django-permission2 Documentation

Release 2.1.0

Malte Gerth <mail@malte-gerth.de>

Jul 31, 2023

USER GUIDE

1 Documentation **3**

2 Installation **5**

3 Usage **7**
- 3.1 Configuration . . . 7
- 3.2 Quick tutorial . . . 7

4 License **11**
- 4.1 Installation . . . 11
  - 4.1.1 Installing django-permission2 . . . 11
  - 4.1.2 Autodiscovery . . . 12
- 4.2 Permissions . . . 12
  - 4.2.1 Apply permission logic . . . 12
  - 4.2.2 Customize permission logic . . . 17
- 4.3 Decorators . . . 17
  - 4.3.1 Class, method, or function decorator . . . 17
- 4.4 Templatetags . . . 17
  - 4.4.1 Override the builtin `if` template tag . . . 17
- 4.5 License . . . 18
- 4.6 Decorators . . . 19
  - 4.6.1 Class based . . . 19
  - 4.6.2 Function based . . . 20
  - 4.6.3 Method based . . . 21
  - 4.6.4 permission_required . . . 21
  - 4.6.5 Decorators utils . . . 22
- 4.7 Logics . . . 22
  - 4.7.1 Base Logic . . . 22
  - 4.7.2 Author logic . . . 23
  - 4.7.3 Collaborators logic . . . 23
  - 4.7.4 GrouIn logic . . . 24
  - 4.7.5 Oneself logic . . . 25
  - 4.7.6 Staff logic . . . 26
- 4.8 Templatetags . . . 26
  - 4.8.1 permissionif . . . 26
- 4.9 Utils . . . 28
  - 4.9.1 Autodiscover . . . 28
  - 4.9.2 field_lookup . . . 28
  - 4.9.3 Handlers utils . . . 29
  - 4.9.4 Logics utils . . . 30

4.9.5 Permissions utils 31
4.10 Backends 32
4.10.1 PermissionBackend 32
4.11 Handlers 33
4.11.1 PermissionHandler 33
4.11.2 LogicalPermissionHandler 33
4.12 Conf 33
4.12.1 permission.conf module 33
4.13 Compat 33
4.13.1 permission.compat module 33
4.14 Module contents 33

5 Indices and tables **35**

Python Module Index **37**

Index **39**

Author
Malte Gerth <mail@malte-gerth.de>

Original Author
Alisue <lambdalisue@hashnote.net>

Supported python versions
Python 3.8, 3.9, 3.10, 3.11

Supported django versions
Django 2.2, 3.2, 4.0, 4.1, 4.2

An enhanced permission library which enables a *logic-based permission system* to handle complex permissions in Django.

CHAPTER

ONE

DOCUMENTATION

http://django-permission2.readthedocs.org/

CHAPTER TWO

INSTALLATION

Use pip like:

```
$ pip install django-permission2
```

CHAPTER

THREE

USAGE

The following might help you to understand as well.

- Basic strategy or so on, Issue #28
- Advanced usage and examples, Issue #26

3.1 Configuration

1. Add `permission` to the `INSTALLED_APPS` in your settings module

```
INSTALLED_APPS = (
    # ...
    'permission',
)
```

2. Add our extra authorization/authentication backend

```
AUTHENTICATION_BACKENDS = (
    'django.contrib.auth.backends.ModelBackend', # default
    'permission.backends.PermissionBackend',
)
```

3. Follow the instructions below to apply logical permissions to django models

3.2 Quick tutorial

Let's assume you wrote an article model which has an `author` attribute to store the creator of the article, and you want to give that author full control permissions (e.g. add, change and delete permissions).

1. Add `import permission; permission.autodiscover()` to your `urls.py` like:

```
from django.conf.urls import patterns, include
from django.urls import path
from django.contrib import admin

admin.autodiscover()

# only add the following line
import permission; permission.autodiscover()
```

(continues on next page)

(continued from previous page)

```
urlpatterns = [
    path('admin/', include(admin.site.urls)),
    # ...
]
```

2. Write perms.py in your application directory like:

```
from permission.logics import AuthorPermissionLogic
from permission.logics import CollaboratorsPermissionLogic

PERMISSION_LOGICS = (
    ('your_app.Article', AuthorPermissionLogic()),
    ('your_app.Article', CollaboratorsPermissionLogic()),
)
```

What you need to do is just applying `permission.logics.AuthorPermissionLogic` to the `Article` model like

```
from django.db import models
from django.contrib.auth.models import User


class Article(models.Model):
    title = models.CharField('title', max_length=120)
    body = models.TextField('body')
    author = models.ForeignKey(User)

    # this is just required for easy explanation
    class Meta:
        app_label='permission'

# apply AuthorPermissionLogic
from permission import add_permission_logic
from permission.logics import AuthorPermissionLogic
add_permission_logic(Article, AuthorPermissionLogic())
```

That's it. Now the following codes will work as expected:

```
user1 = User.objects.create_user(
    username='john',
    email='john@test.com',
    password='password',
)
user2 = User.objects.create_user(
    username='alice',
    email='alice@test.com',
    password='password',
)

art1 = Article.objects.create(
    title="Article 1",
    body="foobar hogehoge",
    author=user1
```

(continues on next page)

(continued from previous page)

```
)
art2 = Article.objects.create(
    title="Article 2",
    body="foobar hogehoge",
    author=user2
)

# You have to apply 'permission.add_article' to users manually because it
# is not an object permission.
from permission.utils.permissions import perm_to_permission
user1.user_permissions.add(perm_to_permission('permission.add_article'))

assert user1.has_perm('permission.add_article') == True
assert user1.has_perm('permission.change_article') == False
assert user1.has_perm('permission.change_article', art1) == True
assert user1.has_perm('permission.change_article', art2) == False

assert user2.has_perm('permission.add_article') == False
assert user2.has_perm('permission.delete_article') == False
assert user2.has_perm('permission.delete_article', art1) == False
assert user2.has_perm('permission.delete_article', art2) == True
```

