django-permission Documentation

Release 0.8.8

Alisue <lambdalisue@hashnote.net>

October 29, 2015

django-permission

Author Alisue <lambdalisue@hashnote.net>

Supported python versions Python 2.6, 2.7, 3.2, 3.3, 3.4

Supported django versions Django 1.2 - 1.8

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

It is developed based on the authentication backend system introduced in Django 1.2. This library does support Django 1.2 and higher.

1.1 Documentation

http://django-permission.readthedocs.org/en/latest/

1.2 Installation

Use pip like:

```
$ pip install django-permission
```

1.3 Usage

The following might help you to understand as well.

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

1.3.1 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

1.3.2 Autodiscovery

This is a new feature, added in django-permission 0.6.0, and the behavior was changed in django-permission 0.6.3. Like django's admin package, django-permission 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-permission 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)
```

Quick tutorial

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

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

admin.autodiscover()
# add this line
import permission; permission.autodiscover()

urlpatterns = patterns('',
    url(r'^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()),
)
```

You can specify a different module or variable name, with `PERMISSION_AUTODISCOVER_MODULE_NAME` or `PERMISSION_AUTODISCOVER_VARIABLE_NAME` respectively.

1.3.3 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

```python
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: From django-permission version 0.8.0, you can specify related object with *field__name* attribute like django queryset lookup. See the working example below:

```python
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",
    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

#
# 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
    # 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 http://django-permission.readthedocs.org/en/latest/_modules/permission/logics/author.html#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: From django-permission version 0.8.0, 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

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 http://django-permission.readthedocs.org/en/latest/_modules/permission/logics/collaborators.html#CollaboratorsPermissionLogic to learn how this logic works.

There are StaffPermissionLogic and GroupInPermissionLogic for `is_staff` or ``group` based permission logic as well.

Customize permission logic

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

1.4 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 cannnot 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
```

1.5 Override the builtin `if` template tag

django-permission 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>
{% 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 %}
```

CHAPTER 2

API documentation

2.1 permission package

2.1.1 Subpackages

permission.decorators package

Submodules

permission.decorators.classbase module

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 instance

An instance of model object or None

`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_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

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

permission.decorators.functionbase module

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 instance

An instance of model object or None

`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 instance

An instance of model object or None

`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_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

