缓存的架构

缓存到底保存到哪里？本地？服务器？服务器几级几台？

黑马头条项目：

1. redis cluster 作为一级外部分布式缓存
2. queryset 查询集(查询结果集) 其中一个作用就是缓存 (起到了本地缓存的作用)

缓存数据

思考 在缓存中保存什么数据？ 数据以什么格式保存？

1. 数据内容

关于Caching at the database query level的说明

缓存的数据是数据库查询的结果

第一次

```
sql = select a.user_id, a.user_name, b.gender, b.birthday from tbl_user as a inner join tbl_profile as b on a.user_id=b.user_id where a.user_id=1;
```

得到数据库的查询结果 result_data

设置缓存 md5(sql) -> 计算结果 'fdh9ihf92dfhowidfhwoho'

'fdh9ihf92dfhowidfhwoho': result_data

以后使用缓存

生成要执行的sql ，

md5(sql) -> 'fdh9ihf92dfhowidfhwoho'

从缓存中尝试读取 'fdh9ihf92dfhowidfhwoho' 的缓存记录，如果有，直接使用，如果没有，再查询数据库

头条项目：

Caching at the object level

不以视图作为缓存数据的思考，以数据库中哪些数据被频繁使用访问，以这一点作为思考点，所以选择使用缓存数据库中数据记录的级别来缓存。

2. 数据保存的形式

- 字符串 (序列化成字符串， json)
  - 对于早已的memcached服务器，只有字符串类型可以选择
- redis中的其他符合类型(hash、zset list)

缓存数据的有效期和淘汰策略

1. 有效期 两点作用
2. 对于缓存数据 一定要设置有效期

通用过期策略

- 定时过期
  - 每个记录单独追踪有效期
- 惰性过期
  - 只在使用数据的时候 判断数据是否过期
- 定期过期
  - 每隔100ms 检查一下有哪些数据过期了

Redis选用的过期策略

- 惰性过期+定期过期
- 对于定期过期，在每100ms时，随机检测一部分数据是否过期

淘汰策略

计算机内存中 (内存淘汰策略)

redis 内存淘汰策略

两种常用算法

- LRU 以操作过的时间选择

```
[                  最近使用的
    {user_4}
    {user_3}
    {user_2}
    {user_1}  X
]                 最早已使用过的


添加 {user_5} ?

[                  最近使用的
    {user_5}
    {user_4}
    {user_3}
    {user_2}
]
操作过user_2的数据后
[                  最近使用的
    {user_2}
    {user_5}
    {user_4}
    {user_3}
]
```

- LFU Least Frequently Used 以次数 频率来选择

```
[
    ({user_4}, 3000)
    ({user_1}, 3)
    ({user_3}, 2680)
    ({user_2}, 50)
]

添加 {user_5} ?

[                  选择使用累计次数最少的淘汰
    ({user_4}, 3000)
    ({user_5}, 1)
    ({user_3}, 2680)
    ({user_2}, 50)
]
操作过user_2的数据后
[
    ({user_4}, 3000)
    ({user_5}, 1)
    ({user_3}, 2680)
    ({user_2}, 51)
]

user_6
```

LFU 效果会好一点，但是有隐含问题

定期衰减 每过一段时间，所有记录的此时减半

头条项目方案

- 缓存数据都设置有效期
- 配置redis，使用volatile-lru -> redis cluster 配置的

缓存模式

- 读缓存
  - cache aside
  - 通读 (多出了一个缓存工具层)
- 写缓存

更新方式

先更新数据库，再删除缓存

user_1 2h + 1

user_2 2h +3

user_3 2h + 6

头条项目缓存数据的设计

1. 用户的基本信息数据
   - 多个用户的数据库记录是保存在redis中的一条还是多条?
   - 字符串 or 复合型

     user_1 user_2 user_3
     - 都保存到redis中一条 X

```
users -> hash {
    1: user_1_cache_data,
    2: user_2_cache_data
}

users -> list [
```

```
    user_1_cache_data, user_2_cache_data
]

users -> zset {
    值                      分数 user_id
    user_1_cache_data       1
    user_2_cache_data       2
}
```

user_id =1

- 从数据的存储角度考虑 可以使用hash 方便
- 从有效期的角度考虑，如果放到一个redis记录中，只能有一个有效期，不是我们的需求，
- 综合来说，不使用多条缓存放到一个redis记录中

- 每个用户一条缓存记录

  user1 user2 user3

```
redis键                 值
user:{user_id} -> hash
user:1  ->  {
    name: python
    photo: xx
    mobile: xxx
}

user:{user_id}  ->  string
user:1 -> json.dumps({})  pickle
```

2. 用户关注列表的缓存

   user1 -> user2 user3 user4

```
redis key                          值
user:{user_id}:following    list
[user_3_id,user_2_id']
                                   zset {
                                       value       score   时间戳 1557986157.1353633
                                       user_3_id
update_timestamp 关注的时间
                                       user_2_id
update_timestamp
                                   }
```

不是字符串的原因，是要考虑到应用程序可能分页获取

redis 性能1s 内可以执行 10000+ 读操作

优先选择zset，冲分利用score分数的价值，可以排序+过滤

3. 持久保存中的 统计数据

- 发布数量
- 关注数量
- 粉丝数量
- 点赞数量

```
redis key                  值
user:{user_id}:count       hash -> {
                             'article_count': 12w
                             'folloing_count': 51
                             "fans_count": 629w
                           }
```

MIS后台管理系统：

文章数量有多到少的用户 显示出来 top10

关注数量由多到少的用户显示 top100

```
redis key                          redis值
count:user:articles    zset -> [
                               值            score
                               user_id_1      12w
                               user_id_2     11w
                          ]

count:user:following    zset -> [
                               值            score
                               user_id_1      12w
                               user_id_2     11w
                          ]
```