CHAPTER

FOUR

LICENSE

4.1 Installation

4.1.1 Installing django-permission2

1. Install latest stable version into your python environment using pip:

```
pip install django-permission2
```

2. Once installed add `permission` to your `INSTALLED_APPS` in settings.py:

```
.. code:: python

    INSTALLED_APPS = (
        ...
        'permission',
    )
```

3. Add our extra authorization/authentication backend

```
AUTHENTICATION_BACKENDS = (
    'django.contrib.auth.backends.ModelBackend', # default
```

(continues on next page)

(continued from previous page)

```
    'permission.backends.PermissionBackend',
)
```

4. Follow the instructions below to apply logical permissions to django models

4.1.2 Autodiscovery

Like django's admin package, django-permission2 automatically discovers the `perms.py` in your application directory **by running ``permission.autodiscover()``**. Additionally, if the `perms.py` module has a `PERMISSION_LOGICS` variable, django-permission2 automatically run the following functions to apply the permission logics.

```
for model, permission_logic_instance in PERMISSION_LOGICS:
    if isinstance(model, str):
        model = get_model(*model.split(".", 1))
    add_permission_logic(model, permission_logic_instance)
```

Note: Autodiscover feature is automatically called. To disable, use *PERMISSION_AUTODISCOVER_ENABLE* setting.

4.2 Permissions

4.2.1 Apply permission logic

Let's assume you wrote an article model which has an `author` attribute to store the creator of the article, and you want to give that author full control permissions (e.g. add, change and delete permissions).

What you need to do is just applying `permission.logics.AuthorPermissionLogic` to the `Article` model like

```
from django.db import models
from django.contrib.auth.models import User


class Article(models.Model):
    title = models.CharField('title', max_length=120)
    body = models.TextField('body')
    author = models.ForeignKey(User)

    # this is just required for easy explanation
    class Meta:
        app_label='permission'

# apply AuthorPermissionLogic
from permission import add_permission_logic
from permission.logics import AuthorPermissionLogic
add_permission_logic(Article, AuthorPermissionLogic())
```

Note: You can specify related object with *field__name* attribute like django queryset lookup. See the working example below:

```
from django.db import models
from django.contrib.auth.models import User


class Article(models.Model):
    title = models.CharField('title', max_length=120)
    body = models.TextField('body')
    project = models.ForeignKey('permission.Project')

    # this is just required for easy explanation
    class Meta:
        app_label='permission'

class Project(models.Model):
    title = models.CharField('title', max_length=120)
    body = models.TextField('body')
    author = models.ForeignKey(User)

    # this is just required for easy explanation
    class Meta:
        app_label='permission'

# apply AuthorPermissionLogic to Article
from permission import add_permission_logic
from permission.logics import AuthorPermissionLogic
add_permission_logic(Article, AuthorPermissionLogic(
    field_name='project__author',
))
```

That's it. Now the following codes will work as expected:

```
user1 = User.objects.create_user(
    username='john',
    email='john@test.com',
    password='password',
)
user2 = User.objects.create_user(
    username='alice',
    email='alice@test.com',
    password='password',
)

art1 = Article.objects.create(
    title="Article 1",
    body="foobar hogehoge",
    author=user1
)
art2 = Article.objects.create(
    title="Article 2",
```

(continues on next page)

(continued from previous page)

```
    body="foobar hogehoge",
    author=user2
)

# Grant the `permission.add_article` permission for user1.
# Use the `perm_to_permission` utility to convert the permission-string to a `Permission`␣
↪object instance.
from permission.utils.permissions import perm_to_permission
user1.user_permissions.add(perm_to_permission('permission.add_article'))

# `add_article` is granted by user permissions
assert user1.has_perm('permission.add_article') == True
assert user2.has_perm('permission.add_article') == False

# `change_article` is not granted by user permissions
assert user1.has_perm('permission.change_article') == False
assert user2.has_perm('permission.change_article') == False

# `change_article` is granted by `AuthorPermissionLogic`
assert user1.has_perm('permission.change_article', art1) == True
# `change_article` is not granted by `AuthorPermissionLogic`
assert user1.has_perm('permission.change_article', art2) == False

# `delete_article` is not granted by user permissions
assert user1.has_perm('permission.delete_article') == False
assert user2.has_perm('permission.delete_article') == False

# `delete_article` is granted by `AuthorPermissionLogic`
assert user1.has_perm('permission.delete_article', art1) == True
# `delete_article` is not granted by `AuthorPermissionLogic`
assert user1.has_perm('permission.delete_article', art2) == False

# `delete_article` is not granted by `AuthorPermissionLogic`
assert user2.has_perm('permission.delete_article', art1) == False
# `delete_article` is granted by `AuthorPermissionLogic`
assert user2.has_perm('permission.delete_article', art2) == True

#
# You may also be interested in django signals to apply 'add' permissions to the
# newly created users.
# https://docs.djangoproject.com/en/dev/ref/signals/#django.db.models.signals.post_save
#
from django.db.models.signals.post_save
from django.dispatch import receiver
from permission.utils.permissions import perm_to_permission

@receiver(post_save, sender=User)
def apply_permissions_to_new_user(sender, instance, created, **kwargs):
    if not created:
        return
    #
    # permissions you want to apply to the newly created user
```

(continues on next page)

(continued from previous page)

```
    # YOU SHOULD NOT APPLY PERMISSIONS EXCEPT PERMISSIONS FOR 'ADD'
    # in this way, the applied permissions are not object permission so
    # if you apply 'permission.change_article' then the user can change
    # any article object.
    #
    permissions = [
        'permission.add_article',
    ]
    for permission in permissions:
        # apply permission
        # perm_to_permission is a utility to convert string permission
        # to permission instance.
        instance.user_permissions.add(perm_to_permission(permission))
```

See `permission.logics.author.AuthorPermissionLogic` to learn how this logic works.