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

permission.decorators.methodbase module

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_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 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
```

permission.decorators.permission_required module

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.

permission.decorators.utils module

Decorator utility module

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

redirect to login

Module contents

permission.logics package

Submodules

permission.logics.author module

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: *`permission.logics.base.PermissionLogic`*

Permission logic class for author based permission system

Methods

`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 boolean

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

permission.logics.base module

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

Bases: `object`

Abstract permission logic class

Methods

`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

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

Note: Sub class must override this method.

permission.logics.collaborators module

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: *`permission.logics.base.PermissionLogic`*

Permission logic class for collaborators based permission system

Methods

`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 boolean

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

permission.logics.groupin module

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: *`permission.logics.base.PermissionLogic`*

Permission logic class for group based permission system

Methods

`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_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 boolean

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

permission.logics.oneself module

Permission logic module to manage users' self-modifications

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

Bases: *`permission.logics.base.PermissionLogic`*

Permission logic class to manage users' self-modifications

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

Methods

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 boolean

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

permission.logics.staff module

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: `permission.logics.base.PermissionLogic`

Permission logic class for is_staff authority based permission system

Methods

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 boolean

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

Module contents

Permission logic module

permission.templatetags package

Submodules

permission.templatetags.patch module

django if templatetag patch

`permission.templatetags.patch.`**`parser_patch`**(*instance*)

permission.templatetags.permissionif module

permissionif templatetag

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

Bases: `django.template.smartif.IfParser`

Permission if parser

Methods

`OPERATORS` **= {'and': <class 'django.template.smartif.Operator'>, '>=': <class 'django.template.smartif.Operator'>, 'no**

use extra operator

`translate_token`(*token*)

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

Bases: *`permission.templatetags.permissionif.PermissionIfParser`*

Methods

`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

Module contents

permission.tests package

Subpackages

permission.tests.test_decorators package

Submodules

permission.tests.test_decorators.test_classbase module

class `permission.tests.test_decorators.test_classbase.`**`PermissionClassDecoratorsTestCase`**(*methoc*

Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

`setUp`()

tearDown()

test_with_get_object()

test_with_get_queryset()

test_with_object()

test_with_queryset()

permission.tests.test_decorators.test_functionbase module

class `permission.tests.test_decorators.test_functionbase.`**`PermissionFunctionDecoratorsTestCase`**

Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

setUp()

tearDown()

test_date_based_object_id()

test_date_based_slug()

test_list_detail_object_id()

test_list_detail_slug()

permission.tests.test_decorators.test_methodbase module

