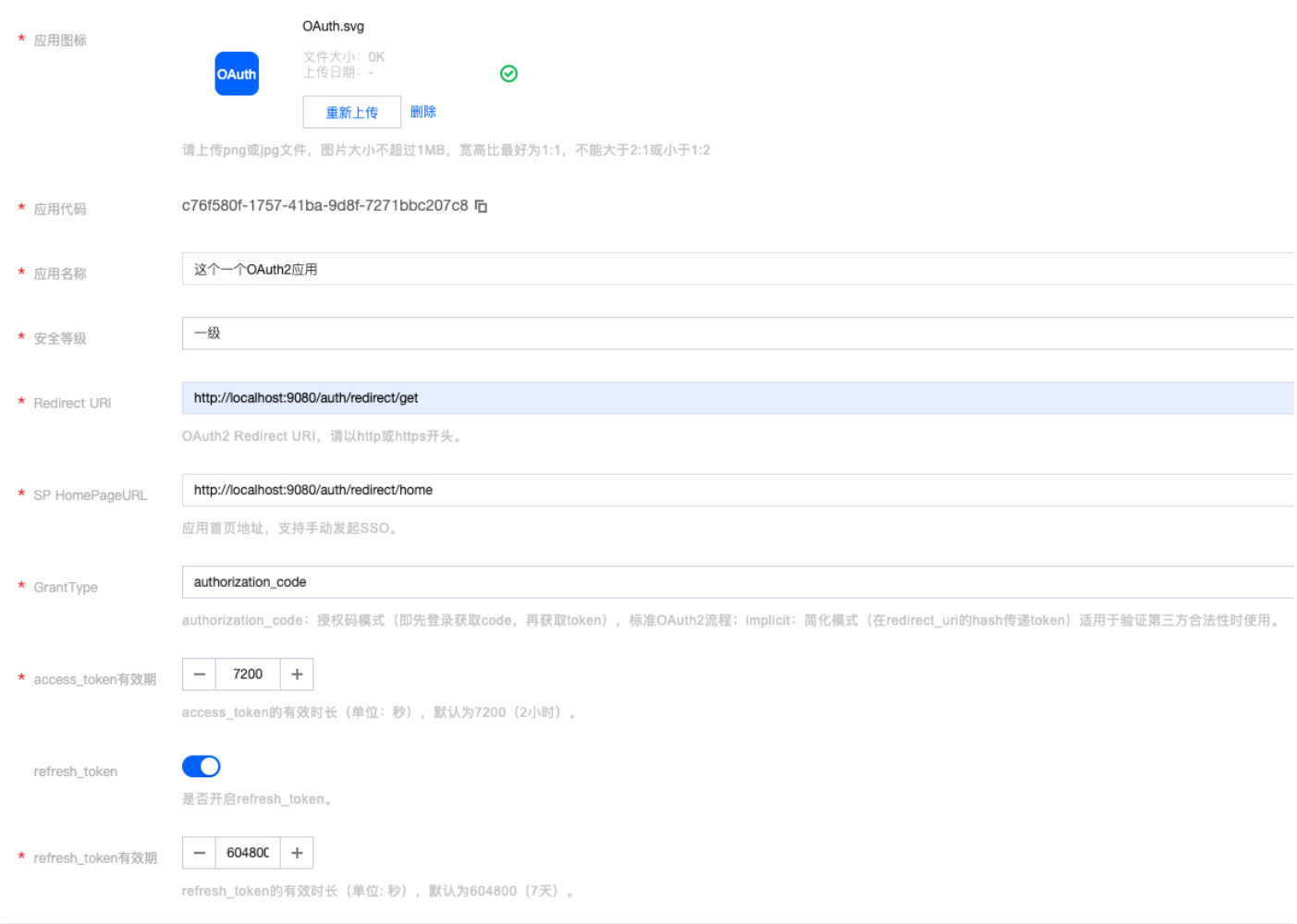
# 创建应用并授权访问

最近更新时间：2022-03-03 09:38:23

## 创建应用

1. 登录 数字身份管控平台（员工版）控制台，在左侧导航栏，单击**应用管理**。
2. 在应用管理页面，单击**应用协议新建**，选择应用协议模板**OAuth2**，单击**下一步：配置参数信息**。

3. 在应用协议新建页面，登记接入应用的相关信息，如下图所示：




**参数说明：**

- Redirect URI：为应用系统的重定向地址，用于接收数字身份管控平台（员工版）以重定向的方式返回的授权码或 access_token 等参数。
- SP HomePageURL：应用系统的登录首页地址。
- GrantType：应用系统获取用户授权的方式，支持 authorization_code、implicit 两种模式；该配置只针对从数字身份管控平台（员工版）门户发起登录的场景，grantType 设置为authorization_code 时，授权成功后返回 code，grantType 设置为 implicit 时，授权成功后返回 access_token。

- access_token 有效期：access_token 的有效时长（单位：秒），默认为7200（2小时）。
- refresh_token：是否开启 refresh_token，该配置只对授权码模式和密码模式有效；如使用其他模式，即使将该配置设置为“开启”，也默认不返回 refresh_token。
- refresh_token 有效期：refresh_token 的有效时长（单位: 秒），默认为604800（7天）。

4. 配置完成后，单击**下一步：完成**，即可创建应用。

## 授权访问

具体操作请参见 授权管理 。

# API 场景介绍

最近更新时间：2021-10-11 16:57:20

本文将介绍数字身份管控平台（员工版）(以下简称 EIAM) 为 OAuth2 应用系统提供的 API 。通过调用这些 API ，您的应用可以完成与用户身份相关的各类操作，例如：用户认证、用户信息的获取，应用的访问控制等。
这里将具体给出应用系统实现如下几种典型业务场景所需的主要工作，不同业务场景可能调用相同的接口，但请求参数和请求响应一般有所不同。

> **注意**
> 在应用接入前，您需要首先在 应用管理页面 完成 OAuth2 协议类型应用的新建与配置，获取应用的 Client ID 、 Client Secret 等配置信息。

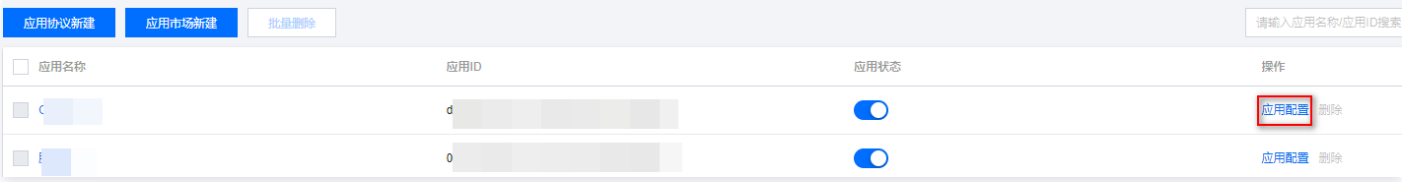
## 前提条件

已完成 创建应用并授权访问 。

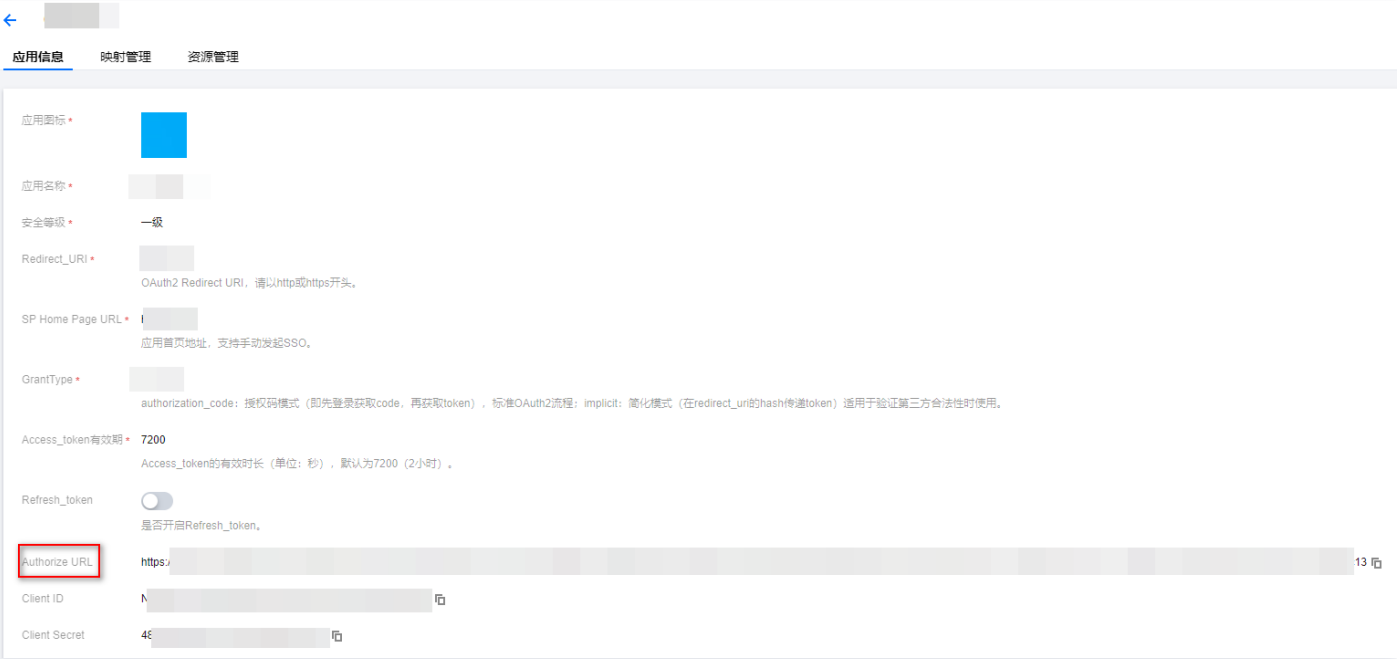
## 单点登录

对于应用实现用户认证的功能，EIAM 提供通过认证门户登录并直接单点登录至应用的能力。基于OAuth2 协议，提供授权码模式、PKCE 模式、简化模式等多种认证授权方式。下面以授权码模式为例展示应用实现用户认证的流程。

1. 登录 数字身份管控平台（员工版）控制台，在左侧导航栏，单击**应用管理**。
2. 在应用管理页面，选择已创建的 OAuth2 应用，单击**应用配置**。




3. 在应用信息页面，获取 Authorize URL，此 URL 为用户在应用系统登录时，应用将用户重定向的认证地址，详情请参见 授权码模式 。




4. 使用该地址首次访问会跳转到登录页面进行用户身份认证，输入账号密码后单击**登录**，认证通过后会重定向到指定地址(redirect_uri)，并在 URL 中携带 code 参数。

5. 使用应用回调地址，接收到回调请求后，可以通过调用 获取 Token 接口，来获取接口中的 access_token 和 refresh_token。
6. 应用建立用户的登录态，将 access_token 和 refresh_token 与登录态关联起来，完成登录。

> **说明**
> 应用系统可自行选择合适的方法建立并保持登录态，例如：Web 应用可以使用 session cookie ，单页应用可以使用 localStorage 或 sessionStorage 。

## 短信验证码二次认证

针对在 EIAM 控制台开启了二次认证功能的情况下，发起的密码登录场景。

> **说明**
> 二次认证功能在 安全设置页面 中开启。

1. 登录 数字身份管控平台（员工版）控制台，在左侧导航栏，单击**应用管理**。
2. 调用 密码模式，触发二次认证将会返回 `mfa_required` 的错误码并返回 `mfa_token` 。
3. 调用 MFA 认证因子质询 接口，输入 `mfa_token` ，该接口会发送短信验证码，并返回 oob 码。
4. 调用 oob 模式的 短信验证码模式，传入 oob 码和短信验证码，验证通过后返回 access_token和 refresh_token。
5. 应用建立用户的登录态，将 access_token 和 refresh_token 与登录态关联起来，完成登录。

> **说明**
> 应用系统可自行选择合适的方法建立并保持登录态，例如：Web 应用可以使用 session cookie ，单页应用可以使用 localStorage 或 sessionStorage 。

## 访问用户资源

在完成用户认证后，应用可以使用 access_token 访问用户资源，例如获取用户基本信息。

1. 假设用户在使用应用系统时，系统需要将用户的账号信息显示在界面上。此时，需要调用相应的接口来获取该信息。
2. 应用系统取出当前登录用户的 access_token ，携带 Token 调用 获取用户信息 接口。
3. 如果接口返回了 HTTP 401 错误码，代表 access_token 已过期，需要使用 refresh_token 调用 令牌刷新接口，获取新的 access_token ，再次调用获取用户信息接口。
4. 应用解析接口返回的用户账号信息，并显示给用户。提供如下获取当前登录用户相关信息的 API：

| 接口功能 | 描述 |
| --- | --- |
| 获取用户基本信息 | 使用合法的授权令牌，获取用户基本信息。 |
| 获取用户授权应用列表 | 使用合法的授权令牌，获取用户有权限访问的应用系统列表。 |
| 获取当前用户指定应用的账号列表 | 使用合法的授权令牌，获取当前用户指定应用的账号列表。 |

## 其他 API

1. 如果 refresh token 存在泄漏的可能，可调用 注销刷新令牌 接口。
2. 用户在应用系统进行注销操作时，可调用 注销接口 。

> **注意**
> 应用系统需要自行清除用户在应用的登录态。

# 调用方式
# 使用认证门户登录
# 授权码模式

最近更新时间：2022-03-23 18:05:14

## 接口描述

获取用户授权，如果用户未登录 EIAM 系统，则要求用户进行身份认证，身份认证成功后返回授权码。接口支持 PKCE 模式，如需进行 PKCE 模式的校验，可传入相应的参数。

## 请求方法

GET

## 请求地址

```
/auth/oauth2/authorize
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| client_id | Query | String | 是 | 客户端 ID，可参考 应用管理页面 >**选定指定应用**> 单击**应用配置**>对应的“Client Id” |
| response_type | Query | String | 是 | 固定值“code” |
| redirect_uri | Query | String | 否 | 授权成功后的重定向地址，如该参数为空，则使用创建应用时设置的“Redirect Uri”作为重定向地址，用于接收 cod e等参数。含特殊字符，需要做 URLEncode。 |
| state | Query | String | 推荐 | 应用系统提供的一个随机字符串，服务器会原样重定向给应用系统，它既能防止CSRF、XSRF， 同时也可以用来对应 SP 初始发起的状态 |
| code_challenge_method | Query | String | 否 | PKCE 模式的必要参数，即 code_challenge 值的计算方法，目前仅支持 SM3，表示 SM3 摘要算法 |
| code_challenge | Query | String | 否 | PKCE模式的必要参数，计算方式：code_challenge = code_challenge_method(code_verifier)。code_verifier 为校验码原文，应用系统需要自行保存 code_verifier，并在后续“获取授权令牌”的请求中带上，用作验证 |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| code | Query | String | 授权码，10分钟内有效 |
| state | Query | String | 应用系统提供的一个随机字符串，服务器会原样重定向给应用系统，它既能防止 CSRF、XSRF， 同时也可以用来对应 SP 初始发起的状态。当入参未传入 state 时，则无此项出参 |

## 返回方式

重定向

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/oauth2/authorize?
client_id=ODQyNGJlYmF****NDZiYWE4YjkwNjU4MzMxOThkMGU&response_type=code&redirect_uri=http%3A%2F%2Flocalhos
t%3A9080%2Fauth%2Fredirect%2Fget&state=baa07350-e2b9-****-a8ae-350325288049
```

**返回示例**

```
http://localhost:9080/auth/redirect/get?code=b9eb0dc2338824****24993e0aca6180&state=baa07350-e2b9-****-
a8ae-350325288049
```

# 简化模式

最近更新时间：2022-03-23 18:05:18

## 接口描述

获取用户授权，如果用户未登录 EIAM 系统，则用户需要进行身份认证，身份认证成功后返回授权令牌。

## 请求方法

GET

## 请求地址

```
/auth/oauth2/authorize
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| client_id | Query | String | 是 | 客户端 ID，可参考 应用管理页面 > 选定指定应用 > 单击应用配置 > 对应的“Client Id” |
| response_type | Query | String | 是 | 固定值“token” |
| redirect_uri | Query | String | 否 | 授权成功后的重定向地址，如该参数为空，则使用创建应用时设置的“Redirect Uri”作为重定向地址，用于接收 access_token 等参数。含特殊字符，需要做 URLEncode |
| state | Query | String | 推荐 | 应用系统提供的一个随机字符串，服务器会原样重定向给应用系统，它既能防止CSRF、XSRF， 同时也可以用来对应 SP 初始发起的状态 |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| access_token | Query | String | access_token |
| expires_in | Query | Long | access_token 有效时间，与创建应用时设置的“access_token有效期”一致。 |
| state | Query | String | 应用系统提供的一个随机字符串，服务器会原样重定向给应用系统，它既能防止 CSRF、XSRF， 同时也可以用来对应 SP 初始发起的状态。当入参未传入state 时，则无此项出参 |

## 返回方式

重定向

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/oauth2/authorize?
client_id=ODQyNGJlYmFjY****ZiYWE4YjkwNjU4MzMxOThkMGU&response_type=token&redirect_uri=http%3A%2F%2Flocalho
st%3A9080%2Fauth%2Fredirect%2Fget&state=baa07350-e2b9-****-a8ae-350325288049
```