Now, assume you add `collaborators` attribute to store collaborators of the article and you want to give them a change permission.

What you need to do is quite simple. Apply `permission.logics.CollaboratorsPermissionLogic` to the `Article` model as follows

```
from django.db import models
from django.contrib.auth.models import User


class Article(models.Model):
    title = models.CharField('title', max_length=120)
    body = models.TextField('body')
    author = models.ForeignKey(User)
    collaborators = models.ManyToManyField(User)

    # this is just required for easy explanation
    class Meta:
        app_label='permission'

# apply AuthorPermissionLogic and CollaboratorsPermissionLogic
from permission import add_permission_logic
from permission.logics import AuthorPermissionLogic
from permission.logics import CollaboratorsPermissionLogic
add_permission_logic(Article, AuthorPermissionLogic())
add_permission_logic(Article, CollaboratorsPermissionLogic(
    field_name='collaborators',
    any_permission=False,
    change_permission=True,
    delete_permission=False,
))
```

Note: You can specify related object with *field_name* attribute like django queryset lookup. See the working example below:

```
from django.db import models
from django.contrib.auth.models import User


class Article(models.Model):
    title = models.CharField('title', max_length=120)
    body = models.TextField('body')
    project = models.ForeignKey('permission.Project')

    # this is just required for easy explanation
    class Meta:
        app_label='permission'

class Project(models.Model):
    title = models.CharField('title', max_length=120)
    body = models.TextField('body')
    collaborators = models.ManyToManyField(User)

    # this is just required for easy explanation
    class Meta:
        app_label='permission'

# apply AuthorPermissionLogic to Article
from permission import add_permission_logic
from permission.logics import CollaboratorsPermissionLogic
add_permission_logic(Article, CollaboratorsPermissionLogic(
    field_name='project__collaborators',
))
```

That's it. Now the following codes will work as expected:

```
user1 = User.objects.create_user(
    username='john',
    email='john@test.com',
    password='password',
)
user2 = User.objects.create_user(
    username='alice',
    email='alice@test.com',
    password='password',
)

art1 = Article.objects.create(
    title="Article 1",
    body="foobar hogehoge",
    author=user1
)
art1.collaborators.add(user2)

assert user1.has_perm('permission.change_article') == False
assert user1.has_perm('permission.change_article', art1) == True
assert user1.has_perm('permission.delete_article', art1) == True
```

(continues on next page)

(continued from previous page)

```
assert user2.has_perm('permission.change_article') == False
assert user2.has_perm('permission.change_article', art1) == True
assert user2.has_perm('permission.delete_article', art1) == False
```

See *permission.logics.collaborators.CollaboratorsPermissionLogic* to learn how this logic works.

There are *permission.logics.staff.StaffPermissionLogic* and `permission.logics.groupinGroupInPermissionLogic` for `is_staff` or `group` based permission logic as well.

4.2.2 Customize permission logic

Your own permission logic class must be a subclass of *permission.logics.base.PermissionLogic* and must override `has_perm(user_obj, perm, obj=None)` method which return boolean value.

4.3 Decorators

4.3.1 Class, method, or function decorator

Like Django's `permission_required` but it can be used for object permissions and as a class, method, or function decorator. Also, you don't need to specify a object to this decorator for object permission. This decorator automatically determined the object from request (so you cannot use this decorator for non view class/method/function but you anyway use `user.has_perm` in that case).

```
>>> from permission.decorators import permission_required
>>> # As class decorator
>>> @permission_required('auth.change_user')
>>> class UpdateAuthUserView(UpdateView):
...     pass
>>> # As method decorator
>>> class UpdateAuthUserView(UpdateView):
...     @permission_required('auth.change_user')
...     def dispatch(self, request, *args, **kwargs):
...         pass
>>> # As function decorator
>>> @permission_required('auth.change_user')
>>> def update_auth_user(request, *args, **kwargs):
...     pass
```

4.4 Templatetags

4.4.1 Override the builtin `if` template tag

django-permission2 overrides the builtin `if` tag, adding two operators to handle permissions in templates. You can write a permission test by using `has` keyword, and a target object with `of` as below.

```
{% if user has 'blogs.add_article' %}
    <p>This user have 'blogs.add_article' permission</p>
```

(continues on next page)

(continued from previous page)

```
{% elif user has 'blog.change_article' of object %}
    <p>This user have 'blogs.change_article' permission of {{object}}</p>
{% endif %}

{# If you set 'PERMISSION_REPLACE_BUILTIN_IF = False' in settings #}
{% permission user has 'blogs.add_article' %}
    <p>This user have 'blogs.add_article' permission</p>
{% elpermission user has 'blog.change_article' of object %}
    <p>This user have 'blogs.change_article' permission of {{object}}</p>
{% endpermission %}
```

Note: You have to add *'permission.templatetags.permissionif'* to *'builtins'* option manually. See - https://docs.djangoproject.com/en/1.9/releases/1.9/#django-template-base-add-to-builtins-is-removed - https://docs.djangoproject.com/en/1.9/topics/templates/#module-django.template.backends.django Or following example:

