用Python建设企业认证和权限控制平台

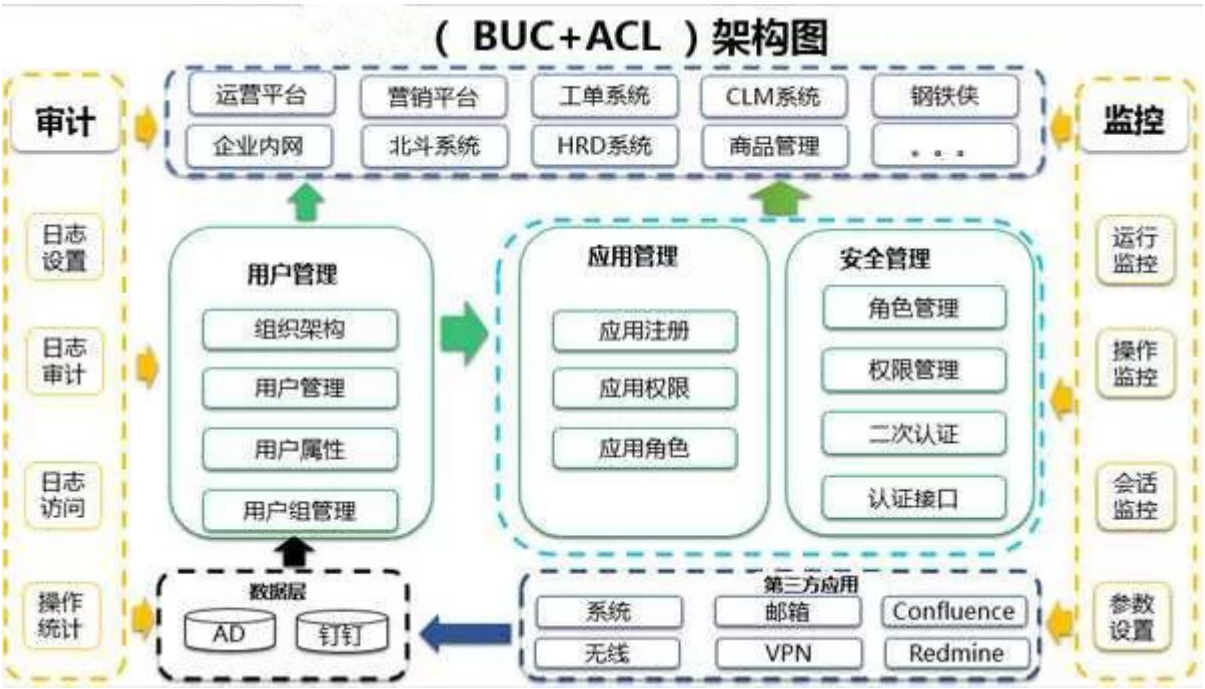
> 程晓飞，Python中文社区专栏作者，高级全栈开发工程师 。2016年加入五阿哥钢铁电商平台，曾供职于知名500强外企，精通Python，Java等多种编程语言及软件开发技术，在企业内部Web系统、自动化工具开发方面有多年经验。

目前大家对Python的了解更多来源是数据分析、Ai、运维工具开发，在行业中使用Python进行web开发，同样也是非常受欢迎的，例如：FaceBook，豆瓣，知乎，饿了么等等，本文主要介绍利用Python进行web开发企业统一用户认证和权限控制平台，提供用户管理、认证和权限接入的能力，避免各个系统重复建设造成资源的浪费。

企业内网，建立在企业内部，为员工提供信息的共享和交流，为业务提供运营和管理的支撑，已是当今企业信息化建设必不可少的一个项目。随着企业的规模越来越大，业务越来越广，系统建设就显得尤为重要。

统一认证系统是企业内网系统建设的基础，主要实现用户管理、身份认证、权限管理和单点登录等功能，以解决企业内网系统建设过程中用户定义模糊、用户身份组织零乱、交叉权限管理和应用系统出口多样性等棘手的问题。

基于此，笔者所在运维技术团队基于Python建设了统一认证（BUC：Back User Center）和权限控制（ACL：Access Control List）平台，这个系统技术架构图如下：



系统的功能

1.用户管理

在企业中，每个用户都有一个唯一的账号进行登录，用户的账号和个人身份信息（包含姓名、邮件等公共属性）会集中保存在内网统一认证系统里。但对于同一个用户在外部系统中的账号，如微信、钉钉、Tower等第三方系统，统一认证系统也可以通过定时同步或实时查询等方式获取到用户的信息。

2.安全管理

对于保存在企业内网中的用户账号进行认证是比较方便的，对于保存在外部系统中的用户账号，可以通过定时同步或实时验证等方式来验证用户的鉴权，最终统一以唯一的身份使用户登录进入系统。

同时，企业内部存在多个不同的业务应用，以所有应用均接入统一认证平台为基础，各个业务系统的用户认证采用单点登录认证模式，一次登录即可在各个业务子系统中完成自动认证并获得相关授权。

