大数据库系统

3.6 Redis的事务

3.4 Redis的事务及消息订阅功能

- 主要内容

  什么是事务

  MySQL事务

  Redis事务

  事务锁

  Redis中的事务锁

3.4.1 Redis事务功能

- 什么是事务
  - ✓ 本质是一组命令的集合
  - ✓ 一个事务中的所有命令都会序列化，按顺序地串行化执行而不会被其它命令插入，不许加塞

现实中有这样的需求吗?

3.4.1 Redis事务功能

- 事务的特性ACID

1、原子性

一个原子事务要么完整执行，要么干脆不执行

2、一致性

数据库要一直处于一致的状态，事务的运行不会改变数据库原本的一致性约束

3、独立性

独立性意味着事务必须在不干扰其他进程或事务的前提下独立执行

4、持久性

持久性表示在某个事务的执行过程中，对数据所作的所有改动都必须在事务成功结束前保存至某种物理存储设备。这样可以保证，所作的修改在任何系统瘫痪时不至于丢失。

3.4.1 Redis事务功能

◆ Mysql中的事务

Mysql是一种关系型数据库，并且支持事务

Mysql中的事务（使用SQL语句）：

- 开启事务：start transaction
- 失败：rollback 回滚
- 成功：commit执行

3.4.1 Redis事务功能

在Mysql中准备数据，account表存着各个用户账户余额，

- Uid表示用户编号
- Uname表示用户姓名
- Money表示用户余额

```
mysql> select * from account;
+------+-------+-------+
| uid  | uname | money |
+------+-------+-------+
|    1 | li    |   100 |
|    2 | wang  |   200 |
|    3 | zhao  |   700 |
+------+-------+-------+
3 rows in set (0.00 sec)

mysql>
```

3.4.1 Redis事务功能

- 如果我们想让用户zhao转账100给用户wang，在不启用事务的情况下，首先我们需要对用户zhao的余额减100

```
mysql> update account set money=money-100 where uid=3;
```

- 但是，在这条语句之后发生了断电等意外情况，这时会发生什么？

Zhao的余额被减100但wang却没收到钱，这是不允许的

因此我们用Mysql的事务功能完成这一系列转账操作

```
mysql> select * from account;
+------+-------+-------+
| uid  | uname | money |
+------+-------+-------+
|    1 | li    |   100 |
|    2 | wang  |   200 |
|    3 | zhao  |   700 |
+------+-------+-------+
3 rows in set (0.00 sec)

mysql>
```

1、启动Mysql事务，先输入start transction，接着输入语句，zhao余额减少100

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> update account set money=money-100 where uid=3;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> select * from account;
+------+-------+-------+
| uid  | uname | money |
+------+-------+-------+
|    1 | li    |   100 |
|    2 | wang  |   200 |
|    3 | zhao  |   600 |
+------+-------+-------+
3 rows in set (0.00 sec)

mysql> update account set money=money+100 where uid=2;
```

2、假定这时，发生了断网等意外情况，在给wang的余额加100时按下回车迟迟没有反应，这时可以使用rollback语句进行回滚，可以看到zhao的余额回滚了

```
mysql> rollback;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from account;
+------+-------+-------+
| uid  | uname | money |
+------+-------+-------+
|    1 | li    |   100 |
|    2 | wang  |   200 |
|    3 | zhao  |   700 |
+------+-------+-------+
3 rows in set (0.00 sec)
```

- 如果刚刚没有发生意外情况，则可以执行commit指令，一次性按序执行所有语句

3.4.1 Redis事务功能

Redis的事务（使用语言： redis命令）

- 开启事务：MULTI
- 失败： DISCARD取消
- 成功：EXEC执行

Mysql中的事务（使用语言：SQL语句）：

- 开启事务：START TRANSACTION
- 失败：ROLLBACK回滚
- 成功：COMMIT执行

3.4.1 Redis事务功能

- **MULTI命令**

  MULTI 不跟参数

  用于标记事务的开始

  Redis会将后续的命令逐个放入队列中，然后才能使用EXEC命令原子化地执行这个命令序列

  返回值是一个简单的字符串，总是OK

- **EXEC命令**

  EXEC不跟参数

  执行事务中放入队列中命令，然后恢复正常的连接状态

  返回值是一个数组，其中的每个元素分别是原子化事务中的每个命令的返回值，如果事务执行中止，那么EXEC命令就会返回一个Null值。

3.4.1 Redis事务功能

在redis中准备数据

- wang的余额有200
- zhao的余额有700

```
redis 127.0.0.1:6379> flushdb
OK
redis 127.0.0.1:6379> set wang 200
OK
redis 127.0.0.1:6379> set zhao 700
OK
redis 127.0.0.1:6379>
```

- 使用Redis事务功能模拟zhao转账100给wang

- Redis中使用multi命令表示事务开始，返回ok之后输入事务内容

