WiseAdministrator

君子博学而日参省乎己，则知明而行无过矣

Flask-flask_jwt_extended组件

安装：

```
pip3 install Flask-JWT-Extended
```

什么是Flask-JWT-Extended

之前已经说过jwt是序列化并加密过的json串，那很明显extend则是对之前功能的拓展。那下面我们就该看看拓展的强大之处。

app.py

```
from flask_jwt_extended import JWTManager


app.config['JWT_SECRET_KEY'] = 'jose'  # token密钥
app.config['JWT_BLACKLIST_ENABLED'] = True  # 黑名单管理
app.config['JWT_BLACKLIST_TOKEN_CHECKS'] = ['access', 'refresh']  # 允许将access and refresh tokens加入黑名单
#注册jwt
jwt = JWTManager(app)
```

接下来，因为此插件剔除了自动生成/auth，我们需要拓展user的功能，生成用户有关的：注册、登录、登出等功能，以及用户令牌认证以及刷新、失效等功能。

user.py

```
#关联包介绍

#request约束
_user_parser = reqparse.RequestParser()
_user_parser.add_argument('username',
                          type=str,
                          required=True,
                          help="This field cannot be blank."
                          )
_user_parser.add_argument('password',
                          type=str,
                          required=True,
                          help="This field cannot be blank."
                          )
#用户登录
class UserLogin(Resource):
    def post(self):
#从parser获取request资料
        data = _user_parser.parse_args()
#找到用户资料
        user = UserModel.find_by_username(data['username'])
#进行用户认证
        if user and safe_str_cmp(user.password, data['password']):
#生成token，认证令牌和刷新令牌
            access_token = create_access_token(identity=user.id, fresh=True)
            refresh_token = create_refresh_token(user.id)
```

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签

我的标签

PYTHON笔记(158)
Linux(22)
数据库(19)
WEB前端(18)
go语言(15)
MySql数据库(6)
未分类(4)
Docker(1)

随笔分类

golang语言

随笔档案

2019年10月(1
2019年8月(4)
2019年7月(2)
2019年6月(4)
2019年5月(4)

```
            return {
                        'access_token': access_token,
                        'refresh_token': refresh_token
                    }, 200
#认证失败
        return {"message": "Invalid Credentials!"}, 401
```

claims

这是jwt的数据存储机制

```
jwt = JWTManager(app)
#user_claims_loader定义我们附加到jwt的有效数据
#在每一个收到jwt保护的端口，我们都可以使用get_jwt_claims检索这些数据
#如下所示
@jwt.user_claims_loader
def add_claims_to_jwt(identity):
    if identity == 1:   # instead of hard-coding, we should read from a config file to get a list of admins instead
        return {'is_admin': True}
    return {'is_admin': False}
```

通过这个示例我们可以模拟用户管理机制，如下对item.py的修改，非管理员用户不可以删除item：

注意下get_jwt_claims，说明在上文。

```
    @jwt_required
    def delete(self, name):
        claims = get_jwt_claims()
```

文章分类

Docker(1)
GIT版本控制(6)
golang语言(14)
Linux(21)
PYTHON基础(44)
python项目那些事(100)
python与数据库(23)
前端(21)

文章档案

2019年10月(10)
2019年9月(12)
2019年8月(65)
2019年7月(66)
2019年6月(35)
2019年5月(19)
2019年4月(17)
2019年3月(20)

GitHub

GitHub

思维导图

```
        if not claims['is_admin']:
            return {'message': 'Admin privilege required.'}, 401

        item = ItemModel.find_by_name(name)
        if item:
            item.delete_from_db()
            return {'message': 'Item deleted.'}
        return {'message': 'Item not found.'}, 404
```

接下来就是一个个介绍特色的时候了：

jwt_optional

官网介绍是，可以可选的保护节点，无论是否发送token，都可以进入节点，然后交给逻辑判断如何处理。比较常见的判定如下：

如果请求中存在访问令牌，则将调用`get_jwt_identity()`具有访问令牌标识的端点。如果请求中不存在访问令牌，则仍将调用此端点，但`get_jwt_identity()`将返回None

示例

```
class ItemList(Resource):
    @jwt_optional
    def get(self):
        user_id = get_jwt_identity()
        items = [item.json() for item in ItemModel.find_all()]
        if user_id:
            return {'items': items}, 200
        return {
            'items': [item['name'] for item in items],
            'message': 'More data available if you log in.'
        }, 200
```

Linux基础命令

最新评论

1. Re:Linux中Xshell常用快捷键
你微信多少，我QQ 346712481，可以多交流下，linUX博大精深
--张刚强
2. Re:Linux中Xshell常用快捷键
@ 张刚强互相学习吧...
--WiseAdministrator
3. Re:Linux中Xshell常用快捷键
总结的不错
--张刚强
4. Re:工作中的常见应用
@ blog_wu科比有点慌~...
--WiseAdministrator
5. Re:工作中的常见应用
一个程序员问科比他为什么这么成功，科比问他：你知道凌晨4点的洛杉矶是什么样子吗？程序员说：不知道，那个时候我还没下班。你问这个干嘛？ 科比：不干嘛！
--blog_wu

评论排行榜

1. 数据库语言(
2. 工作中的常
3. Web项目之

create_refresh_token

这个要追溯到create_access_token。一般来说，我们都是将用户权限和用户基本资料存放在这个TOKEN中，但是当用户权限？或者资料变更时，我们就要去刷新用户的资料，或者说当access_token过期了，我们也要去更新access_token。这一部分可以细看Web API与OAuth：既生access token，何生refresh token

其实这只是定义的问题（不过如果不是自己写TOKEN规则，我们要遵守这个定义）。我们在书写create_refresh_token的逻辑时，不在需要验证用户名和密码即达到了这种效果，如果说个人不这么写逻辑，你也看不出来效果，毕竟两种TOKEN其实是一样的。

```
class TokenRefresh(Resource):
    @jwt_refresh_token_required
    def post(self):
        current_user = get_jwt_identity()
        new_token = create_access_token(identity=current_user, fresh=False)
        return {'access_token': new_token}, 200
```

这里有另一个概念，令牌的新鲜度fresh=False/True。我们可以根据此处的变更灵活处理一些敏感数据。比如说有些数据只有刚输入用户名和密码时的那段有效认证期才可以使用，当令牌刷新时即判定无效。此处匹配装饰器`fresh_jwt_required()`

item.py

POST /item/<name>的逻辑变更。

```
    @fresh_jwt_required
    def post(self, name):
        if ItemModel.find_by_name(name):
            return {'message': "An item with name '{}' already exists.".format(name)},
400


        data = self.parser.parse_args()
```

```
        item = ItemModel(name, **data)

        try:
            item.save_to_db()
        except:
            return {"message": "An error occurred while inserting the item."}, 500

        return item.json(), 201
```

expired_token_loader

invalid_token_loader

重新定义TOKEN回调错误内容，还有更多的请参考Changing Default Behaviors

```
#过期令牌
@jwt.expired_token_loader
def expired_token_callback():
    return jsonify({
        'message': 'The token has expired.',
        'error': 'token_expired'
    }), 401
#无效令牌
@jwt.invalid_token_loader
def invalid_token_callback(error):  # we have to keep the argument here, since it's
passed in by the caller internally
    return jsonify({
        'message': 'Signature verification failed.',
        'error': 'invalid_token'
    }), 401
```