class `permission.tests.test_decorators.test_methodbase.`**`PermissionClassDecoratorsTestCase`**(*metho*

Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

setUp()

tearDown()

test_with_get_object()

test_with_get_queryset()

test_with_object()

test_with_queryset()

permission.tests.test_decorators.test_permission_required module

class `permission.tests.test_decorators.test_permission_required.`**`PermissionDecoratorsTestCase`**`(`

Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

`setUp``()`

`tearDown``()`

`test_class_views``()`

`test_function_views``()`

`test_method_views``()`

`test_permission_required``()`

class `permission.tests.test_decorators.test_permission_required.`**`View`**`(`***kwargs*`)`

Bases: `django.views.generic.base.View`

Methods

`dispatch``(`*request*, **args*, ***kwargs*`)`

`get_object``(`*queryset=None*`)`

`permission.tests.test_decorators.test_permission_required.`**`view_func`**`(`*request*, **args*, ***kwargs*`)`

permission.tests.test_decorators.utils module

`permission.tests.test_decorators.utils.`**`create_mock_class`**`(`*name*, *base*, *instance=None*`)`

`permission.tests.test_decorators.utils.`**`create_mock_handler`**`()`

`permission.tests.test_decorators.utils.`**`create_mock_model`**`()`

`permission.tests.test_decorators.utils.`**`create_mock_queryset`**`(`*obj*`)`

`permission.tests.test_decorators.utils.`**`create_mock_request`**`(`*mock_permission_handler*`)`

`permission.tests.test_decorators.utils.`**`create_mock_view_class`**`(`*view_func*`)`

`permission.tests.test_decorators.utils.`**`create_mock_view_func`**`()`

Module contents

permission.tests.test_logics package

Submodules

permission.tests.test_logics.test_author module

class `permission.tests.test_logics.test_author.`**`PermissionLogicsAuthorPermissionLogicTestCase`**`(`

Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

`setUp``()`

`test_constructor``()`

`test_constructor_with_specifing_any_permission``()`

`test_constructor_with_specifing_change_permission``()`

`test_constructor_with_specifing_delete_permission``()`

`test_constructor_with_specifing_field_name``()`

`test_has_perm_add_with_obj``()`

`test_has_perm_add_with_obj_author``()`

`test_has_perm_add_with_obj_author_diff_field_name``()`

`test_has_perm_add_with_obj_author_non_any``()`

`test_has_perm_add_with_obj_author_non_any_no_change``()`

`test_has_perm_add_with_obj_author_non_any_no_delete``()`

`test_has_perm_add_with_obj_with_anonymous``()`

`test_has_perm_add_without_obj``()`

`test_has_perm_add_without_obj_with_anonymous``()`

`test_has_perm_change_with_obj``()`

`test_has_perm_change_with_obj_author``()`

`test_has_perm_change_with_obj_author_diff_field_name``()`

`test_has_perm_change_with_obj_author_non_any``()`

`test_has_perm_change_with_obj_author_non_any_no_change``()`

`test_has_perm_change_with_obj_author_non_any_no_delete``()`

`test_has_perm_change_with_obj_with_anonymous``()`

`test_has_perm_change_without_obj``()`

`test_has_perm_change_without_obj_with_anonymous``()`

`test_has_perm_delete_with_obj``()`

`test_has_perm_delete_with_obj_author``()`

`test_has_perm_delete_with_obj_author_diff_field_name``()`

test_has_perm_delete_with_obj_non_any()

test_has_perm_delete_with_obj_non_any_no_change()

test_has_perm_delete_with_obj_non_any_no_delete()

test_has_perm_delete_with_obj_with_anonymous()

test_has_perm_delete_without_obj()

test_has_perm_delete_without_obj_with_anonymous()

permission.tests.test_logics.test_base module

class permission.tests.test_logics.test_base.**PermissionLogicsPermissionLogicTestCase**(*methodNan*

Bases: django.test.testcases.TestCase

Attributes

available_apps	
fixtures	

Methods

setUp()

test_constructor()

test_has_perm_add_wiht_obj()

test_has_perm_add_wihtout_obj()

test_has_perm_change_wiht_obj()

test_has_perm_change_wihtout_obj()

test_has_perm_delete_wiht_obj()

test_has_perm_delete_wihtout_obj()

permission.tests.test_logics.test_collaborators module

class permission.tests.test_logics.test_collaborators.**PermissionLogicsCollaboratorsPermissior**

Bases: django.test.testcases.TestCase

Attributes

available_apps	
fixtures	

Methods

setUp()

test_constructor()

test_constructor_with_specifing_any_permission()

test_constructor_with_specifing_change_permission()

