Redis

讲师：潘sir

缓存介绍

产生背景

- 访问量的上升，以及对响应时间的要求提升，单DB无法再满足要求，所以考虑DB拆分(sharding)、读写分离、甚至硬件升级(SSD)等以满足新的业务需求，但仍有缺点：
  1. 性能提升有限，很难达到数量级上的提升，尤其在互联网业务场景下，随着网站的发展，访问量经常会面临十倍、百倍的上涨。
  2. 成本高昂，为了承载N倍的访问量，通常需要N倍的机器，这个代价难以接受
- 大部分的业务场景下，80%的访问量都集中在20%的热数据上(适用二八原则)

解决方案

- 由单层DB的数据存储结构，也变为Cache+DB的结构，也就是引入缓存

缓存优势

1. 提升数据读取速度
2. 提升系统扩展能力，通过扩展缓存，提升系统承载能力
3. 降低存储成本，Cache+DB的方式可以承担原有需要多台DB才能承担的请求量，节省机器成本

缓存方式

根据业务场景分类

1. 懒汉式：写入DB后, 然后把相关的数据也写入Cache
2. 饥饿式：先查询DB里的数据, 然后把相关的数据写入Cache
3. 定期刷新：适合周期性的跑数据的任务，或者列表型的数据，而且不要求绝对实时性

根据技术层面分类

1. 应用内缓存：Map(简单的数据结构)，以及EH Cache(Java第三方库)
2. 缓存组件：Memached，Redis

关系与非关系

	关系型数据库	非关系型数据库
代表	Mysql/oracle/sqlserver	redis/cb/hb/mongodb/es
存储方式	表格式，有行列	大块组合，键值对
存储结构	结构化数据，定义列(数据类型)	非结构化数据，动态结构
存储规范	库->表->值	库->值
存储扩展	纵向扩展，瓶颈在硬件	天然分布式，横向扩展
查询方式	SQL查询	块单元操作，UnQL
事务	ACID原则	BASE原则：基本可用/软事务性/最终一致性
性能	读写性能差，海量数据处理效率低	key-value类型，存储在内存中，读写性能极高

四类NoSQL

1. 键值对：使用hash表，特定的键指向特定的数据。如：redis
2. 列存储：键指向多个列，通常用于分布式存储海量数据(大数据)，如：Hbase
3. 文档型：键值对，但是存储的是半结构化文档，将文档以特定格式(json)存储，允许嵌套键值。优点：比一般键值数据库的查询效率更高，如：CouchBase，MongoDB，Elasticsearch
4. 图形数据库：Neo4J, InfoGrid, Infinite Graph

Redis缓存的优点

1. 高性能Key-Value存储：适合当缓存
2. 丰富的数据结构：string、list、hash、set、zset、hypeloglog
3. 支持数据过期：主动过期+惰性过期
4. 支持多种LRU策略：volatile-lru、volatile-ttl 等
5. 内存管理：tcmaloc、jemalloc
6. 内存存储+磁盘持久化: 数据存在内存，支持多种方式(rdb、aof)进行数据持久化，将数据写入硬盘
7. 支持主从复制
8. 单线程：多路复用方式提高处理效率，常用作分布式锁

Redis操作

Redis环境搭建

- ubuntu下安装reidis：`sudo apt-get -y install redis-server`，安装完成后输入：`redis-cli`即可进入redis命令行模式

- 配置远程连接：

  1. 修改配置文件：/etc/redis/redis.conf，注释这一行 : bind 127.0.0.1
  2. 设置密码：requirepass password
  3. 重启redis服务：`sudo /etc/init.d/redis-server restart`
  4. 防火墙开放端口，允许6379端口：`sudo ufw allow 6379/tcp`
  5. 使用密码登录redis：`./redis-cli -h 127.0.0.1 -p 6379 -a myPassword`

Redis基本操作

基本数据类型

数据类型	说明
String(字符串)	相当于python字符串
Hash(散列)	键值对，类似python字典的键值对
List(列表)	相当于python列表
Set(集合)	无序，元素不重复
Sorted Set(有序集合)	有序，元素不重复