```
redis 127.0.0.1:6379> multi
OK
redis 127.0.0.1:6379> decrby zhao 100
QUEUED
redis 127.0.0.1:6379> incrby wang 100
QUEUED
redis 127.0.0.1:6379>
```

- 可以看到在multi之后输入redis命令，返回的不是ok，而是queued，表示该命令并没有执行，而是放进了队列
- 这时我们使用exec命令完成事务

```
redis 127.0.0.1:6379> multi
OK
redis 127.0.0.1:6379> decrby zhao 100
QUEUED
redis 127.0.0.1:6379> incrby wang 100
QUEUED
redis 127.0.0.1:6379> exec
1) (integer) 600
2) (integer) 300
redis 127.0.0.1:6379>
```

- Exec表示立刻按序执行队列里的所有命令，并且执行过程中不允许执行其他redis命令，并且我们看到了decrby和incrby这两条指令的返回值

3.4.1 Redis事务功能

上述是redis中顺利完成事务的例子，现在讨论一些事务中的意外情况

- 情况1：事务中出现语法错误语句，执行exec指令

```
redis 127.0.0.1:6379> exec
1) (integer) 600
2) (integer) 300
```

```
redis 127.0.0.1:6379> multi
OK
redis 127.0.0.1:6379> decrby zhao 100
QUEUED
redis 127.0.0.1:6379> sdfa
(error) ERR unknown command 'sdfa'
redis 127.0.0.1:6379> exec
(error) EXECABORT Transaction discarded because of previous errors.
```

- 在事务中出现“sdfa”这种无法识别的命令，执行exec之后提示错误，并将事务进行取消，我们再来看下zhao和wang此时的余额

```
redis 127.0.0.1:6379> mget zhao wang
1) "600"
2) "300"
redis 127.0.0.1:6379>
```

- 可以看到，依然为600和300，说明multi之后的decrby没有执行

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> set k1 v1
QUEUED
127.0.0.1:6379> set k2 v2
QUEUED
127.0.0.1:6379> set k3 v3
QUEUED
127.0.0.1:6379> getset k3
(error) ERR wrong number of arguments for 'getset' command
127.0.0.1:6379> set k4 v4
QUEUED
127.0.0.1:6379> set k5 v5
QUEUED
127.0.0.1:6379> exec
(error) EXECABORT Transaction discarded because of previous errors.
127.0.0.1:6379> get k2
(nil)
127.0.0.1:6379> get k5
(nil)
```

- 中间出现了一条“getset k3”语法错误的语句报错，在执行exec之后，事务中错误语句前后的语句全部不执行

- 情况2：事务中存在语法正确，但执行会报错的语句

```
redis 127.0.0.1:6379> exec
1) (integer) 600
2) (integer) 300
```

```
redis 127.0.0.1:6379> multi
OK
redis 127.0.0.1:6379> decrby zhao 100
QUEUED
redis 127.0.0.1:6379> sadd wang pig
QUEUED
redis 127.0.0.1:6379> exec
1) (integer) 500
2) (error) ERR Operation against a key holding the wrong kind of value
```

- 代码中可以看到事务中存在一条“sadd wang pig”语句，这条语句语法上没有问题，由于wang不是集合，因此这条语句执行后会报错。
- 但redis依然把这条语句存到队列中，因为此时并没有exec执行，redis不知道这条语句是否会报错
- 执行了exec之后，redis对“sadd wang pig”报错了，但之前decrby语句执行了

```
redis 127.0.0.1:6379> mget zhao wang
1) "500"
2) "300"
redis 127.0.0.1:6379>
```

3.4.1 Redis事务功能

◆ Redis中事务

1、语法错误的命令，该条命令回车后直接报错；执行exec之后，所有事务中的语句均得不到执行

2、语法正确的而执行后会报错的命令，回车后先存到队列；exec之后会执行正确的语句，并跳过执行后会报错的语句

为什么redis没有回滚功能?

- 没有任何机制能避免程序员自己造成的错误，并且这类错误通常不会在生产环境中出现[1]，所以 Redis 选择了更简单、更快速的无回滚方式来处理事务。
- 这类错误应该由程序员负责

3.4.1 Redis事务功能

- **DISCARD命令**

DISCARD　不跟参数

清除所有在事务中放入队列的命令，然后恢复正常的连接状态。

返回值是一个简单的字符串，总是OK

注意： DISCARD必须和MULTI搭配使用

3.4.1 Redis事务功能

使用Discard取消事务

- 情况1：在incrby wang 100存入队列后取消事务

```
redis 127.0.0.1:6379> mget zhao wang
1) "500"
2) "300"
redis 127.0.0.1:6379>
```

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> decrby zhao 100
QUEUED
127.0.0.1:6379> discard
OK
```

假如出现异常

这时我们再来看下zhao和wang的余额

```
127.0.0.1:6379> discard
OK
127.0.0.1:6379> mget zhao wang
1) "500"
2) "300"
127.0.0.1:6379>
```

与之前完全一致，即取消掉了整个队列

3.4.1 Redis事务功能

- 情况2：在exec之后可否执行discard取消?

