翻页 Pagination 与 缓存 Cache

授课人 令狐东邪

版权声明

九章的所有课程均受法律保护，不允许录像与传播录像
一经发现，将被追究法律责任和赔偿经济损失

课程大纲

九章算法
www.jiuzhang.com

今天需要做什么

01

传统翻页
Page Number
Pagination

02

无限翻页
Endless
Pagination

03

Memcached
的安装与配置

04

Memcached
如何优化
DB 查询

00 前情回顾

前情回顾

在上周的课程中，我们实现了**两个核心功能**：

- 通知模块 Notification
- 图像上传

前情回顾

接下来我们要学习**新的内容**:

- 传统的翻页方法是什么，如何实现?
  - DRF 的 pagination 配置
- 如何实现瀑布流的翻页?
  - Endless Pagination

- 通过缓存优化好友关系与用户信息

01 传统翻页 Page Number Pagination

传统翻页 Page Number Pagination

传统翻页

传统翻页方法：/api/items/?page=1

传统翻页 Page Number Pagination

传统翻页实现

```python
from rest_framework.pagination import PageNumberPagination
from rest_framework.response import Response


class FriendshipPagination(PageNumberPagination):
    page_size = 20
    page_size_query_param = 'size'
    max_page_size = 20

    def get_paginated_response(self, data):
        return Response({
            'total_results': self.page.paginator.count,
            'total_pages': self.page.paginator.num_pages,
            'page_number': self.page.number,
            'has_next_page': self.page.has_next(),
            'results': data,
        })
```

friendships/api/paginations.py

```python
REST_FRAMEWORK = {
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 10,
    'DEFAULT_FILTER_BACKENDS': [
        'django_filters.rest_framework.DjangoFilterBackend',
    ],
}
```

twitter/settings.py

参数说明

page_size： 默认的 page size，也就是 page 没有在 url 参数里的时候。

page_size_query_param： 默认的 page_size_query_param 是 None 表示不允许客户端指定每一页的大小。如果加上这个配置，就表示客户端可以通过 size=10 来指定一个特定的大小用于不同的场景。比如手机端和web端访问同一个API但是需要的 size 大小是不同的。

max_page_size： 允许客户端指定的最大 page_size 是多少

假设每页显示50个问题，如何写获取问题列表中第二页问题的 SQL 语句?

```
SELECT * FROM problem_table LIMIT 50 OFFSET 50
```