基本命令

这里只讲最基本的命令，扩展命令见：https://www.runoob.com/redis

- 操作key：

```
# 创建1个key，这个key的值是 captain

127.0.0.1:6379> set name captain

# 给这个key设置过期时间，60秒

127.0.0.1:6379> expire name 60

# 查询这个key的过期时间

127.0.0.1:6379> ttl name

# 查询key

127.0.0.1:6379> keys name

# 模糊匹配
127.0.0.1:6379> keys *me*

# 删除这个key
127.0.0.1:6379> del name
```

- 添加字符串

```
# 添加一个key，设置他的值为字符串
127.0.0.1:6379> set name captain

# 查询这个key，可以获得其值
127.0.0.1:6379> get name
```

- 练习：以自己的名字创建一个key，并设置过期时间30秒，然后验证该key是否过期
- 添加集合，数据不能重复

```
# 添加一个集合，并赋值
127.0.0.1:6379> sadd age 18
127.0.0.1:6379> sadd age 30,32

# 查询这个key，可以获得集合里的所有值
127.0.0.1:6379> smembers age

# 删除集合中某个值
127.0.0.1:6379> srem age 32
```

- 练习：创建一个集合，添加多个姓名信息
- 添加哈希(Hash)

```
# 添加一个哈希，并创建key-value
127.0.0.1:6379> hset location dcs shenzhen
127.0.0.1:6379> hset location xiaohong guangzhou

# 查询该redis-key下的所有 hash-key
127.0.0.1:6379> hkeys location

# 查询某个key的值
127.0.0.1:6379> hget location panwj

# 查询所有
127.0.0.1:6379> hgetall location

# 删除集合中某个值
127.0.0.1:6379> srem age 32
```

- 练习：创建一个哈希key，添加多个键值对，{"name": "dcs", "age": 15, "loacation": "shenzhen"}

- 补充：redis 数据库db0-db15；默认情况下，redis会生成0-15号共16个db，以供不同情境使用的需要；不同的数据库下，相同的key各自独立

Python操作redis

- 安装第三方库：`pip install redis`

redis连接方式

普通连接/直接连接

```
import redis
# decode_responses=True 自动解码，默认查询出来的数据格式是bytes
# 方式1
r1 = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0)
# r1 = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)
print(type(r1))
r1.set("name", "dcs21")
a = r1.get("name")
print(a ,type(a))

# # 方式2
# r2 = redis.StrictRedis(host='10.211.55.5', port=6379, password='123456',
db=0, decode_responses=True)
# print(type(r2))
```

```
<class 'redis.client.Redis'>
b'dcs21' <class 'bytes'>
```

补充：

1. 通过编码, 从str转换bytes， 比特流=str(串,encoding='utf-8')
2. 通过解码，从bytes转换到str，串=bytes(比特流,encoding='utf-8')

redis连接池

为了避免每次建立，释放连接的开销。直接建立一个连接池，作为参数传给redis，这样可以实现多个redis实例共享一个连接池。

```
import redis
# 创建连接池
# pool = redis.ConnectionPool(host='10.211.55.5', port=6379,
password='123456', db=0)
pool = redis.ConnectionPool(host='10.211.55.5', port=6379, password='123456',
db=0, decode_responses=True)
# 使用连接池进行实例化
r = redis.Redis(connection_pool=pool)
r.set("name", "dcs")
print(r.get("name"))
```

```
dcs
```

补充：性能测试中，可以考虑的性能调优点

对key的操作

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)

# 1. 删除key
r.delete("test_key")
r.set("test_key", 1)

# 2. 判断是否存在，存在则返回1，不存在则返回0
r.exists("test_key")

# 3. 查询key
r.keys("test*")

# 4. 设置/修改过期时间
r.expire("test_key", 300)
# 查询过期时间
r.ttl("test_key")

# 5. 查看类型
print(r.type("test_key"))

# 6. 重命名
print(r.rename("test_key", "test_kk"))

# 7. 移动分区
print(r.move("test_kk", 5))
```

```
string
True
True
```

string基本操作

单个设置

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)
# 1. 在Redis中设置值，默认不存在则创建，存在则修改
r.set('name', 'dcs')
# 2. 获取值
r.get("name")

# 3. 同时设置：key/ttl/value
r.setex("dcs", 100, "hello")
print(r.get("dcs"))
print(r.ttl("dcs"))
```

```
hello
100
```