3.应用管理

不同的业务系统会对用户进行不同的角色划分，不同的角色又会划分出细粒度的权限。角色权限在统一认证系统中保存，在一个地方就可以完成用户授权信息的设定。

4.系统接入

支持各种技术栈的系统接入，包含Java和Python等，其中各技术栈的SDK可以在Github项目中找到。

5.三方应用

第三方应用，包含邮箱，VPN，无线等直接进行SSO登录。

6.系统监控

对系统运行、操作、会话进行监控，保障在受到外部或者内部攻击时，能够及时发现，进行实例回溯。

7.系统审计

对平台所有操作进行审计，在出现系统权限错乱或者安全问题的时候，对平台操作进行审计。

系统架构设计

1. 接口设计

企业内网的统一认证平台建议基于B/S模式设计，后端使用Django框架以快速开发，用DB+LDAP方式完成用户各类信息的存储，保障存储和查询效率。 统一认证的核心问题是鉴权中心和各子系统之间的通信接口问题，用户认证接口协议可以基于标准化HTTP/HTTPS方式实现，并对外提供不同语言的SDK（如Python CAS库、Java Web过滤器等），使得第三方业务系统的接入不完全依赖于特定的开发环境。

2. 安全设计

对于接入系统，认证中心接口协议调用采用HTTPS传输的方式，通信安全问题将转化到HTTPS传输的安全性问题上，而对于HTTPS通道的攻击，可以由单独的网络扫描模块专门负责监控。

对于统一认证和SSO接口参数的信息安全，一方面网站可采用专有加密算法对参数内容进行加密，另一方面，可以采用IP认证策略来保证对接口双方的信任，系统通过通道安全和信息加密双保险的措施来保证统一认证体系的接口安全。

同时系统配有全方面的应用监控和访问日志的审计，当机器发生异常情况或日志审计检测到有可疑入侵行为时，会自动以多种方式通知到运维工程师和相关负责人。

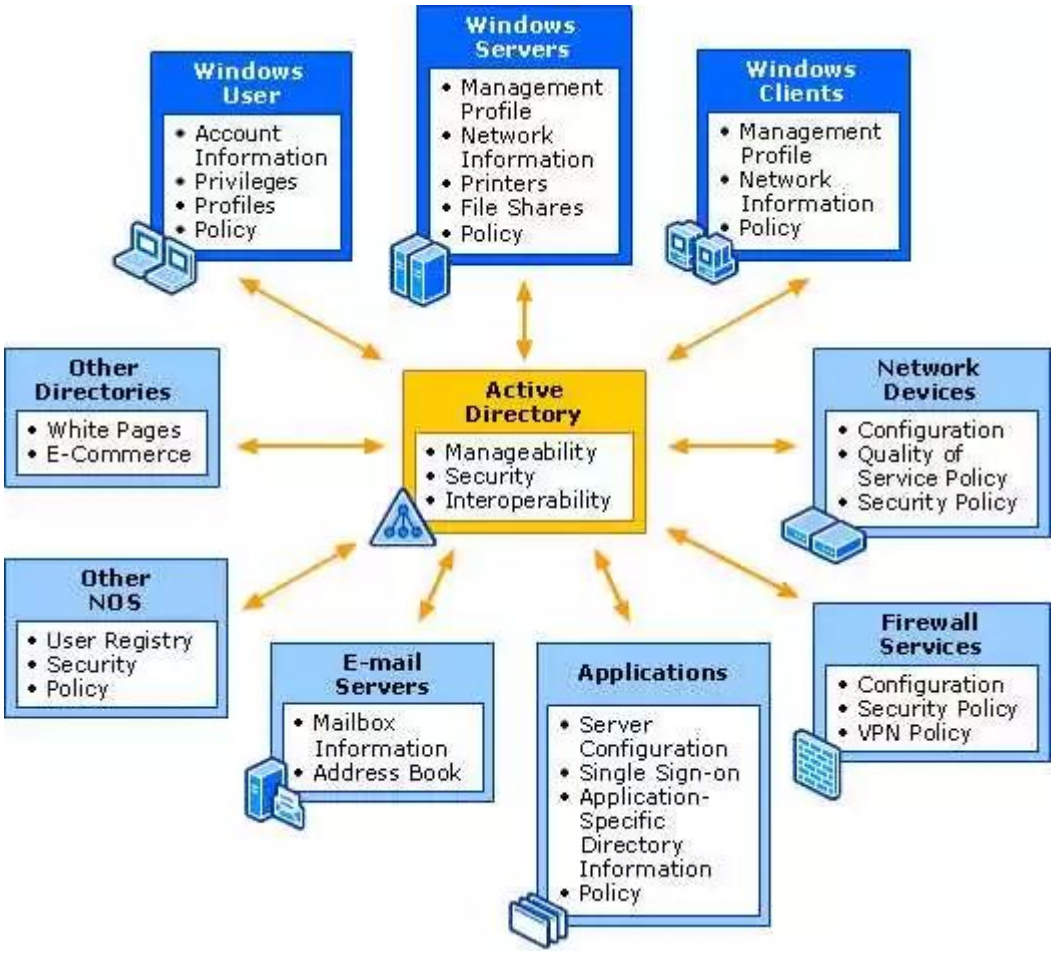
BUC使用技术和实现

1. 用户管理

通常在企业中，每个用户拥有一个唯一的身份标识，即用户名。同时用户在其他内网或外部应用也存在着对应的用户，如果能使用同一个用户身份最为方便；独立的用户可以通过关联不同的系统的用户使之对外呈现为一个用户。用户可以在不同的应用系统中使用，这一切的基础是有一个中央的系统来保存和管理这些用户。常见的解决方案有Windows活动目录和LDAP。