### 返回示例

```
http://localhost:9080/auth/redirect/get?
access_token=c6a40b185c8e350****9509b541&expires_in=7200&state=baa07350-e2b9-****-a8ae-350325288049
```

## 错误码

| httpStatus | Response |
| --- | --- |
| 400 | {"error": "invalid_request", "error_description": "invalid response_type"} |
| 400 | {"error": "invalid_request", "error_description": "missing subject_token parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing client_id parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing response_type parameter"} |
| 403 | {"error": "unauthorized_client", "error_description": "invalid client"} |
| 500 | {"error": "internal_server_error", "error_description": "generate access token fail"} |
| 500 | {"error": "internal_server_error", "error_description": "generate authorization code fail"} |

版权所有：腾讯云计算（北京）有限责任公司

# 获取 Token
# 授权码模式

最近更新时间：2022-03-23 18:05:22

## 接口描述

使用授权码（code）换取授权令牌（access_token）。

## 请求方法

POST

## 请求地址

```
/auth/oauth2/token
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| client_id | Query | String | 是 | 客户端 ID，可参考 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Id” |
| grant_type | Query | String | 是 | 固定值“authorization_code” |
| redirect_uri | Query | String | 否 | 授权成功后的重定向地址。如果“用户授权/authorize”请求带有 redirect_uri 参数，则本步骤请求必须上送 redirect_uri 参数，EIAM 会校验该请求参数值与“用户授权/authorize”上送的 redirect_uri 参数值是否一致。含特殊字符，需要做 URLEncode |
| code | Query | String | 是 | 授权码 |
| client_secret | Query | String | 否 | 客户端 Secret， 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Secret”。含有特殊字符，需要做 URLEncode。如 code_verifier 为空，则 client_secret 必须上送。 |
| code_verifier | Query | String | 否 | PKCE 模式必要参数，code_verifier 校验码原文。可能含有特殊字符，需要做 URLEncode |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| access_token | Body | String | access_token |
| expires_in | Body | Long | access_token 有效时间，与创建应用时设置的“access_token有效期”一致 |
| refresh_token | Body | String | refresh_token |

## 返回方式

重定向

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/oauth2/token?
client_id=ODQyNGJlYmFjY****WE4YjkwNjU4MzMxOThkMGU&grant_type=authorization_code&redirect_uri=http%3A%2F%2F
```

```
localhost%3A9080%2Fauth%2Fredirect%2Fget&code=b9eb0dc2****4993e0aca6180&client_secret=oG412Uk6EdbfXtgU****
UdJht1j%2bq&code_verifier=s9hw2n
```

**返回示例**

```
{
    "access_token": "c6a40b1****058eaed0f59509b541",
    "expires_in": 7200,
    "refresh_token": "6865e92a****01df8081e4562e26c82"
}
```

# 密码模式

最近更新时间：2024-11-14 14:50:32

## 接口描述

校验用户名、密码，验证通过后颁发授权令牌。

## 请求方法

POST

## 请求地址

```
/auth/oauth2/token
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| client_id | Query | String | 是 | 客户端 ID，可参考 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Id” |
| grant_type | Query | String | 是 | 固定值“password” |
| client_secret | Query | String | 是 | 客户端 Secret， 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Secret”。含有特殊字符，需要做 URLEncode |
| username | Query | String | 是 | 用于登录 EIAM 的用户名。可能含有特殊字符，需要做 URLEncode |
| password | Query | String | 是 | 用户名对应的密码，必须输入明文。可能含有特殊字符，需要做 URLEncode |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| access_token | Body | String | access_token |
| expires_in | Body | Long | access_token 有效时间，与创建应用时设置的“access_token有效期”一致 |
| refresh_token | Body | String | 固定为空，该模式不提供 refresh_token |

## 返回方式

重定向

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/oauth2/token?
client_id=ODQyNGJlYmFjYWFmND****wNjU4MzMxOThkMGU&grant_type=password&client_secret=oG412Uk6EdbfXtgUW****t1
j%2bq&username=testuser&password=PPee3****lk!!
```

### 返回示例

```
{
    "access_token": "c6axxxxx5c8exxxxxxxxd0fxxxxxxxxx",
    "expires_in": 7200,
    "refresh_token": ""
```

```
}
```

在二次认证功能启用的情况下，将返回。

```
{
    "error": "mfa_required",
    "error_description":"mfa required",
    "mfa_token": "xxxxx"
}
```

# 客户端模式

最近更新时间：2022-03-24 09:27:20

## 接口描述

校验 client_id、client_secret，校验成功后返回授权令牌。

## 请求方法

POST

## 请求地址

```
/auth/oauth2/token
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| client_id | Query | String | 是 | 客户端 ID，可参考 应用管理页面 > 选定指定应用 > 单击应用配置 > 对应的“Client Id” |
| grant_type | Query | String | 是 | 固定值“client_credentials” |
| client_secret | Query | String | 是 | 客户端 Secret， 应用管理页面 > 选定指定应用 > 单击应用配置 > 对应的“Client Secret”。含有特殊字符，需要做 URLEncode |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| access_token | Body | String | access_token |
| expires_in | Body | Long | access_token 有效时间，与创建应用时设置的“access_token有效期”一致 |
| refresh_token | Body | String | 固定为空，该模式不提供 refresh_token |

## 返回方式

重定向

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/oauth2/token?
client_id=ODQyNGJlYmFjYWF****4YjkwNjU4MzMxOThkMGU&grant_type=client_credentials&client_secret=oG412Uk6Edbf
XtgUW****1j%256
```

### 返回示例

```
{
    "access_token": "c6a40b185c8e3****d0f59509b541",
    "expires_in": 7200,
    "refresh_token": "6865e92a99225****1e4562e26c82"
}
```

# 短信验证码模式

最近更新时间：2022-03-23 18:05:36

## 接口描述

验证授权码、短信验证码，验证通过后颁发授权令牌。

## 请求方法

POST

## 请求地址

```
/auth/oauth2/token
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| client_id | Query | String | 是 | 客户端 ID，可参考 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Id” |
| grant_type | Query | String | 是 | 固定值“mfa_oob” |
| code | Query | String | 是 | 授权码，此处传入 challenge 接口返回的 oob授权码 |
| client_secret | Query | String | 是 | 客户端 Secret， 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Secret”。含有特殊字符，需要做URLEncode。 |
| mfa_token | Query | String | 是 | mfa 令牌，密码模式的 token 接口返回 mfa_required 错误时会同时返回 mfa_token |
| binding_code | Query | String | 是 | 短信验证码 |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| access_token | Body | String | access_token |
| expires_in | Body | Long | access_token 有效时间，与创建应用时设置的“access_token有效期”一致 |
| refresh_token | Body | String | refresh_token |

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/oauth2/token?
client_id=ODQyNGJlYmFjYWF****ZiYWE4YjkwNjU4MzMxOThkMGU&grant_type=mfa_oob&code=test0dc233****224993e0aca61
80&client_secret=test2Uk6E****WlwJfvWUdJht1j%2bq&binding_code=123456&mfa_token=3dgdfg3****2266ffg0acaghdtr
```

### 返回示例

```
{
    "access_token": "c6a40b185c****58eaed0f59509b541",
    "expires_in": 7200,
    "refresh_token": "6865e92a9****8081e4562e26c82"
}
```

# 令牌刷新

最近更新时间：2022-03-23 18:05:41

## 接口描述

使用 refresh_token 换取新的 access_token，刷新成功后原来的 access_token 和 refresh_token 将失效。

## 请求方法

POST

## 请求地址

```
/auth/oauth2/token
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| client_id | Query | String | 是 | 客户端 ID，可参考 应用管理页面 > 选定指定应用 > 单击应用配置 > 对应的“Client Id” |
| grant_type | Query | String | 是 | 固定值“refresh_token”。 |
| client_secret | Query | String | 是 | 客户端 Secret， 应用管理页面 > 选定指定应用 > 单击应用配置 > 对应的“Client Secret”。含有特殊字符，需要做 URLEncode。 |
| refresh_token | Query | String | 是 | refresh_token |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| access_token | Body | String | 新的 access_token |
| expires_in | Body | Long | access_token 有效时间，与创建应用时设置的“access_token有效期”一致 |
| refresh_token | Body | String | 新的 refresh_token |

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/oauth2/token?
client_id=ODQyNGJlYmFjYWFm****WE4YjkwNjU4MzMxOThkMGU&grant_type=refresh_token&client_secret=oG412Uk6****gU
WlwJfvWUdJht1j%2bq&refresh_token=a6a40b185c5****58eaed0f59509b567
```

### 返回示例

```
{
    "access_token": "c6a40b1****e35058eaed0f59509b541",
    "expires_in": 7200,
    "refresh_token": "6865e****2501df8081e4562e26c82"
}
```

## 错误码

| httpStatus | Response |
| --- | --- |
| 400 | {"error": "invalid_request", "error_description": "invalid code"} |
| 400 | {"error": "invalid_request", "error_description": "invalid client"} |
| 400 | {"error": "invalid_request", "error_description": "invalid redirect_uri"} |
| 400 | {"error": "invalid_request", "error_description": "invalid code challenge"} |
| 400 | {"error": "invalid_request", "error_description": "invalid grant_type"} |
| 400 | {"error": "invalid_request", "error_description": "missing username parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing password parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing suject_token parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing suject_token_type parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing refresh_token parameter"} |
| 403 | {"error": "unauthorized_client", "error_description": "invalid user password"} |

版权所有：腾讯云计算（北京）有限责任公司

# 获取用户信息
# 获取用户基本信息

最近更新时间：2022-03-23 18:05:48

## 接口描述

使用合法的授权令牌，获取用户基本信息。

## 请求方法

GET

## 请求地址

```
/auth/oauth2/userinfo
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| access_token | Header | String | 是 | access_token，在 HTTP 头部 Authorization 属性中传入的 Bearer 令牌 |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| sub | Body | String | 用户 id |
| ou_id | Body | String | 用户所属组织机构 id |
| phone_number | Body | String | 带有国家号前缀的用户手机号，例如+86-150****8888 |
| email | Body | String | 邮箱地址。当用户未录入邮箱信息时，该字段为空 |
| username | Body | String | 用户名 |
| nickname | Body | String | 用户昵称。当用户未录入昵称信息时，该字段为空 |

## 接口示例

### 输入示例

```
Authorization:Bearer 566bbf829e381a****dbd45e0699bd3f
https://<auth_domain>/auth/oauth2/userinfo
```

### 返回示例

```
{
    "sub": "10b99f39-8243-****-956f-f6baceb4cb94",
    "ou_id": "32a97f3a-7ca3-****-871f-f7becea3ca63",
    "phone_number": "+86-150****8888",
    "email": "150****8888@qq.com",
    "username": "testuser",
    "nickname": "testuser"
}
```

版权所有：腾讯云计算（北京）有限责任公司

# 获取用户授权应用列表

最近更新时间：2023-12-28 11:36:22

## 接口描述

使用合法的授权令牌，获取用户有权限访问的应用系统列表。

## 请求方法

GET

## 请求地址

```
/auth/oauth2/applist
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| access_token | Header | String | 是 | access_token，在 HTTP 头部 Authorization 属性中传入的 Bearer 令牌 |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| data | Body | List < Object > | - |
| -app_name | Body | String | 应用名称 |
| -app_type | Body | String | 应用类型 |
| -app_login_url | Body | String | 应用登录地址。在用户已登录的情况下，在该地址后面追加 accountName 参数，以指定登录应用的账号，再访问该 URL，即可实现单点登录 |
| -account_list | Body | List< String > | 登录用户对应的应用账号列表 |
| -app_id | Body | String | 应用 id |

## 接口示例

### 输入示例

```
Authorization:Bearer 566bbf****1a0ddbdbd45e0699bd3f
https://<auth_domain>/auth/oauth2/applist
```

### 返回示例

```
{
    "data": [
        {
            "app_name": "网关",
            "app_login_url": "https://t1.**.com:8002/auth/sso/ssoLogin?service=119b197e-0981-****-8f5b-
43b7fbecb41f",
            "app_type": "API",
            "account_list": [
                "testuser"
            ],
            "app_id": "119b197e-0981-****-8f5b-43b7fbecb41f"
```

```
        },
        {
            "app_name": "SAML",
            "app_login_url": "https://t1.**.com:8002/auth/sso/ssoLogin?service=d9131abc-f882-****-8615-
0fcb8df36271",
            "app_type": "SAML2",
            "account_list": [
                "testuser"
            ],
            "app_id": "d9131abc-****-48ba-8615-0fcb8df36271"
        },
        {
            "app_name": "JWT",
            "app_login_url": "https://t1.**.com:8002/auth/sso/ssoLogin?service=012ec090-****-4b31-b7f0-
f47aeaf55116",
            "app_type": "JWT",
            "account_list": [
                "testuser"
            ],
            "app_id": "012ec090-****-4b31-b7f0-f47aeaf55116"
        },
        {
            "app_name": "表单代填",
            "app_login_url": "https://t1.**.com:8002/auth/sso/ssoLogin?service=4b4f55f2-****-4dd7-9437-
9a0dfe3a688e",
            "app_type": "FORM",
            "account_list": [],
            "app_id": "4b4f55f2-****-4dd7-9437-9a0dfe3a688e"
        },
        {
            "app_name": "CAS（标准）",
            "app_login_url": "https://t1.**.com:8002/auth/sso/ssoLogin?service=16c00180-****-4c49-9e61-
605e8ad51459",
            "app_type": "CAS",
            "account_list": [
                "testuser"
            ],
            "app_id": "16c00180-****-4c49-9e61-605e8ad51459"
        },
        {
            "app_name": "OAuth2",
            "app_login_url": "https://t1.**.com:8002/auth/sso/ssoLogin?service=c4a37f27-****-4bd4-9111-
504868c234a7",
            "app_type": "OAUTH2",
            "account_list": [
                "testuser"
            ],
            "app_id": "c4a37f27-****-4bd4-9111-504868c234a7"
        }
    ]
}
```

# 获取当前用户指定应用的账号列表

最近更新时间：2022-03-23 18:05:57

## 接口描述

使用合法的授权令牌，获取当前用户指定应用的账号列表。

## 请求方法

GET

## 请求地址

```
/auth/account/list
```

## 对外开放

是

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| access_token | Header | String | 是 | access_token，在 HTTP 头部Authorization 属性中传入的 Bearer 令牌 |
| application_id | Query | String | 是 | 应用 ID，是应用的全局唯一标识 |

## 返回参数

| 参数 | 参数类型 | 描述 |
| --- | --- | --- |
| app_info | application_info | 应用信息 |
| account_list | Array of String | 用户账号名称列表 |

## 接口示例

### 输入示例

```
Authorization:Bearer 566bbf8*******a0ddbdbd45e0699bd3f
https://<auth_domain>/auth/account/list?application_id=10b99f39-8243-4900-956f-f6baceb4cb94
```

### 返回示例

```
{
    "app_info": {
        "app_icon": "https://online-xxxxxxx.cos.ap-shenzhen-fsi.myqcloud.com/xxxxxxxx/...",
        "app_name": "app1"
    },
    "account_list": [
        "account1",
        "account2"
    ]
}
```

## 错误码

| | |
| --- | --- |
| httpStatus | Response |

| | |
|---|---|
| 400 | {"error": "invalid_request", "error_description": "missing access_token parameter"} |
| 401 | {"error": "invalid_request", "error_description": "invalid_token"} |

# 注销刷新令牌

最近更新时间：2022-03-23 18:06:03

## 接口描述

注销权限令牌（refresh token 或 access token）。

## 请求方法

GET

## 请求地址

```
/auth/oauth2/revoke
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| client_id | Query | String | 是 | 客户端 ID。 |
| client_secret | Query | String | 是 | 客户端 Secret。 |
| token | Query | String | 是 | 需要注销的 refresh token 或 access token。 |
| token_type_hint | Query | String | 否 | 待注销的令牌类型，固定值为 refresh_token，access_token，此参数为提升接口性能参数。<br>• 如果填写此项，则优先在该令牌类型中匹配并注销令牌<br>• 如果未填写此项，则默认优先在refresh_token令牌类型中匹配并注销令牌。<br>• 如果没有匹配到对应令牌，则匹配范围扩大至 refresh_token，access_token 两种类型。 |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| message | Body | String | 操作成功信息。 |
| success | Body | String | 操作成功描述。 |

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/oauth2/revoke?
client_id=ODQyNGJlYmFjYWFmND*****kwNjU4MzMxOThkMGU&client_secret=oG412Uk6*****wJfvWUdJht1j%2bq&token=c6a40
b185c8*****ed0f59509b541&token_type_hint=access_token
```

### 返回示例

```
{
    "success": "ok",
    "message": "invalidate token success"
}
```

# MFA 认证因子质询

最近更新时间：2022-03-23 18:06:08

## 接口描述

获取 MFA 要素，例如短信验证码、一次密码（OTP）等。

## 请求方法

POST

## 请求地址

```
/auth/mfa/challenge
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| client_id | Query | String | 是 | 客户端 ID |
| client_secret | Query | String | 否 | 客户端 Secret |
| mfa_token | Query | String | 是 | mfa 令牌，密码模式的 token 接口返回mfa_required 错误时会同时返回mfa_token |
| challenge_type | Query | String | 是 | MFA 认证类型，目前固定为 oob |
| oob_channel | Query | String | 否 | MFA 认证方式，固定为 sms |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| challenge_type | Body | String | 用户支持的 MFA 认证类型 |
| oob_code | Body | String | oob 授权码 |

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/mfa/challenge?
client_id=ODQyNGJlYm****mNDZiYWE4YjkwNjU4MzMxOThkMGU&client_secret=oG412Uk6EdbfXtgU****WUdJht1j%2bq&mfa_to
ken=7138d426f2fb5****e152a4710720b39&challenge_type=oob&oob_channel=sms
```