```
redis 127.0.0.1:6379> multi
OK
redis 127.0.0.1:6379> decrby zhao 100
QUEUED
redis 127.0.0.1:6379> sadd wang pig
QUEUED
redis 127.0.0.1:6379> exec
1) (integer) 400
2) (error) ERR Operation against a key holding the wrong kind of value
redis 127.0.0.1:6379> discard
(error) ERR DISCARD without MULTI
redis 127.0.0.1:6379>
```

- 在事务执行exec之后发生了报错，使用discard语句无法取消，提示discard必须与multi连用
- 因此，discard命令只能取消还没进行exec执行的指令

3.4.1 Redis事务功能

- 事务锁

例子：

买票的过程

剩余票的数量ticket，用户余额money

购买票的过程：ticket-1，money-100

假设：票只剩一张了，然而在事务执行exec之前，票被人买了，该怎么办?

- 1、设置ticket=1，lisi余额300，wang的余额300

```
127.0.0.1:6379> flushdb
OK
127.0.0.1:6379> set ticket 1
OK
127.0.0.1:6379> set lisi 300
OK
127.0.0.1:6379> set wang 300
OK
```

- 2、开启事务，lisi进行买票过程

```
127.0.0.1:6379> multi
OK
127.0.0.1:6379> decr ticket
QUEUED
127.0.0.1:6379> decrby lisi 100
QUEUED
```

终端1

- 3、在lisi买票的过程中（还没exec），另一个终端，其他人正巧完成了买票，即开启另一个终端模拟其他人买票

```
127.0.0.1:6379> decr ticket
0
127.0.0.1:6379> get ticket
0
```

终端2

- 4、这时，lisi完成了买票（exec执行）

```
127.0.0.1:6379> exec
1) (integer) -1
2) (integer) 200
```

终端1

- 票没买到，钱却付了

3.4.1 Redis事务功能

- 碰到上述情况该如何解决?

1、悲观锁

- 悲观，每次取数据的时候都会认为别人会修改这些数据，所以每次取数据的时候都会给这些数据加锁，别人想拿这些数据就会阻塞
  - 买票一定会有人跟我抢票，我买票的时候一定要给ticket上锁，别人只能等着

- 特点：数据记录很多时并发性差，一致性好
  - 10000条，如果用户A修改第2条，用户B修改第5条，……相互不冲突，本可以同时进行，但用悲观锁，效率低下只能排队一个一个进行修改

3.4.1 Redis事务功能

2、乐观锁

- 取数据的时候，都认为别人不会修改这些数据，所以它不会给这些数据加锁，但是每次更新后提交时都会检查数据是否最新，不是的话则放弃更新
  - 在数据较多时，对每条数据加上个version号，A用户更新后提交时将version +1，B提交该记录时如果发现自己version比现有的低，则重新提取高version的数据重新修改提交，提交后把version+1
  - 买票时不会有人跟我抢，因此我买票时大家也可以买，只需要注意我的ticket提交的值是不是最新的就可以了

- 特点：吞吐量高，并发性好

3.4.1 Redis事务功能

- 两种方法有均有各自应用需求

  悲观锁常用于数据备份、格式化等

  乐观锁常用于多读应用等

- Redis的事务可以启用乐观锁

  负责监测key没有被改动

3.4.1 Redis事务功能

- 事务的 WATCH 命令

WATCH key [key1 key2 ….]

用于监视事务中的键值是否变化，只有所有被WATCH命令监视的数据库键都没有被修改的前提下，事务才能执行成功

有任意一个被监视的数据库键被修改，那么这个事务都会执行失败

数据准备：lisi余额200，ticket剩余1

1、使用watch命令监控键值ticket，并开始买票事务

```
127.0.0.1:6379> set ticket 1
OK
127.0.0.1:6379> set lisi 200
OK
127.0.0.1:6379> watch ticket
OK
127.0.0.1:6379> multi
OK
127.0.0.1:6379> decr ticket
QUEUED
127.0.0.1:6379> decrby lisi 100
QUEUED
127.0.0.1:6379>
```

终端1

2、此时暂时不执行exec，开启终端2模拟另一个用户抢先把票给买了

