中山大学

统一认证服务接入指引

V1.6

网络与信息中心（信息化办公室）

Network and Information Center

前 言

Introduction

本指引旨在为各系统开发及运维人员提供清晰的统一认证服务接入操作说明，涵盖接入授权申请、接口说明、接入方法与示例等内容。请使用者结合实际系统需求，对照指引逐步完成接入配置。通过统一认证服务，实现“一次认证，多系统访问”，提升校园数字化服务体验。

目 录

Contents

1. 引言

1.1 概述

本文档描述了中山大学统一认证服务所使用 CAS、OAuth2.0、OIDC 的协议规范和对接方式。校内应用系统需要接入统一认证服务时，请参考本文档进行对接开发。

应用开发人员可任意选择一种协议进行对接。

1.2 阅读对象

1.2.1 校方

中山大学业务部门信息系统相关的管理及技术人员，包括：

- 应用系统开发人员
- 应用系统运维人员

1.2.2 承建单位

- 产品经理/产品策划/需求负责人/需求分析师
- 系统架构设计师
- 开发工程师
- 实施工程师和运维工程师

1.3 接入申请

请应用系统管理员进入 USC 平台“公共服务平台新增及变更接入”流程填写各项信息。申请审批通过后，由网络与信息中心提供有关认证接入参数。

1.4 专业术语

术语	英文全程	描述
SSO	Single Sign-On	单点登录，一次登录后可访问所有应用
IAM	Identity and Access Management	身份识别与访问管理
CAS	Central Authentication Service	中央认证服务，是由耶鲁大学发起的开源单点登录协议
OAuth2.0	Open Authorization	OAuth 协议的延续版本，授权的标准协议
OIDC	OpenID Connect	基于 OAuth 2.0 的身份验证层，让客户端能够以标准化的方式验证用户身份

以下标识符为文档所使用的占位符，请根据实际情况进行替换。

标识符	描述
${SSO_URL}	SSO 系统实际的域名，请参考 1.5 节描述
${APP_CALLBACK}	接入系统的回调地址，使用 CAS 协议时为 service 参数，使用 OAuth2.0 或 OIDC 时为 redirect_uri 参数
${PROTOCOL}	认证协议，用于接口路径，根据实际情况使用"oauth2.0"或"oidc"

1.5 环境描述

以下为统一身份认证服务的访问域名，请根据实际情况进行设置。

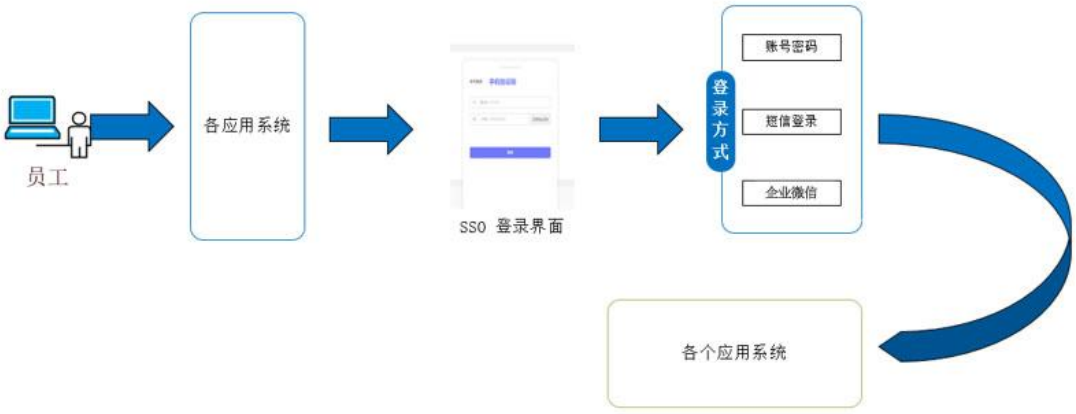
运行环境	域名（含访问协议）

生产	https://cas.sysu.edu.cn
测试(仅限校内访问)	https://iamsso.sysu.edu.cn

2. 接入指引

2.1 认证的总体过程

用户访问应用系统的认证过程如下图所示：



在登录后，认证服务默认返回 NetID。如需其他用户信息，请在申请表单中勾选。其他信息在 CAS 协议的 Ticket 校验接口或 OAuth2.0/OIDC 的用户信息接口中返回，字段名如下：

