Table of Contents

Django-auth-ldap中文文档

本文档基于django-auth-ldap1.2.8翻译而来，各位看官注意版本。

另外只翻译到了配置示例这一节，下面的“引用”以及“变更日志”比较琐碎不翻译了。

更多文章，请移步本人博客：www.dear-shen.com

2016.09.08

安装

这个认证后端使Django能够通过任何LDAP进行用户验证，简单的把 `django_auth_ldap.backend.LDAPBackend` 添加到 `AUTHENTICATION_BACKENDS` 即可。不建议把 `django_auth_ldap` 加入到 `INSTALLED_APPS` 中，除非你要进行单元测试。LDAP的配置就像配置模板一样简单，而且提供了丰富的选项对用户、组、权限进行操作，这些功能依赖python-ldap包。

> 注意：`LDAPBackend` 并不继承自 `ModelBackend`，所以单独的为LDAP用户配置一个组是可能的。然而，如果你想为单独的用户指定权限或把用户添加到django组中，你需要以下两种后端同时存在：
>
> ```
> AUTHENTICATION_BACKENDS = (
>     'django_auth_ldap.backend.LDAPBackend',
>     'django.contrib.auth.backends.ModelBackend',
>     )
> ```

认证

服务端配置

如果你的LDAP服务器不是运行在本机以及默认端口，你需要设置 `AUTH_LDAP_SERVER_URI` 来指明这些信息。这些可以被设定为任何你所使用的LDAP库支持的值。比如openldap准许你使用逗号或空格分割的URL列表。

```
AUTH_LDAP_SERVER_URI = "ldap://ldap.example.com"
```

如果你的LDAP服务器URL是动态的并且十分灵活(译者注：意思就是URL在不同情况下会有不同的值)，你可以提供一个函数(或者任何可调用对象)来返回URL。你应该假设每次请求这个函数都会被调用，这是一种昂贵的开销，所以需要缓存来减少开销。

```
from my_module import find_my_ldap_server

AUTH_LDAP_SERVER_URI = find_my_ldap_server
```

如果你需要配置任何关于python-ldap的选项，可以设置 `AUTH_LDAP_GLOBAL_OPTIONS` 和/或 `AUTH_LDAP_CONNECTION_OPTIONS` 。比如禁用referral这种不常见的情况：

```
import ldap

AUTH_LDAP_CONNECTION_OPTIONS = {
    ldap.OPT_REFERRALS: 0
}
```

搜索绑定

现在你可以和LDAP服务器进行通讯了，下一步就是验证用户名和密码。这里有2种方法：搜索式绑定和直接绑定。第一种方式要么涉及匿名账户要么涉及固定账户并且通过DN(distinguished name)搜索用户，然后进行密码验证来绑定用户。第二种方式直接在用户名中指明DN并且尝试直接绑定用户。

因为LDAP的搜索配置在其他地方，所以提供了 `LDAPSearch` 来对搜索信息进行封装。在这里，过滤参数应该包含 `%(user)s` 占位符。一个简单的搜索绑定配置如下（为了完整性而包含了一些默认值）：

```
import ldap
from django_auth_ldap.config import LDAPSearch

AUTH_LDAP_BIND_DN = ""
AUTH_LDAP_BIND_PASSWORD = ""
AUTH_LDAP_USER_SEARCH = LDAPSearch("ou=users,dc=example,dc=com",
    ldap.SCOPE_SUBTREE, "(uid=%(user)s)")
```

联合搜索

> *1.1版新功能*

如果你需要从多个地方来搜索用户，你可以使用 `LDAPSearchUnion` 。这需要多个 `LDAPSearch` 对象并返回结果的集合，默认情况并没指定搜索的优先级：

```
import ldap
from django_auth_ldap.config import LDAPSearch, LDAPSearchUnion

AUTH_LDAP_USER_SEARCH = LDAPSearchUnion(
    LDAPSearch("ou=users,dc=example,dc=com", ldap.SCOPE_SUBTREE,
"(uid=%(user)s)"),
    LDAPSearch("ou=otherusers,dc=example,dc=com", ldap.SCOPE_SUB
TREE, "(uid=%(user)s)"),
)
```

