2.2 缓存、排行榜、积分接口

开发任务

1. 为获取个人资料接口添加缓存处理
2. 统一为所有数据模型增加缓存处理
    - 任何 model 对象创建时，自动为该对象添加缓存
    - 任何 model 对象创建时，自动更新缓存数据
3. 开发全服人气排行功能
    - 被左滑 -5 分
    - 被右滑 +5 分
    - 被上滑 +7 分
    - 统计全服人气最高的 10 位用户

缓存处理

1.
2. 缓存一般处理流程

```
data = get_from_cache(key)    # 首先从缓存中获取数据
if data is None:
    data = get_from_db()      # 缓存中没有，从数据库获取
    if data is None:
        set_to_cache(key, data)# 将数据添加到缓存，方便下次获取
    else:
        raise Error.....
return data
```

3. Django 的默认缓存接口

```
from django.core.cache import cache

cache.set('a', 123, 10)
a = cache.get('a')
print(a)
x = cache.incr(a)
print(a)
```

4. Django 中使用 Redis 缓存
    - 安装 django_redis: `pip install django_redis`
    - 配置

```
# settings 添加如下配置
CACHES = {
    "default": {
        "BACKEND": "django_redis.cache.RedisCache",
        "LOCATION": "redis://127.0.0.1:6379/1",
        "OPTIONS": {
            "CLIENT_CLASS":
"django_redis.client.DefaultClient",
            "PICKLE_VERSION": -1,
        }
    }
}
```

REDIS 的使用

1. Redis 文档
2. 主要数据类型
    1. NoSQL
    2. KeyValue类的：Memcacche(d), CouchBase, TokyoCabinet
        2. get / set
        3. 约等于Redis的String类型
    - **String 类**: 常用作普通缓存

CMD	Example	Description
set	set('a', 123)	设置值
get	get('a')	获取值
incr	incr('a')	自增
decr	decr('a')	自减
mset	mset(a=123, b=456, c=789)	设置多个值
mget	mget(('a', 'b', 'c'))	获取多个值
setex	setex('kk', 21, 10)	设置值的时候, 同时设置过期时间
setnx	setnx('a', 999)	如果不存在, 则设置该值

    - **Hash 类**: 常用作对象存储

CMD	Example	Description
hset	hset('obj', 'name', 'hello')	在哈希表 obj 中添加一个 name = hello 的值
hget	hget('obj', 'name')	获取哈希表 obj 中的值
hmset	hmset('obj', {'a': 1, 'b': 3})	在哈希表中设置多个值
hmget	hmget('obj', ('a', 'b', 'name'))	获取多个哈希表中的值
hgetall	hgetall('obj')	获取多个哈希表中所有的值
hincrby	hincrby('obj', 'count')	将哈希表中的某个值自增 1
hdecrby	hdecrby('obj', 'count')	将哈希表中的某个值自减 1

- **List 类**: 常用作队列(消息队列、任务队列等)

CMD	Example	Description
lpush	lpush(name, *values)	向列表左侧添加多个元素
rpush	rpush(name, *values)	向列表右侧添加多个元素
lpop	lpop(name)	从列表左侧弹出一个元素
rpop	rpop(name)	从列表右侧弹出一个元素
blpop	blpop(keys, timeout=0)	从列表左侧弹出一个元素, 列表为空时阻塞 timeout 秒
brpop	brpop(keys, timeout=0)	从列表右侧弹出一个元素, 列表为空时阻塞 timeout 秒
llen	llen(name)	获取列表长度
ltrim	ltrim(name, start, end)	从 start 到 end 位置截断列表

- **Set 类**: 常用作去重

CMD	Example	Description
sadd	sadd(name, *values)	向集合中添加元素
sdiff	sdiff(keys, *args)	多个集合做差集
sinter	sinter(keys, *args)	多个集合取交集
sunion	sunion(keys, *args)	多个集合取并集
sismember	sismember(name, value)	元素 value 是否是集合 name 中的成员

smembers	smembers(name)	集合 name 中的全部成员
spop	spop(name)	随机弹出一个成员
srem	srem(name, *values)	删除一个或多个成员

- **SortedSet 类**: 常用作排行处理

CMD	Example	Description
zadd	zadd(name, a=12)	添加一个 a, 值为 12
zcount	zcount(name, min, max)	从 min 到 max 的元素个数
zincrby	zincrby(name, key, 1)	key 对应的值自增 1
zrange	zrange(name, 0, -1, withscores=False)	按升序返回排名 0 到 最后一位的全部元素
zrevrange	zrevrange(name, 0, -1, withscores=False)	按降序返回排名 0 到 最后一位的全部元素
zrem	zrem(name, *value)	删除一个或多个元素

- 面试题：
  - zset有什么特性？怎么用?