test_constructor_with_specifing_delete_permission()

test_constructor_with_specifing_field_name()

test_has_perm_add_with_obj()

test_has_perm_add_with_obj_collaborators()

test_has_perm_add_with_obj_collaborators_diff_field_name()

test_has_perm_add_with_obj_collaborators_non_any()

test_has_perm_add_with_obj_collaborators_non_any_no_change()

test_has_perm_add_with_obj_collaborators_non_any_no_delete()

test_has_perm_add_with_obj_with_anonymous()

test_has_perm_add_without_obj()

test_has_perm_add_without_obj_with_anonymous()

test_has_perm_change_with_obj()

test_has_perm_change_with_obj_collaborators()

test_has_perm_change_with_obj_collaborators_diff_field_name()

test_has_perm_change_with_obj_collaborators_non_any()

test_has_perm_change_with_obj_collaborators_non_any_no_change()

test_has_perm_change_with_obj_collaborators_non_any_no_delete()

test_has_perm_change_with_obj_with_anonymous()

test_has_perm_change_without_obj()

test_has_perm_change_without_obj_with_anonymous()

test_has_perm_delete_with_obj()

test_has_perm_delete_with_obj_collaborators()

test_has_perm_delete_with_obj_collaborators_diff_field_name()

test_has_perm_delete_with_obj_non_any()

test_has_perm_delete_with_obj_non_any_no_change()

test_has_perm_delete_with_obj_non_any_no_delete()

test_has_perm_delete_with_obj_with_anonymous()

test_has_perm_delete_without_obj()

test_has_perm_delete_without_obj_with_anonymous()

permission.tests.test_logics.test_groupin module

class permission.tests.test_logics.test_groupin.**PermissionLogicsAuthorPermissionLogicTestCase**

Bases: django.test.testcases.TestCase

Attributes

available_apps	
fixtures	

Methods

setUp()

test_constructor()

test_constructor_with_specifing_add_permission()

test_constructor_with_specifing_any_permission()

test_constructor_with_specifing_change_permission()

test_constructor_with_specifing_delete_permission()

test_has_perm_add_with_obj()

test_has_perm_add_with_obj_with_anonymous()

test_has_perm_add_with_obj_with_two_groups()

test_has_perm_add_with_obj_without_any_permission()

test_has_perm_add_without_obj()

test_has_perm_add_without_obj_with_anonymous()

test_has_perm_add_without_obj_with_two_groups()

test_has_perm_add_without_obj_without_any_permission()

test_has_perm_change_with_obj()

test_has_perm_change_with_obj_with_anonymous()

test_has_perm_change_with_obj_with_two_groups()

test_has_perm_change_with_obj_without_any_permission()

test_has_perm_change_without_obj()

test_has_perm_change_without_obj_with_anonymous()

test_has_perm_change_without_obj_with_two_groups()

test_has_perm_change_without_obj_without_any_permission()

test_has_perm_delete_with_obj()

test_has_perm_delete_with_obj_with_anonymous()

test_has_perm_delete_with_obj_with_two_groups()

test_has_perm_delete_with_obj_without_any_permission()

test_has_perm_delete_without_obj()

test_has_perm_delete_without_obj_with_anonymous()

test_has_perm_delete_without_obj_with_two_groups()

test_has_perm_delete_without_obj_without_any_permission()

permission.tests.test_logics.test_oneself module

class `permission.tests.test_logics.test_oneself.`**`PermissionLogicsOneselfPermissionLogicTestCas`**

Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

`setUp``()`

`test_constructor``()`

`test_constructor_with_specifying_any_permission``()`

`test_constructor_with_specifying_change_permission``()`

`test_constructor_with_specifying_delete_permission``()`

`test_has_perm_add_with_himself``()`

`test_has_perm_add_with_himself_non_any``()`

`test_has_perm_add_with_himself_non_any_no_change``()`

`test_has_perm_add_with_himself_non_any_no_delete``()`

`test_has_perm_add_with_obj``()`

`test_has_perm_add_with_obj_with_anonymous``()`

`test_has_perm_add_without_obj``()`

`test_has_perm_add_without_obj_with_anonymous``()`

`test_has_perm_change_with_himself``()`

`test_has_perm_change_with_himself_non_any``()`

`test_has_perm_change_with_himself_non_any_no_change``()`

`test_has_perm_change_with_himself_non_any_no_delete``()`

`test_has_perm_change_with_obj``()`

`test_has_perm_change_with_obj_with_anonymous``()`

`test_has_perm_change_without_obj``()`

`test_has_perm_change_without_obj_with_anonymous``()`

`test_has_perm_delete_with_himself``()`