（1）Windows活动目录域服务

使用 Active Directory(R) 域服务 (AD DS) 服务器角色，可以创建用于用户和资源管理的可伸缩、安全及可管理的基础机构，并可以提供对启用目录的应用程序（如 Microsoft(R) Exchange Server）的支持。



AD的功能，来自微软的介绍

AD DS 提供了一个分布式数据库，该数据库可以存储和管理有关网络资源的信息，以及启用了目录的应用程序中特定于应用程序的数据。 运行 AD DS 的服务器称为域控制器。 管理员可以使用 AD DS 将网络元素（如用户、计算机和其他设备）整理到层次内嵌结构。内嵌层次结构包括 Active Directory 林、林中的域以及每个域中的组织单位 (OU)。

域模式的最大好处就是单一的网络登录能力，用户只要在域中有一个账户，就可以在整个网络中漫游。活动目录服务增强了信任关系，扩展了域目录树的灵活性。活动目录把一个域作为一个完整的目录，域之间能够通过一种基于Kerberos认证的可传递的信任关系建立起树状连接，从而使单一账户在该树状结构中的任何地方都有效，这样在网络管理和扩展时就比较轻松。

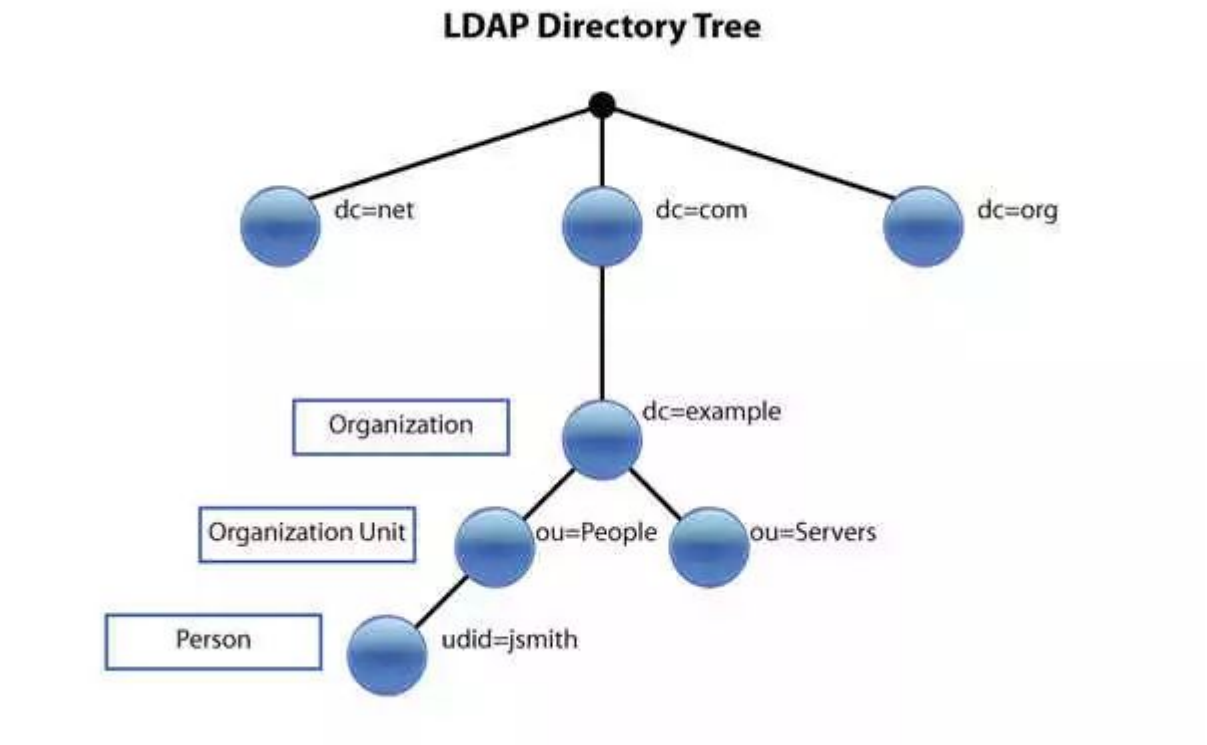
同时，活动目录服务把域又详细划分成组织单元。组织单元是一个逻辑单元，它是域中一些用户和组、文件与打印机等资源对象的集合。组织单元中还可以再划分下级组织单元，下级组织单元能够继承父单元的访问许可权。每一个组织单元可以有自己单独的管理员并指定其管理权限，它们都管理着不同的任务，从而实现了对资源和用户的分级管理。活动目录服务通过这种域内的组织单元树和域之间的可传递信任树来组织其信任对象，为动态活动目录的管理和

扩展带来了极大的方便。

（2）LDAP技术

轻型目录存取协定（英文：Lightweight Directory Access Protocol，缩写：LDAP）是一个开放的，中立的，工业标准的应用协议，通过IP协议提供访问控制和维护分布式信息的目录信息。

目录服务在开发内部网和与互联网程序共享用户、系统、网络、服务和应用的过程中占据了重要地位。例如，目录服务可能提供了组织有序的记录集合，通常有层级结构，例如公司电子邮件目录。同理，也可以提供包含了地址和电话号码的电话簿。