序列化与反序列化

- 序列化：将对象从内存的结构化格式转变为一种可以存储和易于网络传输的序列格式
- 反序列化：将序列化的对象还原
- 跨语言访问
  - 开源的：
    1. msgpack
  - google：
    1. protobuf
       1. 定义接口：数据结构
       2. 生成对应语言的访问类：server端和client端
       3. client端用python，server端用java
       4. 中间都是透明的
  - facebook：thrift
- 读写分离
  - 读和写在不同的服务器上
  - 强依赖于服务器间的数据同步
  - 不同的服务，同步时间是不同的，差的可以到秒级

PYTHON 访问 REDIS

1. 使用 pickle 对 Redis 接口的封装

```
from pickle import dumps, loads

rds = redis.Redis()

def set(key, value):
    # 序列化
    data = dumps(value)
    return rds.set(key, data)

def get(key):
    # 反序列化
    data = rds.get(key)
    return loads(data)
```

2. 动态修改 Python 属性和方法

```
class A:
    m = 128
    def __init__(self):
        self.x = 123

    def add(self, n):
        print(self.x + n)

a = A()

# 动态添加属性 (两种方式)
a.y = 456
setattr(a, 'z', 789)

# 动态添加类属性
A.y = 654

# 类属性和实例属性互不影响
print(A.y, a.y)

# 动态添加实例方法
def sub(self, n):
    print(self.x - n)
A.sub = sub

# 动态添加类方法
@classmethod
def mul(cls, n):
    print(cls.m * n)
A.mul = mul
```

```
# 动态添加静态方法
@staticmethod
def div(x, y):
    print(x / y)
A.div = div

# 属性修改的本质原因
print(A.__dict__, a.__dict__)
```

3. 在 Model 层插入缓存处理
    - Monkey Patch 也叫做“猴子补丁”，是一种编程技巧，旨在运行时为对象动态添加、修改或者替换某项功能

MYSQL/INNODB事务和锁

MySQL/InnoDB的加锁，一直是一个面试中常问的话题。

注：MySQL是一个支持插件式存储引擎的数据库系统。本文基于InnoDB存储引擎。

存储引擎查看

MySQL给开发者提供了查询存储引擎的功能，我这里使用的是MySQL5.6.4，可以使用：

```
SHOW ENGINES
```

乐观锁

用数据版本（Version）记录机制实现，这是乐观锁最常用的一种实现方式。何谓数据版本？即为数据增加一个版本标识，一般是通过为数据库表增加一个数字类型的“version”字段来实现。当读取数据时，将version字段的值一同读出，数据每更新一次，对此version值加1。当我们提交更新的时候，判断数据库表对应记录的当前版本信息与第一次取出来的version值进行比对，如果数据库表当前版本号与第一次取出来的version值相等，则予以更新，否则认为是过期数据。

举例

1、数据库表设计

三个字段，分别是`id,value、version`

```
select id,value,version from TABLE where id=#{id}
update TABLE
set value=2,version=version+1
where id=#{id} and version=#{version};
set autocommit=0;

# 设置完autocommit后，我们就可以执行我们的正常业务了。具体如下：

# 1. 开始事务

begin;/begin work;/start transaction; (三者选一就可以)

# 2. 查询表信息

select status from TABLE where id=1 for update;

# 3. 插入一条数据

insert into TABLE (id,value) values (2,2);

# 4. 修改数据为

update TABLE set value=2 where id=1;

# 5. 提交事务

commit;/commit work;
begin;/begin work;/start transaction;  (三者选一就可以)

SELECT * from TABLE where id = 1  lock in share mode;
update  TABLE set name="www.souyunku.com" where id =1;
[SQL]update  test_one set name="www.souyunku.com" where id =1;
[Err] 1205 - Lock wait timeout exceeded; try restarting transaction
update  test_one set name="www.souyunku.com" where id =1 lock in share
mode;
[SQL]update  test_one set name="www.souyunku.com" where id =1 lock in
share mode;
[Err] 1064 - You have an error in your SQL syntax; check the manual
that corresponds to your MySQL server version for the right syntax to
use near 'lock in share mode' at line 1
SELECT * from TABLE where id = "1"  lock in share mode;  结果集的数据都会
加共享锁
select status from TABLE where id=1 for update;
show OPEN TABLES where In_use > 0;
show processlist
kill id
SELECT * FROM INFORMATION_SCHEMA.INNODB_TRX;
```