传统翻页 Page Number Pagination

传统翻页的优劣

优点： 可以直接跳转到第 x 页

缺点： 如果有新数据被插入时，翻到下一页可能会看到上一页的内容

适用于用户希望查阅很久以前的内容的使用场景

用户获得第一页数据，得到前 3 个 Item

-----page1-----
item0
item1
item2
-----page2-----
item3
item4
item5

用户点击下一页之前新数据被加入：

-----page1-----
new-item
item0
item1
-----page2-----
item2
item3
item4

用户此时再请求
第二页的数据
会看到重复 item

02

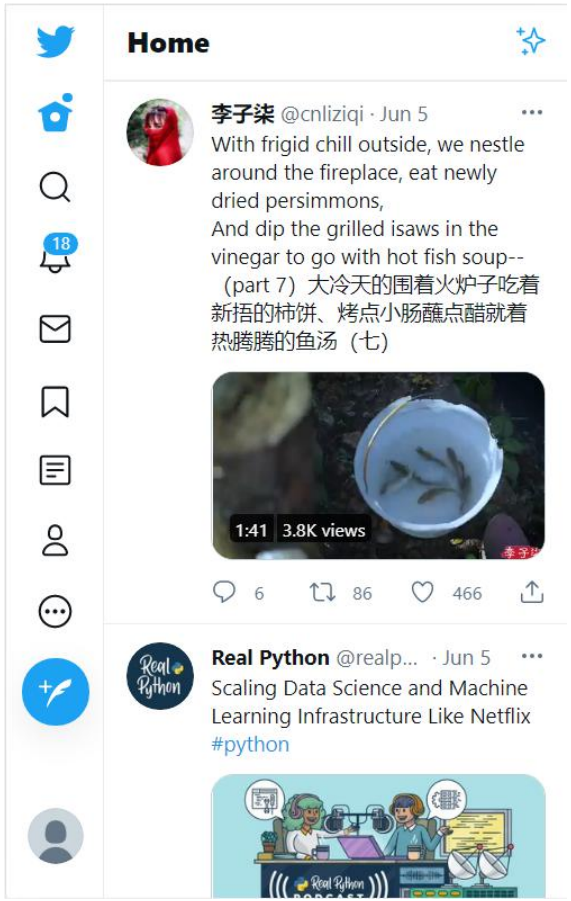
无限翻页
Endless Pagination

无限翻页 Endless Pagination

无限翻页

无限翻页方法：/api/items/?id__lt=xxx

id__lt 代表 id 小于（lt=less than）



无限翻页 Endless Pagination

无限翻页实现

```python
from rest_framework.pagination import BasePagination
from rest_framework.response import Response


class EndlessPagination(BasePagination):
    page_size = 20

    def __init__(self):
        super(EndlessPagination, self).__init__()
        self.has_next_page = False

    def to_html(self):
        pass
```

```python
    def paginate_queryset(self, queryset, request, view=None):
        if 'created_at__gt' in request.query_params:
            created_at__gt = request.query_params['created_at__gt']
            queryset = queryset.filter(id__gt=created_at__gt)
            self.has_next_page = False
            return queryset.order_by('-id')

        if 'created_at__lt' in request.query_params:
            created_at__lt = request.query_params['created_at__lt']
            queryset = queryset.filter(id__lt=created_at__lt)

        queryset = queryset.order_by('-id')[:self.page_size + 1]
        self.has_next_page = len(queryset) > self.page_size
        return queryset[:self.page_size]

    def get_paginated_response(self, data):
        return Response({
            'has_next_page': self.has_next_page,
            'results': data,
        })
```

utils/paginations.py

假设每次上滑显示3个推文，如何写获取下一页推文的 SQL 语句?

```
SELECT * FROM tweets
WHERE user_id=123 AND `id` < 28
LIMIT 3
```

无限翻页 Endless Pagination

News Feed 一般不会采用页码进行翻页
而是采用无限翻页 Endless Pagination

API 设计：
/api/newsfeed/?id__lt=1000
没有 id__lt 表示第一页
有 id__lt 找到所有 id < id__lt 的最顶上的一页的数据（假设倒序）

如何判定有没有下一页?

当 response 里什么数据都没有时表示没有下一页

问：这种方法有什么问题?

无限翻页 Endless Pagination

正确方法：

每次多取一个数据，如果取到，has_next_page=true 放到 response 里传给前端

```
PAGE_SIZE = 20
从 request 中获取 id__lt
if 没有 id__lt
 获得最新的 PAGE_SIZE + 1 条数据
else
 获得 id < id__lt 的最新的 PAGE_SIZE + 1 条数据
如果获取到的数据为 PAGE_SIZE + 1 条，标记 has_next_page = true
否则 has_next_page = false
返回数据：
{
 'has_next_page': 是否能取到第 PAGE_SIZE + 1 条数据
 'items': [...最多 PAGE_SIZE 条数据]
}
```

如何获得更新内容?

GET /api/newsfeed/?id__gt=<id>

找到所有 id > 目前客户端里最新的一条帖子的 id 的所有帖子

03 Memcached 的安装与配置

安装 Memcache

安装

```
sudo apt-get install memcached
```

启动

```
/usr/bin/memcached -u memcache -m 1024 -p 11222 -l 0.0.0.0 -d start
```

- -d：这个参数是让memcached在后台运行
- -m：指定占用多少内存。以M为单位，默认为64M。
- -p：指定占用的端口。默认端口是11211。
- -l：别的机器可以通过哪个ip地址连接到我这台服务器,如果想要让别的机器连接，就必须设置-l 0.0.0.0

Memcached 的安装与配置

配置 Memcache

安装依赖

```
pip install python-memcached
```

在 settings.py 中添加缓存配置

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '192.168.33.10:11211',
        'TIMEOUT': 86400,
    },
    'testing': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '192.168.33.10:11211',
        'TIMEOUT': 86400,
        'PREFIX': 'testing',
    },
}
```

04 Memcached 如何优化 DB 查询

Memcached 如何优化 DB 查询

项目代码

```python
class FriendshipService(object):

    @classmethod
    def get_followers(cls, user):...