LDAP目录树图示，来自网络

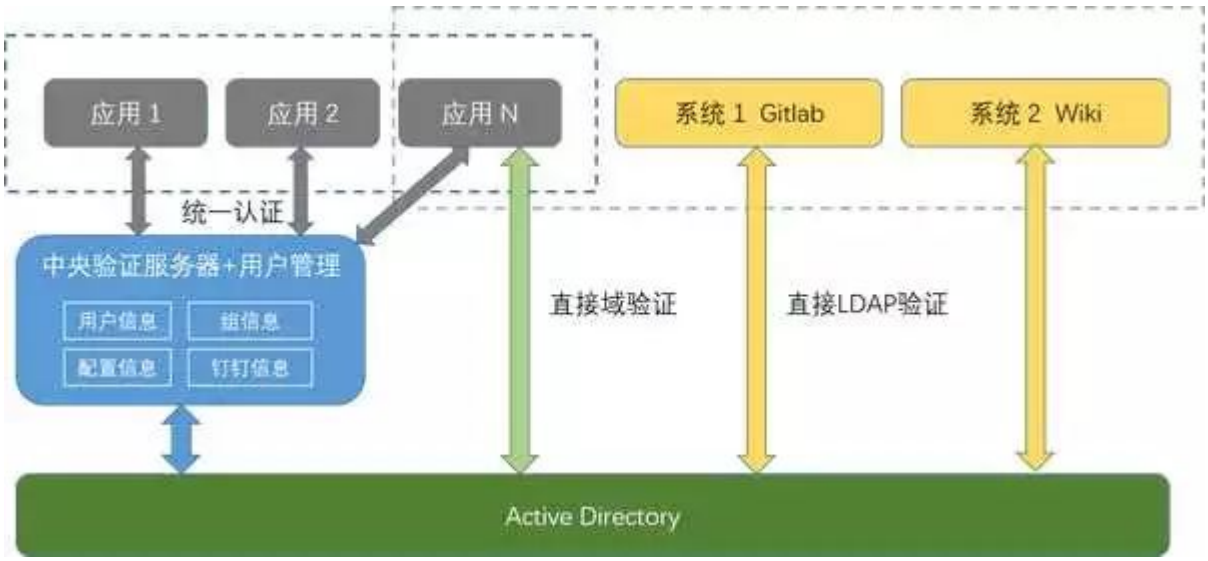
LDAP协议是跨平台的和标准的协议，因此应用程序就不用为LDAP目录放在什么样的服务器上操心了。实际上，LDAP得到了业界的广泛认可，因为它是Internet的标准。厂商都很愿意在产品中加入对LDAP的支持，因为他们根本不用考虑另一端（客户端或服务端）是怎么样的。

LDAP服务器可以是任何一个开放源代码或商用的LDAP目录服务器（或者还可能是具有LDAP界面的关系型数据库），因为可以用同样的协议、客户端连接软件包和查询命令与LDAP服务器进行交互。

与LDAP不同的是，如果软件厂商想在软件产品中集成对DBMS的支持，那么通常都要对每一个数据库服务器单独定制。不像很多商用的关系型数据库，你不必为LDAP的每一个客户端连接或许可协议付费。大多数的LDAP服务器安装起来很简单，也容易维护和优化。

（3）用户管理实践

根据国内企业办公网络的实际情况，用户计算机通常为Windows系统，通通接入到Windows活动目录中进行管理。而且Windows活动目录兼容LDAP协议，我们使用活动目录作为统一保存用户信息的中央系统，再通过LDAP协议使用程序访问域控制器将用户信息同步到统一认证服务器中。



企业内部系统基本结构

由上图架构所示，一方面常见的办公系统（如代码仓库、Wiki等）自身即支持LDAP认证，通过配置Windows AD中的目录/用户搜索规则即完成对登录用户的认证；另一方面自行开发的业务系统通过中央认证服务器提供的接口间接的对Windows AD进行登录用户的认证，即一个用户，一套密码，在多个系统中都可使用。

```
# auth.py
class ActiveDirectoryAuthenticationBackEnd:
    def authenticate(self, username, password, request=None):
        if not username or not password:
            return None
        try:
            # 建立同域控的连接
            server = Server(settings.AD_SERVER_NAME, use_ssl=True)
            # 验证用户使用的用户名和密码
            conn = Connection(server, "%s\\%s" % (settings.AD_DOMAIN, username), p
assword, auto_bind=True,                              authentication=NTLM)
            # 认证通过，返回系统中注册的用户，不存在则创建
            user = conn.bound and self.get_or_create_user(username, conn) or None
            return user
        except LDAPBindError:
            return None

    def get_or_create_user(self, username, conn=None):
        try:
            user = User.objects.get(username=username)
        except User.DoesNotExist:
            if conn is None:
                return None
            # 从域控向中央认证系统中同步注册一个新用户
            if conn.search(USER_BASE_DN, "(sAMAccountName=%s)" % username, SUBTRE
E, attributes=["sn", "givenName"]):
                result = conn.response[0]["attributes"]
                user = User(username=username)
                user.first_name = result["givenName"]
                user.last_name = result["sn"]
                user.email = "%s@%s" % (username, settings.AD_EMAIL_HOST)
                user.save()
            else:
                return None
        return user
```