`test_has_perm_delete_with_himself_non_any_no_change``()`

`test_has_perm_delete_with_himself_non_any_no_delete``()`

`test_has_perm_delete_with_obj``()`

`test_has_perm_delete_with_obj_non_any``()`

`test_has_perm_delete_with_obj_with_anonymous``()`

test_has_perm_delete_without_obj()

test_has_perm_delete_without_obj_with_anonymous()

permission.tests.test_logics.test_staff module

class permission.tests.test_logics.test_staff.**PermissionLogicsStaffPermissionLogicTestCase**(*me*

Bases: django.test.testcases.TestCase

Attributes

available_apps	
fixtures	

Methods

setUp()

test_constructor()

test_constructor_with_specifing_add_permission()

test_constructor_with_specifing_any_permission()

test_constructor_with_specifing_change_permission()

test_constructor_with_specifing_delete_permission()

test_has_perm_add_with_obj()

test_has_perm_add_with_obj_with_anonymous()

test_has_perm_add_with_obj_without_any()

test_has_perm_add_without_obj()

test_has_perm_add_without_obj_with_anonymous()

test_has_perm_add_without_obj_without_any()

test_has_perm_change_with_obj()

test_has_perm_change_with_obj_with_anonymous()

test_has_perm_change_with_obj_without_any()

test_has_perm_change_without_obj()

test_has_perm_change_without_obj_with_anonymous()

test_has_perm_change_without_obj_without_any()

test_has_perm_delete_with_obj()

test_has_perm_delete_with_obj_with_anonymous()

test_has_perm_delete_with_obj_without_any()

test_has_perm_delete_without_obj()

test_has_perm_delete_without_obj_with_anonymous()

test_has_perm_delete_without_obj_without_any()

Module contents

permission.tests.test_templatetags package

Submodules

permission.tests.test_templatetags.test_permissionif module

class `permission.tests.test_templatetags.test_permissionif.`**`PermissionTemplateTagsTestCase`**`(`*met*

Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

`setUp``()`

`tearDown``()`

`test_permissionif_tag``()`

`test_permissionif_tag_and``()`

`test_permissionif_tag_elif``()`

`test_permissionif_tag_else``()`

`test_permissionif_tag_or``()`

`test_permissionif_tag_with_obj``()`

class `permission.tests.test_templatetags.test_permissionif.`**`PermissionTemplateTagsWithBuiltinT`**

Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

`setUp``()`

`tearDown``()`

`test_if_tag``()`

`test_if_tag_and``()`

`test_if_tag_elif``()`

`test_if_tag_else``()`

test_if_tag_or()

test_if_tag_with_obj()

Module contents

permission.tests.test_utils package

Submodules

permission.tests.test_utils.test_field_lookup module

class `permission.tests.test_utils.test_field_lookup.`**`PermissionUtilsFieldLookupTestCase`**(*methodN*

Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

setUp()

test_field_lookup_author()

test_field_lookup_author_username()

test_field_lookup_editors()

test_field_lookup_editors_username()

test_field_lookup_multiple_bridge_author()

test_field_lookup_multiple_bridge_author_username()

test_field_lookup_multiple_bridge_editors()

test_field_lookup_multiple_bridge_editors__name()

test_field_lookup_single_bridge_author()

test_field_lookup_single_bridge_author_username()

test_field_lookup_single_bridge_editors()

test_field_lookup_single_bridge_editors_username()

permission.tests.test_utils.test_handlers module

class `permission.tests.test_utils.test_handlers.`**`PermissionUtilsHandlersTestCase`**(*methodName='run*

Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

setUp()

test_get_handlers()

test_register()

test_register_duplicate()

test_register_non_permission_handler()

test_register_permission_handler_instance()

test_register_with_abstract_model()

test_register_without_specifing_handler()

test_unregister()

test_unregister_absence()

permission.tests.test_utils.test_logics module

class permission.tests.test_utils.test_logics.**PermissionUtilsLogicsTestCase**(*methodName='runTest'*)

Bases: django.test.testcases.TestCase

Attributes

available_apps	
fixtures	

Methods

setUp()

tearDown()

test_add_permission_logic_private_attributes()

test_add_permission_logic_registry()

test_remove_permission_logic_exception()

test_remove_permission_logic_private_attributes()

test_remove_permission_logic_registry()

test_remove_permission_logic_registry_with_class()

permission.tests.test_utils.test_permissions module