### 返回示例

```
{
    "challenge_type": "oob",
    "oob_code": "6275684b3e****12ac40c3db148906ae"
}
```

## 错误码

| httpStatus | Response |
| --- | --- |
| 403 | {"error": "unauthorized_client", "error_description": "invalid client"} |

# 注销

最近更新时间：2024-11-14 14:50:32

## 接口描述

用户登录状态注销，由应用方发起的登出。

> **说明**
> EIAM 完成清除自身会话后，会跳转到合作方应用指定的跳转地址：`https://example.com/logout`。

## 请求方法

GET

## 请求地址

```
/auth/sso/logout
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| access_token | Header | String | 否 | access_token，在 HTTP 头部 Authorization 属性中传入的 Bearer 令牌。如果传入该令牌，则校验令牌的正确性并注销令牌 |
| return_to | RequestParam | String | 否 | 应用希望 EIAM 清除自身会话之后，跳转的地址，由应用具体业务逻辑决定。此参数格式例如：`https://example.com/logout` |

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/sso/logout?return_to=https%3A%2F%2Fexample.com%2Flogout
```

### 返回示例

```
{
    "success": "ok",
    "message": "Logout successfully"
}
```

# 数据结构

最近更新时间：2024-11-14 16:00:42

## application_info

| 参数名称 | 类型 | 是否允许为 null | 描述 |
| --- | --- | --- | --- |
| app_icon | String | 否 | 应用图标 |
| app_name | String | 否 | 应用名称 |

## key_info

| 参数名称 | 类型 | 是否允许为 null | 描述 |
| --- | --- | --- | --- |
| key | String | 否 | 密钥类型，如 RSA |
| e | String | 否 | RSA 公钥 |
| kid | String | 否 | 密钥标识 |
| n | String | 否 | RSA 公钥 |

# JWT 应用对接

## 概述

最近更新时间：2023-12-28 11:36:22

### 协议概述

Json web token（JWT）是为了在网络应用环境间传递声明而执行的一种基于 JSON 的开放标准。该 token 被设计为紧凑且安全的，适用于分布式站点的单点登录（SSO）场景。
JWT 的声明一般被用来在身份提供者和服务提供者间传递被认证的用户身份信息，以便于从资源服务器获取资源，也可以增加一些额外的其它业务逻辑所必须的声明信息，该 token 也可直接被用于认证，也可被加密。

### JWT 的构成

JWT 是由三段信息构成的，将这三段信息文本用 `.` 链接一起就构成了 JWT 字符串，这三部分分别是 Header、Payload、Signature，各部分均为 JSON 字符串 Base64 编码值，如下：

```
eyJhbGciOiJFUzI##############.eyJqdGkiOiJjbzlESHNuOEYtbEdFSnFjcT##############.gHVV8X2JAnese5_SIPmM2I#####
#########
```

- Header
  由 token 的类型（“JWT”）和算法名（例如 HMAC、SHA256 等）两部分组成 JSON 字符串（如下），并对该 JSON 字符传进行 Base64 编码，最终到 Header 信息。

  ```
  {
  'typ': 'JWT',
  'alg': 'HS256'
  }
  ```

- Payload
  即存放有效信息的地方，由多个声明字段组成的 JSON 字符串（如下），并对该 JSON 字符传进行 Base64 编码，最终到 Payload 信息。
  这些有效信息包括：标准中注册的声明、自定义声明。其中，自定义声明，不应包括个人身份等敏感信息。标准中注册的声明为建议使用，但不强制使用，包括：
  - iss：JWT 签发者
    -sub：JWT 所面向的用户
    -aud：接收 JWT 的一方
    -exp：JWT 的过期时间，这个过期时间必须要大于签发时间
    -nbf：定义在什么时间之前，该 JWT 都是不可用的
    -iat：JWT 的签发时间
    -jti：JWT 的唯一身份标识，主要用来作为一次性 token，从而回避重放攻击

    ```
    {
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true
    }
    ```

- Signature
  即签名信息，采用 Header 中指定的签名算法（alg），对（Base64 编码的 Header + "." + Base64 编码的 payload）计算签名信息（如下）。签名过程如需使用密钥，应提前生成指定算法的密钥，并存储于认证服务器。

  ```
  // javascript
  var encodedString = base64UrlEncode(header) + '.' + base64UrlEncode(payload);
  var signature = HMACSHA256(encodedString, 'secret');
  ```

### 应用场景

基于 token 的鉴权机制（单点登录）
类似于 HTTP 协议也是无状态的，它不需要在服务端去保留用户的认证信息或者会话信息。这就意味着基于 token 认证机制的应用不需要去考虑用户在哪一台服务器登录了，这就为应用的扩展提供了便利。

### 运行流程

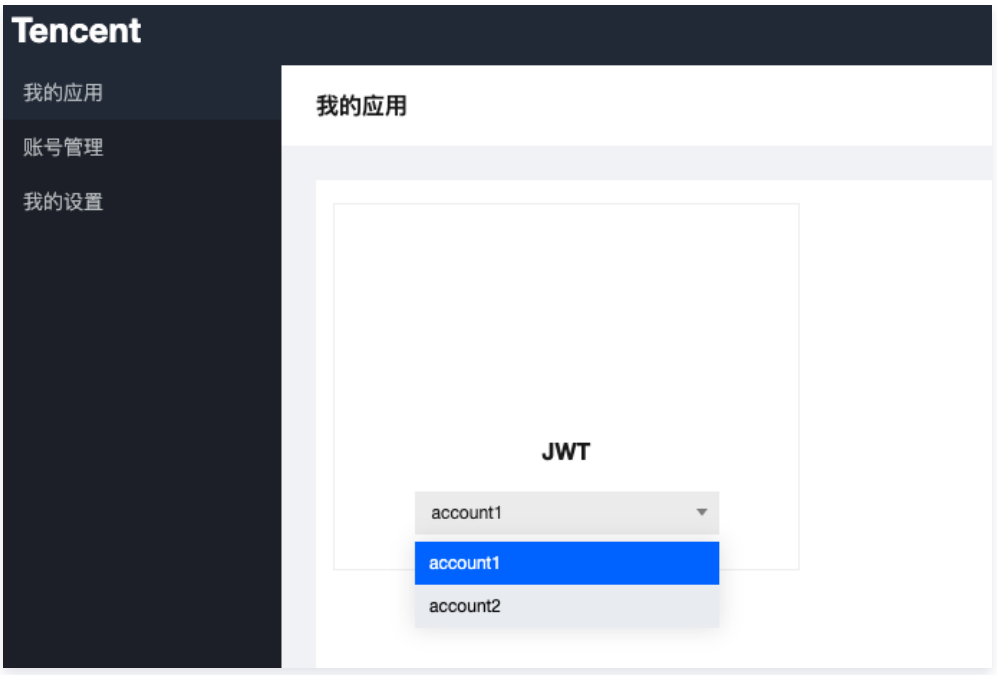
1. 用户使用用户名密码来请求认证服务器。
2. 认证服务器进行验证用户的信息。
3. 认证服务器通过验证后，发送给用户一个 token。
4. 客户端存储 token，并在每次请求时附送上这个 token 值。
5. 应用系统服务端验证 token 值，并返回数据。

## 数字身份管控平台（员工版）功能概述

### 使用 jose4j api 生成 JWT token

生成的 token 中，标准声明字段 subject 的值表示登录者身份。具体取值为：

- 当登录用户没有绑定应用账号（映射关系）时，subject 取值为登录用户名（userName）。
- 当登录用户绑定了唯一一个应用账号（映射关系）时，subject 取值为应用账号名（accountName）。
- 当登录用户绑定了多个应用账号（映射关系）时，登录成功后不会跳转到应用系统，而是跳转到门户应用列表页面，用户选择指定的登录账号后，单击应用图标进行登录，如下：




### 采用 RS256 签名算法

采用 RS256 签名算法生成签名信息，RSA 密钥长度为2048位。密钥对随 JWT 应用创建时一并生成，保存于认证服务器。
应用系统验证 token 的有效性，需要使用该应用的公钥进行验签，可参考 应用管理页面 > **选定指定应用** > 单击**应用配置** >对应的 JWT PublicKey，获取 JSON 格式的公钥信息字符串，如下：

JWT PublicKey {"kty":"RSA","kid":"B903E[redacted],"use":"sig","alg":"RS256","n":"qCfIOHaID6Axv[redacted],"e":"AQAB"}

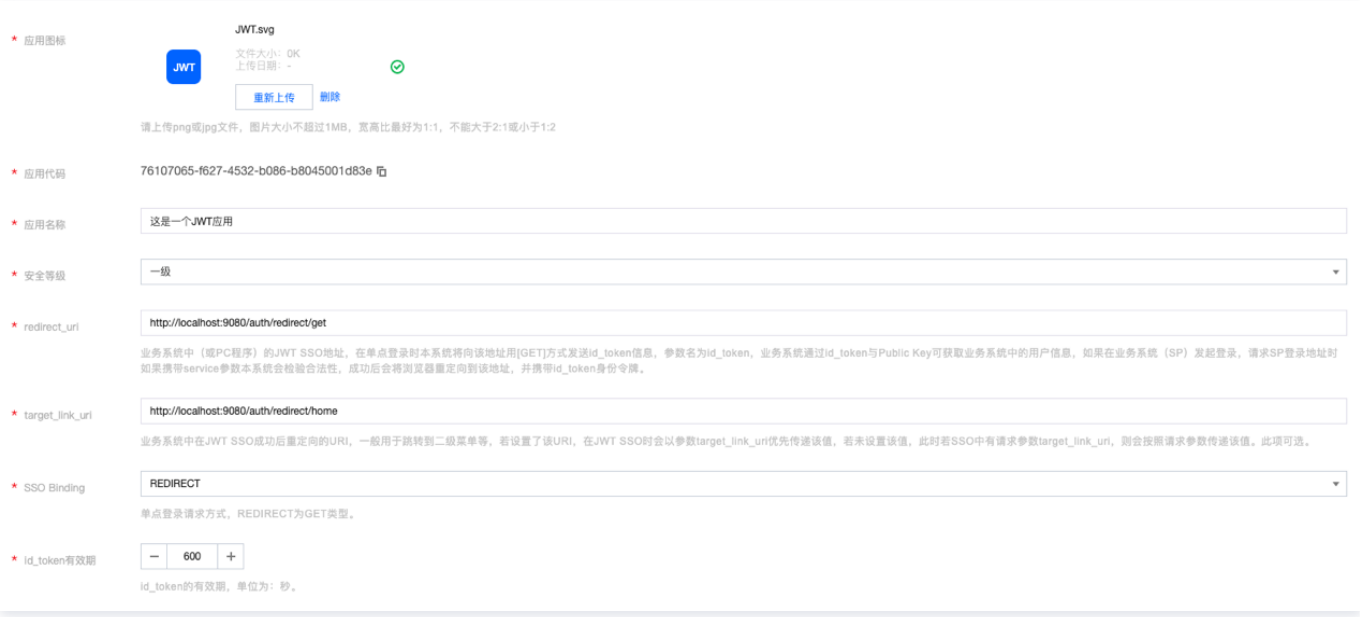
# 对接操作

最近更新时间：2023-12-28 11:36:22

## 创建应用并授权访问

### 创建应用

1. 登录 数字身份管控平台（员工版）控制台，在左侧导航栏，单击**应用管理**。
2. 在应用管理页面，单击**新建应用**，选择应用协议模板**JWT**，单击**下一步**。
3. 在编辑应用信息页面，登记接入应用的相关信息，如下图所示：
   - redirect_uri：为应用系统的重定向地址，用于接收数字身份管控平台（员工版）以重定向的方式返回的 token。
   - target_link_uri：为应用系统单点登录成功后需要重定向的地址，一般用于跳转到二级菜单等。
   - SSO Binding：单点登录请求方式，默认为重定向 REDIRECT。
   - id_token 有效期：id_token 的有效期，单位为：秒。




### 授权访问

具体操作请参见 授权管理 。

## 开发对接步骤

### 步骤1：获取 JWT token

应用系统发起请求数字身份管控平台（员工版），数字身份管控平台（员工版）校验用户身份生成后生成 JWT token，通过重定向的方式将 token 返回应用系统。
请求方式：GET
请求 Url：

```
https://<auth.domain>/auth/sso/ssoLogin?service=<service>
```

请求地址也可参考 应用管理页面 > 选定指定应用 > 单击**应用配置** >对应的“应用登录地址”。
请求参数说明：

| 参数 | 参数位置 | 类型 | 是否必选 | 描述 | 示例值 |
| --- | --- | --- | --- | --- | --- |
| service | Query | String | 是 | 应用 ID，应用系统的唯一标志 | ajgetgndtbaskbampvclw |

重定向回应用系统 Url：

```
http(s)://<redirect_uri>?id_token=<token>
```

返回参数说明：

| 参数 | 参数位置 | 类型 | 是否必选 | 描述 | 示例值 |
| --- | --- | --- | --- | --- | --- |
| id_token | Query | String | 是 | JWT token，有效时间与创建应用是设置的 id_token 有效期一致 | - |

返回的JWT token中，包含如下属性值：

| claim 属性名 | 说明 |
| --- | --- |
| sub | 数字身份管控平台（员工版）中的用户名。 |
| subjectId | 数字身份管控平台（员工版）中的用户 ID。 |
| secondaryAccount | 用户对应的应用账号，多个应用账号以竖线分隔。 |

## 步骤2：验证 JWT 并获取 token 中的账号相关信息

应用系统侧，使用 jose4j api 对 token 进行验证，验证成功后获取 token 中的账号相关信息。
应用系统 pom.xml 引入 maven 依赖：

```
<dependency>
    <groupId>org.bitbucket.b_c</groupId>
    <artifactId>jose4j</artifactId>
    <version>0.7.2</version>
</dependency>
```

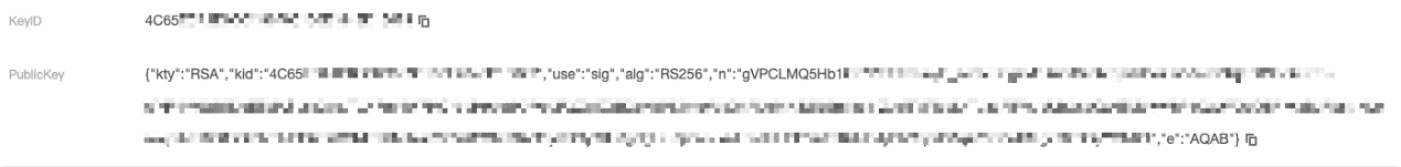
验证 token，并从 token 中获取各声明字段的信息，参考代码：

```
/**
  * 验证 JWT
  *
  * @param keyId 密钥 ID
  * @param jwtTicket JWT 票据
  * @param publicKeyStr 公钥
  */
public JwtClaims validateToken(String keyId, String jwtTicket, String publicKeyStr) throws Exception {
    JsonWebSignature jws = new JsonWebSignature();
    jws.setCompactSerialization(jwtTicket);
    jws.setKey(JsonWebKey.Factory.newJwk(publicKeyStr).getKey());

    boolean verifySignature = jws.verifySignature();
    if (!verifySignature) {
       logger.error("Invalid signature! jwtTicket:{} keyId: {} ", jwtTicket, keyId);
        throw new Exception("Invalid signature!");
    }

    String payload = jws.getPayload();
    JwtClaims claims = JwtClaims.parse(payload);
    NumericDate expirationTime = claims.getExpirationTime();
    if (expirationTime == null) {
       logger.error("Invalid signature! jwtTicket:{} keyId: {} ", jwtTicket, keyId);
       throw new Exception("Invalid signature!");
    }
    if (!expirationTime.isAfter(NumericDate.now())) {
       logger.error("Jwt ticket expired! jwtTicket:{} keyId: {} ", jwtTicket, keyId);
```

```
        throw new Exception("Jwt ticket expired!");
    }

    String loginSubject = claims.getSubject();
    System.out.println("login subject is " + loginSubject);
    return claims;
}
```

上述参考代码中：

- jwtTicket：为步骤1获取的 JWT token.
- keyId：可参考 应用管理页面 > 选定指定应用 > 单击**应用配置** > 对应的 JWT KeyID。
- publicKeyStr：可参考 应用管理页面 > 选定指定应用 > 单击**应用配置** >对应的 JWT PublicKey。