2. 身份认证

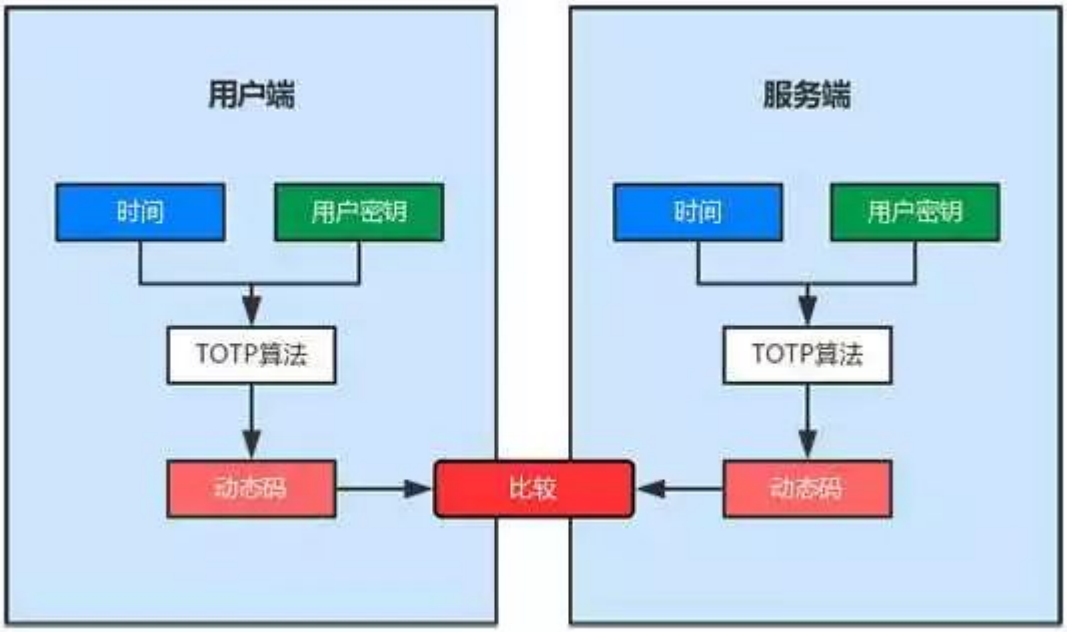
（1）通过外部应用认证

外部应用，如即时通讯软件钉钉等，这些应用存有单独的一套用户凭证，通过应用提供的免登服务，将应用中的用户与统一认证服务器中的用户进行一一对应，当用户在外部应用中登录后，自动获得在企业内应用的已登录状态。

（2）通过TOTP动态验证码认证

OTP (One-Time Password)，一次性密码，也称动态口令。它是使用密码技术实现在客户端和服务器之间共享秘密的一种认证技术，是一种强认证技术，是增强目前静态口令认证的一种非常方便的技术手段，是一种重要的双因素认证技术。

TOTP (Time-base One-Time Password)，基于时间的一次性密码，也称时间同步的动态密码。当在一些用户不方便输入密码或者忘记密码的场景中，我们可以使用TOTP进行认证。服务器和用户各自保管共同的密钥，通过比对基于时间分片与哈希计算出的动态数字验证码即可完成对用户身份的认证。主流实现为Google Authenticator（Google身份验证器），阿里的身份宝也兼容该算法。



TOTP算法图示

（3）双因子认证

双因子认证（Two-Factor Authentication）是指结合密码以及实物（信用卡、SMS手机、令牌或指纹等生物标志）两种元素对用户进行认证的方法。

动态验证码流程图示

结合上面使用的TOTP验证码，对于安全级别较高的应用或资源路径、或是系统探测到风险较高的操作时，即可以对用户重定向至双因子认证页面，进一步保障系统安全。

3. 单点登录

主要实现方式：

（1）共享 cookie

利用同一域名下的cookie共享为基础，将session id写入共享cookie，在实现了后台session共享存储和访问后，不同的应用之间即实现了单点登录。

（2）Broker-based （基于经纪人）

在一个基于经纪人的 SSO 解决方案中，有一个集中的认证和用户帐号管理的服务器。经纪人能被用于进一步请求的电子的身份存取。中央数据库的使用减少了管理的代价，并为认证提供一个公共和独立的"第三方"。例如 Kerberos 、 Sesame 、 IBM KryptoKnight （凭证库思想) 等。

（3）Agent-based（基于代理人）

在这种解决方案中，有一个自动地为不同的应用程序认证用户身份的代理程序。这个代理程序需要设计有不同的功能。比如，它可以使用口令表或加密密钥来自动地将认证的负担从用户移开。代理人被放在服务器上面，在服务器的认证系统和客户端认证方法之间充当一个 " 翻译 " ，例如 SSH 等。