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'OPTIONS': {
            'builtins': ['permission.templatetags.permissionif'],
        },
    },
]
```

4.5 License

4.6 Decorators

4.6.1 Class based

permission_required decorator for generic classbased view from django 1.3

`permission.decorators.classbase.`**`get_object_from_classbased_instance`**(*instance*, *queryset*, *request*, **args*, ***kwargs*)

Get object from an instance of classbased generic view

Parameters

- **`instance`** (*`instance`*) – An instance of classbased generic view
- **`queryset`** (*`instance`*) – A queryset instance
- **`request`** (*`instance`*) – A instance of HttpRequest

Returns
An instance of model object or None

Return type
instance

`permission.decorators.classbase.`**`permission_required`**(*perm*, *queryset=None*, *login_url=None*, *raise_exception=False*)

Permission check decorator for classbased generic view

This decorator works as class decorator DO NOT use `method_decorator` or whatever while this decorator will use `self` argument for method of classbased generic view.

Parameters

- **`perm`** (*`string`*) – A permission string
- **`queryset`** (*`queryset or model`*) – A queryset or model for finding object. With classbased generic view, `None` for using view default queryset. When the view does not define `get_queryset`, `queryset`, `get_object`, or `object` then `obj=None` is used to check permission. With functional generic view, `None` for using passed queryset. When non queryset was passed then `obj=None` is used to check permission.

Examples

```
>>> @permission_required('auth.change_user')
>>> class UpdateAuthUserView(UpdateView):
...     pass
```

4.6.2 Function based

permission_required decorator for generic function view

`permission.decorators.functionbase.`**`get_object_from_date_based_view`**(*request*, **args*, ***kwargs*)

Get object from generic date_based.detail view

Parameters
`request` (*`instance`*) – An instance of HttpRequest

Returns
An instance of model object or None

Return type
instance

`permission.decorators.functionbase.`**`get_object_from_list_detail_view`**(*request*, **args*, ***kwargs*)

Get object from generic list_detail.detail view

Parameters
`request` (*`instance`*) – An instance of HttpRequest

Returns
An instance of model object or None

Return type
instance

`permission.decorators.functionbase.`**`permission_required`**(*perm*, *queryset=None*, *login_url=None*, *raise_exception=False*)

Permission check decorator for function-base generic view

This decorator works as function decorator

Parameters

- **`perm`** (*`string`*) – A permission string
- **`queryset`** (*`queryset or model`*) – A queryset or model for finding object. With class-based generic view, `None` for using view default queryset. When the view does not define `get_queryset`, `queryset`, `get_object`, or `object` then `obj=None` is used to check permission. With functional generic view, `None` for using passed queryset. When non queryset was passed then `obj=None` is used to check permission.

Examples

```
>>> @permission_required('auth.change_user')
>>> def update_auth_user(request, *args, **kwargs):
...     pass
```

4.6.3 Method based

permission_required decorator for generic classbased/functionbased view

`permission.decorators.methodbase.`**`permission_required`**(*perm*, *queryset=None*, *login_url=None*, *raise_exception=False*)

Permission check decorator for classbased/functionbased generic view

This decorator works as method or function decorator DO NOT use `method_decorator` or whatever while this decorator will use `self` argument for method of classbased generic view.

Parameters

- **`perm`** (*`string`*) – A permission string
- **`queryset`** (*`queryset or model`*) – A queryset or model for finding object. With classbased generic view, `None` for using view default queryset. When the view does not define `get_queryset`, `queryset`, `get_object`, or `object` then `obj=None` is used to check permission. With functional generic view, `None` for using passed queryset. When non queryset was passed then `obj=None` is used to check permission.

Examples

```
>>> # As method decorator
>>> class UpdateAuthUserView(UpdateView):
>>>     @permission_required('auth.change_user')
>>>     def dispatch(self, request, *args, **kwargs):
...         pass
>>> # As function decorator
>>> @permission_required('auth.change_user')
>>> def update_auth_user(request, *args, **kwargs):
...     pass
```

4.6.4 permission_required

Decorator module for permission

`permission.decorators.permission_required.`**`permission_required`**(*perm*, *queryset_or_model=None*, *login_url=None*, *raise_exception=False*)

Permission check decorator for classbased/functional generic view

This decorator works as class, method or function decorator without any modification. DO NOT use `method_decorator` or whatever while this decorator will use `self` argument for method of classbased generic view.

Parameters

- **`perm`** (*`string`*) – A permission string
- **`queryset_or_model`** (*`queryset or model`*) – A queryset or model for finding object. With classbased generic view, `None` for using view default queryset. When the view does not define `get_queryset`, `queryset`, `get_object`, or `object` then `obj=None` is used to check permission. With functional generic view, `None` for using passed queryset. When non queryset was passed then `obj=None` is used to check permission.

Examples