# CAS 应用对接

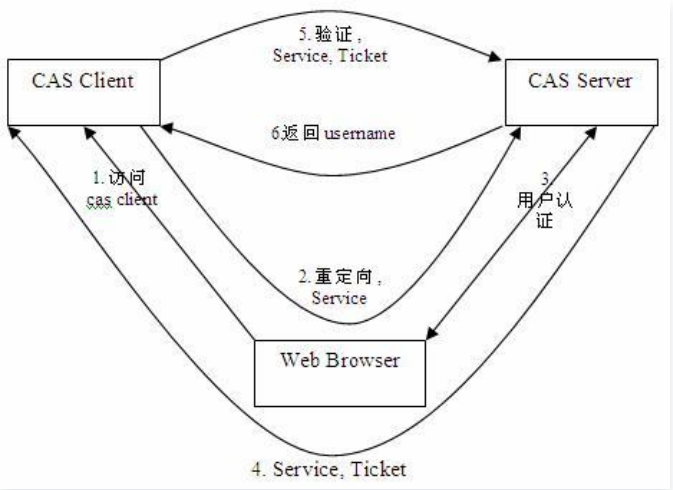
## 概述

最近更新时间：2023-12-28 11:36:22

### 协议概述

CAS（Central Authentication Service）中央认证服务，一种独立开放指令协议，旨在为 Web 应用系统提供一种可靠的单点登录方法。

#### 运行流程




1. CAS Client 与受保护的客户端应用部署在一起，以 Filter 方式保护受保护的资源，对于访问受保护资源的每个 Web 请求，CAS Client 会分析该请求的 HTTP 请求中是否包含 Service Ticket。
2. CAS Client 发现如果没有包含 Service Ticket，则说明当前用户尚未登录，于是将请求重定向到指定好的 CAS Server 登录地址，并传递 Service（也就是要访问的目的资源地址），以便登录成功过后转回该地址。
3. 用户输入认证信息，如果登录成功，CAS Server 随机产生一个相当长度、唯一、不可伪造的 Service Ticket，并缓存以待将来验证。
4. 之后系统自动重定向到 Service 所在地址。
5. CAS Client 在拿到 Service 和新产生的 Ticket 后，调用 CAS Server 提供的接口对 Service Ticket 进行核实，以确保 Service Ticket 的合法性。
6. Service Ticket 核实通过后，CAS Server 返回对应的登录用户身份。
   在该协议中，所有交互均应采用 SSL 协议，确保安全性。

### 数字身份管控平台（员工版）功能概述

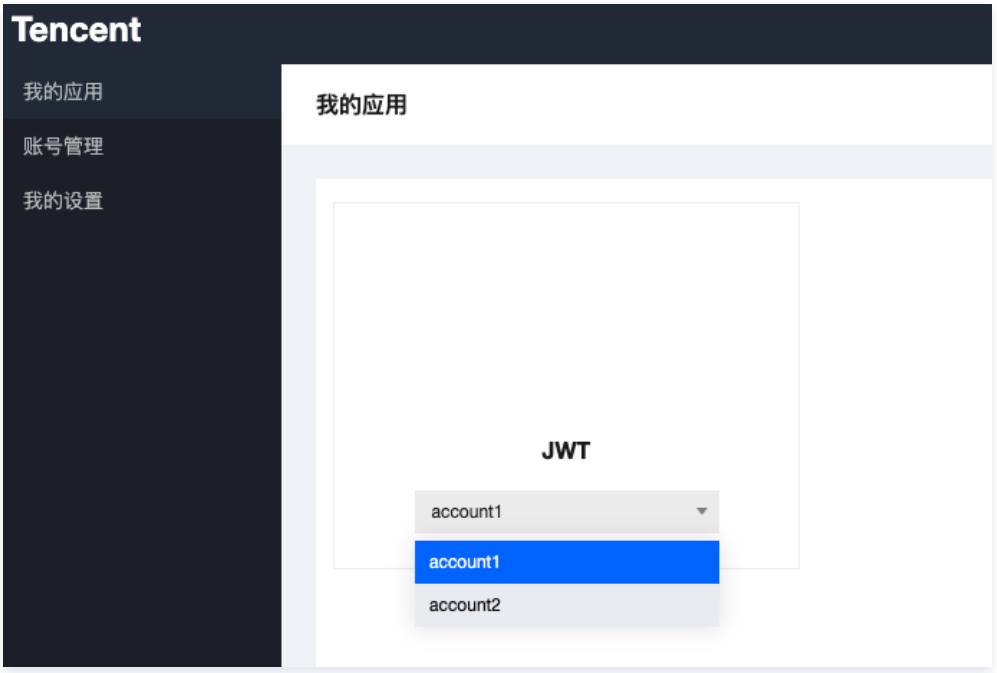
#### 生成 Service Ticket

生成随机字符串作为 Service Ticket，并通过重定向的方式将 Service Ticket 返回应用系统。

#### 验证 Service Ticket

提供后端接口，对上送的 Service Ticket 进行验证，确定该 Service Ticket 是否由 数字身份管控平台（员工版）所颁发。验证成功后，返回登录用户身份，以 XML 格式返回，`xmlns:cas="http://www.yale.edu/tp/cas"`。
登录身份具体取值为：

- 当登录用户没有绑定应用账号（映射关系）时，用户身份取值为登录用户名（userName）。
- 当登录用户绑定了唯一一个应用账号（映射关系）时，用户身份取值为应用账号名（accountName）。
- 当登录用户绑定了多个应用账号（映射关系）时，登录成功后不会跳转到应用系统，而是跳转到门户应用列表页面，用户选择指定的登录账号后，单击应用图标进行登录，如下：


Tencent
我的应用
账号管理
我的设置
我的应用
JWT
account1
account1
account2

# 对接操作

最近更新时间：2023-12-28 11:36:22

## 创建应用并授权访问

### 创建应用

1. 登录 数字身份管控平台（员工版）控制台，在左侧导航栏，单击**应用管理**。
2. 在应用管理页面，单击**新建应用**，选择应用协议模板**CAS（标准）**，单击**下一步**。
3. 在编辑应用信息页面，登记接入应用的相关信息，如下图所示：
   - ServerNames：为应用系统的重定向地址，用于接收数字身份管控平台（员工版）以重定向的方式返回的参数。
   - TargetUrl：应用系统发起单点登录时的地址，一般为应用系统首页地址。

### 授权访问

具体操作请参见 授权管理 。

## 开发对接步骤

### 步骤1：获取 Service Ticket

应用系统发起请求数字身份管控平台（员工版），数字身份管控平台（员工版）校验用户身份生成后生成 Service Ticket，通过重定向的方式将 ticket 返回应用系统。
请求方式：GET
请求Url：

```
https://<auth.domain>/auth/sso/ssoLogin?service=<service>
```

请求地址可参考 应用管理页面 > 选定指定应用 > 单击**应用配置** >对应的“应用登录地址”。
请求参数说明：

| 参数 | 参数位置 | 类型 | 是否必选 | 描述 | 示例值 |
| --- | --- | --- | --- | --- | --- |

| | | | | | |
|---|---|---|---|---|---|
| service | Query | String | 是 | 应用 ID，应用系统的唯一标志 | fa4a90ff-d2df-47da-abb7-9e680d62a469 |

重定向回应用系统Url：

```
http(s)://<redirect_uri>?ticket=<ticket>
```

返回参数说明：

| 参数 | 参数位置 | 类型 | 是否必选 | 描述 | 示例值 |
|---|---|---|---|---|---|
| ticket | Query | String | 是 | Service Ticket | - |

## 步骤2：验证 Service Ticket 并获取对应的账号名

请求数字身份管控平台（员工版）对 Service Ticket 进行验证，验证成功后，返回登录用户身份。
请求方式：GET
请求Url：

```
https://<auth_domain>/auth/sso/cas/verify?service=<service>&ticket=<ticket>
```

请求地址也可参考 应用管理页面 > 选定指定应用 > 单击**应用配置** >对应的 CAS validation URL Prefix。
请求参数说明：

| 参数 | 参数位置 | 类型 | 是否必选 | 描述 | 示例值 |
|---|---|---|---|---|---|
| service | Query | String | 是 | 应用 ID，应用系统的唯一标志 | fa4a90ff-d2df-47da-abb7-9e680d62a469 |
| ticket | Query | String | 是 | 步骤1获取的 Service Ticket | - |

返回示例：

```
<cas:serviceResponse xmlns:cas="http://www.yale.edu/tp/cas">
      <cas:authenticationSuccess>
         <cas:user>iamadmin</cas:user>
         <cas:attributes>
         <cas:secondaryAccount>cas-user-001</cas:secondaryAccount>
         <cas:subjectId>e2a05b30-3448-4631-8fdd-050a3a52e59c</cas:subjectId>
         </cas:attributes>
         </cas:authenticationSuccess>
    </cas:serviceResponse>
```

返回参数说明：

| 参数 | 参数位置 | 类型 | 是否必选 | 描述 | 示例 |
|---|---|---|---|---|---|
| cas:user | XML | String | 是 | 登录用户名 | - |
| cas:subjectId | XML | String | 是 | 用户名 ID | - |
| cas:secondaryAccount | XML | String | 是 | 用户对应的应用账号，多个应用账号以竖线分隔 | - |

# 表单代填应用对接
# 概述

最近更新时间：2024-12-04 15:06:42

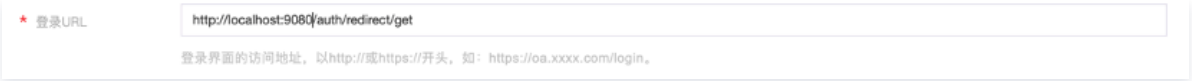
## 协议概述

表单代填就是在浏览器端通过模拟登录页面表单提交，进行登录操作。表单代填实现单点登录，优点在于应用系统可以不做任何改造。在应用系统不支持任何标准单点登录协议，也不可进行系统改造的情况下，可考虑使用表单代填的方式实现单点登录。

## 数字身份管控平台（员工版）功能概述

### 自定义代填表单

- action
  创建应用或配置应用时设置的“登录URL”，即登录请求提交地址。




- input 域
  创建应用或配置应用时，除了定义代填登录表单的用户名、密码 input 域的 name 属性外，还可以自定义表单中的其他 input 域的 name，及其对应的 value。

- 登录提交方式
  支持 POST、GET 两种方式。

如上图所示配置，等价于如下代填表单：

```
<form action="http://localhost:9080/auth/redirect/get" method="GET" target="_self">
      <input name="username" type="text" value=""/>
      <input name="password" type="text" value=""/>
      <input name="type" type="text" value="simple"/>
      <input name="source" type="text" value="thirdparty"/>
</form>
```

### 账号映射关系

建立登录用户与应用账号的映射关系，是使用表单代填方式进行单点登录的前提。创建“映射关系”时，应输入正确的应用系统侧的账号、以及账号对应的密码（必填）。

- 如未创建任何“映射关系”，发起登录时提示“未找到账号”。
- 当登录用户绑定了唯一一个应用账号（映射关系）时，登录名、登录密码 input 域的 value 取值该应用账号名。
- 当登录用户绑定了多个应用账号（映射关系），发起登录时提示“存在多个账号”。

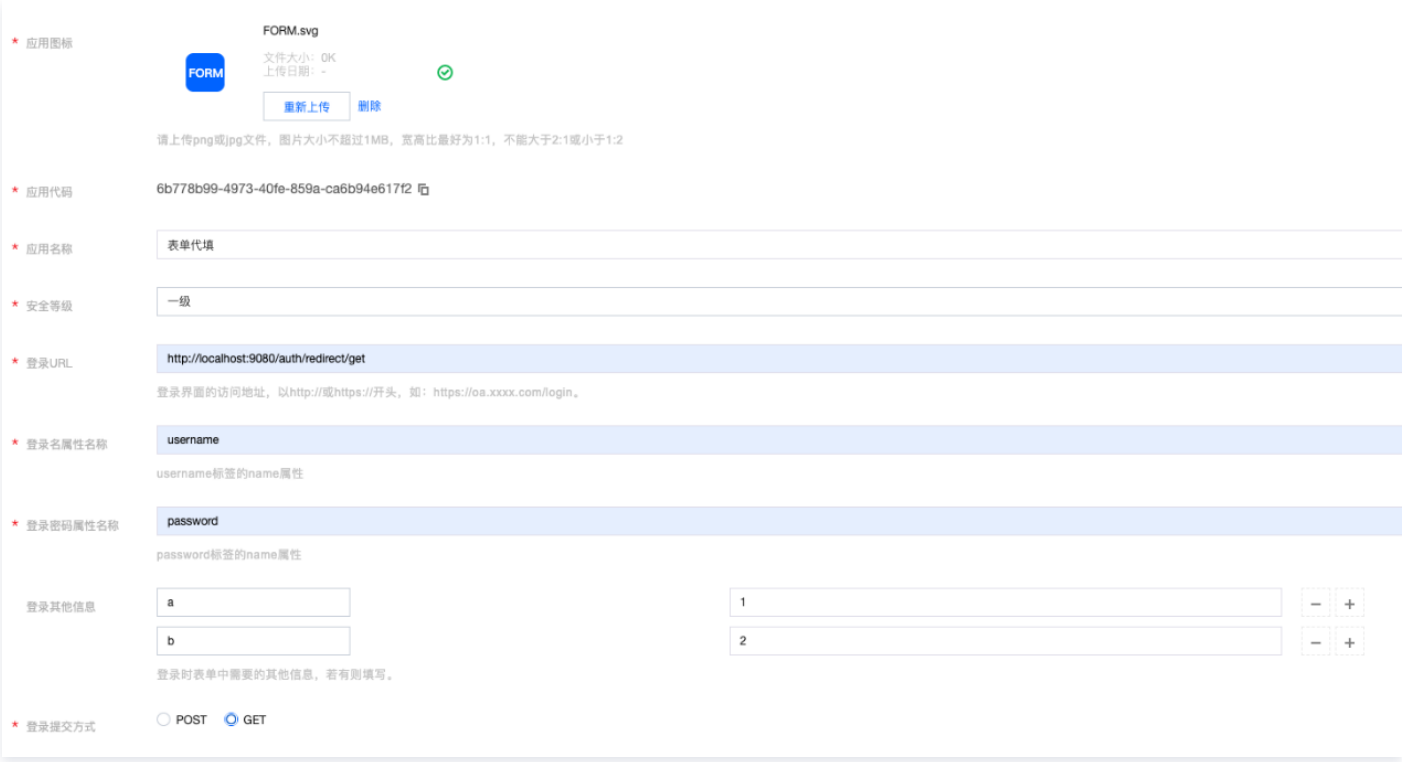
# 对接操作

最近更新时间：2023-12-28 11:36:22

## 创建应用、授权访问、创建映射关系

### 创建应用

1. 登录 数字身份管控平台（员工版）控制台，在左侧导航栏，单击**应用管理**。
2. 在应用管理页面，单击**新建应用**，选择应用协议模板**表单代填**，单击**下一步**。
3. 在编辑应用信息页面，登记接入应用的相关信息，如下图所示：




> **说明**
> 各输入项的具体意义，可参看上文 自定义代填表单 。

### 授权访问

具体操作请参见 授权管理 。

### 创建映射关系

具体操作请参见 账号管理-管理映射 。

## 开发对接步骤

应用系统发起请求数字身份管控平台（员工版），数字身份管控平台（员工版）校验用户身份后，模拟提交账号、密码、以及其他登录信息到应用系统的登录请求地址，由应用系统处理后续的登录流程。
请求方式：GET
请求 URL：

```
https://<auth.domain>/auth/sso/ssoLogin?service=<service>
```

请求地址也可参考 应用管理页面 > **选定指定应用** > 单击**应用配置** >对应的 “应用登录地址”。
请求参数说明：

| 参数 | 参数位置 | 类型 | 是否必选 | 描述 | 示例值 |
| --- | --- | --- | --- | --- | --- |
| service | Query | String | 是 | 应用 ID，应用系统的唯一标志 | fa4a90ff-d2df-47da-abb7-9e680d62a469 |

# SAML 应用对接

# 概述

最近更新时间：2023-12-28 11:36:22

## 协议概述

安全断言标记语言（Security Assertion Markup Language，SAML）是一个基于 XML 的开源标准数据格式，它在当事方之间交换身份验证和授权数据，尤其是在身份提供者和服务提供者之间交换。

