接口频次限制/流量过滤

需求:

vod端有大量的饱和式请求，对服务器带来巨大的压力。现要求过滤掉其中一部分，保证原有业务稳定。

实践:

接口频率的限制可以从nginx, 业务层来进行限制。

为了业务的稳定运行，这次采用的是redis-lua来进行实现。

实现思路:

1.把预先写好的redis-lua脚本缓存到redis里，得到sha1。

ratelimit.lua

```
--- evalsha shaname keynum [k1...kn] [arg1...argn]
local key = KEYS[1]
local limit = tonumber(ARGV[1])
local expire_time = ARGV[2]

local current = tonumber(redis.call('get', key) or "0")
if current > 0 then
 if current + 1 > limit then
 return 0
 else
 redis.call("INCR", key)
```

```
 return 1
 end
else
 redis.call("SET", key, 1)
 redis.call("EXPIRE", key, expire_time)
 return 1
end
```

```
redis-cli script load "$(cat ratelimit.lua)"
"1012ff91db24c879d926f3a38cb21a3fd9062e55"
```

2.把装饰器放置在函数头部，按要求填参。根据业务场景来设定参数

在这里限制每2秒至多响应2个请求，超出阀值的会被抛掉。可以根据实际场景调整参数。

每过2秒key被销毁，表示又可以开始接收请求。

decorator：

```
# 异步限流策略 2个/2秒
# key: 部门:项目:路由:MAC_ID 例: wow:wow_user:Test:ratelimit:00E07E00A258
# 装饰器这里的参数都是根据Lua定义的参数来配置的
# sha1: 就是redis-cli script load "$(cat ratelimit.lua)"的返回值
# 1: 表示key的个数
# limit: 限制次数
# expire: key过期时间

def async_ratelimit(key_str, limit=2, expire=10,
sha1="84dac7d57ddfbf015c31655b5fe28269a88b2f60"):
 def decorator(func):
 async def wrap(*args, **kw):
 self = args[0]
 mac_id = self.get_argument('mac_id')
 assert mac_id
 key = key_str % (self.__class__.__name__, func.__name__,
self.get_argument('mac_id'))
 raw = control.ctrl.rs.evalsha(sha1, 1, key, limit, expire)
 if raw == 0:
```

```
18  return self.send_json(errcode=10001, errmsg='访问过于频繁')
19  else:
20  await func(*args, **kw)
21  return wrap
22  return decorator
```

调用示例：

```
1 @async_ratelimit('wow:wow_user:%s:%s:%s')
2 async def ratelimit(self, method):
3  try:
4  mac_id = self.get_argument('mac_id')
5  except Exception as e:
6  logging.error(e)
7  return self.send_json(errcode=10001)
8  return self.send_json()
```

参考网文：

- https://segmentfault.com/a/1190000016552464 lua scrip
- http://www.cnblogs.com/zhenbianshu/p/8416162.html lua scrip
- https://blog.csdn.net/seesun2012/article/details/80764497 秒

流量过滤策略

需求：

客户端请求服务器时接口已经是最新的内容了，返回相同内容无意义。还会带来流量资费的上涨，解决这块带来的问题。

实践：

平台介绍：tornado

思路：

tornado发包是由send_json封装的。在响应请求的时，取字段data摘要。这里有个插曲（python的字典是无序的，json.dumps()的时候可能每次都不一样，使用json.dumps(data, srot_keys=True)）先对数据排序）

将摘要(x-response-md5)返回给客户端。下次发包时，客户端会上传md5值，在返回时对新旧md5进行校验。

md5一致代表数据未曾变更，随返回'数据未变化'。

```
# handler
class Test(AsyncHandler):
async def md5_filter(self, method):
 try:
 md5 = self.get_argument('md5', '')
 logging.info(md5)
 except Exception as e:
 logging.error(e)
 return self.send_json(errcode=10001)
 return self.send_json(dict(data={'apple': 'iphone XR', 'huawei': 'meta 20', 'sony': 'x10', 'xiaomi': 'x10'}))


# base.py
def send_json(self, data={}, errcode=200, errmsg='', status_code=200):
 # 流量监测
 json_str = json.dumps(res, default=self.json_format, sort_keys=True)
 compare_md5 = hashlib.md5(json_str.encode(encoding='utf-8')).hexdigest()
 md5 = self.get_argument('md5', '')
 if md5 and compare_md5 == md5:
 json_str = '{"errcode": 200, "errmsg": "服务正常","md5": "%s"}' % compare_md5
 else:
 json_str = json_str[:-1] + ', "md5": "%s"}' % compare_md5
 ...
```