- 参数说明：`set(name, value, ex=None, px=None, nx=False, xx=False)`
    1. ex：过期时间秒
    2. px：过期时间-毫秒
    3. nx：设为True时，则当name不存在才操作，通setnx
    4. xx：设为True时，则当name存在时才操作

批量设置

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)
# 批量设置值
keys = {
    "name1": "dcs",
    "name2": "houhc"
}
a = r.mset(keys)
print(a)

# 批量查询
kk = ("name1", "name2")
b = r.mget(kk)
print(b)
```

```
True
['dcs', 'houhc']
```

自增

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)
r.set("freq", 1)
# 增加指定的amount(默认1)，若key不存在则创建
print(r.incr("freq"))
print(r.incr("freq",amount=2))
```

```
2
4
```

应用：统计页面点击数

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)

# 初始化
r.set("static_click", 100)
r.incr("static_click")
```

```
101
```

补充用法

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)

r.set("name", "dcs123")

# 1. 设置新值，打印原值
print(r.getset("name","houhc"))

# 2. 根据字节获取
print(r.getrange("name",0,3))

# 3. 从指定字符串索引开始向后替换，如果新值太长时，则向后添加
print(r.setrange("name",4,"zzzzzzz"))
print(r.get("name"))

# 4. 获取长度
```

```
print(r.strlen("name"))

# 5. 追加内容
print(r.append("name","scl"))
print(r.get("name"))
```

```
dcs123
houh
11
houhzzzzzzz
11
14
houhzzzzzzzscl
string
```

```
300
```

list操作

添加操作

lpush/rpush，注意：

1. 添加时不会进行去重处理
2. 如果key本身不存在，会自动创建

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)
# 删除key
r.delete("dcs_list")
# 1. lpush往列表左边添加元素，left
r.lpush("dcs_list",2)
r.lpush("dcs_list", 2, 3, 4, 5)
# ======================

# 2. rpush往列表右边添加，right
r.rpush("dcs_list", 6, 7, 8)

print(r.lrange("dcs_list", 0, 7))
```

```
['5', '4', '3', '2', '2', '6', '7', '8']
```

- lpushx/rpushx：判断key是否存在，只有存在才会添加

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)
# 删除key
r.delete("dcs_list")
# 1. lpushx往列表左边添加元素
r.lpush("dcs_list",2)
r.lpushx("dcs_list", 2)

# 2. rpushx往列表右边添加
r.rpushx("dcs_list", 6)

print(r.lrange("dcs_list", 0, 7))
```

```
['2', '2', '6']
```

修改与删除

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)
# 1. 获取长度
r.delete("dcs_list")
r.rpush("dcs_list", 1,2,3,4,5,6,7,8,9)

# 2. 指定位置插入，在"dcs_list"中找到元素 4 ，然后在它前面添加 "hello"
# BEFORE/AFTER
```

```
r.linsert("dcs_list", "BEFORE", "4", "hello")
print(r.lrange("dcs_list", 0, 10))