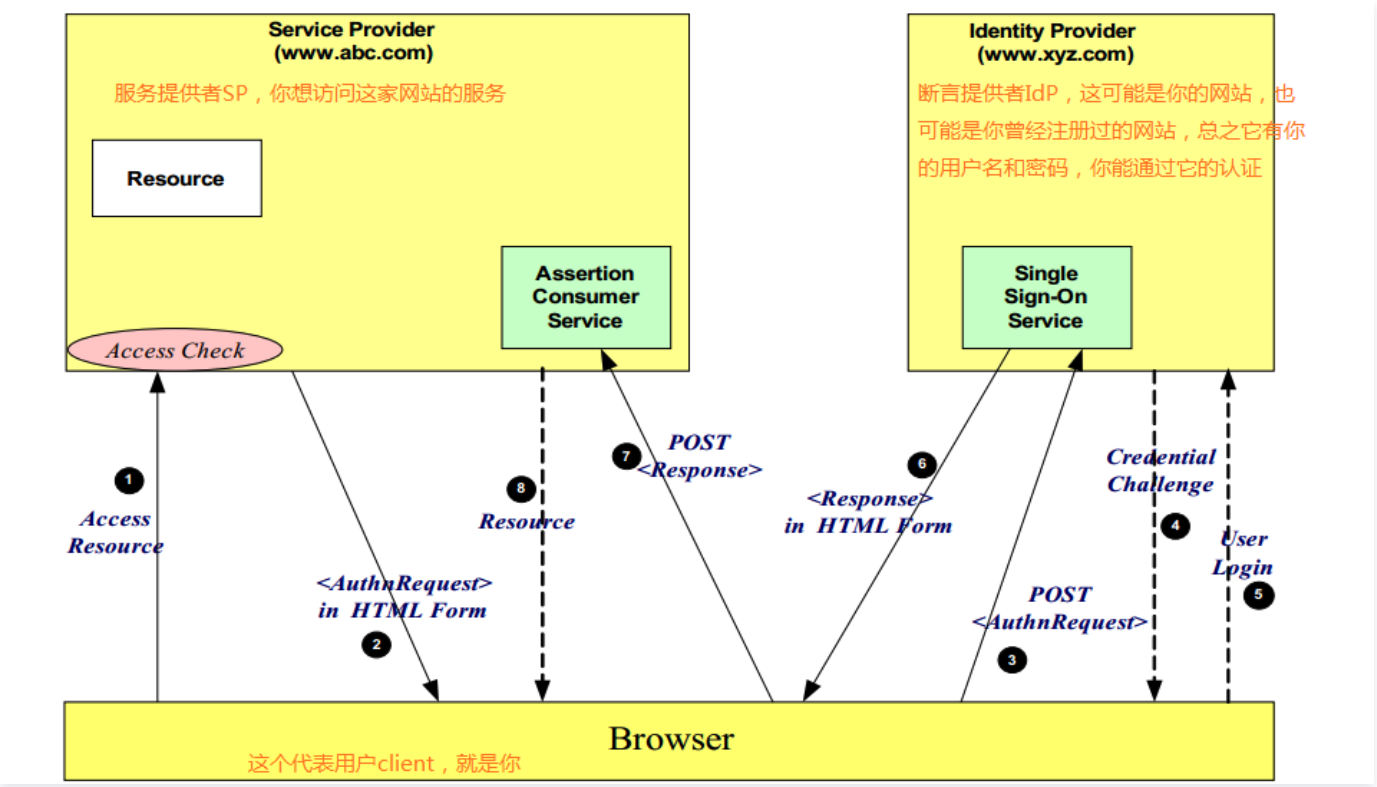
## 名词说明

- 委托人：SAML 规范定义的三个角色之一，通常为一名用户。
- IdP：Identity Provider，身份提供者，SAML 规范定义的三个角色之一，通常指身份鉴别服务器。
- SP：Service Providers，服务提供者，SAML 规范定义的三个角色之一。
- Assert：断言，是指 IdP 认证用户后，发送认证结果给 SP 的一种形式，以表示委托人的基本信息、权限信息等。

## 应用场景

SAML 解决的最重要的需求是网页浏览器单点登录。在 SAML 协议中，一旦用户身份被 IdP 认证过后，该用户再去访问其他在主站注册过的应用（SP）时，都可以直接登录，而不用再输入身份和口令。

## 运行流程




1. 委托人访问 SP 的某个受保护资源，例如，浏览器打开 http://www.abc.com/resource1。
2. SP 发现当前委托人没有认证信息，需要进行身份认证，则生成一个 SAML 认证请求数据包，并发送请求到 IdP。
3. 委托人进行身份认证。
4. IdP 确认身份认证通过后，生成断言，断言可包括但不限于委托人姓名、手机号码、邮箱、权限等信息。
5. IdP 使用私钥对断言进行签名，然后将签名信息、用户身份、自定义断言属性等包装成特定 XML 格式的返回报文，通过 POST 方式重定向回 SP。
6. SP 验证返回报文中的签名信息，并从报文中获取断言，进而完成后续登录流程。

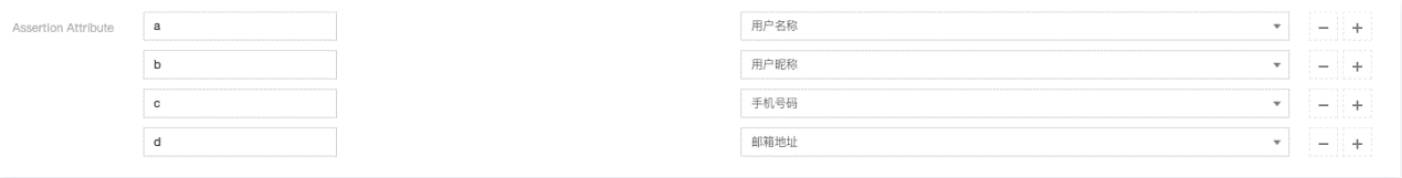
## 数字身份管控平台（员工版）功能概述

### 协议版本

目前支持 `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent` 。

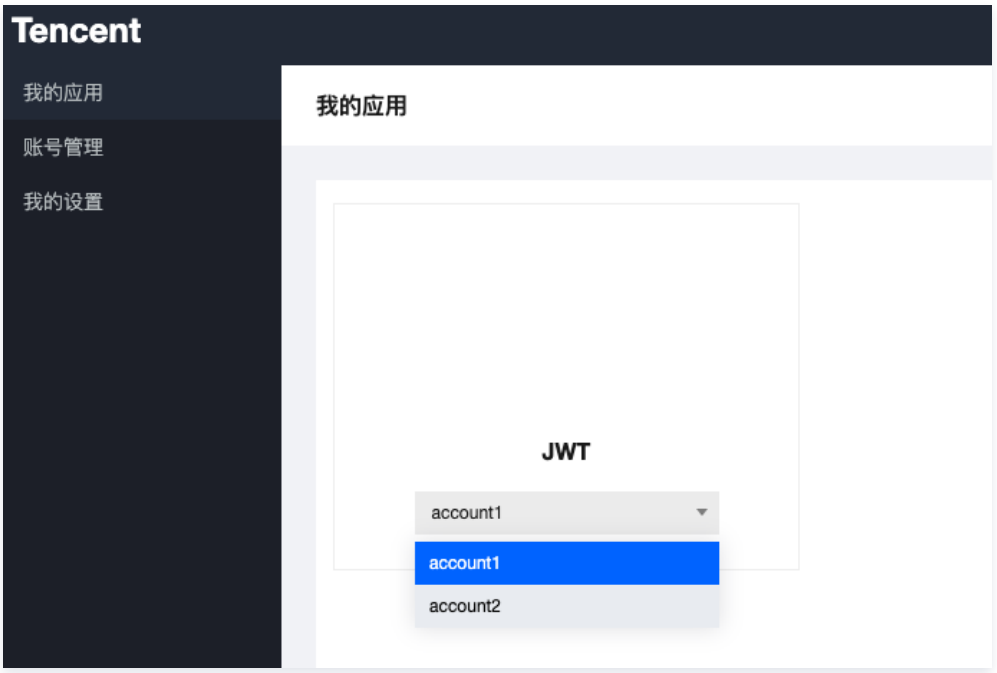
### 自定义断言属性

创建应用或配置应用时设置的 Assertion Attribute，表示断言应包含的属性。属性名可根据实际情况填写，属性值支持用户名称、用户昵称、手机号码、邮箱地址，如下：




### 其他断言信息

- 签名信息（Signature）
  采用 RSA 数字证书私钥，对断言信息进行数字签名。
- 用户身份（Subject）
  具体取值为：
  - 当登录用户没有绑定应用账号（映射关系）时，subject 取值为登录用户名（userName）。
  - 当登录用户绑定了唯一一个应用账号（映射关系）时，subject 取值为应用账号名（accountName）。
  - 当登录用户绑定了多个应用账号（映射关系）时，登录成功后不会跳转到应用系统，而是跳转到门户应用列表页面，用户选择指定的登录账号后，单击应用图标进行登录，如下：



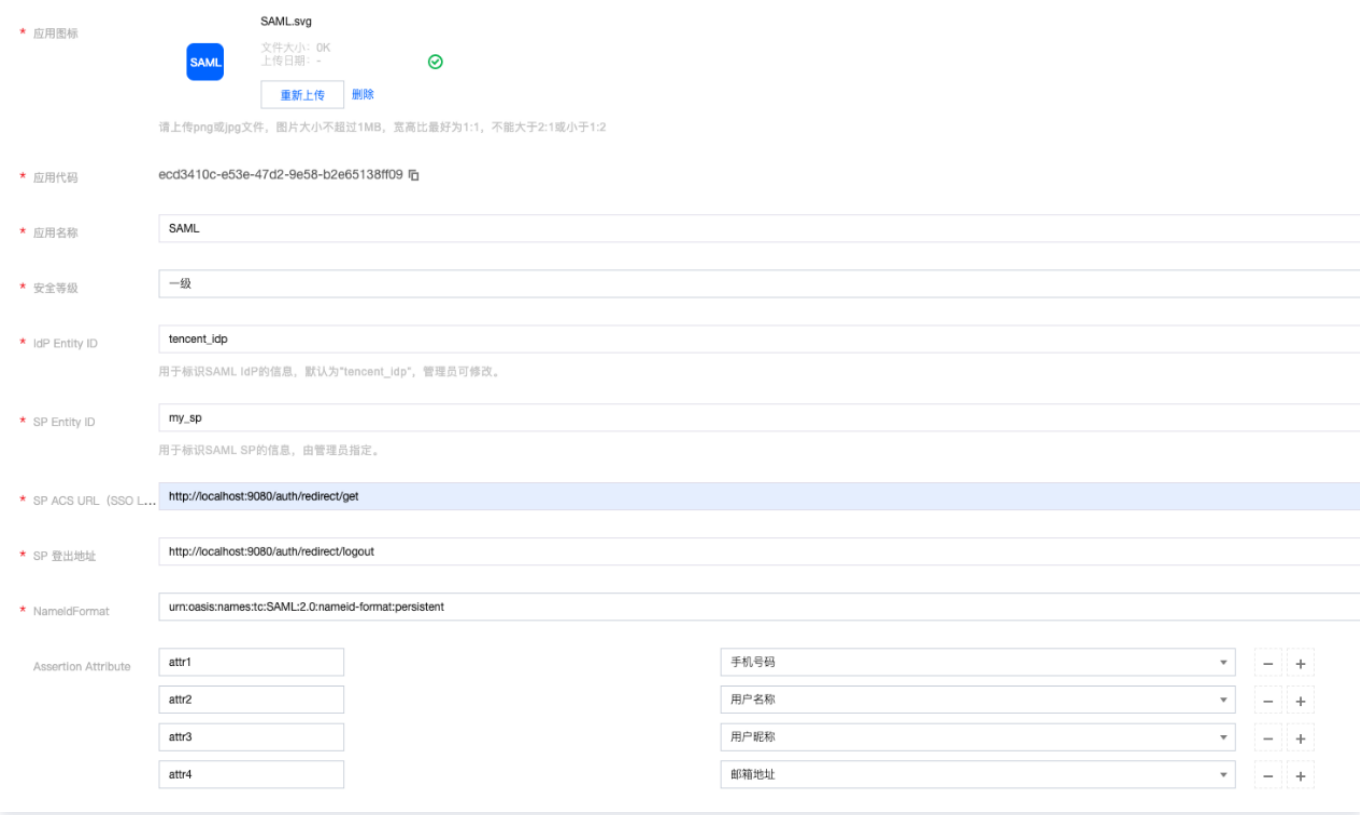
# 对接操作

最近更新时间：2023-12-28 11:36:22

## 创建应用、授权访问

### 创建应用

1. 登录 数字身份管控平台（员工版）控制台，在左侧导航栏，单击**应用管理**。
2. 在应用管理页面，单击**新建应用**，选择应用协议模板**SAML2**，单击**下一步**。
3. 在编辑应用信息页面，登记接入应用的相关信息，如下图所示：
   - IdP Entity ID：用于标识 SAML IdP 的信息，默认为 tencent_idp，会在 SAML 协议返回的 XML 报文中体现。
   - SP Entity ID：用于标识 SAML SP 的信息，会在 SAML 协议返回的XML报文中体现。
   - SP ACS URL：为应用系统的重定向地址，用于接收数字身份管控平台（员工版）以重定向的方式返回的 SAML 协议 XML 报文。
   - SP 登出地址：应用系统的“退出”请求地址。
   - NameIdFormat：默认为 `urn:oasis:names:tc:SAML:2.0:nameid-format:persistent` 。
   - Assertion Attribute：自定义的断言属性，可参考 自定义断言属性 。




### 授权访问

具体操作请参见 授权管理 。

## 开发对接步骤

### 步骤1：获取 SAML 返回报文（SAMLResponse）

应用系统发起请求数字身份管控平台（员工版），数字身份管控平台（员工版）校验用户身份生成后生成 XML 格式的 SAML 返回报文，并通过 POST方 式重定向返回应用系统。
请求方式：GET
请求 Url：

```
https://<auth.domain>/auth/sso/ssoLogin?service=<service>
```

请求地址也可参考 应用管理页面 > 选定指定应用 > 单击**应用配置** >对应的 “SP发起地址”。
请求参数说明：

| 参数 | 参数位置 | 类型 | 是否必选 | 描述 | 示例值 |
|---|---|---|---|---|---|
| service | Query | String | 是 | 应用 ID，应用系统的唯一标志 | fa4a90ff-d2df-47da-abb7-9e680d62a469 |

重定向回应用系统 Url（POST 方式）：

```
http(s)://<redirect_uri>?SAMLResponse=<SAMLResponse>
```

返回参数说明：

| 参数 | 参数位置 | 类型 | 是否必选 | 描述 | 示例值 |
|---|---|---|---|---|---|
| SAMLResponse | Body | String | 是 | Base64 编码后的 XML 格式返回报文 | - |

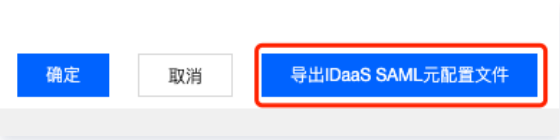
## 步骤2：验证签名并获取返回报文中的断言信息

应用系统侧，可使用 opensaml api 对返回报文（SAMLResponse）中的签名信息进行验签，验签成功后获取返回报文中的断言信息，包括用户身份、自定义属性等。
应用系统 pom.xml 引入 maven 依赖：

```
<dependency>
    <groupId>org.apache.velocity</groupId>
    <artifactId>velocity</artifactId>
    <version>1.6.2</version>
  </dependency>
  <dependency>
    <groupId>org.opensaml</groupId>
    <artifactId>opensaml-security-api</artifactId>
    <version>3.2.0</version>
 </dependency>
<dependency>
    <groupId>org.opensaml</groupId>
    <artifactId>opensaml-saml-api</artifactId>
    <version>3.2.0</version>
 </dependency>
 <dependency>
    <groupId>org.opensaml</groupId>
    <artifactId>opensaml-core</artifactId>
    <version>3.2.0</version>
 </dependency>
 <dependency>
    <groupId>org.opensaml</groupId>
    <artifactId>opensaml-saml-impl</artifactId>
    <version>3.2.0</version>
 </dependency>
 <dependency>
    <groupId>io.github.openfeign</groupId>
    <artifactId>feign-core</artifactId>
    <version>11.0</version>
 </dependency>
 <dependency>
    <groupId>net.shibboleth.utilities</groupId>
    <artifactId>java-support</artifactId>
    <version>7.2.0</version>
```

```
  </dependency>
```

获取验签所用的公钥证书。
可参考 应用管理页面 > 选定指定应用 > 单击**应用配置** > 单击**导出IDaaS SAML元配置文件**，从下载的 XML 文件中获取。




验签，并从返回报文中获取身份信息、断言信息，参考代码如下：
其中，certificateS 参数为验签所用的公钥证书。

```
import org.apache.commons.codec.binary.Base64;
import org.opensaml.Configuration;
import org.opensaml.DefaultBootstrap;
import org.opensaml.saml2.core.*;
import org.opensaml.saml2.core.impl.*;
import org.opensaml.xml.io.*;
import org.opensaml.xml.security.x509.BasicX509Credential;
import org.w3c.dom.*;
import org.opensaml.xml.*;
import org.apache.commons.codec.binary.Base64;

import java.io.*;
import java.security.*;
import java.security.cert.*;
import java.security.spec.*;
import javax.xml.parsers.*;


public class SAMLResponseHandler {

  private static final String certificateS = "MIIENTCCAx2gA***************iEAblF1hYy5dDb/Fjc3W0Q==";

  public void handle(String responseMessage) {

   // Read certificate
   CertificateFactory certificateFactory = CertificateFactory.getInstance("X.509");
   InputStream inputStream = new ByteArrayInputStream(Base64.decodeBase64(certificateS.getBytes("UTF-
8")));
   X509Certificate certificate = (X509Certificate) certificateFactory.generateCertificate(inputStream);
   inputStream.close();


   BasicX509Credential credential = new BasicX509Credential();
   KeyFactory keyFactory = KeyFactory.getInstance("RSA");
   X509EncodedKeySpec publicKeySpec = new X509EncodedKeySpec(certificate.getPublicKey().getEncoded());
   PublicKey key = keyFactory.generatePublic(publicKeySpec);
   credential.setPublicKey(key);

   // Parse response
   byte[] base64DecodedResponse = Base64.decodeBase64(responseMessage);

   ByteArrayInputStream is = new ByteArrayInputStream(base64DecodedResponse);
   DocumentBuilderFactory documentBuilderFactory = DocumentBuilderFactory.newInstance();
   documentBuilderFactory.setNamespaceAware(true);
   DocumentBuilder docBuilder = documentBuilderFactory.newDocumentBuilder();
   Document document = docBuilder.parse(is);
   Element element = document.getDocumentElement();
```

```
    UnmarshallerFactory unmarshallerFactory = Configuration.getUnmarshallerFactory();
    Unmarshaller unmarshaller = unmarshallerFactory.getUnmarshaller(element);
    XMLObject responseXmlObj = unmarshaller.unmarshall(element);
    Response responseObj = (Response) responseXmlObj;
    Assertion assertion = responseObj.getAssertions().get(0);
    String subject = assertion.getSubject().getNameID().getValue();
    String issuer = assertion.getIssuer().getValue();
    String audience =
assertion.getConditions().getAudienceRestrictions().get(0).getAudiences().get(0).getAudienceURI();
    String statusCode = responseObj.getStatus().getStatusCode().getValue();

    org.opensaml.xml.signature.Signature sig = assertion.getSignature();
    org.opensaml.xml.signature.SignatureValidator validator = new
org.opensaml.xml.signature.SignatureValidator(credential);
    validator.validate(sig);

   }
  }
```