直接绑定

省略搜索过程，直接指定 `AUTH_LDAP_USER_DN_TEMPLATE` 来构造用户的DN信息，这里应该有占位符 `%(user)s` ，如果上面的例子中使用了 `ldap.SCOPE_ONELEVEL` ，那么下面的写法更加直接高效：

```
AUTH_LDAP_USER_DN_TEMPLATE = "uid=%(user)s,ou=users,dc=example,dc=com"
```

注意

LDAP在匹配用户DN信息时候是十分灵活的， `LDAPBackend` 通过在创建django用户时候强制把用户名转换成小写并去除空格来适应这种灵活性。

有些LDAP配置准许用户没有密码，为了防止出错， `LDAPBackend` 默认拒绝任何没有密码的验证请求。你可以通过设置 `AUTH_LDAP_PERMIT_EMPTY_PASSWORD = True` 来关闭这一特性。

默认情况下，对LDAP所有的操作都是以 `AUTH_LDAP_BIND_DN` 和 `AUTH_LDAP_BIND_PASSWORD` 设定的帐号进行的而不是当前用户的。否则LDAP将在登录请求时使用正在验证的用户凭证绑定，而其他请求使用默认用户的凭证。所以根据djanog view的不同你也许会看到不同的LDAP参数。如果你愿意为了获取绑定时用户的参数而接受这种不一致，可以设置 `AUTH_LDAP_BIND_AS_AUTHENTICATING_USER` 参数。

默认情况下，LDAP链接是未加密的并且不尝试去保护比如密码一类的敏感信息。如果和本地网络进行LDAP通讯也许没问题，如果需要进行安全链接你可以使用 `ldaps://` 前缀或者启用StartTLS扩展。一般情况下推荐后者：

```
AUTH_LDAP_START_TLS = True
```

如果 `LDAPBackend` 收到了python_ldap引发的 `LDAPError` 错误，一般情况下仅会记录下这个错误到日志中，如果你需要任何特殊的处理，可以添加信号处理代码到 `django_auth_ldap.backend.ldap_error` 中。这个信号处理代码可以用任何你喜欢的方式处理异常，包括引发其他异常等。

操作组

组类型

在LDAP中操作组可能有些棘手，因为他们的种类实在是太多了。我们提供了一个可扩展的API用来操作组并实现了一些常见的类型。`LDAPGroupType`是所有子类的基类可以用来为特定的分组机制来确定组成员，下面的4个子类覆盖了几种最常见的分组机制：

- PosixGroupType
- NISGroupType
- MemberDNGroupType
- NestedMemberDNGroupType

`posixGroup`和`nisNetgroup`有些特殊，所以他们有自己的类。剩下的2种通过组对象来存储一个由其成员DN组成的列表，包括`groupOfNames`、`groupOfUniqueNames`、`Active Directory groups`等等。这种嵌套的结构准许组包含另一个组，你喜欢嵌套多少层都可以。不过为了更清晰以及可读性，我们提供了下面几个子类：

- GroupOfNamesType
- NestedGroupOfNamesType
- GroupOfUniqueNamesType
- NestedGroupOfUniqueNamesType
- ActiveDirectoryGroupType
- NestedActiveDirectoryGroupType
- OrganizationalRoleGroupType
- NestedOrganizationalRoleGroupType

发现组

开始之前，你需要提供一些关于LDAP组的基本信息。`AUTH_LDAP_GROUP_SEARCH`是一个`LDAPSearch`对象用来识别相关联的组对象。也就是说，我们也许需要知道关于用户都可能属于哪些组的信息(比如嵌套组的

情况)。`AUTH_LDAP_GROUP_TYPE`是一个组类型的实例，这个类型要和`AUTH_LDAP_GROUP_SEARCH`返回的一样。所有其他地方对于组的配置都必须是这个类型并且是搜索结果的一部分。(这段翻译的有些别扭)

```
import ldap
from django_auth_ldap.config import LDAPSearch, GroupOfNamesType

AUTH_LDAP_GROUP_SEARCH = LDAPSearch("ou=groups,dc=example,dc=com"
,
    ldap.SCOPE_SUBTREE, "(objectClass=groupOfNames)"
)
AUTH_LDAP_GROUP_TYPE = GroupOfNamesType()
```