```
>>> # As class decorator
>>> @permission_required('auth.change_user')
>>> class UpdateAuthUserView(UpdateView):
...     pass
>>> # As method decorator
>>> class UpdateAuthUserView(UpdateView):
...     @permission_required('auth.change_user')
...     def dispatch(self, request, *args, **kwargs):
...         pass
>>> # As function decorator
>>> @permission_required('auth.change_user')
>>> def update_auth_user(request, *args, **kwargs):
...     pass
```

Note: Classbased generic view is recommended while you can regulate the queryset with `get_queryset()` method. Detecting object from passed kwargs may not work correctly.

4.6.5 Decorators utils

Decorator utility module

`permission.decorators.utils.`**`redirect_to_login`**(*request*, *login_url=None*, *redirect_field_name='next'*)

redirect to login

4.7 Logics

4.7.1 Base Logic

`class permission.logics.base.`**`PermissionLogic`**

Bases: `object`

Abstract permission logic class

`get_full_permission_string`(*perm*)

Return full permission string (app_label.perm_model)

`has_perm`(*user_obj*, *perm*, *obj=None*)

Check if user have permission (of object)

Parameters

- **`user_obj`** (*`django user model instance`*) – A django user model instance which be checked
- **`perm`** (*`string`*) – *app_label.codename* formatted permission string
- **`obj`** (*`None or django model instance`*) – None or django model instance for object permission

Returns

- *boolean* – Whether the specified user have specified permission (of specified object).
- .. *note::* – Sub class must override this method.

4.7.2 Author logic

Permission logic module for author based permission system

`class permission.logics.author.`**`AuthorPermissionLogic`**(*field_name=None*, *any_permission=None*, *change_permission=None*, *delete_permission=None*)

Bases: `PermissionLogic`

Permission logic class for author based permission system

`has_perm`(*user_obj*, *perm*, *obj=None*)

Check if user have permission (of object)

If the user_obj is not authenticated, it return `False`.

If no object is specified, it return `True` when the corresponding permission was specified to `True` (changed from v0.7.0). This behavior is based on the django system. https://code.djangoproject.com/wiki/RowLevelPermissions

If an object is specified, it will return `True` if the user is specified in `field_name` of the object (e.g. `obj.author`). So once user create an object and the object store who is the author in `field_name` attribute (default: `author`), the author can change or delete the object (you can change this behavior to set `any_permission`, `change_permissino` or `delete_permission` attributes of this instance).

Parameters

- **`user_obj`** (*`django user model instance`*) – A django user model instance which be checked
- **`perm`** (*`string`*) – *app_label.codename* formatted permission string
- **`obj`** (*`None or django model instance`*) – None or django model instance for object permission

Returns

Whether the specified user have specified permission (of specified object).

Return type

boolean

4.7.3 Collaborators logic

Permission logic module for collaborators based permission system

`class permission.logics.collaborators.`**`CollaboratorsPermissionLogic`**(*field_name=None*, *any_permission=None*, *change_permission=None*, *delete_permission=None*)

Bases: `PermissionLogic`

Permission logic class for collaborators based permission system

has_perm(*user_obj*, *perm*, *obj=None*)

Check if user have permission (of object)

If the user_obj is not authenticated, it return `False`.

If no object is specified, it return `True` when the corresponding permission was specified to `True` (changed from v0.7.0). This behavior is based on the django system. https://code.djangoproject.com/wiki/RowLevelPermissions

If an object is specified, it will return `True` if the user is found in `field_name` of the object (e.g. `obj.collaborators`). So once the object store the user as a collaborator in `field_name` attribute (default: `collaborators`), the collaborator can change or delete the object (you can change this behavior to set `any_permission`, `change_permission` or `delete_permission` attributes of this instance).

Parameters

- **user_obj** (*django user model instance*) – A django user model instance which be checked
- **perm** (*string*) – *app_label.codename* formatted permission string
- **obj** (*None or django model instance*) – None or django model instance for object permission

Returns
Whether the specified user have specified permission (of specified object).

Return type
boolean

4.7.4 GrouIn logic

Permission logic module for group based permission system

class permission.logics.groupin.**GroupInPermissionLogic**(*group_names*, *any_permission=None*, *add_permission=None*, *change_permission=None*, *delete_permission=None*)

Bases: *PermissionLogic*

Permission logic class for group based permission system

has_perm(*user_obj*, *perm*, *obj=None*)

Check if user have permission (of object)

If the user_obj is not authenticated, it return `False`.

If no object is specified, it return `True` when the corresponding permission was specified to `True` (changed from v0.7.0). This behavior is based on the django system. https://code.djangoproject.com/wiki/RowLevelPermissions

If an object is specified, it will return `True` if the user is in group specified in `group_names` of this instance. This permission logic is used mainly for group based role permission system. You can change this behavior to set `any_permission`, `add_permission`, `change_permission`, or `delete_permission` attributes of this instance.

Parameters

- **user_obj** (*django user model instance*) – A django user model instance which be checked

- **perm** (*string*) – *app_label.codename* formatted permission string
- **obj** (*None or django model instance*) – None or django model instance for object permission

Returns
Whether the specified user have specified permission (of specified object).

Return type
boolean

4.7.5 Oneself logic

Permission logic module to manage users' self-modifications

class permission.logics.oneself.**OneselfPermissionLogic**(*any_permission=None*, *change_permission=None*, *delete_permission=None*)

Bases: *PermissionLogic*

Permission logic class to manage users' self-modifications

Written by quasiyoke. https://github.com/lambdalisue/django-permission/pull/27

has_perm(*user_obj*, *perm*, *obj=None*)

Check if user have permission of himself

If the user_obj is not authenticated, it return `False`.

If no object is specified, it return `True` when the corresponding permission was specified to `True` (changed from v0.7.0). This behavior is based on the django system. https://code.djangoproject.com/wiki/RowLevelPermissions

If an object is specified, it will return `True` if the object is the user. So users can change or delete themselves (you can change this behavior to set `any_permission`, `change_permissino` or `delete_permission` attributes of this instance).

Parameters

- **user_obj** (*django user model instance*) – A django user model instance which be checked
- **perm** (*string*) – *app_label.codename* formatted permission string
- **obj** (*None or django model instance*) – None or django model instance for object permission

Returns
Whether the specified user have specified permission (of specified object).

Return type
boolean

4.7.6 Staff logic

Permission logic module for author based permission system

class permission.logics.staff.**StaffPermissionLogic**(*any_permission=None*, *add_permission=None*, *change_permission=None*, *delete_permission=None*)

Bases: *PermissionLogic*

Permission logic class for is_staff authority based permission system

has_perm(*user_obj*, *perm*, *obj=None*)

Check if user have permission (of object)

If the user_obj is not authenticated, it return `False`.

If no object is specified, it return `True` when the corresponding permission was specified to `True` (changed from v0.7.0). This behavior is based on the django system. https://code.djangoproject.com/wiki/RowLevelPermissions

If an object is specified, it will return `True` if the user is staff. The staff can add, change or delete the object (you can change this behavior to set `any_permission`, `add_permission`, `change_permission`, or `delete_permission` attributes of this instance).

Parameters

- **user_obj** (*django user model instance*) – A django user model instance which be checked
- **perm** (*string*) – *app_label.codename* formatted permission string
- **obj** (*None or django model instance*) – None or django model instance for object permission

Returns
Weather the specified user have specified permission (of specified object).

Return type
boolean

4.8 Templatetags

4.8.1 permissionif

permissionif templatetag

class permission.templatetags.permissionif.**PermissionIfParser**(*tokens*)

Bases: `IfParser`

Permission if parser

`OPERATORS = {'!=': <class 'django.template.smartif.infix.<locals>.Operator'>, '<': <class 'django.template.smartif.infix.<locals>.Operator'>, '<=': <class 'django.template.smartif.infix.<locals>.Operator'>, '==': <class 'django.template.smartif.infix.<locals>.Operator'>, '>': <class 'django.template.smartif.infix.<locals>.Operator'>, '>=': <class 'django.template.smartif.infix.<locals>.Operator'>, 'and': <class 'django.template.smartif.infix.<locals>.Operator'>, 'has': <class 'django.template.smartif.infix.<locals>.Operator'>, 'in': <class 'django.template.smartif.infix.<locals>.Operator'>, 'is': <class 'django.template.smartif.infix.<locals>.Operator'>, 'is not': <class 'django.template.smartif.infix.<locals>.Operator'>, 'not': <class 'django.template.smartif.prefix.<locals>.Operator'>, 'not in': <class 'django.template.smartif.infix.<locals>.Operator'>, 'of': <class 'django.template.smartif.infix.<locals>.Operator'>, 'or': <class 'django.template.smartif.infix.<locals>.Operator'>}`

use extra operator

translate_token(*token*)

class permission.templatetags.permissionif.**TemplatePermissionIfParser**(*parser*, **args*, ***kwargs*)

Bases: `PermissionIfParser`

create_var(*value*)

error_class

alias of `TemplateSyntaxError`

permission.templatetags.permissionif.**do_permissionif**(*parser*, *token*)

Permission if templatetag

Examples