campusId: 学工号

departmentCode: 部门编号

departmentName: 部门名称

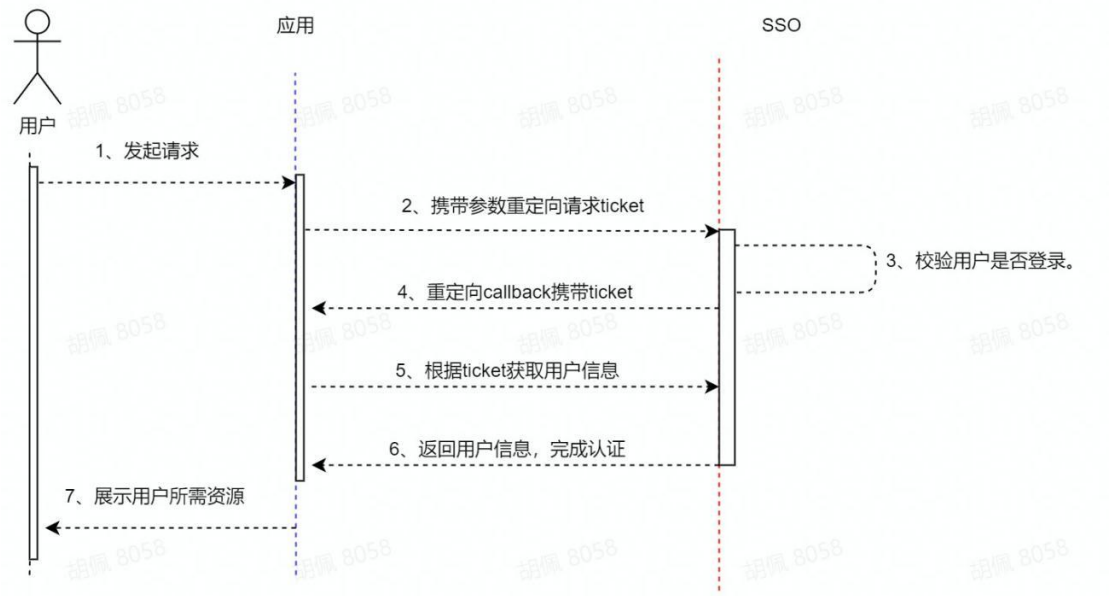
identity: 用户类型

name: 用户姓名

umsStatus: 在校状态（返回中文状态名）

2.2 CAS 认证

2.2.1 认证流程



1. 用户浏览器访问应用 CAS 入口地址，然后携带回调地址（service 参数）跳转到统一认证页面。

2. 如用户未登录，则在统一认证页面完成登录；如用户已为登录态，则无需再次登录。之后，认证服务为本次认证生成 Ticket（票根）。

3. 用户浏览器携带 Ticket 跳转到 service 参数所在的地址。

4. 应用获取 Ticket，通过认证校验接口检查 Ticket 合法性，并获取用户信息。

5. 校验成功后并获取用户信息后，认证完成。应用可临时缓存用户基本内容。在登出前，应用可通过会话或其他方式识别当前登录用户。

2.2.2 接口说明

以下是 CAS 认证中所涉及的接口。参数中黑体字为必填参数，否则为可选参数。

认证登录	【GET】 ${SSO_URL}/esc-sso/login
参数	**service**：应用的回调地址，即${APP_CALLBACK}
返回	认证成功后，重定向到应用回调地址

应用回调（由应用提供）	【GET】 ${APP_CALLBACK}
参数	**ticket**: 本次认证的票根
处理过程与结果	校验 ticket 成功后跳转到应用业务页面

检查票根	【GET】 ${SSO_URL}/esc-sso/serviceValidate 【GET】 ${SSO_URL}/esc-sso/p3/serviceValidate (兼容接口)
参数	**ticket**: 本次认证的票根 **service**: 在认证登录接口中传递的地址 format: 返回格式，默认为 xml，可传 json
返回	返回票根有效性和用户基本信息。用户基本信息为申请表中所选择的，默认仅返回 NetID。 示例： <cas:authenticationSuccess> <cas:user>用户 NetID</cas:user> <cas:attributes> <cas:departmentName>部门名称 </cas:departmentName> <cas:isFromNewLogin>false</cas:isFromNewLogin> <cas:authenticationDate> 2024-08-01T00:56:30.649Z[UTC] </cas:authenticationDate> <cas:name>姓名</cas:name> <cas:campusId>012345</cas:campusId> …… </cas:attributes> </cas:authenticationSuccess>

