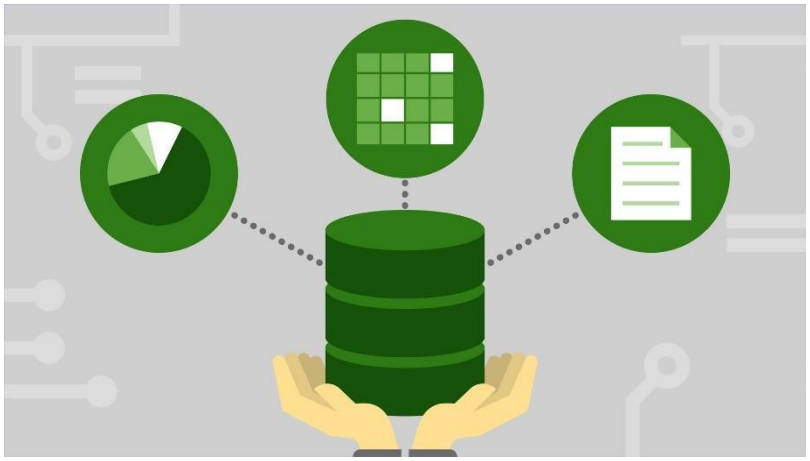
به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده برق و کامپیوتر



آزمایشگاه پایگاه داده

دستور کار شماره 8

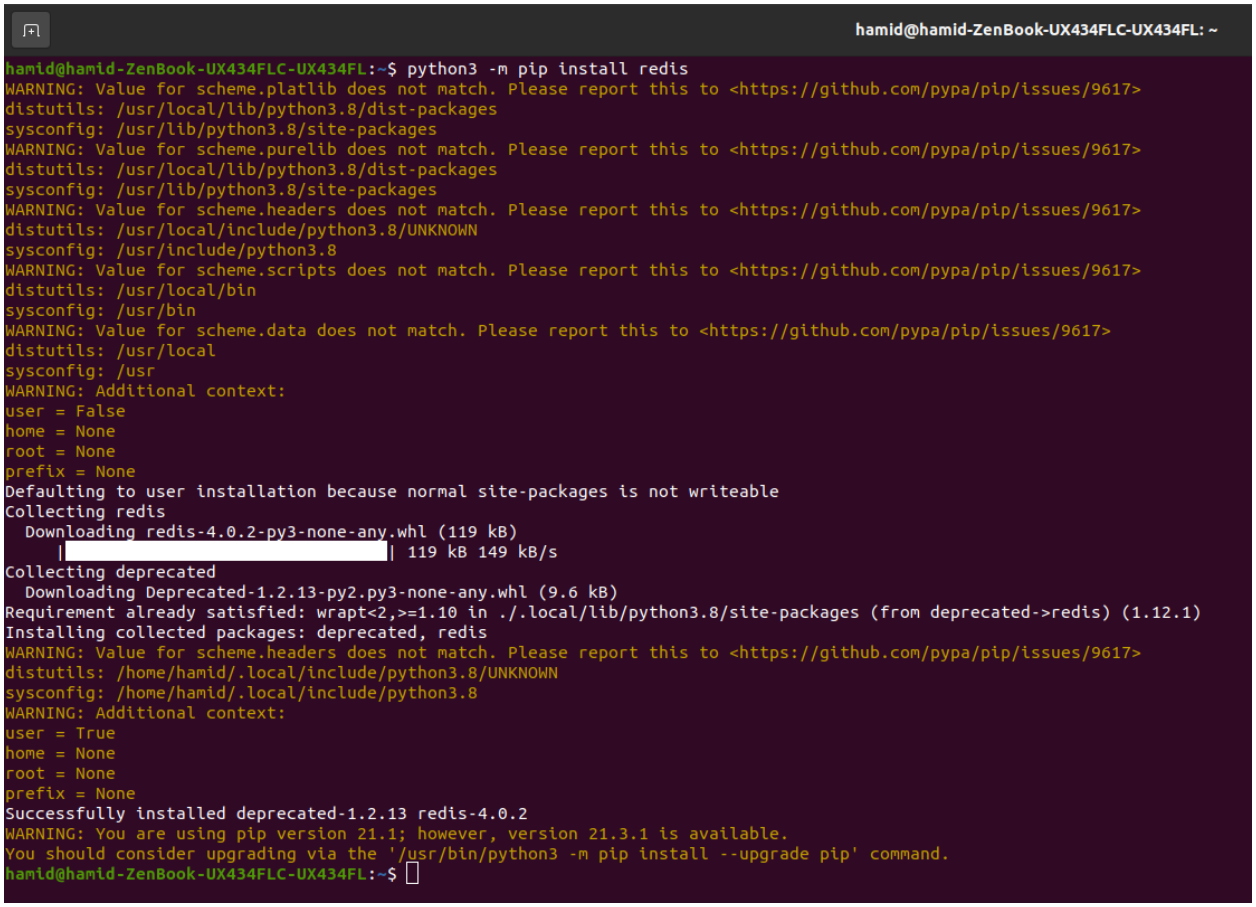
حمیدرضا خدادادی

810197499

دی ماه ۱۴۰۰