权限控制

最简单的组应用就是控制哪些组的用户可以登录。如果设置了`AUTH_LDAP_REQUIRE_GROUP`那么只有这个组的成员才会登录成功。`AUTH_LDAP_DENY_GROUP`正好相反，这个组的用户登录都会被拒绝。

```
AUTH_LDAP_REQUIRE_GROUP = "cn=enabled,ou=groups,dc=example,dc=co
m"
AUTH_LDAP_DENY_GROUP = "cn=disabled,ou=groups,dc=example,dc=com"
```

如果配置了组信息，则可以使用`user.ldap_user.group_dns`或`user.ldap_user.group_names`来获取用户的组信息。更多信息请看下2章。

用户

控制访问系统对于防止外部来源访问是非常有效的，但Django的认证模块和用户模型紧密相连。在用户登录之后，我们需要创建一个对象在数据库中存储他们。因为LDAP搜索是不区分大小写的，默认搜索Django已经存在用户的实现使用了 `iexact` 查询，并且使用小写的用户名来创建新用户。如果想重写这个方法，可以参考 `get_or_create_user()` ，如果想修改代理模型，可以参考 `get_user_model()` 。

> 注意：在Django1.5之前，用户对象总是User类的实例。现在的版本则支持通过修改 `AUTH_USER_MODEL` 来自定义用户模型。在1.1.4版本后的django-auth-ldap将遵守用户自定义的用户模型。

很明显 `username` 是用户唯一必须的字段。默认的User类对于用户名中包含的字符有严格的限定，所以 `LDAPBackend` 包含了2个函数 `ldap_to_django_username()` 和 `django_to_ldap_username()` 用来转换LDAP用户名和Django用户名。当你的LDAP用户名中包含逗点时候你或许会需要这个。你可以通过继承 `LDAPBackend` 来实现自己的处理函数。默认情况 `username` 是不可修改的。通过 `LDAPBackend` 验证的用户对象将会有一个**ldap_username**属性，值和LDAP的用户名一样。同理，Django的 `username` 或者 `get_username()` 也是如此。

> 注意：通过 `LDAPBackend` 创建的用户将会被设置一个不可用的密码，这仅仅发生在用户被创建时。所以如果你在Django中设置了一个合法的用户密码，这个用户即使被LDAP拒绝，也能够通过 `ModelBackend` (如果配置了)登录。通常不建议这么做，除了希望某些用户在LDAP服务不可用时也能登录的情况下。

扩充用户

你可以通过 `Django signals: django_auth_ldap.backend.populate_user` 和 `django_auth_ldap.backend.populate_user_profile` 来任意扩充你的用户模型，他们在用户被创建以及属性映射被应用后触发。你可以使用这点来按照你喜欢的方式从LDAP目录传递信息给用户和profile对象。用户实例将在信号处理完成时自动保存。

如果你需要的参数默认不包含在LDAP搜索结果中，请参考 `AUTH_LDAP_USER_ATTRLIST` 。

> 注意：Django1.7后并不直接支持配置用户。在这些版本中， `populate_user_profile` 不会被触发。

简明参数

如果你仅仅是想简单的从LDAP目录直接获取少量用户参数到Django中，可以使用 `AUTH_LDAP_USER_ATTR_MAP` 和 `AUTH_LDAP_PROFILE_ATTR_MAP` ，他们使用字典来建立用户模型和LDAP服务属性名(不区分大小写)的映射：

```
AUTH_LDAP_USER_ATTR_MAP = {"first_name": "givenName", "last_name"
: "sn"}
AUTH_LDAP_PROFILE_ATTR_MAP = {"home_directory": "homeDirectory"}
```