（4）Token-based

口令认证，比如 FTP 、邮件服务器的登录认证，这是一种简单易用的方式，实现一个口令在多种应用当中使用。

- 基于网关
- 基于 SAML
- Ticket-based（基于票据）

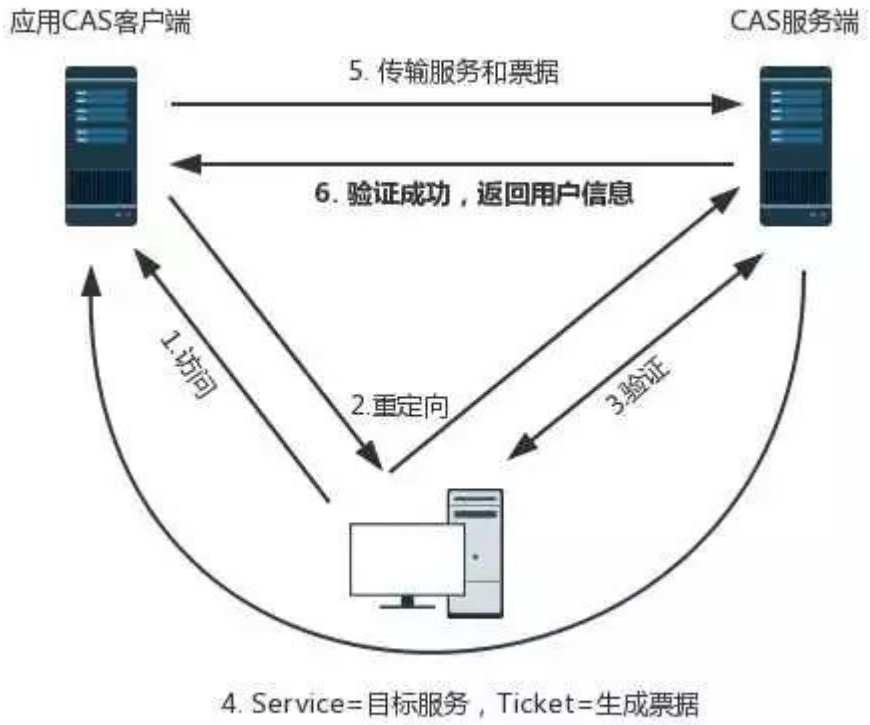
4. BUC实践

在我们的内网应用中，最终选择了CAS协议作为单点登录的方案。CAS（Central Authentication Service）是 Yale 大学发起的一个企业级的、开源的项目，旨在为 Web 应用系统提供一种可靠的单点登录解决方法。CAS开始于2001年，并在2004年12月正式成为JA-SIG的一个项目。

CAS的主要特点有：

- 开源
- 支持多种认证机制：Active Directory、JAAS、JDBC、LDAP、X.509等
- 安全策略：使用票据（Ticket）来实现支持的认证协议
- 支持授权：可以决定哪些服务可以请求和验证服务票据
- 提供高可用性
- 支持多种客户端及SDK： Java， .Net，PHP，Python，nodejs 等
- 服务端也有多种语言实现

（1）登录验证流程



用户、CAS客户端、服务端三方交互过程

```
# urls.py
# 在配置文件中将/cas 路径挂载至根路径，即可通过/cas/login来访问中央认证页面，后台的认证过程会通过已有的认证服务进行
urlpatterns = [
……
url(r"^cas/", include("mama_cas.urls", namespace="cas")),
……
]
```

（2）安全扩展

当CAS服务端完成了对用户和CAS客户端的验证之后，CAS服务端将验证后的用户信息传输给CAS客户端（目标应用），同时也可根据配置返回该应用下的附属用户信息，如用户拥有的该应用下的角色、权限和属性。目标应用根据服务器返回的用户信息进一步检查用户可访问的资源，适当的展示业务视图。

```
# settings.py
MAMA_CAS_SERVICES = [
    {
        'SERVICE': '.*',
        'CALLBACKS': [
            'base.cas_callbacks.user_profile_attributes',
            'security.cas_callbacks.user_security_attributes',
# 按需加入回调函数
        ]
    }
]
# user_profile_attributes
# 返回用户的基本信息
def user_profile_attributes(user, service):
return {
“username”: get_username(user),
“name”: get_display_name(user)
}
# user_security_attributes.py
# 返回用户在该业务/服务中拥有的角色、权限和访问路径信息
def user_security_attributes.py(user, service):
return {
    “roles”: get_roles(service, user),
“permissions”: get_permissions(service, user),
“urls”: get_urls(service, user)
}
```