```
{% if user has 'blogs.add_article' %}
    <p>This user have 'blogs.add_article' permission</p>
{% elif user has 'blog.change_article' of object %}
    <p>This user have 'blogs.change_article' permission of {{object}}</p>
{% endif %}

{# If you set 'PERMISSION_REPLACE_BUILTIN_IF = False' in settings #}
{% permission user has 'blogs.add_article' %}
    <p>This user have 'blogs.add_article' permission</p>
{% elpermission user has 'blog.change_article' of object %}
    <p>This user have 'blogs.change_article' permission of {{object}}</p>
{% endpermission %}
```

permission.templatetags.permissionif.**has_operator**(*context*, *x*, *y*)

'has' operator of permission if

This operator is used to specify the user object of permission

permission.templatetags.permissionif.**of_operator**(*context*, *x*, *y*)

'of' operator of permission if

This operator is used to specify the target object of permission

permission.templatetags.permissionif.**replace_builtin_if**(*replace=False*)

4.9 Utils

4.9.1 Autodiscover

`permission.utils.autodiscover.`**`autodiscover`**`(`*module_name=None*`)`

Autodiscover INSTALLED_APPS perms.py modules and fail silently when not present. This forces an import on them to register any permissions bits they may want.

`permission.utils.autodiscover.`**`discover`**`(`*app*, *module_name=None*`)`

Automatically apply the permission logics written in the specified module.

Examples

Assume if you have a `perms.py` in `your_app` as:

```
from permission.logics import AuthorPermissionLogic
PERMISSION_LOGICS = (
    ('your_app.your_model', AuthorPermissionLogic),
)
```

Use this method to apply the permission logics enumerated in `PERMISSION_LOGICS` variable like:

```
>>> discover('your_app')
```

4.9.2 field_lookup

A module to lookup field of object.

`permission.utils.field_lookup.`**`field_lookup`**`(`*obj*, *field_path*`)`

Lookup django model field in similar way of django query lookup.

Parameters

- **obj** (*instance*) – Django Model instance
- **field_path** (*str*) – '__' separated field path

Example

```
>>> from django.db import model
>>> from django.contrib.auth.models import User
>>> class Article(models.Model):
>>>     title = models.CharField('title', max_length=200)
>>>     author = models.ForeignKey(User, null=True,
>>>             related_name='permission_test_articles_author')
>>>     editors = models.ManyToManyField(User,
>>>             related_name='permission_test_articles_editors')
>>> user = User.objects.create_user('test_user', 'password')
>>> article = Article.objects.create(title='test_article',
...                                  author=user)
>>> article.editors.add(user)
```

(continues on next page)

(continued from previous page)