# 3. 重新赋值，指定索引位置
r.lset("dcs_list",  3, "world")
print(r.lrange("dcs_list", 0, 10))

# 4. 删除指定位置
r.lrem("dcs_list",  3, "world")
print(r.lrange("dcs_list", 0, 10))
```

```
['1', '2', '3', 'hello', '4', '5', '6', '7', '8', '9']
['1', '2', '3', 'world', '4', '5', '6', '7', '8', '9']
['1', '2', '3', '4', '5', '6', '7', '8', '9']
```

补充操作

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)
# 获取长度
r.delete("dcs_list")
r.rpush("dcs_list", 1,2,3,4,5,6,7,8,9)

# 1. 获取指定位置的值
print(r.lindex("dcs_list", 1))

# 2. 分片获取，指定范围
print(r.lrange("dcs_list",0,-1))

# 1. 移除第一个元素，返回该值，  rpop/lpop
print(r.rpop("dcs_list"))
print(r.lrange("dcs_list", 0, 10))

# 4. 批量截取，注意，该方法执行返回的是True
print(r.ltrim("dcs_list",0, 2))
print(r.lrange("dcs_list", 0, 10))

# 5. 右出左入，调个头
print(r.rpoplpush("dcs_list", "dcs_list"))
print(r.lrange("dcs_list", 0, 10))
```

```
2
['1', '2', '3', '4', '5', '6', '7', '8', '9']
9
['1', '2', '3', '4', '5', '6', '7', '8']
True
['1', '2', '3']
3
['3', '1', '2']
```

set集合操作

添加操作

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)
r.delete("dcs_set")
# 1. 添加多个元素，不能重复
r.sadd("dcs_set", "age", "name")
# 2. 查询
r.smembers("dcs_set")

# 3. 获取元素个数
r.scard("dcs_set")
```

```
2
```

移除

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)

r.sadd("a", "hello")
r.sadd("b", "world")
# 1. 从一个集合移到另一个集合
print(r.smove("a", "b", "hello"))
print(r.smembers("a"))
print(r.smembers("b"))

# 2. 从集合左边移除，并返回该值
r.delete("dcs_set")
r.sadd("dcs_set", 1,2,3,4)
```

```python
print(r.smembers("dcs_set"))
print(r.spop("dcs_set"))
print(r.smembers("dcs_set"))
print(r.spop("dcs_set"))
print(r.smembers("dcs_set"))

# 3. 移除指定的值
r.delete("dcs_set")
r.sadd("dcs_set", 1,2,3,4)
r.srem("dcs_set", 3)
print(r.smembers("dcs_set"))

# 4. 求集合交集
r.delete("dcs_set")
r.sadd("dcs_set1", 1,2,3,)
r.sadd("dcs_set2", 3,4)
print(r.sinter("dcs_set1", "dcs_set2"))

# 5. 求集合的并集
r.delete("dcs_set")
r.sadd("dcs_set1", 1,2,3,)
r.sadd("dcs_set2", 3,4)
print(r.sunion("dcs_set1", "dcs_set2"))
```

```
True
set()
{'world', 'hello'}
{'2', '3', '1', '4'}
1
{'2', '3', '4'}
2
{'3', '4'}
{'2', '1', '4'}
{'3'}
{'2', '3', '1', '4'}
```

- 应用场景：社交用户分群，用户标签，用户画像
- 练习：求两个集合：lol/wzry 的共同爱好者；求整个群体的爱好者

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)

r.sadd('circle:game:lol','user:lijh', 'user:xiep', 'user:aqiang')
r.sadd('circle:game:wzry','user:xiep', 'user:azhen')

# 获取两个群体的共同爱好者
print(r.sinter('circle:game:lol', 'circle:game:wzry'))

# 获取game游戏的所有用户
print(r.sunion('circle:game:lol', 'circle:game:wzry'))
```

```
{'user:xiep'}
{'user:aqiang', 'user:lijh', 'user:azhen', 'user:xiep'}
```

思考：如果同样的数据，key和value互换：

```
r.sadd('user:xiep', 'game:lol', 'game:wzry')
r.sadd('user:lijh', 'game:lol')
```

hash(字典)操作

添加操作

```
import redis
r = redis.Redis(host='10.211.55.5', port=6379, password='123456', db=0,
decode_responses=True)

# 1. 添加单个
r.hset("dcs_dict", "name", "dcs")
# 全部取出
print(r.hgetall("dcs_dict"))
# 取出单个
print(r.hget("dcs_dict", "name"))
# 取出全部key
print(r.hkeys("dcs_dict"))
# 求长度
print(r.hlen("dcs_dict"))
# 取出所有value
print(r.hvals("dcs_dict"))

a = {
    "name": "ddd",
    "age": 18
}
# 2. 添加多个
```

```
r.hmset("dcs_dict", a)
print(r.hgetall("dcs_dict"))