    @classmethod
    def get_following_user_id_set(cls, from_user_id):
        key = FOLLOWINGS_PATTERN.format(user_id=from_user_id)
        user_id_set = cache.get(key)
        if user_id_set is not None:
            return user_id_set

        friendships = Friendship.objects.filter(from_user_id=from_user_id)
        user_id_set = set([
            fs.to_user_id
            for fs in friendships
        ])
        cache.set(key, user_id_set)
        return user_id_set
```

friendships\services.py

当用户的 following_user_id_set 增加时，我们改怎么更新缓存?

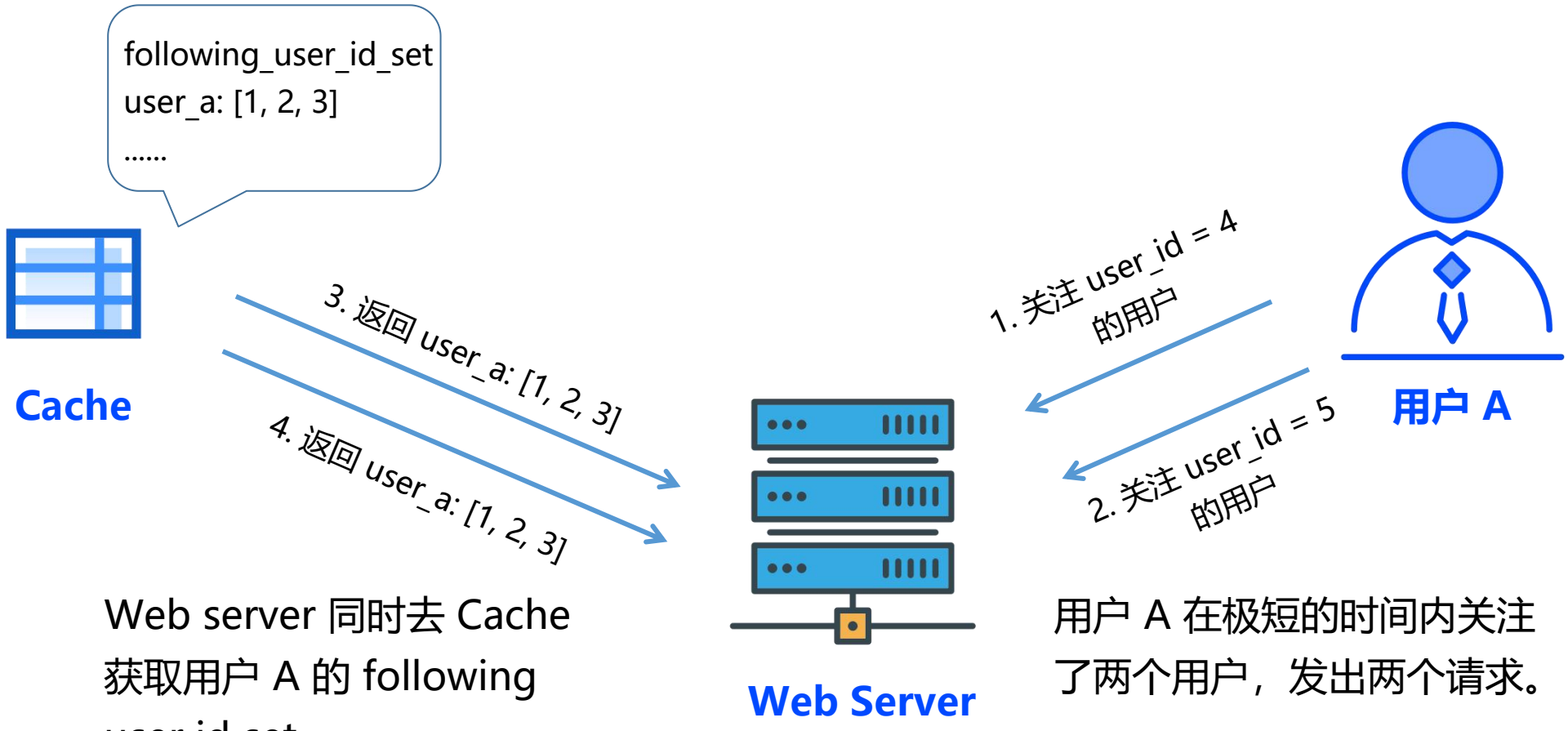
A. cache.set

B. cache.delete 