```
>>> assert 'test_article' == field_lookup(article, 'title')
>>> assert 'test_user' == field_lookup(article, 'user__username')
>>> assert ['test_user'] == list(field_lookup(article,
...                                            'editors__username'))
```

4.9.3 Handlers utils

A utilities of permission handler

class permission.utils.handlers.**PermissionHandlerRegistry**

Bases: `object`

A registry class of permission handler

get_handlers()

Get registered handler instances

Returns
permission handler tuple

Return type
tuple

register(*model*, *handler=None*)

Register a permission handler to the model

Parameters

- **model** (*django model class*) – A django model class
- **handler** (*permission handler class, string, or None*) – A permission handler class or a dotted path

Raises

- **ImproperlyConfigured** – Raise when the model is abstract model
- **KeyError** – Raise when the model is already registered in registry The model cannot have more than one handler.

unregister(*model*)

Unregister a permission handler from the model

Parameters
model (*django model class*) – A django model class

Raises
KeyError – Raise when the model have not registered in registry yet.

4.9.4 Logics utils

Permission logic utilities

`permission.utils.logics.`**`add_permission_logic`**`(`*model*, *permission_logic*`)`

Add permission logic to the model

Parameters

- **`model`** (*`django model class`*) – A django model class which will be treated by the specified permission logic
- **`permission_logic`** (*`permission logic instance`*) – A permission logic instance which will be used to determine permission of the model

Examples

```
>>> from django.db import models
>>> from permission.logics import PermissionLogic
>>> class Mock(models.Model):
...     name = models.CharField('name', max_length=120)
>>> add_permission_logic(Mock, PermissionLogic())
```

`permission.utils.logics.`**`remove_permission_logic`**`(`*model*, *permission_logic*, *fail_silently=True*`)`

Remove permission logic to the model

Parameters

- **`model`** (*`django model class`*) – A django model class which will be treated by the specified permission logic
- **`permission_logic`** (*`permission logic class or instance`*) – A permission logic class or instance which will be used to determine permission of the model
- **`fail_silently`** (*`boolean`*) – If *True* then do not raise KeyError even the specified permission logic have not registered.

Examples

```
>>> from django.db import models
>>> from permission.logics import PermissionLogic
>>> class Mock(models.Model):
...     name = models.CharField('name', max_length=120)
>>> logic = PermissionLogic()
>>> add_permission_logic(Mock, logic)
>>> remove_permission_logic(Mock, logic)
```

4.9.5 Permissions utils

Permission utility module.

In this module, term *perm* indicate the identifier string permission written in 'app_label.codename' format.

`permission.utils.permissions.`**`get_app_perms`**`(`*model_or_app_label*`)`

Get permission-string list of the specified django application.

Parameters
`model_or_app_label` (*`model class or string`*) – A model class or app_label string to specify the particular django application.

Returns
A set of perms of the specified django application.

Return type
set

Examples

```
>>> perms1 = get_app_perms('auth')
>>> perms2 = get_app_perms(Permission)
>>> perms1 == perms2
True
```

`permission.utils.permissions.`**`get_model_perms`**`(`*model*`)`

Get permission-string list of a specified django model.

Parameters
`model` (*`model class`*) – A model class to specify the particular django model.

Returns
A set of perms of the specified django model.

Return type
set

Examples

```
>>> sorted(get_model_perms(Permission)) == [
...     'auth.add_permission',
...     'auth.change_permission',
...     'auth.delete_permission'
... ]
True
```

`permission.utils.permissions.`**`get_perm_codename`**`(`*perm*, *fail_silently=True*`)`

Get permission codename from permission-string.

Examples

```
>>> get_perm_codename('app_label.codename_model')
'codename_model'
>>> get_perm_codename('app_label.codename')
'codename'
>>> get_perm_codename('codename_model')
'codename_model'
>>> get_perm_codename('codename')
'codename'
>>> get_perm_codename('app_label.app_label.codename_model')
'app_label.codename_model'
```

`permission.utils.permissions.`**`perm_to_permission`**(*perm*)

Convert a permission-string to a permission instance.

Examples

```
>>> permission = perm_to_permission('auth.add_user')
>>> permission.content_type.app_label
'auth'
>>> permission.codename
'add_user'
```

`permission.utils.permissions.`**`permission_to_perm`**(*permission*)

Convert a permission instance to a permission-string.

Examples