# 3. 删除指定的字段
r.hdel("dcs_dict", "name")
print(r.hgetall("dcs_dict"))
```

```
{'age': '18', 'name': 'dcs'}
dcs
['age', 'name']
2
['18', 'dcs']
{'age': '18', 'name': 'ddd'}
{'age': '18'}
```

结合业务开发TsmsRedis

- 练习：发起充值接口，校验充值是否成功
- 需要封装方法：查询账户余额 `def get_account(self, user="dcs"):`

```
    def get_account(self, user="dcs"):
        """获取充值金额"""
        key = "tsms:%s:account" % user
        res = self.rds.get(key)
        logging.info("{} = {}".format(key, res))
        return int(res)
```

参考答案：

```
from tsms.tsms_db import TsmsDB
from tsms.tsms_base import Tsmstest
from tsms.tsms_rds import TsmsRedis
import unittest
import random
from time import sleep
from tenacity import retry, stop_after_attempt, wait_exponential
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

class TestCharge(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.db = TsmsDB()
        cls.ts = Tsmstest()
        cls.rds = TsmsRedis()
```

```
    @classmethod
    def tearDownClass(cls):
        pass

    def setUp(self):
        self.history_money = self.rds.get_account("dcs")

    def test_send_1(self):
        # 调接口
        charge_money = random.randint(1, 999)
        data = {"user": "dcs", "charge": charge_money}
        self.ts.req_post('charge', data, user="root")
        assert self.ts.status_code == 200
        assert self.ts.text == "ok"
        # 验证充值是否正确
        now_money = self.rds.get_account("dcs")
        assert charge_money == now_money - self.history_money

if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

```
test_send_1 (__main__.TestCharge) ... 2019-11-16 16:21:26,657 INFO
tsms:dcs:account = 1191:
2019-11-16 16:21:26,658 INFO     [当前被调用方法是]: req_post:
2019-11-16 16:21:26,658 INFO     [url_type is]: charge:
2019-11-16 16:21:26,659 INFO     [data is]: {'user': 'dcs', 'charge': 667}:
2019-11-16 16:21:26,660 INFO     [user is]: root:
2019-11-16 16:21:26,660 INFO     [当前请求的地址是]:
http://127.0.0.1:5001/v2/charge:
2019-11-16 16:21:26,661 INFO     [发送内容是]: {'user': 'dcs', 'charge': 667}
<class 'dict'>:
2019-11-16 16:21:26,718 INFO     [返回码是:] 200:
2019-11-16 16:21:26,719 INFO     [执行结果为]: ok:
2019-11-16 16:21:26,720 INFO     tsms:dcs:account = 1858:
ok

----------------------------------------------------------------------
Ran 1 test in 0.066s

OK
```

- 练习：发送一条消息，校验计费减少，记频增加
- 需要封装方法：查询频次和ttl：`def get_freq(self, phone):`

```
    def get_freq(self, phone):
        key = "freq:%s" % phone
        res = self.rds.get(key)
        ttl = self.rds.ttl(key)
        logging.info("{} = {} ttl = {}".format(key, res, ttl))
        return int(res), int(ttl)
```

```
from tsms.tsms_db import TsmsDB
from tsms.tsms_base import Tsmstest
from tsms.tsms_rds import TsmsRedis
import unittest
from time import sleep
from tenacity import retry, stop_after_attempt, wait_exponential
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

class TestSendtt(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.db = TsmsDB()
        cls.ts = Tsmstest()
        cls.rds = TsmsRedis()

    @classmethod
    def tearDownClass(cls):
        pass

    def setUp(self):
        self.history_money = self.rds.get_account("dcs")

    @retry(stop=stop_after_attempt(5), wait=wait_exponential(multiplier=1,
max=10))
    def check_db(self, phone):
        real_res = self.db.tsms_select("sms_send", "mobile,status,consume",
uuid=self.ts.json["uuid"])[0]
        assert real_res["mobile"] == phone
        assert real_res["status"] == "success"
        assert real_res["consume"] == 1

    def test_send_1(self):
        # 调接口
        phone = self.ts.gen_phones(1)
        data = {"sign_id": 424, "temp_id": 180, "mobiles": phone}
        self.ts.req_post('message', data)
        assert self.ts.status_code == 200
        assert isinstance(self.ts.json["uuid"], str)
```

```
        # 检查redis
        now_money = self.rds.get_account("dcs")
        # 校验金额是否计费准确
        assert now_money == self.history_money - 1
        freq, ttl = self.rds.get_freq(phone[0])
        assert freq == 1
        assert 599 <= ttl <= 600

        # 查数据库
        self.check_db(phone[0])


if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```

- 注意点：
    1. ttl是个时间，而时间本身就是变化，断言时需要使用范围，而不是一个确定的值
    2. 要注意断言顺序，需要重试的尽量放在最后
- 练习3：校验超频，正常思路：一直循环；引发问题：当限制过大，执行效率极低；所以考虑通过修改redis的值来实现
    1. 校验发送一次，计次+1
    2. 假如限制为20，则需要修改值为19，请求两次，验证：第一次通过，第二次触发异常，即可完成测试
- 需要封装方法：修改记频：`def set_freq(self, phone, value):`