**SAMLReponse 示例：**


```
PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiPz4KPHNhbWwycDpSZXNwb25zZSBEZXN0aW5hdGlvbj0iaHR0cDovL2xvY2
FsaG9zdDo5MDgwL2F1dGgvcmVkaXJlY3QvcG9zdCIgSUQ9Il83MjIzMjdhMGYwYWRmOGY2OGM0ODJhNzU2OTkxYTAwZCIgSXNzdWVJbnN0
YW50PSIyMDIxLTA1LTEzVDA5OjQzOjEzLjEzMloiIFZlcnNpb249IjIuMCIgeG1sbnM6c2FtbDJwPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0
FNTDoyLjA6cHJvdG9jb2wiPjxzYW1sMjpJc3N1ZXIgeG1sbnM6c2FtbDI9InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIuMDphc3NlcnRp
b24iPnRlbmNlbnRfaWRwPC9zYW1sMjpJc3N1ZXI+PHNhbWwycDpTdGF0dXM+PHNhbWwycDpTdGF0dXNDb2RlIFZhbHVlPSJ1cm46b2FzaX
M6bmFtZXM6dGM6U0FNTDoyLjA6c3RhdHVzOlN1Y2Nlc3MiLz48L3NhbWwycDpTdGF0dXM+PHNhbWwyOkFzc2VydGlvbiBJRD0iX2ZjNzc0
N2Q4ZTk2OTUzZDNhNTliMzljMjhmOWFjODExIiBJc3N1ZUluc3RhbnQ9IjIwMjEtMDUtMTNUMDk6NDM6MTMuMDY3WiIgVmVyc2lvbj0iMi
4wIiB4bWxuczpzYW1sMj0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOmFzc2VydGlvbiIgeG1sbnM6eHNkPSJodHRwOi8vd3d3Lncz
Lm9yZy8yMDAxL1hNTFNjaGVtYSI+PHNhbWwyOklzc3Vlcj50ZW5jZW50X2lkcDwvc2FtbDI6SXNzdWVyPjxkczpTaWduYXR1cmUgeG1sbn
M6ZHM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvMDkveG1sZHNpZyMiPgo8ZHM6U2lnbmVkSW5mbz4KPGRzOlNpZ25hdHVyZU1ldGhvZC
BBbGdvcml0aG09Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvMDkveG1sZHNpZyNyc2Etc2hhMSIvPgo8ZHM6UmVmZXJlbmNlIFVSST0iI19m
Yzc3NDdkOGU5Njk1M2QzYTU5YjM5YzI4ZjlhYzgxMSI+CjxkczpUcmFuc2Zvcm1zPgo8ZHM6VHJhbnNmb3JtIEFsZ29yaXRobT0iaHR0cD
ovL3d3dy53My5vcmcvMjAwMC8wOS94bWxkc2lnI2VudmVsb3BlZC1zaWduYXR1cmUiLz4KPGRzOlRyYW5zZm9ybSBBbGdvcml0aG09Imh0
dHA6Ly93d3cudzMub3JnLzIwMDEvMTAveG1sLWV4Yy1jMTRuIyI+PGVjOkluY2x1c2l2ZU5hbWVzcGFjZXMgUHJlZml4TGlzdD0ieHNkIi
B4bWxuczplYz0iaHR0cDovL3d3dy53My5vcmcvMjAwMS8xMC94bWwtZXhjLWMxNG4jIi8+PC9kczpUcmFuc2Zvcm0+CjwvZHM6VHJhbnNm
b3Jtcz4KPGRzOkRpZ2VzdE1ldGhvZCBBbGdvcml0aG09Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvMDQveG1sZW5jI3NoYTI1NiIvPgo8ZH
M6RGlnZXN0VmFsdWU+MzA2a3M4OXBDYUFNZzVVdnlCelBCeDlNMWl1TEVwWkxPZExkaWpIa3RVaz08L2RzOkRpZ2VzdFZhbHVlPgo8L2Rz
OlJlZmVyZW5jZT4KPC9kczpTaWduZWRJbmZvPgo8ZHM6U2lnbmF0dXJlVmFsdWU+ClJqcVhhL1JtSlZCcGZYNEJmNlZWSFQ2dTRVNW1rdV
pZV1ZkeWZoS0oremhHNlhSTjZ0Nk9KQWpKOFBoNlZzVk9XUlZnV2JqT000QVQKTWEvZGNMdnpHcTI2MVJtcDgzcEhWeFoxZXhEcXQ1TkZX
aUpQNkhrQVI2V0hxOFVSbENRNkEzTVAyOUZFY053bWRXVXVuMTgwcW10dwpRcXUwTWlXSW5nZC9BTkVBUUtrPQo8L2RzOlNpZ25hdHVyZV
ZhbHVlPgo8L2RzOlNpZ25hdHVyZT48c2FtbDI6U3ViamVjdD48c2FtbDI6TmFtZUlEIEZvcm1hdD0iMi4wX3BlcnNpc3RlbnQiPmlhbWFk
bWluPC9zYW1sMjpOYW1lSUQ+PHNhbWwyOlN1YmplY3RDb25maXJtYXRpb24gTWV0aG9kPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FNTDoyLj
A6Y206YmVhcmVyIj48c2FtbDI6U3ViamVjdENvbmZpcm1hdGlvbkRhdGEgTm90T25PckFmdGVyPSIyMDIxLTA1LTEzVDA5OjQ4OjEzLjEy
M1oiIFJlY2lwaWVudD0iaHR0cDovL2xvY2FsaG9zdDo5MDgwL2F1dGgvcmVkaXJlY3QvcG9zdCIvPjwvc2FtbDI6U3ViamVjdENvbmZpcm
1hdGlvbj48L3NhbWwyOlN1YmplY3Q+PHNhbWwyOkNvbmRpdGlvbnM+PHNhbWwyOkF1ZGllbmNlUmVzdHJpY3Rpb24+PHNhbWwyOkF1ZGll
bmNlPnRlc3R0ZXN0dGVzdDI8L3NhbWwyOkF1ZGllbmNlPjwvc2FtbDI6QXVkaWVuY2VSZXN0cmljdGlvbj48L3NhbWwyOkNvbmRpdGlvbn
M+PHNhbWwyOkF1dGhuU3RhdGVtZW50IEF1dGhuSW5zdGFudD0iMjAyMS0wNS0xM1QwOTo0MzoxMy4xMjVaIj48c2FtbDI6QXV0aG5Db250
ZXh0Lz48L3NhbWwyOkF1dGhuU3RhdGVtZW50PjxzYW1sMjpBdHRyaWJ1dGVTdGF0ZW1lbnQ+PHNhbWwyOkF0dHJpYnV0ZSBOYW1lPSJzZW
NvbmRhcnlBY2NvdW50Ij48c2FtbDI6QXR0cmlidXRlVmFsdWUgeG1sbnM6eHNpPSJodHRwOi8vd3d3LnczLm9yZy8yMDAxL1hNTFNjaGVt
YS1pbnN0YW5jZSIgeHNpOnR5cGU9InhzZDpzdHJpbmciPnNhbWwtdXNlci0xfHNhbWwtdXNlci0yPC9zYW1sMjpBdHRyaWJ1dGVWYWx1ZT
48L3NhbWwyOkF0dHJpYnV0ZT48c2FtbDI6QXR0cmlidXRlIE5hbWU9InN1YmplY3RJZCI+PHNhbWwyOkF0dHJpYnV0ZVZhbHVlIHhtbG5z
OnhzaT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS9YTUxTY2hlbWEtaW5zdGFuY2UiIHhzaTp0eXBlPSJ4c2Q6c3RyaW5nIj5lMmEwNWIzMC
0zNDQ4LTQ2MzEtOGZkZC0wNTBhM2E1MmU1OWM8L3NhbWwyOkF0dHJpYnV0ZVZhbHVlPjwvc2FtbDI6QXR0cmlidXRlPjwvc2FtbDI6QXR0
cmlidXRlU3RhdGVtZW50Pjwvc2FtbDI6QXNzZXJ0aW9uPjwvc2FtbDJwOlJlc3BvbnNlPg==
```


**对 SAMLReponse 进行 Base64 解码后的 XML 返回报文：**

```
 <?xml version="1.0" encoding="UTF-8"?>
<saml2p:Response Destination="http://localhost:9080/auth/redirect/post"
ID="_722327a0f0adf8f68c482a756991a00d" IssueInstant="2021-05-13T09:43:13.132Z" Version="2.0"
    xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol">
    <saml2:Issuer
        xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion">tencent_idp
    </saml2:Issuer>
    <saml2p:Status>
        <saml2p:StatusCode Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
    </saml2p:Status>
    <saml2:Assertion ID="_fc7747d8e96953d3a59b39c28f9ac811" IssueInstant="2021-05-13T09:43:13.067Z"
Version="2.0"
        xmlns:saml2="urn:oasis:names:tc:SAML:2.0:assertion"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
        <saml2:Issuer>tencent_idp</saml2:Issuer>
        <ds:Signature
            xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
            <ds:SignedInfo>
                <ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                <ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
                <ds:Reference URI="#_fc7747d8e96953d3a59b39c28f9ac811">
                    <ds:Transforms>
                        <ds:Transform Algorithm="http://www.w3.org/2000/09/xmldsig#enveloped-signature"/>
                        <ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#">
                            <ec:InclusiveNamespaces PrefixList="xsd"
                                xmlns:ec="http://www.w3.org/2001/10/xml-exc-c14n#"/>
                            </ds:Transform>
                        </ds:Transforms>
                        <ds:DigestMethod Algorithm="http://www.w3.org/2001/04/xmlenc#sha256"/>
                        <ds:DigestValue>306ks89pCaAMg5UvyBzPBx9M1iuLEpZLOdLdijHktUk=</ds:DigestValue>
                    </ds:Reference>
                </ds:SignedInfo>
                <ds:SignatureValue>
RjqXa/RmJVBpfX4Bf6VVHT6u4U5mkuZYWVdyfhKJ+zhG6XRN6t6OJAjJ8Ph6VsVOWRVgWbjOM4AT
Ma/dcLvzGq261Rmp83pHVxZ1exDqt5NFWiJP6HkAR6WHq8URlCQ6A3MP29FEcNwmdWUun180qmtw
Qqu0MiWIngd/ANEAQKk=
</ds:SignatureValue>
            </ds:Signature>
            <saml2:Subject>
                <saml2:NameID Format="2.0_persistent">iamadmin</saml2:NameID>
                <saml2:SubjectConfirmation Method="urn:oasis:names:tc:SAML:2.0:cm:bearer">
                    <saml2:SubjectConfirmationData NotOnOrAfter="2021-05-13T09:48:13.123Z"
Recipient="http://localhost:9080/auth/redirect/post"/>
                </saml2:SubjectConfirmation>
            </saml2:Subject>
            <saml2:Conditions>
                <saml2:AudienceRestriction>
                    <saml2:Audience>testtesttest2</saml2:Audience>
                </saml2:AudienceRestriction>
            </saml2:Conditions>
            <saml2:AuthnStatement AuthnInstant="2021-05-13T09:43:13.125Z">
                <saml2:AuthnContext/>
            </saml2:AuthnStatement>
            <saml2:AttributeStatement>
                <saml2:Attribute Name="secondaryAccount">
                    <saml2:AttributeValue
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="xsd:string">saml-
user-1|saml-user-2
                    </saml2:AttributeValue>
                </saml2:Attribute>
```

```
                <saml2:Attribute Name="subjectId">
                    <saml2:AttributeValue
                        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:type="xsd:string">e2a05b30-3448-4631-8fdd-050a3a52e59c
                    </saml2:AttributeValue>
                </saml2:Attribute>
            </saml2:AttributeStatement>
        </saml2:Assertion>
    </saml2p:Response>
```

其中，

```
saml2p:Status 节点，表示 IdP 的相应码，“Success”表示成功；
```

- saml2:Assertion节点，断言信息，包括 saml2:Issuer、ds:Signature、saml2:Subject、saml2:Conditions、saml2:AttributeStatement 等子节点；
- saml2:Issuer 节点，对应创建应用或配置应用时，设置的“IDaaS IdentityId”；
- ds:Signature 节点，子节点 ds:DigestValue 为签名信息；
- saml2:Subject 节点，为登录用户名，或用户对应的应用账号；
- saml2:Conditions 节点，对应创建应用或配置应用时，设置的“SP Entity ID”；
- saml2:AttributeStatement 节点，为系统定义属性以及用户自定义属性。系统定义属性，包括：
  - subjectId，用户 ID；
  - secondaryAccount，表示用户对应的应用账号，多个应用账号以竖线分隔。
  - 用户自定义属性，对应创建应用或配置应用时，设置的“Assertion Attribute”，具体请参看上文“自定义断言属性”。

# OIDC 应用对接

## 概述

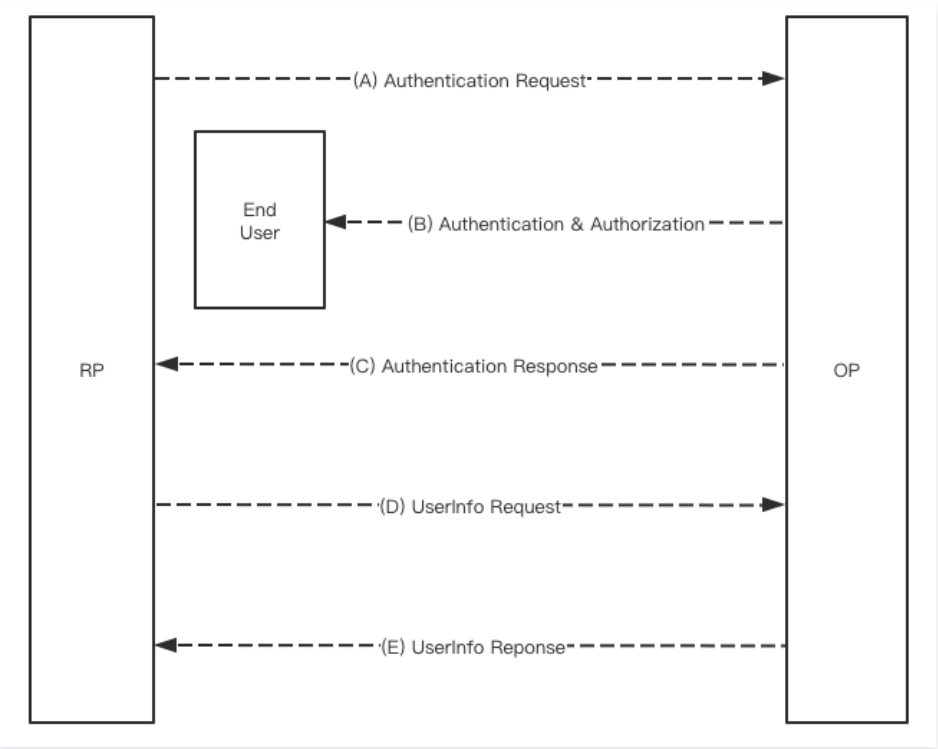
最近更新时间：2021-10-13 17:23:41

OIDC（OpenID Connect）是一个基于OAuth 2.0协议的身份认证标准协议。

### 名词说明

- OpenID Provider（OP）：OIDC 授权服务器，负责签发 ID Token。
- Relying Party（RP）：请求方的应用。
- End-User（EU）：用户，ID Token 的信息中会包含终用户的信息。
- ID Token：JWT 格式的授权 Claims。Claims 中包含用户信息字段（例如名称或电子邮件地址）和其他元数据。
- UserInfo Endpoint：用户信息接口，通过 ID Token 访问时返回用户信息。

### 运行流程




步骤说明：

1. 请求方应用发送认证请求到 OIDC 授权服务器。
2. OIDC 授权服务器对用户进行认证与授权。
3. OIDC 授权服务器返回认证信息，包含 ID Token、Access Token和Refresh Token。
4. 请求方应用携带 Access Token 发送请求到 UserInfo Endpoint。
5. UserInfo Endpoint 返回 End-User 的 Claims。

## OIDC 功能

### 实现 OIDC 场景模式

创建 OIDC 应用时指定的 grantType，只针对从数字身份管控平台（员工版）门户发起登录的场景，grantType 设置默认支持 authorization_code ，授权成功后返回 code；OIDC 应用不支持简化模式（ implicit ）。
对于从应用系统发起登录的四种基础模式，除简化模式以外均支持，应用发起登录重定向或接口调用时，根据实际情况传入各模式对应的 response_type 参数或 grant_type 参数即可。