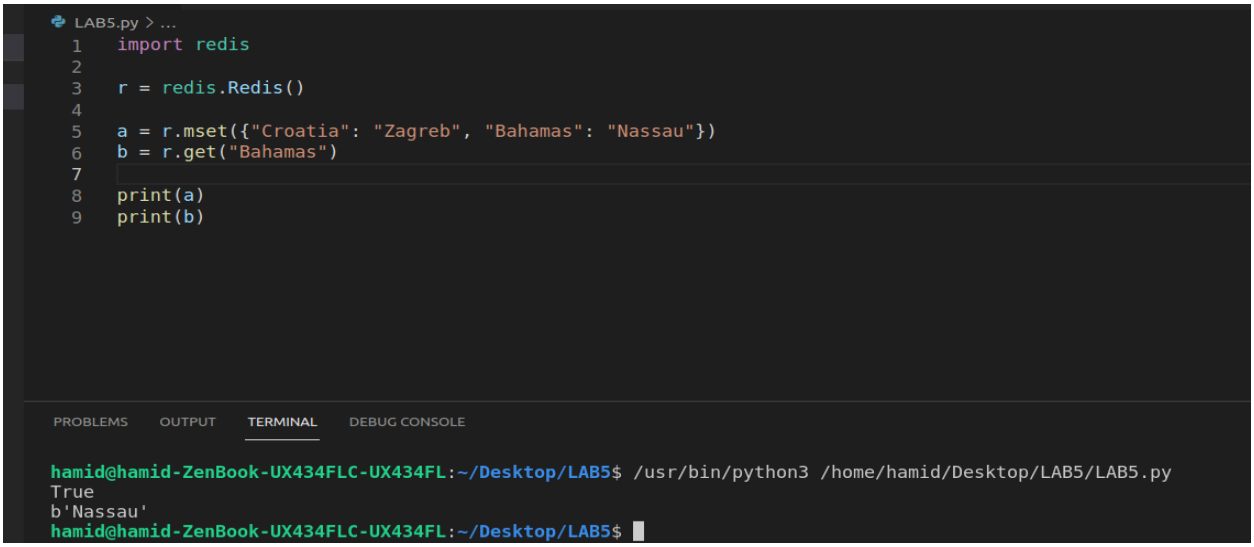
گزارش دستور کار انجام شده

در این گزارش کار می خواهیم با redis-py کار کنیم؛ کلاینت پایتونی که به ما این امکان را می دهد تا از طریق یک API کاربر Python با Redis در تعامل باشیم. پس در ابتدا کتابخانه redis-py را نصب می کنیم.



