Flask Restful API权限管理设计与实现

geekpy 关注

0.579 2017.08.10 16:21:42 字数 1,063 阅读 4,595

在使用flask设计restful api的时候，有一个很重要的问题就是如何进行权限管理，以及如何进行角色的定义，在网上找了一下没有发现有类似的资料，虽然有些针对网站进行的权限管理设计，但是跟restful api接口的权限管理还是有很多不同的，于是乎自己动手，丰衣足食。为方便后来者，特撰此文！

geekpy
总资产20
关注

权限的设计

从本质上思考，我需要为每个API接口设定相应的权限，所以针对API的权限列表跟普通网站的权限设计是不同的，普通网站的权限设计是针对某个功能，比如是否可以comment功能，通常的权限定义如下：

```
class Permission:
    """
    权限表
    """
    COMMENT = 0x01  # 评论
    MODERATE_COMMENT = 0x02  # 移除评论
```

但是针对restful api，我们更希望权限是针对我们的api接口，而restful api接口是跟我们路由的

```
comment_permission = { comments : { post : True,  get : True,  delete : False}}
```

角色的设计

通常，我们在做网站的角色设计时会将角色存储在数据库当中，并会通过或运算(|)赋予角色以特定权限，如下：

```
class Role(db.Model):
    """
    用户角色
    """
    id = db.Column(db.Integer, primary_key=True)
    # 该用户角色名称
    name = db.Column(db.String(164))
    # 该用户角色是否为默认
    default = db.Column(db.Boolean, default=False, index=True)
    # 该用户角色对应的权限
    permissions = db.Column(db.Integer)
    # 该用户角色和用户的关系
    # 角色为该用户角色的所有用户
    users = db.relationship('User', backref='role', lazy='dynamic')

    @staticmethod
    def insert_roles():
        """
        创建用户角色
        """
        roles = {
            # 定义了两个用户角色(User, Admin)
            'User': (Permission.COMMENT, True),
            'Admin': (Permission.COMMENT |
                      Permission.MODERATE_COMMENT, False)
        }
```

```
            role.permissions = roles[r][0]
            role.default = roles[r][1]
            db.session.add(role)
            db.session.commit()
```

这里其实我一直没有搞明白，为什么要将角色存储于数据库当中，在我看来这只会导致更多的I/O操作从而影响系统的性能，因此我在设计角色的时候根本没有考虑存储到数据库中，角色的数据结构在系统运行时，直接存在内存当中，这样在接口调用时，可以直接使用角色相关的数据结构。而且由于我们的权限设计也不太相同，所以我针对restful api设计的Role如下：

```
USER = 1
ADMIN = 2
VISITOR = 3

Role = {
    USER: {
        "comment": {"post": True, "patch": True, "get": True, "delete": True},
        "share": {"post": True}
    },
    ADMIN: {
        "comment": {"post": True, "patch": True, "get": True, "delete": True},
        "share": {"post": True}
    },
    VISITOR: {
        "comment": {"get": True},
        "share": {"post": True}
    }
}
```

用户可以被赋予特定的role，如下：

那么接口如何判断用户是否有权限访问呢：

首先用户访问接口时都会带有用户信息，restful api一般是通过token来表明身份，系统通过token来获取用户的信息，比如用户名，然后我们可以通过用户名来获取用户的角色role，假设我们访问的接口是comments endpoint的post接口，那么就可以如下判断：

```
def access_control(user):
    """判断用户是否有访问权限，有就返回True，没有返回False"""

    # 首先要获取到API的endpoint和http method，此处代码省略
    ...

    role = user.get('role', VISITOR)
    try:
        if not Role[role][endpoint][http_method]:
            return False
        return True
    except KeyError:
        return False
```

由于基本所有的接口都需要access control，那么我们把上边的代码稍作改变，让它成为一个decorator，同时，user信息也可以直接获取而不需要从参数传递，如下：

```
from functools import wraps

def get_role():
    # 这里get_resource_by_name用于从数据库中获取该用户的信息，这个需要自己去定义
    # 另外我们可以在登录验证的时候或者token验证的时候讲user name存储于全局变量g中，这样我们可以随时获取i
    user = UserModel.get_resource_by_name(g.user_name)
    return user.get("role", VISITOR)

def access_control(func):
```

写下你的评论... 评论6 赞27 ...

```
            if not Role[role][endpoint][http_method]:
                return make_response(
                    jsonify({'error': 'no permission'}), 403)
            return func(*args, **kwargs)
        except KeyError:
            return make_response(
                jsonify({'error': 'no permission'}), 403)
    return wrap_func

```

以下是一个获取图片resource的使用示例

```
from flask_restful import Resource

class ImageResource(Resource):
    def __init__(self):
        super(ImageResource, self).__init__()

    @token_auth.login_required
    @access_control
    def get(self, resource_id):
        response = resource_get(resource_id)
        return response
```

这里另外一个decortor @token_auth.login_required用于token验证，大家可以先不用理会。到这里我们已经可以针对每个接口自动判断该用户是否有权限访问了，而所有权限的变化，都可以通过修改Role中的权限来进行更改，而不需要更改原来的代码，很爽吧，有木有?
不过，笔者在项目中还遇到了另外一个问题，有时候针对一个接口所有的user都应该有权限，但是针对特定的resource，只能resource owner可以操作，举个栗子，比如我们要删除某个评论，但

```python
def get_resource_owner():
    """获取resource的owner"""
    # 自定义，代码省略
    ...

def owner_permission_required(func):
    @wrap(func)
    def wrap_func(*args, **kwargs):
        if g.user_name == get_resource_owner():
            return func(*args, **kwargs)
        return make_response(
            jsonify({'error': 'no permission'}), 403)
    return wrap_func
```

使用如下：

```python
from flask_restful import Resource

class CommentResource(Resource):
    def __init__(self):
        super(CommentResource, self).__init__()

    @token_auth.login_required
    @access_control
    @owner_permission_required
    @marshal_with(image_fields)
    def delete(self, resource_id):
        response = resource_delete(resource_id)
        return response
```