2.2.3 接入方法与示例

一般情况下，使用 Java 体系框架的应用无需手动实现认证的全流程。下面以 Java 应用为例，说明集成 CAS Client 组件（jar 包）实现单点登录的方法。

1. Maven 引入 CAS Client

```
<dependency>
    <groupId>org.apereo.cas.client</groupId>
    <artifactId>cas-client-core</artifactId>
    <version>x.x.x</version>
</dependency>
```

旧版本的 groupId 为 org.jasig.cas.client，请根据实际情况选择对应版本。非 Maven 项目请参考对应的包管理方法。

2. 添加对应 Filter

org.apereo.cas.client.authentication.AuthenticationFilter

org.apereo.cas.client.validation.Cass30ProxyReceivingTicketValidationFilter

(或 Cas20ProxyReceivingTicketValidationFilter)

org.apereo.cas.client.session. SingleSignOutFilter

（注：旧版本对应包路径为 org.jasig.cas.client，下同）

添加 Filter 的方法有很多，最原始的是通过项目 web.xml 添加：

SingleSignOutFilter：

```
<filter>
    <filter-name>CAS Single Sign Out Filter</filter-name>
    <filter-class>org.apereo.cas.client.session.SingleSignOutFilter</filter-class>
</filter>
...
<filter-mapping>
    <filter-name>CAS Single Sign Out Filter</filter-name>
    <url-pattern>/cas/logout(根据应用实际情况修改)</url-pattern>
</filter-mapping>
...
<listener>
```

```
    <listener-class>
        org.apereo.cas.client.session.SingleSignOutHttpSessionListener
    </listener-class>
</listener>
```

AuthenticationFilter：

```
<filter>
  <filter-name>CAS Authentication Filter</filter-name>
  <filter-class>
    org.apereo.cas.client.authentication.AuthenticationFilter
  </filter-class>
  <init-param>
    <param-name>casServerUrlPrefix</param-name>
    <param-value>${SSO_URL} /esc-sso/login</param-value>
  </init-param>
  <init-param>
    <param-name>serverName</param-name>
    <param-value>http(s)://ip:port（根据应用实际情况修改）</param-value>
  </init-param>
</filter>
<filter-mapping>
    <filter-name>CAS Authentication Filter</filter-name>
    <url-pattern>/cas（根据实际情况修改）</url-pattern>
</filter-mapping>
```

Cas30ProxyReceivingTicketValidationFilter：

```
<filter>
  <filter-name>CAS Validation Filter</filter-name>
  <filter-class>
    org.apereo.cas.client.validation.Cas30ProxyReceivingTicketValidationFilter
  </filter-class>
```

```
        <init-param>
            <param-name>casServerUrlPrefix</param-name>
            <param-value> ${SSO_URL}/esc-sso</param-value>
        </init-param>
        <init-param>
            <param-name>serverName</param-name>
            <param-value> http(s)://ip:port（根据应用实际情况修改）</param-value>
        </init-param>
    </filter>
    <filter-mapping>
        <filter-name>CAS Validation Filter</filter-name>
        <url-pattern>/cas（根据应用实际情况修改，通常只需要拦截入口地址）
</url-pattern>
    </filter-mapping>
```