Module contents

Submodules

permission.tests.compatibility module

class `permission.tests.compatibility.`**`TestRunner`**(*pattern=None, top_level=None, verbosity=1, interactive=True, failfast=False, keepdb=False, reverse=False, debug_sql=False, **kwargs*)

Bases: `django.test.runner.DiscoverRunner`

Methods

`setup_test_environment`(***kwargs*)

`teardown_test_environment`(***kwargs*)

permission.tests.models module

class `permission.tests.models.`**`Article`**(*id, title, content, author, editor, single_bridge, created_at*)

Bases: `django.db.models.base.Model`

Attributes

Methods

exception **`DoesNotExist`**

Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `Article.`**`MultipleObjectsReturned`**

Bases: `django.core.exceptions.MultipleObjectsReturned`

`Article.`**`author`**

`Article.`**`authors`**

`Article.`**`editor`**

`Article.`**`editors`**

`Article.`**`get_next_by_created_at`**(**moreargs, **morekwargs*)

`Article.`**`get_previous_by_created_at`**(**moreargs, **morekwargs*)

`Article.`**`multiple_bridge`**

`Article.`**`objects`** **= <django.db.models.manager.Manager object>**

`Article.`**`single_bridge`**

class `permission.tests.models.`**`Bridge`**(*id, author*)

Bases: `django.db.models.base.Model`

Attributes

Methods

exception `DoesNotExist`
Bases: `django.core.exceptions.ObjectDoesNotExist`

exception `Bridge.MultipleObjectsReturned`
Bases: `django.core.exceptions.MultipleObjectsReturned`

`Bridge.author`

`Bridge.editors`

`Bridge.objects` **= <django.db.models.manager.Manager object>**

`Bridge.permission_test_multiple_bridge`

`Bridge.permission_test_signgle_bridge`

permission.tests.test_backends module

class `permission.tests.test_backends.PermissionPermissionBackendTestCase`(*methodName='runTest'*)
Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

`setUp()`

`tearDown()`

`test_authenticate()`

`test_constructor()`

`test_has_module_perms()`

`test_has_perm_with_nil_permission`(**args*, ***kwargs*)

`test_has_perm_with_nil_permission_raise`(**args*, ***kwargs*)

`test_has_perm_with_nil_permission_raise_with_user`(**args*, ***kwargs*)

`test_has_perm_with_nil_permission_with_user`(**args*, ***kwargs*)

`test_has_perm_with_obj()`

`test_has_perm_without_obj()`

permission.tests.test_handlers module

class `permission.tests.test_handlers.PermissionLogicalPermissionHandlerTestCase`(*methodName='run*
Bases: `django.test.testcases.TestCase`

Attributes

available_apps	
fixtures	

Methods

setUp()

test_constructor_with_app_label()

test_has_perm_non_related_permission()

test_has_perm_permission_logics_called()

class permission.tests.test_handlers.**PermissionPermissionHandlersTestCase**(*methodName='runTest'*)
Bases: django.test.testcases.TestCase

Attributes

available_apps	
fixtures	

Methods

setUp()

test__get_app_perms_with_app_label()

test__get_app_perms_with_model()

test__get_model_perms()

test_constructor_with_app_label()

test_constructor_with_model()

test_get_supported_app_labels()

test_get_supported_app_labels_with_excludes()

test_get_supported_app_labels_with_excludes_change()

test_get_supported_app_labels_with_includes()

test_get_supported_app_labels_with_includes_change()

test_get_supported_permissions()

test_get_supported_permissions_with_excludes()

test_get_supported_permissions_with_excludes_change()

test_get_supported_permissions_with_includes()

test_get_supported_permissions_with_includes_change()

test_has_module_perms_fail()

test_has_module_perms_success()

test_has_perm_add_wiht_obj()

test_has_perm_add_wihtout_obj()

test_has_perm_change_wiht_obj()

test_has_perm_change_wihtout_obj()

test_has_perm_delete_wiht_obj()

test_has_perm_delete_wihtout_obj()

permission.tests.utils module

permission.tests.utils.**create_anonymous**(***kwargs*)

permission.tests.utils.**create_article**(*title*, *user=None*, *bridge=None*)

permission.tests.utils.**create_bridge**(*user=None*, *editors=None*)

permission.tests.utils.**create_group**(*name*, *user=None*)

permission.tests.utils.**create_permission**(*name*, *model=None*)

permission.tests.utils.**create_user**(*username*, ***kwargs*)

Module contents

permission.utils package

Submodules

permission.utils.autodiscover module

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')
```

