day04笔记

Redis事务

特点

```
1. 单独的隔离操作：事务中的所有命令会被序列化、按顺序执行，在执行的过程中不会被其他客户端发送来的命令打断
2. 不保证原子性：redis中的一个事务中如果存在命令执行失败，那么其他命令依然会被执行，没有回滚机制
```

事务命令

```
1、MULTI  # 开启事务
2、命令1  # 执行命令
3、命令2 ... ...
4、EXEC  # 提交到数据库执行
4、DISCARD # 取消事务
```

使用步骤

```
# 开启事务
127.0.0.1:6379> MULTI
OK
# 命令1入队列
127.0.0.1:6379> INCR n1
QUEUED
# 命令2入队列
127.0.0.1:6379> INCR n2
QUEUED
# 提交到数据库执行
127.0.0.1:6379> EXEC
1) (integer) 1
2) (integer) 1
```

事务中命令错误处理

```
# 1、命令语法错误，命令入队失败，直接自动discard退出这个事务
   这个在命令在执行调用之前会发生错误。例如，这个命令可能有语法错误（错误的参数数量，错误的命令名）
   处理方案：客户端发生了第一个错误情况，在exec执行之前发生的。通过检查队列命令返回值:如果这个命令回答这个队列的命令是正确的，否者redis会返回一个错误。如果那里发生了一个队列命令错误，大部分客户端将会退出并丢弃这个事务

# 2、命令语法没错，但类型操作有误，则事务执行调用之后失败，无法进行事务回滚
    从我们施行了一个由于错误的value的key操作（例如对着String类型的value施行了List命令操作）
    处理方案：发生在EXEC之后的是没有特殊方式去处理的：即使某些命令在事务中失败，所有的其他命令都将会被执行。
127.0.0.1:6379> MULTI
OK
```

```
127.0.0.1:6379> set num 10
QUEUED
127.0.0.1:6379> LPOP num
QUEUED
127.0.0.1:6379> exec
1) OK
2) (error) WRONGTYPE Operation against a key holding the wrong kind of value
127.0.0.1:6379> get num
"10"
127.0.0.1:6379>
```

为什么redis不支持事务回滚

- 观点

```
1、Redis的内部极其简单和快速，来源于它不需要回滚功能
2、在生产环境中，通常回滚并不能解决来自编程的错误。举个例子，你本来想+1，却+2了，又或者+在错误的类型上，回滚并不能解决。由于无法提供一个避免程序员自己的错误，而这种错误在产品中并不会出现，所以选择一个简单和快速的方法去支持事务
```

pipeline补充

python使用pipeline()与execute()批量进行批量操作

示例

```python
import redis

# 创建连接池并连接到redis
pool = redis.ConnectionPool(host = '192.168.153.150',db=0,port=6379)
r = redis.Redis(connection_pool=pool)

# 第一组
pipe = r.pipeline()
pipe.set('fans',50)
pipe.incr('fans')
pipe.incrby('fans',100)
pipe.execute()

# 第二组
pipe.get('fans')
pipe.get('pwd')
# [b'151', b'123']
result = pipe.execute()
print(result)
```

Redis常见问题汇总

- **Redis优点**

```
1、读写速度快. 数据存放在内存中
2、支持数据类型丰富,string,hash,list,set,sorted
3、支持事务
4、可以用于缓存,消息队列,按key设置过期时间,到期后自动删除
5、支持数据持久化(将内存数据持久化到磁盘),支持AOF和RDB两种持久化方式,从而进行数据恢复操作,可以有效地防止数据丢失
5、支持主从(master-slave)复制来实现数据备份,主机会自动将数据同步到从机
```

- **来介绍一下redis中的数据类型**

类型	特点	使用场景
string	简单key-value类型，value可为字符串和数字	常规计数（微博数, 粉丝数等功能）
hash	是一个string类型的field和value的映射表，hash特别适合用于存储对象	存储部分可能需要变更的数据（比如用户信息）
list	有序可重复列表	关注列表，粉丝列表，消息队列等
set	无序不可重复列表	存储并计算关系（如微博，关注人或粉丝存放在集合，可通过交集、并集、差集等操作实现如共同关注、共同喜好等功能）
sorted set	每个元素带有分值的集合	各种排行榜

- **redis中的持久化方案**

```
# RDB
快照形式，定期把内存中的数据保存到磁盘。Redis默认支持的持久化方案。速度快但是服务器断电的时候会丢失部分数据

# AOF
把所有对redis数据库增删改操作的命令保存到文件中。数据库恢复时把所有的命令执行一遍即可。
# 两种持久化方案同时开启使用AOF文件来恢复数据库.能保证数据的完整性,但是速度慢。
```

- **使用过Redis分布式锁么，它是什么回事?**

```
1、从redis2.8开始，set命令集成了两个参数，nx和ex，先拿nx来争抢锁，抢到之后，再用ex参数给锁加一个过期时间防止锁无法释放，造成死锁
2、redis分布式锁原理见图
```

- **缓存穿透**

```
# 原理
缓存和数据库都没有的数据，而用户反复发起请求， 如 假的用户ID

# 场景
比如发起为id为“-1”的数据或id为特别大不存在的数据。这时的用户很可能是攻击者，攻击会导致数据库压力过大

# 解决方案：
    1、请求校验，接口层增加校验，如对id做基础校验，id<=0的直接拦截
    2、都无法取到数据时也可以将key-value对写为key-null，缓存有效时间比如30秒左右，这样可以防止攻击用户反复用同一个id暴力攻击
```

- **缓存击穿**

```
# 原理
缓存没有，数据库有，一般是缓存时间到期， 顺势并发太大

#解决方案
1、热点数据不过期
2、上锁：重新设计缓存的使用方式，当我们通过key去查询数据时，首先查询缓存，如果没有，就通过分布式锁进行加锁，取得锁的进程查DB并设置缓存，然后解锁；其他进程如果发现有锁就等待，然后等解锁后返回缓存数据或者再次查询DB
```

- **缓存雪崩**

```
# 原理
缓存中大批量数据过期，导致瞬时大批量不同请求注入DB

# 解决方案
解决方案
1、缓存设置随机时间（避免缓存设置相近的有效期；为有效期增加随机值）
2、热点数据不过期
```