
Flask-RBAC Documentation

Release 0.2.0

shonenada

May 27, 2017

Contents

1	Installation	3
2	Quick Start	5
2.1	Configuration Your Application	5
2.2	Mode Setting	5
2.3	Set Role Model	6
2.4	Set User Model	7
2.5	Set User Loader	8
2.6	Set Access Rules	8
3	API Reference	9
3.1	Flask.ext.rbac.RBAC	9
3.2	Flask.ext.rbac.model.RoleMixin	11
3.3	Flask.ext.rbac.model.UserMixin	12
4	Release Changes	13
4.1	Change Logs	13
5	Author & Contributor	15

Flask-RBAC provides a Role-based Access Control module in Flask applications. It can help you to control different role of users to access your website.

- *Installation*
- *Quick Start*
 - *Configuration Your Application*
 - *Mode Setting*
 - *Set Role Model*
 - *Set User Model*
 - *Set User Loader*
 - *Set Access Rules*
- *API Reference*
 - *Flask.ext.rbac.RBAC*
 - *Flask.ext.rbac.model.RoleMixin*
 - *Flask.ext.rbac.model.UserMixin*
- *Release Changes*
- *Author & Contributor*

CHAPTER 1

Installation

Your can use pip to install Flask-RBAC:

```
$ pip install flask-rbac
```


CHAPTER 2

Quick Start

This documents show how to create Flask-RBAC extension easily and quickly.

Configuration Your Application

As same as many Flask extensions, you need to configuration your application:

```
from flask import Flask
from flask_rbac import RBAC

app = Flask(__name__)
rbac = RBAC(app)
```

or you can configuration using factory method:

```
from flask import Flask
from flask_rbac import RBAC

rbac = RBAC()

def create_app():
    app = Flask(__name__)

    rbac.init_app(app)

    return app
```

Mode Setting

There are two modes for Flask-RBAC, `RBAC_USE_WHITE` decide whether use white list to check the permission. And it set `False` to default.

<code>RBAC_USE_WHITE = True</code>	Only allowing rules can access the resources. This means, all deny rules and rules you did not add cannot access the resources.
<code>RBAC_USE_WHITE = False</code>	Only denying rules cannot access the resources.

Change it using:

```
app.config['RBAC_USE_WHITE'] = True
```

Set Role Model

Flask-RBAC implements some methods need by Flask-RBAC in RoleMixin class. You can use RoleMixin as your role model:

```
class Role(RoleMixin):
    pass

anonymous = Role('anonymous')
```

However, if your application is working under SQLAlchemy, and you want to save the roles in database, you need to override the Role class to adapt your application, here is an example:

```
from flask_rbac import RoleMixin
from your_package.app import db

roles_parents = db.Table(
    'roles_parents',
    db.Column('role_id', db.Integer, db.ForeignKey('role.id')),
    db.Column('parent_id', db.Integer, db.ForeignKey('role.id'))
)

class Role(db.Model, RoleMixin):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(20))
    parents = db.relationship(
        'Role',
        secondary=roles_parents,
        primaryjoin=(id == roles_parents.c.role_id),
        secondaryjoin=(id == roles_parents.c.parent_id),
        backref=db.backref('children', lazy='dynamic')
    )

    def __init__(self, name):
        RoleMixin.__init__(self)
        self.name = name

    def add_parent(self, parent):
        # You don't need to add this role to parent's children set,
        # relationship between roles would do this work automatically
        self.parents.append(parent)

    def add_parents(self, *parents):
        for parent in parents:
            self.add_parent(parent)

@staticmethod
```

```
def get_by_name(name):
    return Role.query.filter_by(name=name).first()
```

After create role model, you can add your model to Flask-RBAC:

```
rbac.set_role_model(Role)
```

Or use decorator to set role model for Flask-RBAC:

```
@rbac.as_role_model
class Role(RoleMixin):
    # codes go here
```

Set User Model

Same as the RoleMixin, UserMixin implements some methods for Flask-RBAC, You can extend it directly:

```
from flask_rbac import UserMixin

class User(UserMixin):
    pass

a_user = User()
```

