数据库性能

redis 1ms 10000

mysql 0.12s

缓存雪崩

1. 加锁方式 setnx -> set if not exist

```
def get_cache_data():
    # 尝试获取锁
    lock_flag = False
    while not lock_flag:
        lock_flag =
redis.setnx('article:1:info:lock')
        if lock_flag:
            ret = redis.get('article:1:info')
            if not ret:
                ret = db.query.all()
                redis.set(ret)
            redis.delete('article:1:info:lock')
    return ret

```

2. 使用队列方式

```
queue = []

def get_cache_data():
    ret = redis.get('article:1:info')
    if not ret:
        ret = db.query.all()
        redis.set(ret)
```

```
    return ret

queue.append(get_cache_data, get_cache_data, get_cache_data)

# 队列的消费进程
func = queue.pop()
func()
```

缓存编写

以用户资料缓存数据为例

```
class UserCache(object):

    redis_key = ''user:1:profile'

    def get_user_cache_data()  查询
        redis.get(redis_key)

    def clear_user_cache_data()  删除
        redis.delete(redis_key)

    def exists_user_id() 判断是否存在
        redis.get(redis_key)
```

```
POST  /followings/<user_id>  user_id=1
```

类的构造

1. 确定是否要保存数据，即数据选择类属性还是对象属性
2. 选择方法

- 如果要构造的方法中需要处理对象属性和类属性，选择对象方法
- 如果仅需要处理类属性，选择类方法
- 如果在方法中不处理类属性与对象数据，仅从逻辑角度考虑应该封装到一起，则选择静态方法

用户 我的页面 个人信息接口

```
GET /user

response
{
    "message": "OK",
    "data": {
        "user_id": 123,
        "username":
        "photo":
        "art_count":
        "following_count":
        "fans_count":
    }
}
```