```
>>> permission = Permission.objects.get(
...     content_type__app_label='auth',
...     codename='add_user',
... )
>>> permission_to_perm(permission)
'auth.add_user'
```

4.10 Backends

4.10.1 PermissionBackend

A handler based permission backend

4.11 Handlers

4.11.1 PermissionHandler

Abstract permission handler class

4.11.2 LogicalPermissionHandler

Permission handler class which use permission logics to determine the permission

4.12 Conf

4.12.1 permission.conf module

django-permission2 application configure

4.13 Compat

4.13.1 permission.compat module

`permission.compat.`**`is_anonyomus`**`(`*user_obj*`)`

`permission.compat.`**`is_authenticated`**`(`*user_obj*`)`

`permission.compat.`**`isstr`**`(`*x*`)`

4.14 Module contents

`permission.`**`has_permissionif_in_builtins`**`()`

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

permission, 33
permission.backends.PermissionBackend, 32
permission.compat, 33
permission.conf, 33
permission.decorators.classbase, 19
permission.decorators.functionbase, 20
permission.decorators.methodbase, 21
permission.decorators.permission_required, 21
permission.decorators.utils, 22
permission.handlers.LogicalPermissionHandler, 33
permission.handlers.PermissionHandler, 33
permission.logics.author, 23
permission.logics.base, 22
permission.logics.collaborators, 23
permission.logics.groupin, 24
permission.logics.oneself, 25
permission.logics.staff, 26
permission.templatetags.permissionif, 26
permission.utils.autodiscover, 28
permission.utils.field_lookup, 28
permission.utils.handlers, 29
permission.utils.logics, 30
permission.utils.permissions, 31

INDEX

A

add_permission_logic() (*in module permission.utils.logics*), 30
AuthorPermissionLogic (*class in permission.logics.author*), 23
autodiscover() (*in module permission.utils.autodiscover*), 28

C

CollaboratorsPermissionLogic (*class in permission.logics.collaborators*), 23
create_var() (*permission.templatetags.permissionif.TemplatePermissionIfParser method*), 27

D

discover() (*in module permission.utils.autodiscover*), 28
do_permissionif() (*in module permission.templatetags.permissionif*), 27

E

error_class (*permission.templatetags.permissionif.TemplatePermissionIfParser attribute*), 27

F

field_lookup() (*in module permission.utils.field_lookup*), 28

G

get_app_perms() (*in module permission.utils.permissions*), 31
get_full_permission_string() (*permission.logics.base.PermissionLogic method*), 22
get_handlers() (*permission.utils.handlers.PermissionHandlerRegistry method*), 29
get_model_perms() (*in module permission.utils.permissions*), 31
get_object_from_classbased_instance() (*in module permission.decorators.classbase*), 19
get_object_from_date_based_view() (*in module permission.decorators.functionbase*), 20
get_object_from_list_detail_view() (*in module permission.decorators.functionbase*), 20
get_perm_codename() (*in module permission.utils.permissions*), 31
GroupInPermissionLogic (*class in permission.logics.groupin*), 24

H

has_operator() (*in module permission.templatetags.permissionif*), 27
has_perm() (*permission.logics.author.AuthorPermissionLogic method*), 23
has_perm() (*permission.logics.base.PermissionLogic method*), 22
has_perm() (*permission.logics.collaborators.CollaboratorsPermissionLog method*), 23
has_perm() (*permission.logics.groupin.GroupInPermissionLogic method*), 24
has_perm() (*permission.logics.oneself.OneselfPermissionLogic method*), 25
has_perm() (*permission.logics.staff.StaffPermissionLogic method*), 26
has_permissionif_in_builtins() (*in module permission*), 33

I

is_anonyomus() (*in module permission.compat*), 33
is_authenticated() (*in module permission.compat*), 33
isstr() (*in module permission.compat*), 33

M

module
- permission, 33
- permission.backends.PermissionBackend, 32
- permission.compat, 33
- permission.conf, 33
- permission.decorators.classbase, 19

permission.decorators.functionbase, 20
permission.decorators.methodbase, 21
permission.decorators.permission_required, 21
permission.decorators.utils, 22
permission.handlers.LogicalPermissionHandler, 33
permission.handlers.PermissionHandler, 33
permission.logics.author, 23
permission.logics.base, 22
permission.logics.collaborators, 23
permission.logics.groupin, 24
permission.logics.oneself, 25
permission.logics.staff, 26
permission.templatetags.permissionif, 26
permission.utils.autodiscover, 28
permission.utils.field_lookup, 28
permission.utils.handlers, 29
permission.utils.logics, 30
permission.utils.permissions, 31

O

of_operator() (*in module permission.templatetags.permissionif*), 27
OneselfPermissionLogic (*class in permission.logics.oneself*), 25
OPERATORS (*permission.templatetags.permissionif.PermissionIfParser attribute*), 26

P

perm_to_permission() (*in module permission.utils.permissions*), 32
permission
    module, 33
permission.backends.PermissionBackend
    module, 32
permission.compat
    module, 33
permission.conf
    module, 33
permission.decorators.classbase
    module, 19
permission.decorators.functionbase
    module, 20
permission.decorators.methodbase
    module, 21
permission.decorators.permission_required
    module, 21
permission.decorators.utils
    module, 22
permission.handlers.LogicalPermissionHandler
    module, 33
permission.handlers.PermissionHandler
    module, 33
permission.logics.author
    module, 23
permission.logics.base
    module, 22
permission.logics.collaborators
    module, 23
permission.logics.groupin
    module, 24
permission.logics.oneself
    module, 25
permission.logics.staff
    module, 26
permission.templatetags.permissionif
    module, 26
permission.utils.autodiscover
    module, 28
permission.utils.field_lookup
    module, 28
permission.utils.handlers
    module, 29
permission.utils.logics
    module, 30
permission.utils.permissions
    module, 31
permission_required() (*in module permission.decorators.classbase*), 19
permission_required() (*in module permission.decorators.functionbase*), 20
permission_required() (*in module permission.decorators.methodbase*), 21
permission_required() (*in module permission.decorators.permission_required*), 21
permission_to_perm() (*in module permission.utils.permissions*), 32
PermissionHandlerRegistry (*class in permission.utils.handlers*), 29
PermissionIfParser (*class in permission.templatetags.permissionif*), 26
PermissionLogic (*class in permission.logics.base*), 22

R

redirect_to_login() (*in module permission.decorators.utils*), 22
register() (*permission.utils.handlers.PermissionHandlerRegistry method*), 29
remove_permission_logic() (*in module permission.utils.logics*), 30
replace_builtin_if() (*in module permission.templatetags.permissionif*), 27

S

StaffPermissionLogic (*class in permission.logics.staff*), 26

T

TemplatePermissionIfParser (*class in permission.templatetags.permissionif*), 27

translate_token() (*permission.templatetags.permissionif.PermissionIfParser method*), 27

U

unregister() (*permission.utils.handlers.PermissionHandlerRegistry method*), 29