### 支持令牌刷新

提供令牌刷新接口，应用使用 refresh_token 换取新的 access_token、id_token。

### 支持 PKCE 模式

授权码模式下，支持 PKCE 校验码验证。PKCE 模式适用于功能逻辑主要在客户端完成的 Native App。因为 Native App 没有浏览器那样的 cookie 支持，CP 服务器没有 session 这样的东西来保存 state 参数从而防止 CSRF，所以使用 PKCE 的 code_verifier 和 code_challenge 来防止 CSRF。
步骤如下：
这里主要考虑到的是，Native App 必须经过浏览器才能进行 redirect 操作，在下图的第③步中，由于使用了 App 的 custom schemer，如果有恶意 App 注册了同一个 schemer，则不能保证第③步的重定向会回到您的 App。

# 创建 OIDC 应用

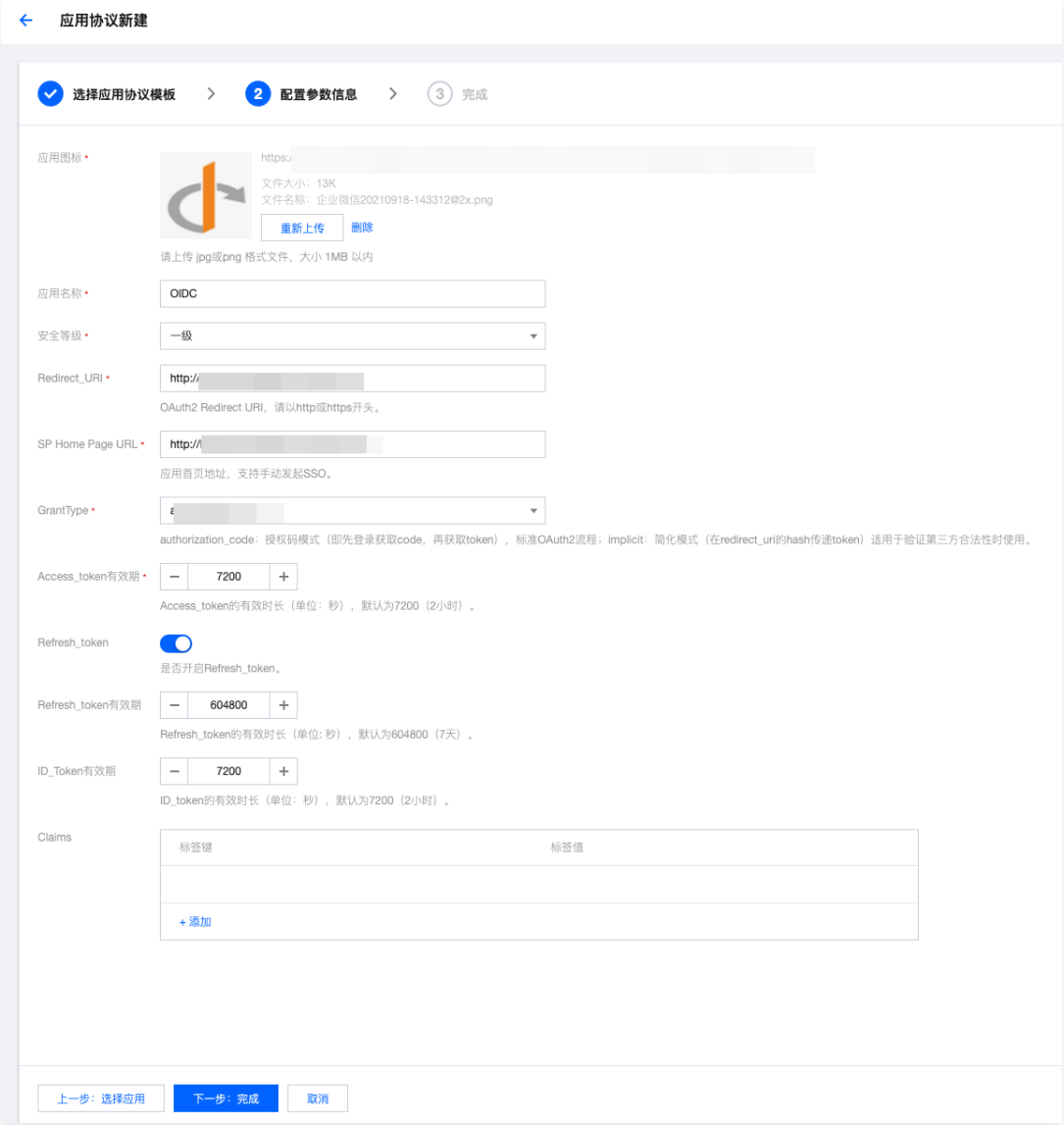
最近更新时间：2021-10-13 17:23:48

1. 登录 数字身份管控平台（员工版）控制台，在左侧导航栏，单击**应用管理**。
2. 在应用管理页面，单击**应用协议新建**，选择应用协议模板**OIDC**，单击**下一步：配置参数信息**。

3. 在应用协议新建页面，登记接入应用的相关信息，如下图所示：

**说明**

- Redirect_URI：为应用系统的重定向地址，用于接收 EIAM 以重定向的方式返回的授权码或 access_token 等参数。
- SP Home Page URL：为应用系统的登录首页地址。
- GrantType：为应用系统获取用户授权的方式，OIDC 应用仅支持 authorization_code 模式，授权成功后返回 code；该配置只针对从 EIAM 门户发起登录的场景。
- Access_token有效期：为 access_token 的有效时长（单位：秒），默认为7200（2小时）。
- Refresh_token：是否开启 refresh_token，该配置只对授权码模式和密码模式有效；如使用其他模式，即使将该配置设置为“开启”，也默认不返回 refresh_token。
- ID_Token有效期：为 refresh_token 的有效时长（单位: 秒），默认为604800（7天）。
- Claims：为用户自定义 id_token 中的属性，目前可定义属性包括用户名称、用户昵称、手机号码、邮箱地址。




4. 配置完成后，单击**下一步：完成**，即可创建应用。

> **说明**
> 创建完成后，需要对应用进行授权操作，具体操作请参见 授权管理 。

# API 场景介绍

最近更新时间：2021-10-14 09:47:31

本文将介绍数字身份管控平台（员工版）(以下简称 EIAM) 为 OIDC 应用系统提供的 API 。通过调用相关 API ，应用可以完成与用户身份相关的各类操作，例如：用户认证、用户信息的获取等。

> **注意**
> 在应用接入前，您需要首先在 应用管理页面 完成 OIDC 协议类型应用的新建与配置，获取应用的 Client ID 、 Client Secret 等配置信息。

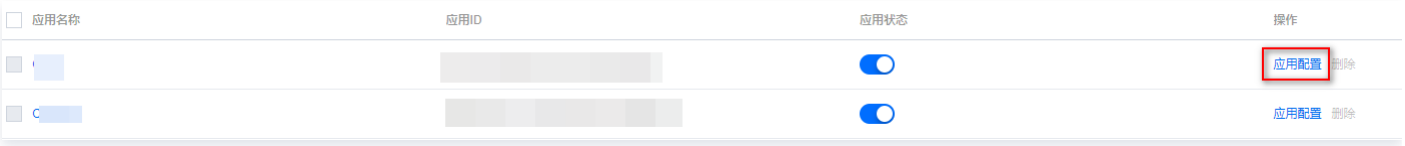
## 前提条件

已完成 创建 OIDC 应用 。

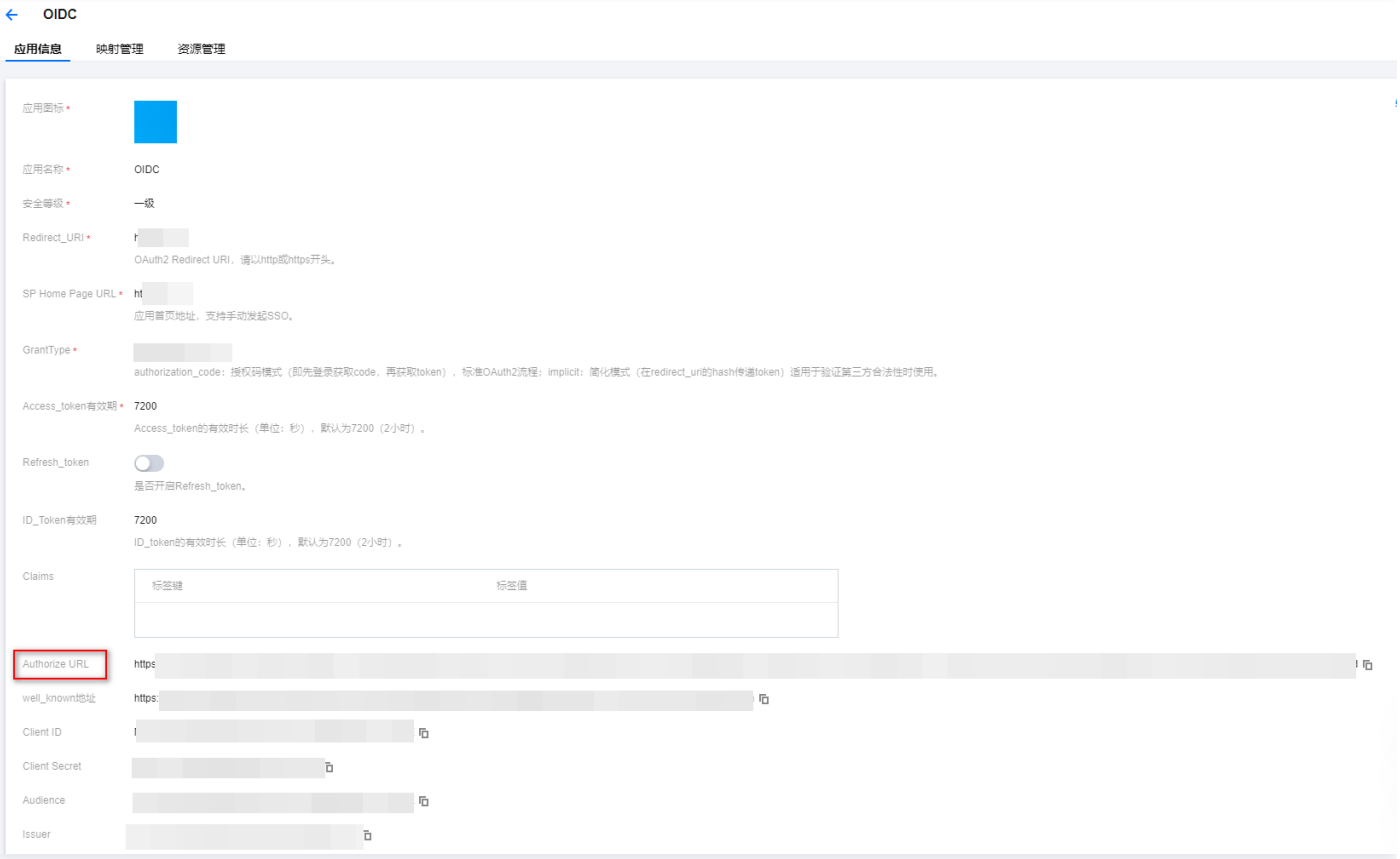
## 单点登录

对于应用实现用户认证的功能，EIAM 产品提供通过认证门户登录并直接单点登录至应用的能力。基于 OIDC 应用协议，提供授权码模式、PKCE 模式等多种认证授权方式。下面以授权码模式为例展示 OIDC 应用实现用户认证的流程。

1. 登录 数字身份管控平台（员工版）控制台 ，在左侧导航栏，单击**应用管理**。
2. 在应用管理页面，选择已创建的 OIDC 应用，单击**应用配置**。




3. 在应用信息页面，获取 Authorize URL，此 URL 为用户在应用系统登录时，应用将用户重定向的认证地址，详情请参见 授权码模式 。



4. 首次访问会跳转到登录页面进行用户身份认证，输入账号密码后单击**登录**，认证通过后重定向到指定地址(redirect_uri)，并在URL 中携带 code 参数。

5. 使用应用回调地址，接收到回调请求后，可以通过调用 获取 Token 接口，来获取 id_token、access_token 和 refresh_token。
6. 应用建立用户的登录态，将 id_token、access_token 和 refresh_token 与登录态关联起来，完成登录。

> **说明**
> 应用系统可自行选择合适的方法建立并保持登录态，例如：Web 应用可以使用 session cookie ，单页应用可以使用 localStorage 或 sessionStorage 。

## 短信验证码二次认证

针对在 EIAM 控制台开启了二次认证功能的情况下，发起的密码登录场景。

> **说明**
> 二次认证功能在 安全设置页面 中开启。

1. 登录 数字身份管控平台（员工版）控制台，在左侧导航栏，单击**应用管理**。
2. 调用 密码模式，触发二次认证将会返回 `mfa_required` 的错误码并返回 `mfa_token` 。
3. 调用 MFA 认证因子质询 接口，输入 `mfa_token` ，该接口会发送短信验证码，并返回 oob 码。
4. 调用 oob 模式的 短信验证码模式，传入 oob 码和短信验证码，验证通过后返回 id_token、access_token 和 refresh_token。
5. 应用建立用户的登录态，将 id_token、access_token 和 refresh_token 与登录态关联起来，完成登录。

> **说明**
> 应用系统可自行选择合适的方法建立并保持登录态，例如：Web 应用可以使用 session cookie ，单页应用可以使用 localStorage 或 sessionStorage 。

## 其他 API

用户应用系统可调用 获取 OpenID Provider 配置信息接口，获取 OIDC 授权服务器的配置信息。

# 调用方式
# 获取授权码

最近更新时间：2022-03-23 18:00:43

## 接口描述

获取用户授权，如果用户未登录 EIAM 系统，则要求用户进行身份认证，身份认证成功后返回授权码。接口支持 PKCE 模式，如需进行 PKCE 模式的校验，可传入相应的参数。

## 请求方法

GET

## 请求地址

```
/auth/oauth2/authorize
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| client_id | Query | String | 是 | 客户端 ID，可参见 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Id” |
| response_type | Query | String | 是 | 固定值“code”。 |
| redirect_uri | Query | String | 否 | 授权成功后的重定向地址，如该参数为空，则使用创建应用时设置的“Redirect Uri”作为重定向地址，用于接收 code 等参数。含特殊字符，需要做 URLEncode |
| state | Query | String | 推荐 | 应用系统提供的一个随机字符串，服务器会原样重定向给应用系统，它既能防止 CSRF、XSRF， 同时也可以用来对应 SP 初始发起的状态 |
| scope | Query | String | 是 | OIDC 应用系统必须上送该参数，参数值应以 openid 开头 |
| code_challenge_method | Query | String | 否 | PKCE 模式的必要参数，即 code_challenge 值的计算方法，目前仅支持 SM3，表示 SM3 摘要算法 |
| code_challenge | Query | String | 否 | PKCE 模式的必要参数，计算方式：code_challenge = code_challenge_method(code_verifier)。code_verifier 为校验码原文，应用系统需要自行保存code_verifier，并在获取授权令牌的请求中带上，用作验证 |
| nonce | Query | String | 否 | 应用系统提供的一个随机字符串，服务器会在 /token 请求返回的 id_token 中原样携带该参数，供应用系统进行校验，以防止重放冲击 |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| code | Query | String | 授权码，10分钟内有效 |
| state | Query | String | 应用系统提供的一个随机字符串，服务器会原样重定向给应用系统，它既能防止 CSRF、XSRF， 同时也可以用来对应 SP 初始发起的状态。当入参未传入 state 时，则无此项出参 |

## 返回方式

重定向

## 接口示例

**输入示例**

```
https://<auth_domain>/auth/oauth2/authorize?
client_id=ODQyNGJlYmF****NDZiYWE4YjkwNjU4MzMxOThkMGU&response_type=code&redirect_uri=http%3A%2F%2Flocalhos
t%3A9080%2Fauth%2Fredirect%2Fget&state=baa07350-e2b9-****-a8ae-
350325288049&scope=openidasd6786hj824****24993e0achjty&nonce=xxxxxxxx
```

**返回示例**

```
http://localhost:9080/auth/redirect/get?code=b9eb0dc2338824****24993e0aca6180&state=baa07350-e2b9-****-
a8ae-350325288049
```

**错误码**

| httpStatus | Response |
| --- | --- |
| 400 | {"error": "invalid_request", "error_description": "invalid response_type"} |
| 400 | {"error": "invalid_request", "error_description": "missing subject_token parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing client_id parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing response_type parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing scope parameter"} |
| 403 | {"error": "unauthorized_client", "error_description": "invalid client"} |
| 500 | {"error": "internal_server_error", "error_description": "generate access token fail"} |
| 500 | {"error": "internal_server_error", "error_description": "generate authorization code fail"} |

# 获取 Token
# 授权码模式

最近更新时间：2022-03-23 18:00:53

### 接口描述

使用授权码（code）换取授权令牌（access_token）。

### 请求方法

POST

### 请求地址

```
/auth/oauth2/token
```

### 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| Authorization | Header | String | 是 | 接口鉴权值，`Basic base64encode(<client_id>:(client_secret))`。鉴权模式为 `client_secret_basic` 时，必须上送该参数 |
| client_id | Query | String | 是 | 客户端 ID，可参见 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Id” |
| grant_type | Query | String | 是 | 固定值“authorization_code” |
| redirect_uri | Query | String | 否 | 授权成功后的重定向地址。如果“用户授权/authorize”请求带有 redirect_uri 参数，则本步骤请求必须上送 redirect_uri 参数，EIAM 会校验该请求参数值与“用户授权/authorize”上送的 redirect_uri 参数值是否一致。含特殊字符，需要做 URLEncode |
| code | Query | String | 是 | 授权码 |
| client_secret | Query | String | 否 | 客户端 Secret，可参见 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应“Client Secret”。含有特殊字符，需要做 URLEncode。如 code_verifier 为空，则 client_secret 必须上送 |
| code_verifier | Query | String | 否 | PKCE 模式必要参数，code_verifier 校验码原文。可能含有特殊字符，需要做 URLEncode |

### 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| access_token | Body | String | access_token |
| expires_in | Body | Long | access_token有效时间，与创建应用时设置的“access_token有效期”一致 |
| refresh_token | Body | String | refresh_token |
| id_token | Body | String | id_token |

### 输入示例

#### client_secret_post 鉴权模式