其他结构的项目请参考对应的配置方法。

3. 访问 Cas Authentication Filter 所拦截的请求，即自动跳转到认证页面，后续流程将由 CAS Client 处理。

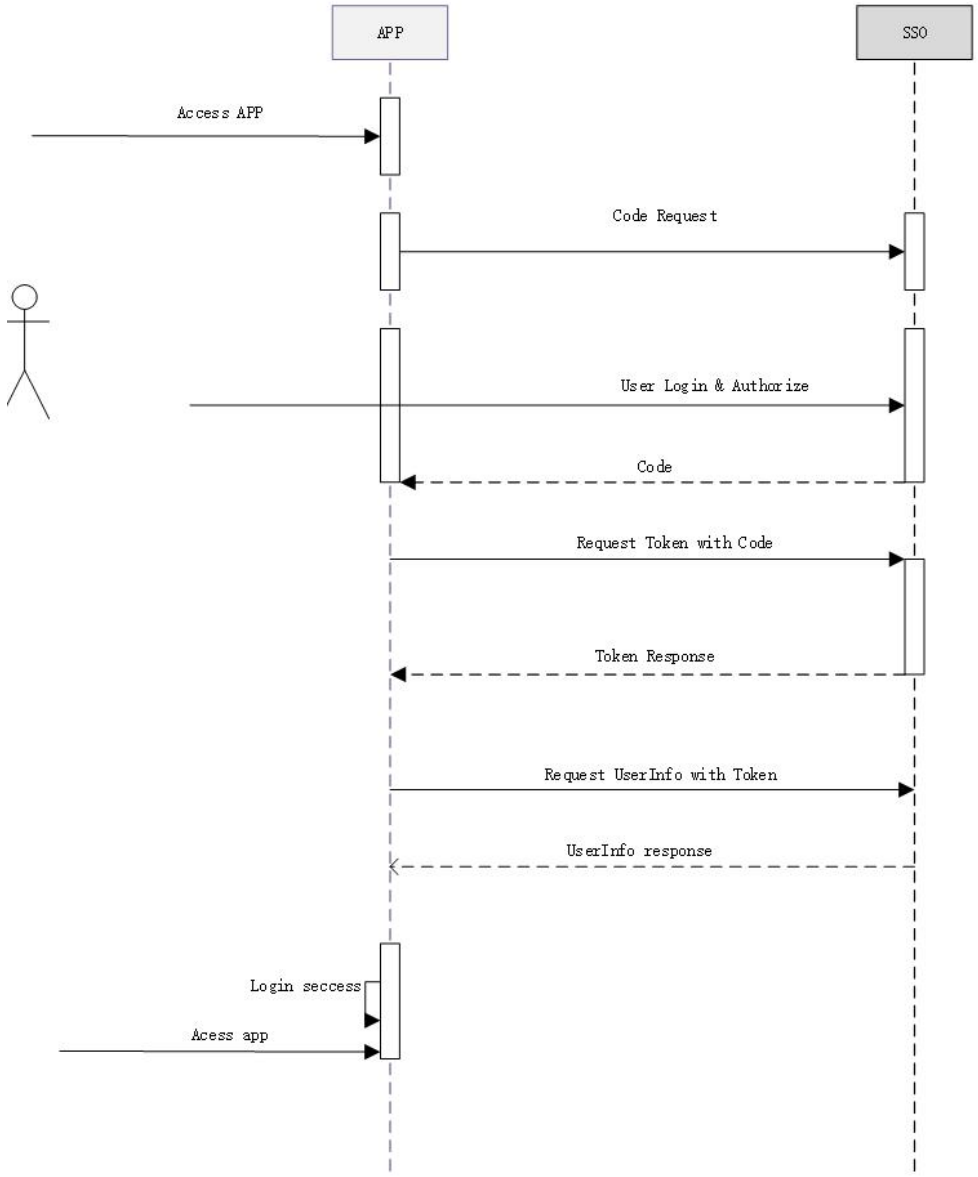
更多的配置方法请参考官方文档（https://apereo.github.io/cas，客户端配置部分）。如本文档阐述的内容与官方文档有冲突，请以官方文档为准。

除了集成 CAS Client 进行对接外，还可以使用 Spring Boot、Spring Security CAS 等组件简化开发流程，如需集成，请开发者参考官方文档。

2.3 Oath2.0/OIDC 认证

2.3.1 认证过程

用户、应用系统和统一认证服务的交互过程遵从 OAuth2 授权码模式，如下图所示：



1. 用户浏览器访问应用系统首页，可手动或自动重定向到 OAuth2.0/OIDC 认证登录页面（获取 Code 请求）。

2. 认证后，由统一认证服务生成 Code。

3. 用户浏览器将携带 Code 跳转到应用回调地址。

4. 应用系统获取 Code 后，访问 Token 接口获取 Access Token 和 Refresh Token（OIDC 协议则是 ID Token）。

5. 应用携带 Access Token 访问用户信息接口获取数据，认证完成。对于 OIDC 协议，可直接解释 ID Token 获取用户 NetID，但不含用户姓名、部门等其他信息。如需其他信息，则仍然需要携带 Access Token 访问用户信息接口。

