Ask Company

리액트와 함께 장고 시작하기 / 장고 DRF

Throttling (최대 호출 횟수 제한하기)

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

용어 정리

Rate : 지정 기간 내에 허용할 최대 호출 횟수

Scope : 각 Rate에 대한 별칭 (alias)

Throttle : 특정 조건 하에 최대 호출 횟수를 결정하는 로직이 구현된 클래스

https://www.django-rest-framework.org/api-guide/throttling/

기본 제공 Throttle

AnonRateThrottle

인증 요청에는 제한을 두지 않고, 비인증 요청에는 IP 단위로 횟수 제한

디폴트 scope : 'anon'

UserRateThrottle

인증 요청에는 유저 단위로 횟수를 제한하고, 비인증 요청에는 IP 단위로 횟수 제한

디폴트 scope : 'user'

ScopedRateThrottle

인증 요청에는 유저 단위로 횟수를 제한하고, 비인증 요청에는 IP 단위로 횟수 제한

각 APIView내 throttle_scope 설정을 읽어, APIView 별로 서로 다른 Scope을 적용

설정 예

디폴트 설정

```
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_CLASSES': [],
    'DEFAULT_THROTTLE_RATES': {
        'anon': None,
        'user': None,
    },
}
```

설정 예

```
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_CLASSES': [
        'rest_framework.throttling.UserRateThrottle',
    ],
    'DEFAULT_THROTTLE_RATES': {
        'user': '10/day',
    },
}
```

ViewSet 예

```
from rest_framework.throttling import UserRateThrottle

class PostViewSet(ViewSet):
    throttle_classes = UserRateThrottle
```

최대 호출 횟수 제한을 넘긴다면?

429 Too Many Requests 응답

예외 메세지에 API 활용이 가능한 시점을 알려줍니다.
→ 이는 Throttle의 wait 멤버함수를 통해 계산

Cache

Cache

매 요청 시마다 cache에서 timestamp list를 get/set → 캐시 성능이 중요

SimpleRateThrottle에서는 다음과 같이 디폴트 캐시 설정

```
# rest_framework/throttling.py

from django.core.cache import cache as default_cache

class SimpleRateThrottle(BaseThrottle):
    cache = default_cache
```

장고의 Cache 지원

기본 settings의 디폴트 캐시 : 로컬 메모리 캐시

다양한 캐시 지원

Memcached 서버 지원 : django.core.cache.backends.MemcachedCache 혹은 PyLibMCCache

데이터베이스 캐시 : django.core.cache.backends.DatabaseCache

파일 시스템 캐시 : django.core.cache.backends..FileBasedCache

로컬 메모리 캐시 : django.core.cache.backends.LocMemCache

더미 캐시 : django.core.cache.backends.dummy.DummyCache → 실제로 캐시를 수행하진 않습니다.

redis를 활용한 캐시

django-redis-cache : https://github.com/sebleier/django-redis-cache

Throttle별 캐시 설정

settings.CACHES의 "default" 사용

Throttle 클래스 별로 다른 캐시 설정 지원

```
from django.core.cache import caches

class CustomAnonRateThrottle(AnonRateThrottle):
    cache = caches['alternate']
```

https://docs.djangoproject.com/en/2.2/ref/settings/#caches

https://www.django-rest-framework.org/api-guide/throttling/#setting-up-the-cache

설정

Rates 포맷

포맷 : "숫자/간격"

숫자 : 지정 간격 내의 최대 요청 제한 횟수

간격 : 지정 문자열의 첫 글자만 사용. "d", "day", "ddd" 모두 Day로서 사용
"s" : 초, "m" : 분, "h" : 시, "d" : 일

Rates 제한 메커니즘

SingleRateThrottle에서는 요청한 시간의 timestamp를 list로 유지

매 요청시마다

1. cache에서 timestamp list를 가져옵니다.
2. 체크 범위 밖의 timestamp 값들은 모두 버립니다.
3. timestamp list의 크기가 허용범위보다 클 경우, 요청을 거부합니다.
4. timestamp list의 크기가 허용범위보다 작을 경우, 현재 timestamp를 list에 추가하고, cache에 다시 저장합니다.

클라이언트 IP

X-Forwarded-For 헤더와 REMOTE_ADDR 헤더를 참조해서, 확정

우선순위 : X-Forwarded-For > REMOTE_ADDR

API 별로 서로 다른 Rate 적용하기 (1/2)

```python
# 프로젝트/settings.py
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_CLASSES': [],
    'DEFAULT_THROTTLE_RATES': {
        'contact': '1000/day',
        'upload': '20/day',
    },
}

# myapp/throttles.py
class CotactRateThrottle(UserRateThrottle):
    scope = 'contact'

class UploadRateThrottle(UserRateThrottle):
    scope = 'upload'
```

```python
# myapp/views.py
class ContactListView(APIView):
    throttle_classes = [CotactRateThrottle]

class ContactDetailView(APIView):
    throttle_classes = [ContactRateThrottle]

class UploadView(APIView):
    throttle_classes = [UploadRateThrottle]
```

API 별로 서로 다른 Rate 적용하기 (2/2)

직전 코드를 ScopedRateThrottle를 통해 간결하게 변경

```
# 프로젝트/settings.py
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_CLASSES': [
        'rest_framework.throttling.ScopedRateThrottle',
    ],
    'DEFAULT_THROTTLE_RATES': {
        'contact': '1000/day',
        'upload': '20/day',
    },
}

# myapp/views.py
class ContactListView(APIView):
    throttle_scope = 'contact'

class ContactDetailView(APIView):
    throttle_scope = 'contact'

class UploadView(APIView):
    throttle_scope = 'upload'
```

유저 별로 Rate 다르게 적용하기

```
REST_FRAMEWORK = {
    'DEFAULT_THROTTLE_RATES': {
        'premium_user': '1000/day',
        'light_user': '10/day',
    },
}
```

```
from rest_framework import viewsets
from .serializers import PostSerializer
from .throttling import PremiumThrottle
from .models import Post

class PostViewSet(viewsets.ModelViewSet):
    queryset = Post.objects.all()
    serializer_class = PostSerializer

    throttle_classes = [PremiumThrottle]
    premium_scope = 'premium_user'
    light_scope = 'light_user'

    def perform_create(self, serializer):
        serializer.save(author=self.request.user)
```

```python
from rest_framework.throttling import UserRateThrottle

class PremiumLightThrottle(UserRateThrottle):
    def __init__(self):
        pass

    def allow_request(self, request, view):
        premium_scope = getattr(view, 'premium_scope', None)
        light_scope = getattr(view, 'light_scope', None)

        if request.user.profile.is_premium_user:
            if not premium_scope:
                return True
            self.scope = premium_scope
        else:
            if not light_scope:
                return True
            self.scope = light_scope

        self.rate = self.get_rate()
        self.num_requests, self.duration = self.parse_rate(self.rate)

        return super().allow_request(request, view)
```

User에 따라 scope이 달라지기에, 생성자에서는 get_rate()를 수행하지 않도록 했습니다.

premium_scope 설정이 없다면, 제한을 두지 않습니다.

light_scope 설정이 없다면, 제한을 두지 않습니다.

인생은 짧습니다.
파이썬/장고를 쓰세요.

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

-

Ask Company