```
127.0.0.1:6379> decr ticket
0
```

终端2

3、回到终端1，执行exec完成全部买票过程

```
127.0.0.1:6379> exec
(nil)
127.0.0.1:6379> mget ticket lisi
1) "0"
2) "200"
```

终端1

发现exec执行后返回nil，执行失败，查看票数和lisi的余额发现，票已经被买走了，但lisi的钱并没有扣除，符合实际规律

3.4.1 Redis事务功能

- 在执行了exec或discard命令后，会自动取消watch监控

  下次事务时必须重新再写watch

- Watch可以用来监视多个key

  如watch ticket lisi

  有任意一个被监视的key被修改，那么事务都会执行失败

3.4.1 Redis事务功能

- UNWATCH命令

UNWATCH　　不跟参数

取消所有WATCH命令对数据库键的监视

EXEC或DISCARD命令执行后自动取消所有监控，不必再执行UNWATCH命令

例：在准备开始事务前，已经监控后发现临时有其他用户修改了citys

127.0.0.1:6379> LPUSH citys "wuhan" "changsan" "kunming"

(integer) 3

127.0.0.1:6379> WATCH citys

OK

127.0.0.1:6379> LPUSH citys "hangzhou"

(integer) 4

127.0.0.1:6379> UNWATCH

OK

127.0.0.1:6379> WATCH citys

OK

127.0.0.1:6379>MULTI

……

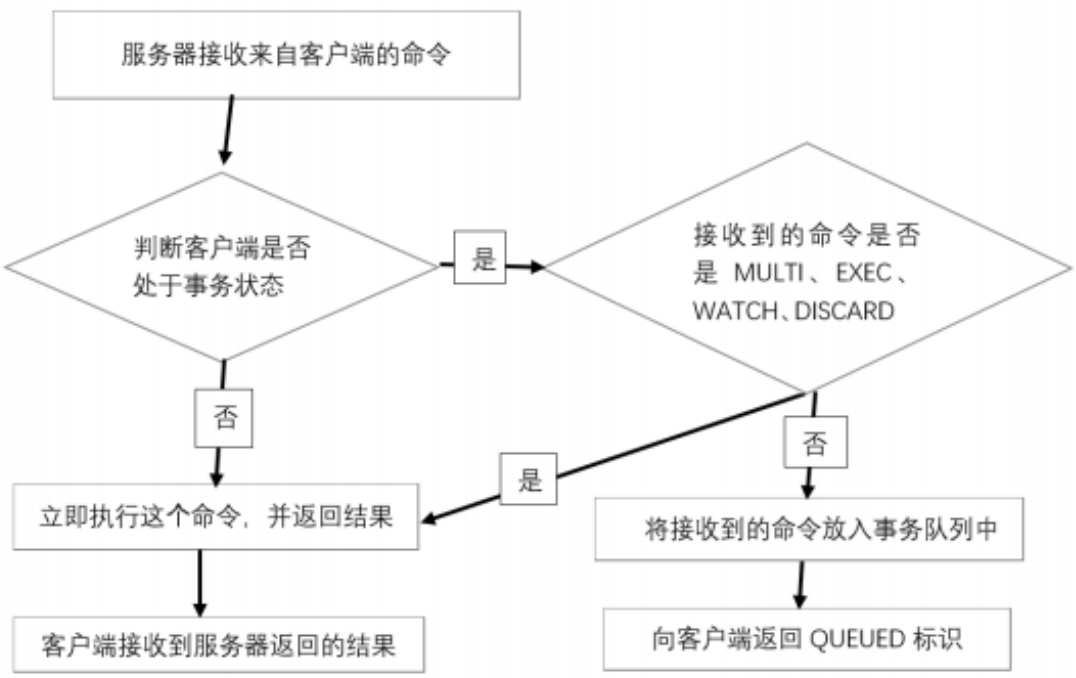
127.0.0.1:6379>EXEC

OK

已经开始监控city，但开始事务前发现citys被改了；如果此时再开始事务就无效了，所以这时先取消监控后，再重新监控

3.4.1 Redis事务功能

- Redis事务过程



·当服务器接收到客户端发送过来的命令是MULTI、EXEC、WATCH、DISCARD 、UNWATCH中的任意一个时，服务器会立即执行这个命令

·否则服务器不会立即执行这个命令，而是将该命令放入一个事务队列中，然后返回QUEUED标识给客户端

总结

1、什么是事务?

2、关系型数据库事务的4个特性

3、两种事务锁

4、Redis事务语句

- MULTI
- EXEC
- DISCARD
- WATCH
- UNWATCH

5、redis事务与关系型数据库事务的不同

3.4.1 Redis事务功能

面试经常问的一个问题

Redis是否支持事务?

答：redis是部分支持事务或支持简单事务；

- 事务的原子性要求：一个原子事务要么完整执行，要么干脆不执行。
- 在redis事务中语法错误的命令不会存到队列，该条命令回车后直接报错，exec之后所有事务中的语句得不到执行；而语法没错而执行后会报错的命令，会先存到队列中，exec后会执行正确的语句，跳过执行后会报错的语句
- 因此不完全满足原子性特征