```
    def set_freq(self, phone, value):
        """设置频率计数"""
        key = "freq:%s" % phone
        self.rds.setex(key, 600, value)
        res = self.get_freq(phone)
        logging.info("[now freq is]: {}".format(res))
        return res
```

- 检查点：
    1. 计费是否正常
    2. 计频是否正常
    3. 脚本修改频率是否成功
    4. 最后一次是否发送成功
    5. 超出限制这次是否发送失败
    6. 最终结果是否为：over_freq

```
from tsms.tsms_db import TsmsDB
from tsms.tsms_base import Tsmstest
from tsms.tsms_rds import TsmsRedis
```

```
import unittest
from time import sleep
from tenacity import retry, stop_after_attempt, wait_exponential
import logging
logging.basicConfig(level=logging.INFO, format='%(asctime)-16s %(levelname)-8s
%(message)s')

class TestSendtt(unittest.TestCase):
    @classmethod
    def setUpClass(cls):
        cls.db = TsmsDB()
        cls.ts = Tsmstest()
        cls.rds = TsmsRedis()

    @classmethod
    def tearDownClass(cls):
        pass

    def setUp(self):
        self.history_money = self.rds.get_account("dcs")

    @retry(stop=stop_after_attempt(5), wait=wait_exponential(multiplier=1,
max=10))
    def check_db(self, phone):
        real_res = self.db.tsms_select("sms_send", "mobile,status,consume",
uuid=self.ts.json["uuid"])[0]
        assert real_res["mobile"] == phone
        assert real_res["status"] == "success", "{}
{}".format(real_res["status"], "success")
        assert real_res["consume"] == 1

    def test_send_1(self):
        # 调接口
        phone = self.ts.gen_phones(1)
        data = {"sign_id": 424, "temp_id": 180, "mobiles": phone}
        self.ts.req_post('message', data)
        assert self.ts.status_code == 200
        assert isinstance(self.ts.json["uuid"], str)

        # 检查redis
        now_money = self.rds.get_account("dcs")
        # 校验金额是否计费准确
        assert now_money == self.history_money - 1
        freq, ttl = self.rds.get_freq(phone[0])
        assert freq == 1
        assert 599 <= ttl <= 600

        # 查数据库
        self.check_db(phone[0])
```

```
        # 修改频次
        self.rds.set_freq(phone[0], 19)
        # 第一次发
        self.ts.req_post('message', data)
        assert self.ts.status_code == 200
        assert isinstance(self.ts.json["uuid"], str)
        freq, ttl = self.rds.get_freq(phone[0])
        assert freq == 20

        # 第二次发
        self.ts.req_post('message', data)
        assert self.ts.status_code == 403
        assert self.ts.json == {'error': 'ER:0036', 'message': 'send freq out
of limit'}

        # 最终结果
        send_status = self.db.tsms_select("sms_send", "status",
mobile=phone[0])
        logging.info(send_status)
        assert {'status': 'over_freq'} in send_status

if __name__ == '__main__':
    # 执行本suite
    unittest.main(argv=['ignored', '-v'], exit=False)
```