```python
@classmethod
def invalidate_following_cache(cls, from_user_id):
    key = FOLLOWINGS_PATTERN.format(user_id=from_user_id)
    cache.delete(key)
```

friendships\services.py

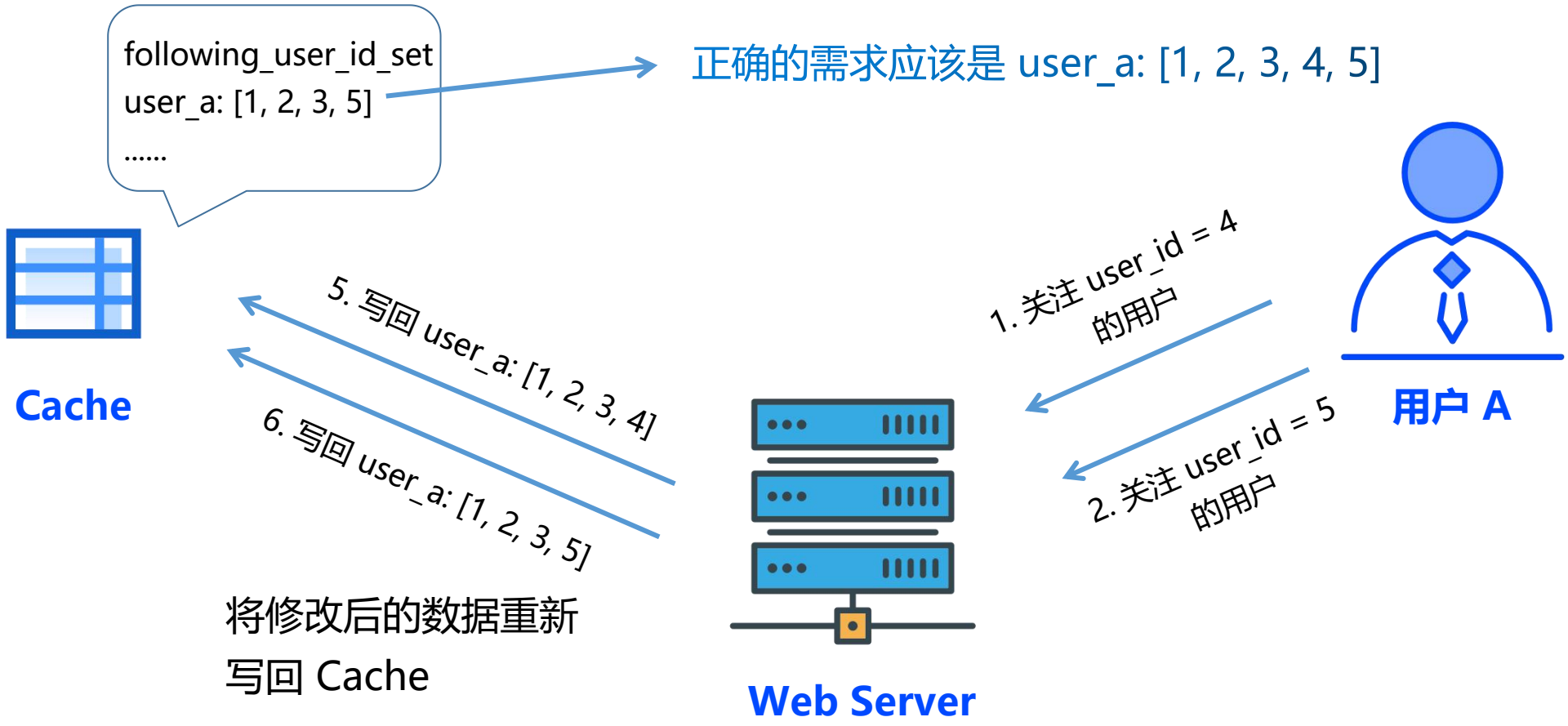
Memcached 如何优化 DB 查询



Web server 同时去 Cache 获取用户 A 的 following user id set

用户 A 在极短的时间内关注了两个用户，发出两个请求。

Memcached 如何优化 DB 查询



Memcached 如何优化 DB 查询


```
40      @classmethod
41      def get_following_user_id_set(cls, from_user_id):
42          key = FOLLOWINGS_PATTERN.format(user_id=from_user_id)
43          user_id_set = cache.get(key)
44          if user_id_set is not None:
45              return user_id_set
46
47          friendships = Friendship.objects.filter(from_user_id=from_user_id)
48          user_id_set = set([
49              fs.to_user_id
50              for fs in friendships
51          ])
52          cache.set(key, user_id_set)
53          return user_id_set
```

当我们做了上面的优化之后，还是会出现数据不一致的情况。假设有两个进程，进程1执行代码到第51行，此时进程1中的变量 user_id_set = [2, 3, 4]。进程2修改了数据库中的数据， 此时数据库中 user_id_set = [2, 3, 4, 5]，进程2执行完毕。接着进程1继续执行，52行代码设置缓存中的 user_id_set = [2, 3, 4]，此时数据库和缓存中的数据出现不一致。