Well, if your application works under SQLAlchemy:

```
from flask_rbac import UserMixin
from your_package.app import db

users_roles = db.Table(
    'users_roles',
    db.Column('user_id', db.Integer, db.ForeignKey('user.id')),
    db.Column('role_id', db.Integer, db.ForeignKey('role.id'))
)

class User(db.Model, UserMixin):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(30), nullable=False)
    # Other columns
    roles = db.relationship(
        'Role',
        secondary=users_roles,
        backref=db.backref('roles', lazy='dynamic')
    )

    def add_role(self, role):
        self.roles.append(role)

    def add_roles(self, roles):
        for role in roles:
            self.add_role(role)

    def get_roles(self):
        for role in self.roles:
            yield role
```

Same as role model, you should add user model to Flask-RBAC:

```
rbac.set_user_model(User)
```

Or using decorator:

```
@rbac.as_user_model  
class User(UserMixin):  
    # codes go here
```

Set User Loader

Flask-RBAC need to know who is current user, so it requires you to provide a function which tells it who is current user.

Flask-RBAC will load current user from [Flask-Login](#) if you have install it by default.

If you save current user in *flask.g*, here is an example for you:

```
from flask import g, current_app  
  
@app.route('/signin', methods=['POST'])  
@rbac.allow(['anomous'], methods=['POST'])  
def signin():  
    # Sign in logic...  
    g.current_user = user  
  
def get_current_user():  
    with current_app.request_context():  
        return g.current_user  
  
rbac.set_user_loader(get_current_user)
```

Set Access Rules

You can use *allow* and *deny* to add rules to Flask-RBAC:

```
@app.route('/')  
@rbac.allow(['anonymous'], methods=['GET'])  
def index():  
    # your codes.  
    pass  
  
@app.route('/account/signin', methods=['GET', 'POST'])  
@rbac.deny(['logged_user'], methods=['GET', 'POST'])  
def signin():  
    # show sign in page or handle sign in request.  
    pass
```

The code above adding two rules:

- Allows user of *anonymous* role to *GET /*.
- Deny user of *logged_user* role to *GET* and *POST /account/signin*.

CHAPTER 3

API Reference

Flask.ext.rbac.RBAC

```
class flask_rbac.__init__.RBAC (app=None, **kwargs)
```

This class implements role-based access control module in Flask. There are two way to initialize Flask-RBAC:

```
app = Flask(__name__)
rbac = RBAC(app)
```

or:

```
rbac = RBAC
def create_app():
    app = Flask(__name__)
    rbac.init_app(app)
    return app
```

Parameters

- **app** – the Flask object
- **role_model** – custom role model
- **user_model** – custom user model
- **user_loader** – custom user loader, used to load current user
- **permission_failed_hook** – called when permission denied.

```
allow(roles, methods, with_children=True)
```

This is a decorator function.

You can allow roles to access the view func with it.

An example:

```
@app.route('/website/setting', methods=['GET', 'POST'])
@rbac.allow(['administrator', 'super_user'], ['GET', 'POST'])
def website_setting():
    return Response('Setting page.')
```

Parameters

- **roles** – List, each name of roles. Please note that, *anonymous* is referred to anonymous. If you add *anonymous* to the rule, everyone can access the resource, unless you deny other roles.
- **methods** – List, each name of methods. methods is valid in ['GET', 'POST', 'PUT', 'DELETE']
- **with_children** – Whether allow children of roles as well. True by default.

as_role_model (*model_cls*)

A decorator to set custom model or role.

Parameters *model_cls* – Model of role.

as_user_model (*model_cls*)

A decorator to set custom model or user.

Parameters *model_cls* – Model of user.

deny (*roles, methods, with_children=False*)

This is a decorator function.

You can deny roles to access the view func with it.

An example:

```
@app.route('/article/post', methods=['GET', 'POST'])
@rbac.deny(['anonymous', 'unactivated_role'], ['GET', 'POST'])
def article_post():
    return Response('post page.')
```

Parameters

- **roles** – List, each name of roles.
- **methods** – List, each name of methods. methods is valid in ['GET', 'POST', 'PUT', 'DELETE']
- **with_children** – Whether allow children of roles as well. True by default.

exempt (*view_func*)

Exempt a view function from being checked permission. It is useful when you are using white list checking.

Example:

```
@app.route('/everyone/can/access')
@rbac.exempt
def everyone_can_access():
    return 'Hello~'
```

Parameters *view_func* – The view function going to be exempted.

get_app (*reference_app=None*)

Helper method that implements the logic to look up an application.

has_permission (*method, endpoint, user=None*)

Return does the current user can access the resource. Example:

```
@app.route('/some_url', methods=['GET', 'POST'])
@rbac.allow(['anonymous'], ['GET'])
def a_view_func():
    return Response('Blah Blah...')
```

If you are not logged.

rbac.has_permission('GET', 'a_view_func') return True. *rbac.has_permission('POST', 'a_view_func')* return False.