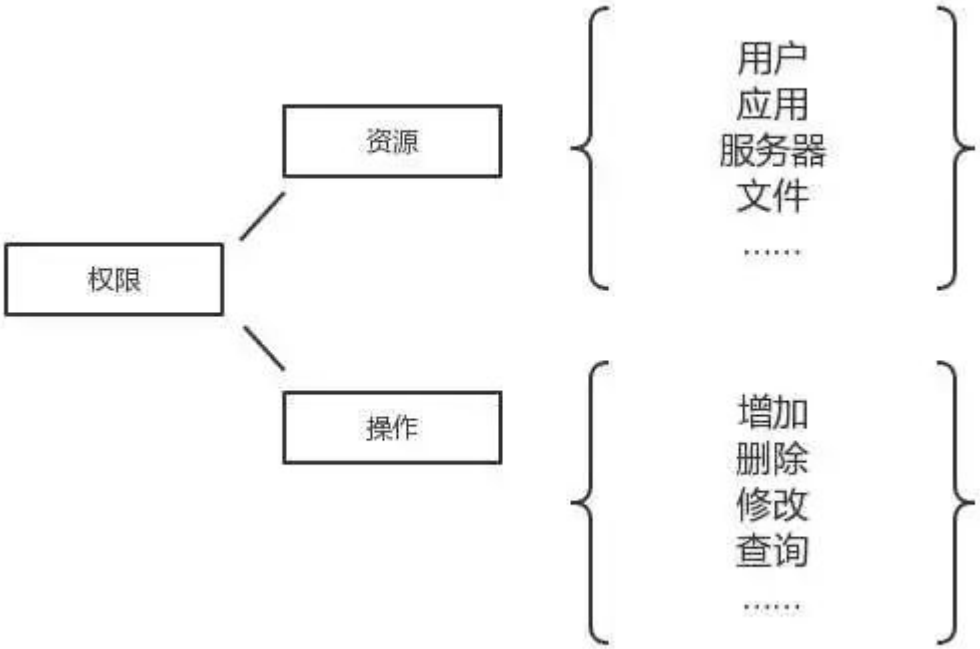
ACL使用技术和实现

在现代企业，尤其是互联网企业中，产品业务繁多，对数据安全、访问控制都提出了很高的要求，基于用户组织结构、汇报线等传统的分组模式已经无法适应和满足多变的互联网扁平化管理模式的需要，因此我们选择了基于角色和权限的动态分组来设计和实现企业中不用应用可以共享的安全访问管理系统。