只有字符串字段可以映射到属性上，布尔型字段可以通过组成员来定义：

```
AUTH_LDAP_USER_FLAGS_BY_GROUP = {
    "is_active": "cn=active,ou=groups,dc=example,dc=com",
    "is_staff": ["cn=staff,ou=groups,dc=example,dc=com",
                 "cn=admin,ou=groups,dc=example,dc=com"],
    "is_superuser": "cn=superuser,ou=groups,dc=example,dc=com"
}

AUTH_LDAP_PROFILE_FLAGS_BY_GROUP = {
    "is_awesome": ["cn=awesome,ou=groups,dc=example,dc=com"]
}
```

如果给的是一个组构成的列表，那么所有属于这些组成员的标志都会被设置成True。

> 注意：Django1.7后并不直接支持配置用户。在这些版本中， `LDAPBackend` 将忽视 `profile-related` 设置。

修改用户

默认情况下，所有已经建立了映射的字段在每次用户登录时候都会被更新。想要关闭这个特性，设置 `AUTH_LDAP_ALWAYS_UPDATE_USER` 为False即可。如果你需要在认证过程外部扩充用户——比如在用户第一次登录之前创建一个相关的模型对象，你可以调用 `django_auth_ldap.backend.LDAPBackend.populate_user()` ，也可以创建一个 `LDAPBackend` 对象来自由的实现所需功能。如果在LDAP服务器中不存在此用户 `populate_user()` 函数返回 `None` ，否则返回 `User` 对象。

```
from django_auth_ldap.backend import LDAPBackend

user = LDAPBackend().populate_user('alice')
if user is None:
    raise Exception('No user named alice')
```

直接属性访问

如果你需要访问复合属性或者有上面的情况不满足你的需求时候，你可以直接访问原生的LDAP参数。 `user.ldap_user` 是一个拥有下列4种公共属性的对象。如果配置了组，那么组属性也是如此。

- dn: 用户的专有名称
- attrs: 用户的LDAP参数信息
- group_dns: 用户所属组的专有名称集合
- group_names: 用户所属组的简单名称集合，如果设置了 `AUTH_LDAP_MIRROR_GROUPS` 将会使用名字

python-ldap使用utf8编码来返回参数信息，为了方便程序将尝试使用unicode进行解码，如果解码失败则原样返回;这也许在获取类似 `Active Directory's objectSid` 这类二进制的属性时被使用。

权限

组不仅仅可以用来设置用户的 `is_*` 字段，还有2种方法把LDAP的组成员信息转换成Django的权限信息。

最后，这2种方法都需要一种机制在LDAP组和Django组之间建立映射。 `LDAPGroupType` 实现了一种算法将LDAP的组信息转换成Django的组信息。用户可以通过继承 `LDAPGroupType` 来修改默认的行为。所有的内部实现都把 `name_attr` 存入了 `__init__` ，并指定了来自LDAP的哪一个参数映射到Django的组名字。默认情况下使用 `cn` 参数。

直接使用组

侵入性最小的方法实现组权限映射就是设置 `AUTH_LDAP_FIND_GROUP_PERMS` 为 `True` 。 `LDAPBackend` 将找出所有LDAP用户组并把它们和Django组进行映射，并且为这些组加载相应的权限。你需要自己将组信息和权限相关联，通常使用自带的admin界面。

为了减少与LDAP的链接请求， `LDAPBackend` 可以使用Django的缓存功能存储一份LDAP用户组信息的副本在缓存中。通过设置 `AUTH_LDAP_CACHE_GROUPS` 为 `True` 来启用这一特性。也可以设置 `AUTH_LDAP_GROUP_CACHE_TIMEOUT` 来控制缓存的超时时间。

```
AUTH_LDAP_CACHE_GROUPS = True
AUTH_LDAP_GROUP_CACHE_TIMEOUT = 300
```

组镜像

第二种把LDAP组信息转换成Django权限信息的方法就是创建组镜像。如果 `AUTH_LDAP_MIRROR_GROUPS` 是 `True` ，则每次用户登录时候 `LDAPBackend` 都会使用LDAP组信息来更新数据库。任何不存在的组都将被创建并且Django的组信息将被更新来匹配LDAP的组信息。注意如果LDAP包含组嵌套结构，Django数据库将以扁平化的形式来表示。为了使组镜像生效，你需要把 `ModelBackend` 也作为认证方式之一。