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_LOCKS;
SELECT * FROM INFORMATION_SCHEMA.INNODB_LOCK_WAITS;
kill 进程ID
```

（1）按同一顺序访问对象。

（2）避免事务中的用户交互。

（3）保持事务简短并在一个批处理中。

（4）使用低隔离级别。

（5）使用绑定连接。

死锁

下列方法有助于最大限度地降低死锁：

虽然不能完全避免死锁，但可以使死锁的数量减至最少。将死锁减至最少可以增加事务的吞吐量并减少系统开销，因为只有很少的事务回滚，而回滚会取消事务执行的所有工作。由于死锁时回滚而由应用程序重新提交。

（1） 互斥条件：一个资源每次只能被一个进程使用。

（2） 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。

（3） 不剥夺条件:进程已获得的资源，在未使用完之前，不能强行剥夺。

（4） 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。

如果系统资源充足，进程的资源请求都能够得到满足，死锁出现的可能性就很低，否则就会因争夺有限的资源而陷入死锁。其次，进程运行推进顺序与速度不同，也可能产生死锁。 产生死锁的四个必要条件：

杀死进程

3：查看当前等锁的事务

2：查看当前锁定的事务

1：查看当前的事务

第二种：

3.杀死进程id（就是上面命令的id列）

2.查询进程（如果您有SUPER权限，您可以看到所有线程。否则，您只能看到您自己的线程）

1.查询是否锁表

第一种：

解除正在死锁的状态有两种方法：

死锁（Deadlock） 所谓死锁：是指两个或两个以上的进程在执行过程中，因争夺资源而造成的一种互相等待的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。由于资源占用是互斥的，当某个进程提出申请资源后，使得有关进程在无外力协助下，永远分配不到必需的资源而无法继续运行，这就产生了一种特殊现象死锁。

行锁

行级锁都是基于索引的，如果一条SQL语句用不到索引是不会使用行级锁的，会使用表级锁。

行级锁的缺点是：由于需要请求大量的锁资源，所以速度慢，内存消耗大。

在实际应用中，要特别注意InnoDB行锁的这一特性，不然的话，可能导致大量的锁冲突，从而影响并发性能。

前面提到过，在Innodb引擎中既支持行锁也支持表锁，那么什么时候会锁住整张表，什么时候或只锁住一行呢？ 只有通过索引条件检索数据，InnoDB才使用行级锁，否则，InnoDB将使用表锁！

Innodb中的行锁与表锁

innodb 的行锁是在有索引的情况下,没有索引的表是锁定全表的.

如何加表锁

表锁

由于对于表中,id字段为主键，就也相当于索引。执行加锁时，会将id这个索引为1的记录加上锁，那么这个锁就是行锁。

可以参考之前演示的共享锁,排它锁语句

名词解释：若某个事物对某一行加上了排他锁，只能这个事务对其进行读写，在此事务结束之前，其他事务不能对其进行加任何锁，其他进程可以读取,不能进行写操作，需等待其释放。

排他锁：

名词解释：共享锁又叫做读锁，所有的事务只能对其进行读操作不能写操作，加上共享锁后在事务结束之前其他事务只能再加共享锁，除此之外其他任何类型的锁都不能再加了。

共享锁：

注意：行级锁都是基于索引的，如果一条SQL语句用不到索引是不会使用行级锁的，会使用表级锁。

行锁又分共享锁和排他锁,由字面意思理解，就是给某一行加上锁，也就是一条记录加上锁。

行锁

使用方式：在需要执行的语句后面加上for update就可以了

读取为什么要加读锁呢：防止数据在被读取的时候被别的线程加上写锁，

若事务 1 对数据对象A加上X锁，事务 1 可以读A也可以修改A，其他事务不能再对A加任何锁，直到事物 1 释放A上的锁。这保证了其他事务在事物 1 释放A上的锁之前不能再读取和修改A。排它锁会阻塞所有的排它锁和共享锁

排它锁是悲观锁的一种实现，在上面悲观锁也介绍过。

排他锁 exclusive lock（也叫writer lock）又称写锁。

排它锁

加上共享锁后，对于update,insert,delete语句会自动加排它锁。

在查询语句后面增加 **LOCK IN SHARE MODE，Mysql会对查询结果中的每行都加共享锁，当没有其他线程对查询结果集中的任何一行使用排他锁时，可以成功申请共享锁，否则会被阻塞。其他线程也可以读取使用了共享锁的表，而且这些线程读取的是同一个版本的数据。**

加上共享锁后，也提示错误信息

如果在超时前，执行 `commit`，此更新语句就会成功。

此时，操作界面进入了卡顿状态，过了超时间，提示错误信息

然后在另一个查询窗口中，对id为1的数据进行更新

打开第一个查询窗口

如果事务T对数据A加上共享锁后，则其他事务只能对A再加共享锁，不能加排他锁。获得共享锁的事务只能读数据，不能修改数据

共享锁又称**读锁 read lock**，是读取操作创建的锁。其他用户可以并发读取数据，但任何事务都不能对数据进行修改（获取数据上的排他锁），直到已释放所有共享锁。

共享锁

我们可以使用命令设置MySQL为非autocommit模式：

要使用悲观锁，我们必须关闭mysql数据库的自动提交属性，因为MySQL默认使用autocommit模式，也就是说，当你执行一个更新操作后，MySQL会立刻将结果进行提交。

使用，排它锁 举例

说到这里，由悲观锁涉及到的另外两个锁概念就出来了，它们就是共享锁与排它锁。**共享锁和排它锁是悲观锁的不同的实现**，它俩都属于悲观锁的范畴。

与乐观锁相对应的就是悲观锁了。悲观锁就是在操作数据时，认为此操作会出现数据冲突，所以在进行每次操作时都要通过获取锁才能进行对相同数据的操作，这点跟java中的synchronized很相似，所以悲观锁需要耗费较多的时间。另外与乐观锁相对应的，悲观锁是由数据库自己实现了的，要用的时候，我们直接调用数据库的相关语句就可以了。