permission.utils.field_lookup module

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

Lookup django model field in similar way of django query lookup

Args: 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)
        >>> aritcle.editors.add(user)
        >>> assert 'test_article' == field_lookup(article, 'title')
        >>> assert 'test_user' == field_lookup(article, 'user__username')
        >>> assert ['test_user'] == list(field_lookup(article,
        ...                                           'editors__username'))
```

permission.utils.handlers module

A utilities of permission handler

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

Bases: object

A registry class of permission handler

Methods

get_handlers()

Get registered handler instances

Returns tuple

permission handler 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 or None

A permission handler class

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

handler : permission handler class or None

A permission handler class

Raises **KeyError**

Raise when the model have not registered in registry yet.

permission.utils.logics module

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)
```

permission.utils.permissions module

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 *perm* (a string in format of 'app_label.codename') 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 set

A set of perms of the specified django application.

Examples

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

`permission.utils.permissions.`**`get_model_perms`**(*model*)
Get *perm* (a string in format of 'app_label.codename') list of the specified django model.

Parameters **model** : model class

A model class to specify the particular django model.

Returns set

A set of perms of the specified django model.

Examples

```
>>> sorted(get_model_perms(Permission)) == ['auth.add_permission', 'auth.change_permission', 'au
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'
True
>>> get_perm_codename('app_label.codename') == 'codename'
True
>>> get_perm_codename('codename_model') == 'codename_model'
True
>>> get_perm_codename('codename') == 'codename'
True
>>> get_perm_codename('app_label.app_label.codename_model') == 'app_label.codename_model'
True
```

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

Convert a identifier string permission format in 'app_label.codename' (teremd as *perm*) to a django permission instance.

Examples

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

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

Convert a django permission instance to a identifier string permission format in 'app_label.codename' (termed as *perm*).

Examples

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

Module contents

2.1.2 Submodules

2.1.3 permission.backends module

Logical permission backends module

class `permission.backends.`**`PermissionBackend`**

Bases: `object`

A handler based permission backend

Methods

`authenticate`(*username*, *password*)

Always return `None` to prevent authentication within this backend.

`has_module_perms`(*user_obj*, *app_label*)

Check if user have permission of specified app based on registered handlers.

It will raise `ObjectDoesNotExist` exception when the specified string permission does not exist and `PERMISSION_CHECK_PERMISSION_PRESENCE` is `True` in `settings` module.

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

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

Raises **django.core.exceptions.ObjectDoesNotExist**

If the specified string permission does not exist and `PERMISSION_CHECK_PERMISSION_PRESENCE` is `True` in `settings` module.

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

Check if user have permission (of object) based on registered handlers.

It will raise `ObjectDoesNotExist` exception when the specified string permission does not exist and `PERMISSION_CHECK_PERMISSION_PRESENCE` is `True` in `settings` module.

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

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

Raises **django.core.exceptions.ObjectDoesNotExist**

If the specified string permission does not exist and `PERMISSION_CHECK_PERMISSION_PRESENCE` is `True` in `settings` module.

`supports_anonymous_user` **= True**

`supports_inactive_user` **= True**

`supports_object_permissions` **= True**

2.1.4 permission.compat module

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

2.1.5 permission.conf module

django-permission application configure

2.1.6 permission.handlers module

class `permission.handlers.`**`LogicalPermissionHandler`**(*model*)

Bases: *`permission.handlers.PermissionHandler`*

Permission handler class which use permission logics to determine the permission

Attributes

Methods

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

Check if user have permission (of object) based on specified models's `_permission_logics` attribute.

The result will be stored in user_obj as a cache to reduce method call.

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

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

class `permission.handlers.`**`PermissionHandler`**(*model_or_app_label*)

Bases: `object`

Abstract permission handler class

Attributes

Methods

`excludes`

`get_supported_app_labels`()

Get app labels which this handler can treat. Specified with *`includes`* and *`excludes`* of this instance.

Returns set

A set instance of app_label

get_supported_permissions()
Get permissions which this handler can treat. Specified with `includes` and `excludes` of this instance.

Returns set

A set instance of *app_label.codename* formatted permission strings

has_module_perms(*user_obj*, *app_label*)
Check if user have permission of specified app

Parameters **user_obj** : django user model instance

A django user model instance which be checked

app_label : string

Django application name

Returns boolean

Wheter the specified user have any permissions of specified app

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

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

Note: Sub class must override this method.

includes

2.1.7 permission.models module

2.1.8 Module contents

django-permission

CHAPTER 3

Indices and tables

- genindex
- modindex
- search