这2种方式区别在于，第一种方式每次接收到请求都会去LDAP服务器或缓存进行查询，而第二种则是仅在用户认证时候被更新，这不太适合那种有长链接的网站。

非LDAP用户

`LDAPBackend` 还有一个能力就是管理非认证(authenticate)用户的授权(authorization)。比如你或许使用 `RemoteUserBackend` 来映射已经在外部认证的用户到Django用户。通过设置 `AUTH_LDAP_AUTHORIZE_ALL_USERS` ， `LDAPBackend` 将按照普通的方法映射这些用户到LDAP用户为了提供授权信息。注意这对于 `AUTH_LDAP_MIRROR_GROUPS` 是无效的;组镜像是一种认证机制而不是授权机制。

多LDAP后端配置

> 1.1版新特性

你也许注意到了，之前我们所有的配置都有`AUTH_LDAP_`前缀，这是默认选项，也可以通过继承`LDAPBackend`来进行自定义配置。这么做的主要情形就是你需要创建2个分别可以操作的后端时。比如，你需要2个LDAP后端分别进行认证的时候，这里有个简短的例子:

```
# mypackage.ldap

from django_auth_ldap.backend import LDAPBackend

class LDAPBackend1(LDAPBackend):
    settings_prefix = "AUTH_LDAP_1_"

class LDAPBackend2(LDAPBackend):
    settings_prefix = "AUTH_LDAP_2_"
```

```
# settings.py

AUTH_LDAP_1_SERVER_URI = "ldap://ldap1.example.com"
AUTH_LDAP_1_USER_DN_TEMPLATE = "uid=%(user)s,ou=users,dc=example
,dc=com"

AUTH_LDAP_2_SERVER_URI = "ldap://ldap2.example.com"
AUTH_LDAP_2_USER_DN_TEMPLATE = "uid=%(user)s,ou=users,dc=example
,dc=com"

AUTHENTICATION_BACKENDS = (
    "mypackage.ldap.LDAPBackend1",
    "mypackage.ldap.LDAPBackend2",
)
```

常见规则如下：Django将尝试对每一个后端进行认证，直到验证成功。当用户在某个特定的后端验证成功，在这个会话的有效期内这个用户都会被链接到这个后端上。

> 注意：由于 `AUTH_LDAP_GLOBAL_OPTIONS` 具有全局特性，所以它将忽略设置的前缀。不管有多少个后端，这个属性只在第一次载入LDAP模块时被设置。

日志

`LDAPBackend` 使用标准库中的 `logging` 模块实现了 `django_auth_ldap` 来记录调试和警告信息，如果你需要调试信息来调试配置问题，你应该添加一个logger来控制它。注意logger默认等级是NOTSET，你需要改变默认等级来获取调试信息：

```
import logging

logger = logging.getLogger('django_auth_ldap')
logger.addHandler(logging.StreamHandler())
logger.setLevel(logging.DEBUG)
```

性能

`LDAPBackend` 在设计上尽量避免每次请求都访问LDAP服务器，当然这依赖于你如何进行配置。如果你比较注重阻塞或延迟问题，这有一些小建议减小它们的影响：

1. 缓存组信息：如果 `AUTH_LDAP_FIND_GROUP_PERMS` 被设置成 `True` ，则默认情况下每次请求都会重载用户的组信息，这是最安全的选择，当用户的所属组信息被改变时会立即生效，但开销比价大。如果可能的话，设置 `AUTH_LDAP_CACHE_GROUPS` 为 `True` 来禁用这种特性。另外你可以考虑使用 `AUTH_LDAP_MIRROR_GROUPS` 和 `ModelBackend` 来支持组权限设置。
2. 不要使用 `user.ldap_user.*` ：这些属性仅仅在per-request请求时被缓存，如果可以传递LDAP属性给User或profile对象，它们仅仅在登录时候被更新。 `user.ldap_user.attrs` 在每次接受请求时都会触发与LDAP的链接。如果你没使用 `AUTH_LDAP_USER_DN_TEMPLATE` ，那么 `user.ldap_user.dn` 也将触发与LDAP的链接。
3. 使用简单的组类型：某些分组机制比其他的机制开销更大。这常常超出你的控制，但一定注意比如 `NestedGroupOfNamesType` 这类复杂的组类型所提供的功能并不是免费的，这些功能通常会生成更大量、更复杂的LDAP查询。
4. 使用直接绑定：使用 `AUTH_LDAP_USER_DN_TEMPLATE` 比 `AUTH_LDAP_USER_SEARCH` 更有效率。特别是存在2台LDAP的服务的登录操作时(一代用于绑定，一台用于查询)。