2.3.2 接口说明

以下为 OAuth2.0/OIDC 认证过程中的接口描述。参数中黑体字为必填参数，否则为可选参数。

认证登录（获取授权码）	【GET】 ${SSO_URL}/esc-sso/${PROTOCOL}/authorize
参数	**client_id**: 应用 ID，申请后由网络中心提供 **response_type**: code，表示获取授权码 **redirect_uri**: 应用回调地址，即认证后接收授权码的地址，必须和申请表上的一致 state: 额外参数，应用的状态标识（可传地址或状态码，地址需转移），将原样传递到回调地址中
返回/结果	认证成功后，用户浏览器携带授权码重定向到应用回调地址

应用回调（由应用提供）	【GET】 ${APP_CALLBACK}
参数	**code**: 本次认证的授权码 state: 应用在授权码接口中所传递的参数
处理过程与结果	应用在该接口中，应调用 Token 接口获取访问 Token，然后再调用用户接口获取用户信息（使用 OIDC 协议的可酌情调用），成功后跳转到应用业务页面

获取 Token	【POST】 ${SSO_URL}/esc-sso/${PROTOCOL}/accessToken
参数	**grant_type**: authorization_code，表示根据授权码获取 Token **client_id**: 应用 ID，申请后由网络中心提供 **client_secret**: 应用秘钥，申请后由网络中心提供 **code**: 回调接口中接收的授权码 **redirect_uri**: 与授权码接口中的参数一致
返回/结果	JSON 格式数据

	{
	"access_token": "……",
	"token_type": "bearer",
	"expires_in": 600,
	"id_token": "……", // 选用 OIDC 时返回
	"refresh_token": "……" // 按需返回
	}
	id_token 为 JWT，解释后请对比签名，以防止篡改。详细的解释过程请参看 RFC 7519 或其他有关文档

获取用户信息	【GET】 ${SSO_URL}/esc-sso/${PROTOCOL}/profile
说明	如果只需要获取 NetID，则使用 OIDC 获取 ID Token 后即可，无需再次调用本接口
参数	**access_token:** Token 接口中获取到的值 或在 Header 中使用 Bearer Token 的方式传递

返回/结果	JSON 格式数据，默认只返回 NetID，其他信息需在申请表中勾选
	OAuth2.0 协议返回：
	{
	"attributes": {
	"token_gtime": 1763624431092,
	"token_expired": "600"
	… // 申请表中勾选的其他属性
	},
	"id": "…" // NetID
	}
	OIDC 协议返回：
	{

	"sub": "…", // NetID "auth_time": 1762846806, … // 申请的其他属性 }

2.3.3 接入方法与示例

使用 Spring 框架的应用无需编写 OAuth2.0/OIDC 协议的全过程，集成 Spring Security OAuth2.0 Client 组件可简化对接过程。使用非 Java 体系框架或非 Spring 框架的应用，可参考对应的 OAuth2.0/OIDC 协议对接方法。

下面以 Maven、Spring 框架应用为例，说明实现 OAuth2.0/OIDC 协议的方法。

1. Maven 引入

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-oauth2-client</artifactId>
    <version>${spring-boot.version}</version>
</dependency>
```

2. 在 application.yml 配置协议参数（也可以是其他格式的配置文件）

```
security:
    oauth2:
      client:
        registration:
          sysu:
            client-id: ...
            client-secret: ...
            authorization-grant-type: authorization_code
            redirect-uri: "{baseUrl}/login/oauth2/code/sysu"
            scope: openid,profile
        provider:
```

```
sysu:
  issuer-uri: https://iamsso.sysu.edu.cn/esc-sso/oidc
```

运行程序，即可对接统一认证服务。另外，本方法配套的示例程序托管在网络与信息中心 GitLab 中（http://172.22.43.132/inc/authentication-demo）。如有需要，可申请账号获取示例代码。