Parameters

- **method** – The method wait to check.
- **endpoint** – The application endpoint.
- **user** – user who you need to check. Current user by default.

init_app (*app*)

Initialize application in Flask-RBAC. Adds (RBAC, app) to flask extensions. Adds hook to authenticate permission before request.

Parameters **app** – Flask object

set_hook (*hook*)

Set hook which called when permission is denied If you haven't set any hook, Flask-RBAC will call:

```
abort(403)
```

Parameters **hook** – Hook function

set_role_model (*model*)

Set custom model of role.

Parameters **model** – Model of role.

set_user_loader (*loader*)

Set user loader, which is used to load current user. An example:

```
from flask_login import current_user
rbac.set_user_loader(lambda: current_user)
```

Parameters **loader** – Current user function.

set_user_model (*model*)

Set custom model of User

Parameters **model** – Model of user

Flask.ext.rbac.model.RoleMixin

class flask_rbac.model.RoleMixin (*name=None*)

This provides implementations for the methods that Flask-RBAC wants the role model to have.

Parameters `name` – The name of role.

add_parent (`parent`)

Add a parent to this role, and add role itself to the parent's children set. you should override this function if neccessary.

Example:

```
logged_user = RoleMixin('logged_user')
student = RoleMixin('student')
student.add_parent(logged_user)
```

Parameters `parent` – Parent role to add in.

add_parents (*`parents`)

Add parents to this role. Also should override if neccessary. Example:

```
editor_of_articles = RoleMixin('editor_of_articles')
editor_of_photonews = RoleMixin('editor_of_photonews')
editor_of_all = RoleMixin('editor_of_all')
editor_of_all.add_parents(editor_of_articles, editor_of_photonews)
```

Parameters `parents` – Parents to add.

static get_by_name (`name`)

A static method to return the role which has the input name.

Parameters `name` – The name of role.

get_name ()

Return the name of this role

Flask.ext.rbac.model.UserMixin

class flask_rbac.model.`UserMixin` (`roles=[]`)

This provides implementations for the methods that Flask-RBAC wants the user model to have.

Parameters `roles` – The roles of this user should have.

add_role (`role`)

Add a role to this user.

Parameters `role` – Role to add.

add_roles (*`roles`)

Add roles to this user.

Parameters `roles` – Roles to add.

CHAPTER 4

Release Changes

Change Logs

Release 0.2.1 (Apr 4, 2014)

(Thanks suggestion from tonyseek)

- Make flask.ext.login.current_user to be default user.
- Add *as_role_model* and *as_user_model* decorator to set role and user model.

Release 0.2.0 (Jan 31, 2014)

- Add *exempt* method.

Release 0.1.0 (Jan 27, 2014)

- First public release.

CHAPTER 5

Author & Contributor

- Yaoda Liu <shonenada@gmail.com>

Index

A

add_parent() (flask_rbac.model.RoleMixin method), 12
add_parents() (flask_rbac.model.RoleMixin method), 12
add_role() (flask_rbac.model.UserMixin method), 12
add_roles() (flask_rbac.model.UserMixin method), 12
allow() (flask_rbac.__init__.RBAC method), 9
as_role_model() (flask_rbac.__init__.RBAC method), 10
as_user_model() (flask_rbac.__init__.RBAC method), 10

D

deny() (flask_rbac.__init__.RBAC method), 10

E

exempt() (flask_rbac.__init__.RBAC method), 10

G

get_app() (flask_rbac.__init__.RBAC method), 10
get_by_name() (flask_rbac.model.RoleMixin static method), 12
get_name() (flask_rbac.model.RoleMixin method), 12

H

has_permission() (flask_rbac.__init__.RBAC method), 11

I

init_app() (flask_rbac.__init__.RBAC method), 11

R

RBAC (class in flask_rbac.__init__), 9
RoleMixin (class in flask_rbac.model), 11

S

set_hook() (flask_rbac.__init__.RBAC method), 11
set_role_model() (flask_rbac.__init__.RBAC method), 11
set_user_loader() (flask_rbac.__init__.RBAC method), 11
set_user_model() (flask_rbac.__init__.RBAC method), 11