```
https://<auth_domain>/auth/oauth2/token?
client_id=ODQyNGJlYmFjY****WE4YjkwNjU4MzMxOThkMGU&grant_type=authorization_code&redirect_uri=http%3A%2F%2F
```

```
localhost%3A9080%2Fauth%2Fredirect%2Fget&code=b9eb0dc2****4993e0aca6180&client_secret=oG412Uk6EdbfXtgU****
UdJht1j%2bq&code_verifier=s9hw2n
```

### client_secret_basic 鉴权模式

```
Authorization: 'Basic WTJKbFlUUmhZVGszWV****bnZ1Qjh0SS9Jd1VFODFudw=='
https://<auth_domain>/auth/oauth2/token?
grant_type=authorization_code&redirect_uri=http%3A%2F%2Flocalhost%3A9080%2Fauth%2Fredirect%2Fget&code=b9eb
0dc2****4993e0aca6180&code_verifier=s9hw2n
```

### 返回示例

```
{
    "access_token": "c6a40b1****058eaed0f59509b541",
    "expires_in": 7200,
    "refresh_token": "6865e92a****01df8081e4562e26c82",
    "id_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJhc2Rhc2QiLCJpYXQiOiIyMDIxLTA5LTExIDAwOjAwOjAwIiwiZXhwIjoi
MjAyMS0wOS0yOSAwMDowMDowMCIsImF1ZCI6ImFzZGFzZCIsInN1YiI6ImFzZGFzZCJ9.aWErSpZtZ*****jbu_zPqt-wD_6-
fP_nWnb68E"
}
```

# 密码模式

最近更新时间：2022-03-23 18:00:59

## 接口描述

使用授权码（code）换取授权令牌（access_token）。

## 请求方法

POST

## 请求地址

```
/auth/oauth2/token
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| Authorization | Header | String | 是 | 接口鉴权值，`Basic base64encode(<client_id>:(client_secret))`。鉴权模式为 `client_secret_basic` 时，必须上送该参数 |
| client_id | Query | String | 是 | 客户端 ID，可参见 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Id” |
| grant_type | Query | String | 是 | 固定值“password” |
| client_secret | Query | String | 是 | 客户端 Secret，可参见 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Secret”。含有特殊字符，需要做 URLEncode。 |
| username | Query | String | 是 | 用于登录 EIAM 的用户名。可能含有特殊字符，需要做 URLEncode |
| password | Query | String | 是 | 用户名对应的密码，必须输入明文。可能含有特殊字符，需要做 URLEncode |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| access_token | Body | String | access_token |
| id_token | Body | String | id_token |
| expires_in | Body | Long | access_token 有效时间，与创建应用时设置的“access_token有效期”一致 |
| refresh_token | Body | String | 固定为空，该模式不提供 refresh_token |

## 输入示例

### client_secret_post 鉴权模式

```
https://<auth_domain>/auth/oauth2/token?
client_id=ODQyNGJlYmFjYWFmND****wNjU4MzMxOThkMGU&grant_type=password&client_secret=oG412Uk6EdbfXtgUW****t1
j%2bq&username=testuser&password=PPee3****lk!!
```

### client_secret_basic 鉴权模式

```
Authorization: 'Basic WTJKbFlUUmhZVGszWV****bnZ1Qjh0SS9Jd1VFODFudw=='
https://<auth_domain>/auth/oauth2/token?grant_type=password&username=testuser&password=PPee3****lk!!
```

## 返回示例

```
{
    "access_token": "c6a40b1****058eaed0f59509b541",
    "expires_in": 7200,
    "refresh_token": "6865e92a****01df8081e4562e26c82",
    "id_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJhc2Rhc2QiLCJpYXQiOiIyMDIxLTA5LTExIDAwOjAwOjAwIiwiZXhwIjoi
MjAyMS0wOS0yOSAwMDowMDowMCIsImF1ZCI6ImFzZGFzZCIsInN1YiI6ImFzZGFzZCJ9.aWErSpZtZ*****jbu_zPqt-wD_6-
fP_nWnb68E"
}
```

**在二次认证功能启用的情况下，将返回**

```
{
    "error": "mfa_required",
    "error_description":"mfa required",
    "mfa_token": "xxxxx"
}
```

# 客户端模式

最近更新时间：2022-03-23 18:02:48

## 接口描述

校验 client_id、client_secret，校验成功后返回授权令牌。

## 请求方法

POST

## 请求地址

```
/auth/oauth2/token
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| Authorization | Header | String | 是 | 接口鉴权值，`Basic base64encode(<client_id>:(client_secret))`。鉴权模式为 `client_secret_basic` 时，必须上送该参数 |
| client_id | Query | String | 是 | 客户端 ID，可参见 应用管理页面 > 选定指定应用 > 单击应用配置 > 对应的“Client Id”。鉴权模式为 `client_secret_post` 时，必须上送该参数 |
| grant_type | Query | String | 是 | 固定值“client_credentials” |
| client_secret | Query | String | 是 | 客户端 Secret，可参见 应用管理页面 > 选定指定应用 > 单击应用配置 > 对应的“Client Secret”。含有特殊字符，需要做 URLEncode。鉴权模式为 `client_secret_post` 时，必须上送该参数 |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| access_token | Body | String | access_token |
| id_token | Body | String | id_token |
| expires_in | Body | Long | access_token有效时间，与创建应用时设置的“access_token有效期”一致 |
| refresh_token | Body | String | 固定为空，该模式不提供 refresh_token |

## 输入示例

### client_secret_post 鉴权模式

```
https://<auth_domain>/auth/oauth2/token?
client_id=ODQyNGJlYmFjYWF****4YjkwNjU4MzMxOThkMGU&grant_type=client_credentials&client_secret=oG412Uk6Edbf
XtgUW****1j%256
```

### client_secret_basic 鉴权模式

```
Authorization: 'Basic WTJKbFlUUmhZVGszWV****bnZ1Qjh0SS9Jd1VFODFudw=='
https://<auth_domain>/auth/oauth2/token?grant_type=client_credentials
```

## 返回示例

```
{
    "access_token": "c6a40b1****058eaed0f59509b541",
    "expires_in": 7200,
    "refresh_token": "6865e92a****01df8081e4562e26c82"
}
```

**说明**
client_credentials 模式不会返回 id_token。

# 短信验证码模式

最近更新时间：2022-03-23 18:02:55

## 接口描述

验证授权码、短信验证码，验证通过后颁发授权令牌。

## 请求方法

POST

## 请求地址

```
/auth/oauth2/token
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| Authorization | Header | String | 是 | 接口鉴权值，`Basic base64encode(<client_id>:(client_secret))` 。鉴权模式为 `client_secret_basic` 时，必须上送该参数 |
| client_id | Query | String | 是 | 客户端 ID，可参见 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Id”。鉴权模式为 client_secret_post 时，必须上送该参数 |
| grant_type | Query | String | 是 | 固定值“mfa_oob” |
| code | Query | String | 是 | 授权码，此处传入 challenge 接口返回的 oob 授权码 |
| client_secret | Query | String | 是 | 客户端 Secret，可参见 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Secret”。含有特殊字符，需要做 URLEncode。鉴权模式为 client_secret_post 时，必须上送该参数 |
| mfa_token | Query | String | 是 | mfa 令牌，密码模式的 token 接口返回 `mfa_required` 错误时会同时返回 `mfa_token` |
| binding_code | Query | String | 是 | 短信验证码 |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| access_token | Body | String | access_token |
| id_token | Body | String | id_token |
| expires_in | Body | Long | access_token 有效时间，与创建应用时设置的“access_token有效期”一致 |
| refresh_token | Body | String | refresh_token |

## 输入示例

### client_secret_post 鉴权模式

```
https://<auth_domain>/auth/oauth2/token?
client_id=ODQyNGJlYmFjYWF****ZiYWE4YjkwNjU4MzMxOThkMGU&grant_type=mfa_oob&code=test0dc233****224993e0aca61
80&client_secret=test2Uk6E****WlwJfvWUdJht1j%2bq&binding_code=123456&mfa_token=fg233****224993e0sdf55dfrgf
```

### client_secret_basic 鉴权模式

```
Authorization: 'Basic WTJKbFlUUmhZVGszWV****bnZ1Qjh0SS9Jd1VFODFudw=='
https://<auth_domain>/auth/oauth2/token?
grant_type=mfa_oob&code=test0dc233****224993e0aca6180&binding_code=123456&mfa_token=fg233****224993e0sdf55
dfrgf
```

### 返回示例

```
{
    "access_token": "c6a40b1****058eaed0f59509b541",
    "expires_in": 7200,
    "refresh_token": "6865e92a****01df8081e4562e26c82",
    "id_token":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJhc2Rhc2QiLCJpYXQiOiIyMDIxLTA5LTExIDAwOjAwOjAwIiwiZXhwIjoi
MjAyMS0wOS0yOSAwMDowMDowMCIsImF1ZCI6ImFzZGFzZCIsInN1YiI6ImFzZGFzZCJ9.aWErSpZtZ*****jbu_zPqt-wD_6-
fP_nWnb68E"
}
```

# 令牌刷新

最近更新时间：2022-03-23 18:03:01

## 接口描述

使用 refresh_token 换取新的 access_token，刷新成功后原来的 id_token、access_token 和 refresh_token 将失效。

## 请求方法

POST

## 请求地址

```
/auth/oauth2/token
```

## 请求参数

| 参数 | 参数位置 | 类型 | 是否必填 | 描述 |
| --- | --- | --- | --- | --- |
| Authorization | Header | String | 是 | 接口鉴权值，`Basic base64encode(<client_id>:(client_secret))`。鉴权模式为 `client_secret_basic` 时，必须上送该参数 |
| client_id | Query | String | 是 | 客户端 ID，可参见 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Id”。鉴权模式为 client_secret_post 时，必须上送该参数 |
| grant_type | Query | String | 是 | 固定值“refresh_token” |
| client_secret | Query | String | 是 | 客户端 Secret，可参见 应用管理页面 > **选定指定应用** > 单击**应用配置** > 对应的“Client Secret”。含有特殊字符，需要做 URLEncode。鉴权模式为 client_secret_post 时，必须上送该参数 |
| refresh_token | Query | String | 是 | refresh_token |

## 返回参数

| 参数 | 参数位置 | 类型 | 描述 |
| --- | --- | --- | --- |
| access_token | Body | String | 新的 access_token |
| id_token | Body | String | 新的 id_token |
| expires_in | Body | Long | access_token 有效时间，与创建应用时设置的“access_token有效期”一致 |
| refresh_token | Body | String | 新的 refresh_token |

## 输入示例

### client_secret_post 鉴权模式

```
https://<auth_domain>/auth/oauth2/token?
client_id=ODQyNGJlYmFjYWFm****WE4YjkwNjU4MzMxOThkMGU&grant_type=refresh_token&client_secret=oG412Uk6****gU
WlwJfvWUdJht1j%2bq&refresh_token=a6a40b185c5****58eaed0f59509b567
```

### client_secret_basic 鉴权模式

```
Authorization: 'Basic WTJKbFlUUmhZVGszWV****bnZ1Qjh0SS9Jd1VFODFudw=='
```

```
https://<auth_domain>/auth/oauth2/token?
grant_type=refresh_token&refresh_token=a6a40b185c5****58eaed0f59509b567
```

### 返回示例


```
{
    "access_token": "c6a40b1****e35058eaed0f59509b541",
    "expires_in": 7200,
    "refresh_token": "6865e****2501df8081e4562e26c82",
    "id_token":
"fyJhbGoisdfIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJhc2Rhc2QiLCJpYXQiOiIyMDIxLTA5LTExIDAwOjAwOjAwIiwiZXhwIjoi
MjAyMS0wOS0yOSAwMDowMDowMCIsImF1ZCI6ImFzZGFzZCIsInN1YiI6ImFzZGFzZCJ9.aWErSpZtZ*****jbu_zPqt-wD_6-
fP_nWnb098"
}
```

# 错误码

最近更新时间：2021-10-13 17:29:55

| httpStatus | Response |
| --- | --- |
| 400 | {"error": "invalid_request", "error_description": "invalid code"} |
| 400 | {"error": "invalid_request", "error_description": "invalid client"} |
| 400 | {"error": "invalid_request", "error_description": "invalid redirect_uri"} |
| 400 | {"error": "invalid_request", "error_description": "invalid code challenge"} |
| 400 | {"error": "invalid_request", "error_description": "invalid grant_type"} |
| 400 | {"error": "invalid_request", "error_description": "missing username parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing password parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing suject_token parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing suject_token_type parameter"} |
| 400 | {"error": "invalid_request", "error_description": "missing refresh_token parameter"} |
| 401 | {"error": "invalid_request", "error_description": "invalid_token"} |
| 403 | {"error": "unauthorized_client", "error_description": "invalid user password"} |

版权所有：腾讯云计算（北京）有限责任公司

# 获取 JWT 公钥

最近更新时间：2021-10-13 18:02:28

EIAM 返回的是 JWT 格式的 ID Token 。请参考 OIDC 官方文档 对 JWT 进行解密与验证。验证所需的公钥通过调用获取 JWT 公钥接口获得。

## 接口描述

JWT 公钥用于对 JWT 格式的 ID Token 和 Access Token 进行解密。

## 请求方法

GET

## 请求地址

```
/auth/oauth2/jwks
```

## 返回参数

| 参数 | 参数类型 | 描述 |
| --- | --- | --- |
| keys | Array of key_info | 包含公钥的数组 |

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/oauth2/jwks
```

### 返回示例


```
{
  "keys" : [ {
    "kty" : "RSA",
    "e" : "AQAB",
    "kid" : "52dd697c-b8f9-****-ad38-21db07a1575b",
    "n" : "w31qUM4pR-vWQyhstx0BFc-
W6ntEPX****MR0LS4NCsFN40cFUhXV_hrfYkxmfhKpQF_4FQXdQLLBC2iliaNCFUWrGM1_i_Q6gSSrjt2D6Lzkc50-
6T2ORhnI5kNxvMqQLPxLgZzS-izB5ouazHn0BFzZJJeE9LAh__hmGSUjSF36w1YEp8GI5HVO5T-
RBKTMsRUdm3NyTSB3xyN_uC****svM_bOIUHGqiEAhdwpm7Pa93Wu0VtZBn5JmaeZi9OIcmcyWad6mrON_sltm5a15nDNhz2SYu8ICWH4u
E1ItXtg713I8txP8TAAwid_iPS21DQZr2LJwqB0KM026w"
  } ]
}
```

# 获取 OpenID Provider 配置信息

最近更新时间：2021-10-13 17:30:03

## 接口描述

应用可以通过此接口获取 OIDC 授权服务器的配置信息。具体的配置信息及其含义请参考 OpenID Connect Discovery 标准 。

> **注意**
> 如果不使用此接口，则需手动在应用本地存储一份配置信息。

## 请求方法

GET

## 请求地址

```
/auth/.well-known/openid-configuration
```

## 返回参数

| 参数 | 参数类型 | 描述 |
| --- | --- | --- |
| issuer | String | OIDC Issuer |
| authorization_endpoint | String | OAuth 2.0 获取授权的服务地址 |
| token_endpoint | String | OAuth 2.0 获取 Token 的服务地址 |
| jwks_uri | String | 获取 JWT 公钥的服务地址 |
| grant_types_supported | Array of String | 支持的 OAuth 2.0  Grant Type |
| response_types_supported | Array of String | OAuth 2.0 获取授权服务支持的 Response Type |
| token_endpoint_auth_methods_supported | Array of String | OAuth 2.0 获取 Token 服务支持的认证方式 |
| subject_types_supported | Array of String | 支持的 OIDC  Subject Identifier Type |
| id_token_signing_alg_values_supported | Array of String | 支持的 JWS 签名算法 |
| scopes_supported | Array of String | 支持的 OAuth 2.0  Scope |

## 接口示例

### 输入示例

```
https://<auth_domain>/auth/.well-known/openid-configuration
```

### 返回示例

```
{
  "issuer" : "https://xxxTENANT.PORTAL.DOMAIN",
  "authorization_endpoint" : "https://xxxTENANT.PORTAL.DOMAIN/oauth2/authorize",
  "token_endpoint" : "https://xxxTENANT.PORTAL.DOMAIN/oauth2/token",
  "token_endpoint_auth_methods_supported" : [ "client_secret_basic", "client_secret_post" ],
  "jwks_uri" : "https://xxxTENANT.PORTAL.DOMAIN/oauth2/jwks",
  "response_types_supported" : [ "code" ],
  "grant_types_supported" : [ "authorization_code", "client_credentials", "refresh_token" ],
  "subject_types_supported" : [ "public" ],
  "id_token_signing_alg_values_supported" : [ "RS256" ],
```

```
  "scopes_supported" : [ "openid" ]
}
```