1. 权限

权限是针对资源和操作层面的最小安全访问控制单元，例如：

- 按资源分，可以设置访问设备A、访问设备B等。
- 按操作分，可以设置读取文件，写入文件等。

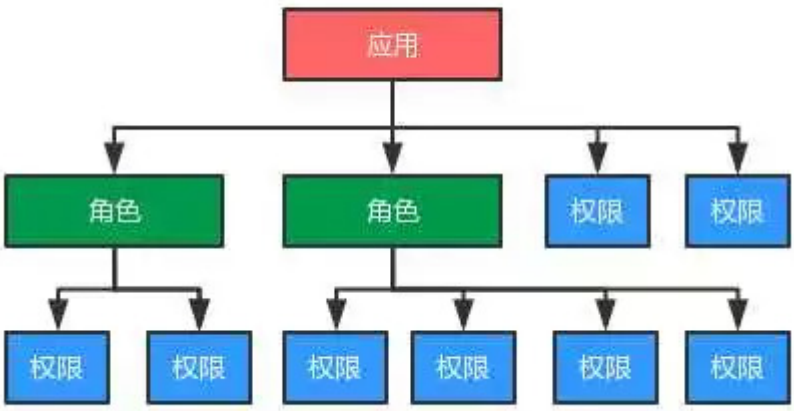


权限分类示意图

2. 角色

角色是针对应用使用者来设置的，可分为管理员、技术人员，普通用户等，也可按区域分为华北员工、华南员工等。

角色是一系列权限的集合，拥有某角色的用户即应当自动拥有该角色下包含的权限。

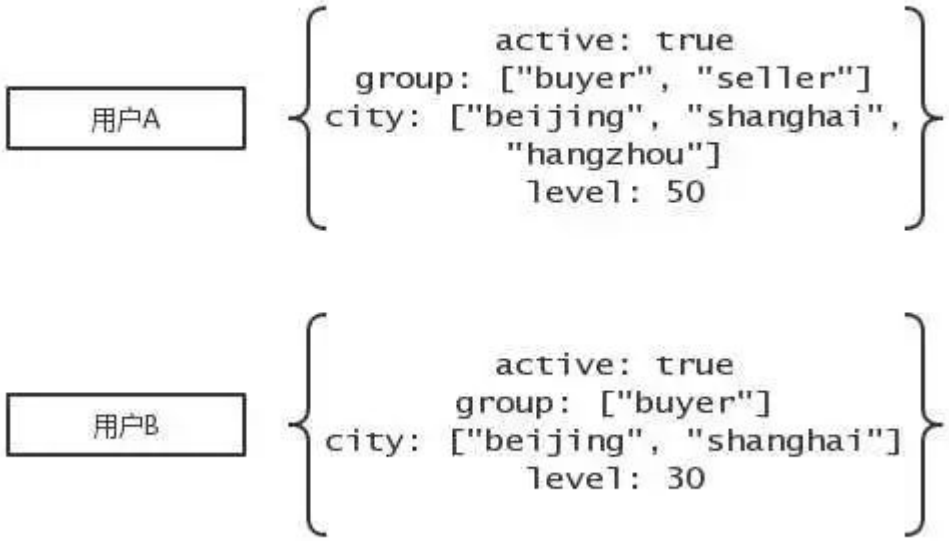


角色与权限关系示意图

3. 属性

属性是针对用户层面下设置的独立的安全设置，用来扩展和实现更细粒度的自定义安全设置数据，如将可访问数据细化到数据库中的表、数据表中的行、列上。

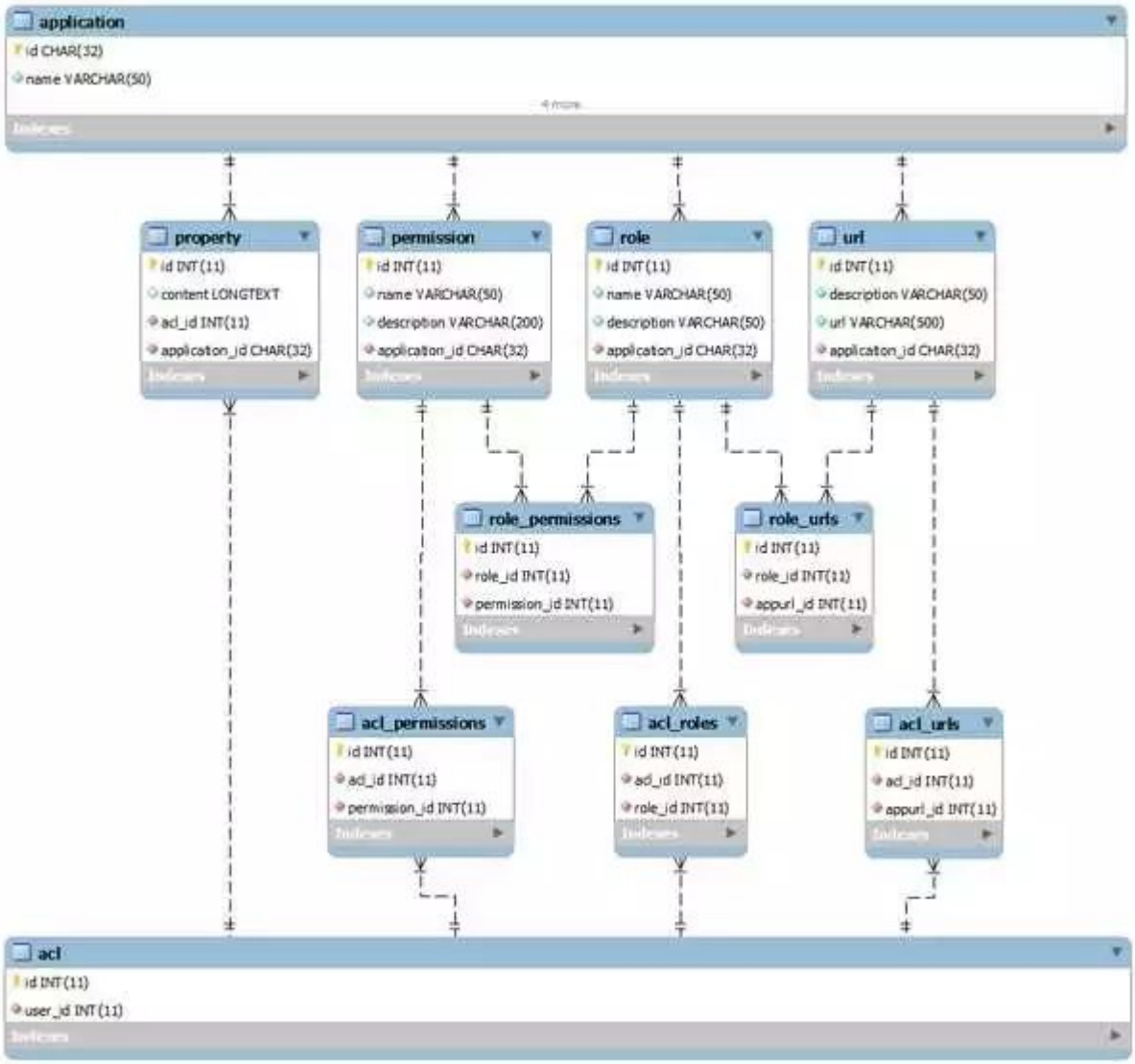
得益于JSON的兼容性，可以很灵活的存储下这些自定义的结构化数据。



用户属性示意图

4. ACL实践

数据库建模



依模型图可以看出，一个应用可划分多个角色、权限、路径和属性，其中角色又可包含同应用下的权限和路径。一个用户对应一个ACL，通过将不同的控制单元授予用户，即可完成用户的访问控制配置。

配套功能设计

为了使访问控制的整套机制良好的运转起来，相关辅助和配套的功能也是不可缺少的，这里列举一些我们已经投入使用的功能：

（1）应用授权的分级、分组管理

不用业务应用的负责人可以分别对自己负责的业务进行授权管理，不会产生冲突和越权。

（2）应用菜单可见性、可访问性的集成

业务应用中的各子功能可以和预先设置的权限一对一或一对多映射，具有相应权限的用户才可以访问和使用相应的功能，前后台设置保持同步。

（3）应用下角色权限申请提交、授权变更、授权完成等自动化流程

基本的权限审批流减小了业务应用负责人和使用者之间的沟通成本，同时也记录了权限获取的记录，为日后的安全审计提供了可查的数据。

（4）应用访问日志收集、分析、审计、报警等

应用访问日志记录了更为详细的用户访问和操作记录，为安全审计提供了更完备的数据支持，同时也支持以一定的逻辑来分析和发现潜在的安全泄露风险。

总结

本文总结介绍了针对企业内网门户的统一用户管理、认证和授权管理的系统的组成部分和常见实现方法。使用开源CAS产品搭建的统一身份认证系统和定制化开发的安全访问控制系统在企业内网平台上得到了很好地实践和应用，各部门的业务系统也已稳定接入并使用，目前运行良好。随着系统规模和业务的增长，这一套平台仍可能会面临新的问题和挑战，这也使得我们在收集用户反馈的同时不断的进行重构和增强，以保障企业业务的稳定发展。

项目地址：

https://github.com/cangelzz/cas-demo-django-server

https://github.com/cangelzz/cas-demo-flask-client

https://github.com/cangelzz/cas-demo-java-client