配置示例

这有一份从 `settings.py` 中提取的完整配置示例，展示了所有的特性。在例子中，我们对目录中所有的用户进行验证，另外我们还对django group(ou=django,ou=groups,dc=example,dc=com)进行了特殊处理。记住，如果你仅仅需要一个简单的验证，那么下面大部分选项都不是必须的。为了完整我们包含了些默认配置。

(译者注：代码注释不翻译了)

```
import ldap
from django_auth_ldap.config import LDAPSearch, GroupOfNamesType


# Baseline configuration.
AUTH_LDAP_SERVER_URI = "ldap://ldap.example.com"

AUTH_LDAP_BIND_DN = "cn=django-agent,dc=example,dc=com"
AUTH_LDAP_BIND_PASSWORD = "phlebotinum"
AUTH_LDAP_USER_SEARCH = LDAPSearch("ou=users,dc=example,dc=com",
    ldap.SCOPE_SUBTREE, "(uid=%(user)s)")
# or perhaps:
# AUTH_LDAP_USER_DN_TEMPLATE = "uid=%(user)s,ou=users,dc=example
,dc=com"

# Set up the basic group parameters.
AUTH_LDAP_GROUP_SEARCH = LDAPSearch("ou=django,ou=groups,dc=exam
ple,dc=com",
    ldap.SCOPE_SUBTREE, "(objectClass=groupOfNames)"
)
AUTH_LDAP_GROUP_TYPE = GroupOfNamesType(name_attr="cn")

# Simple group restrictions
AUTH_LDAP_REQUIRE_GROUP = "cn=enabled,ou=django,ou=groups,dc=exa
mple,dc=com"
AUTH_LDAP_DENY_GROUP = "cn=disabled,ou=django,ou=groups,dc=examp
le,dc=com"
```

```
# Populate the Django user from the LDAP directory.
AUTH_LDAP_USER_ATTR_MAP = {
    "first_name": "givenName",
    "last_name": "sn",
    "email": "mail"
}

AUTH_LDAP_PROFILE_ATTR_MAP = {
    "employee_number": "employeeNumber"
}

AUTH_LDAP_USER_FLAGS_BY_GROUP = {
    "is_active": "cn=active,ou=django,ou=groups,dc=example,dc=co
m",
    "is_staff": "cn=staff,ou=django,ou=groups,dc=example,dc=com"
,
    "is_superuser": "cn=superuser,ou=django,ou=groups,dc=example
,dc=com"
}

AUTH_LDAP_PROFILE_FLAGS_BY_GROUP = {
    "is_awesome": "cn=awesome,ou=django,ou=groups,dc=example,dc=
com",
}

# This is the default, but I like to be explicit.
AUTH_LDAP_ALWAYS_UPDATE_USER = True

# Use LDAP group membership to calculate group permissions.
AUTH_LDAP_FIND_GROUP_PERMS = True

# Cache group memberships for an hour to minimize LDAP traffic
AUTH_LDAP_CACHE_GROUPS = True
AUTH_LDAP_GROUP_CACHE_TIMEOUT = 3600


# Keep ModelBackend around for per-user permissions and maybe a
local
# superuser.
AUTHENTICATION_BACKENDS = (
```

```
    'django_auth_ldap.backend.LDAPBackend',
    'django.contrib.auth.backends.ModelBackend',
)
```