در ابتدا بررسی می کنیم که سرور ردیس فعال است. پس از آن یک کد ساده را در پایتون اجرا می کنیم. اتصال سوکت TCP و استفاده مجدد از آن در پشت صحنه انجام می شود و ما با استفاده از متدهایی در اینستنس کلاس r دستورات Redis را فراخوانی می کنیم. مشاهده می کنیم که نوع شیء برگشتی هم از نوع بایت پایتون است و استرینگ نیست. در اینجا، r.mset() و r.get() را فراخوانی کردیم که با MSET و

GET در API اصلی Redis مطابقت دارند. و HGETALL تبدیل به r.hgetall() و PING تبدیل به r.ping() می شود. ما می توانیم به متد Redis() آرگومان هم بدهیم. پارامتر db شماره پایگاه داده است. و می توانیم چندین پایگاه داده را در Redis به طور همزمان مدیریت کنیم و هر کدام با یک عدد صحیح شناسایی می شوند. حداکثر تعداد پایگاه های داده به طور پیش فرض 16 است. و هنگامی که redis-cli را از خط فرمان اجرا می کنیم، از پایگاه داده شماره 0 شروع می کند.



در ردیس، انواع کلیدهای مجاز redis-py، کلیدهای بایت، str، int یا float هستند که آن ها را به متد هایش پاس می دهیم. برای مثال فرض می کنیم که می خواهیم از تاریخ های تقویم به عنوان کلید استفاده کنیم. که در عکس های زیر مشاهده می کنیم اگر تایپ آن معادل date باشد، با خطا رو به رو خواهیم شد. پس باید به صراحت شی تاریخ پایتون را به str تبدیل کنیم؛ که این کار را می توانیم با isoformat() انجام دهیم. خود Redis فقط string ها را به عنوان کلید مجاز قبول می کند. و redis-py همانطور که در بالا گفته شد، در انواع کلید پایتون که می‌پذیرد کمی آزادتر است، ولی در نهایت همه چیز را قبل از ارسال به سرور Redis به بایت تبدیل می‌کند.

LAB5.py > ...

```python
import redis
import datetime

r = redis.Redis()

a = r.mset({"Croatia": "Zagreb", "Bahamas": "Nassau"})
b = r.get("Bahamas")


today = datetime.date.today()
visitors = {"dan", "jon", "alex"}

c = r.sadd(today, *visitors)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/LAB5.py
Traceback (most recent call last):
  File "/home/hamid/Desktop/LAB5/LAB5.py", line 13, in <module>
    c = r.sadd(today, *visitors)
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/commands/core.py", line 2080, in sadd
    return self.execute_command('SADD', name, *values)
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/client.py", line 1071, in execute_command
    return conn.retry.call_with_retry(
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/retry.py", line 32, in call_with_retry
    return do()
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/client.py", line 1072, in <lambda>
    lambda: self._send_command_parse_response(conn,
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/client.py", line 1050, in _send_command_parse_response
    conn.send_command(*args)
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/connection.py", line 735, in send_command
    self.send_packed_command(self.pack_command(*args),
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/connection.py", line 784, in pack_command
    for arg in map(self.encoder.encode, args):
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/connection.py", line 109, in encode
    raise DataError("Invalid input of type: '%s'. Convert to a "
redis.exceptions.DataError: Invalid input of type: 'date'. Convert to a bytes, string, int or float first.
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

LAB5.py > ...

```python
import redis
import datetime

r = redis.Redis()

a = r.mset({"Croatia": "Zagreb", "Bahamas": "Nassau"})
b = r.get("Bahamas")


today = datetime.date.today()
visitors = {"dan", "jon", "alex"}

stoday = today.isoformat()
print(stoday)

c = r.sadd(stoday, *visitors)
print(c)

d = r.smembers(stoday)
print(d)

e = r.scard(today.isoformat())
print(e)
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/LAB5.py
2021-12-24
0
{b'jon', b'dan', b'alex'}
3
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

حال در ادامه به بررسی یک مثال PyHats.com خواهیم پرداخت.

می خواهیم از Redis برای مدیریت برخی از کاتالوگ محصولات، موجودی، و شناسایی ترافیک برای PyHats.com استفاده کنیم.

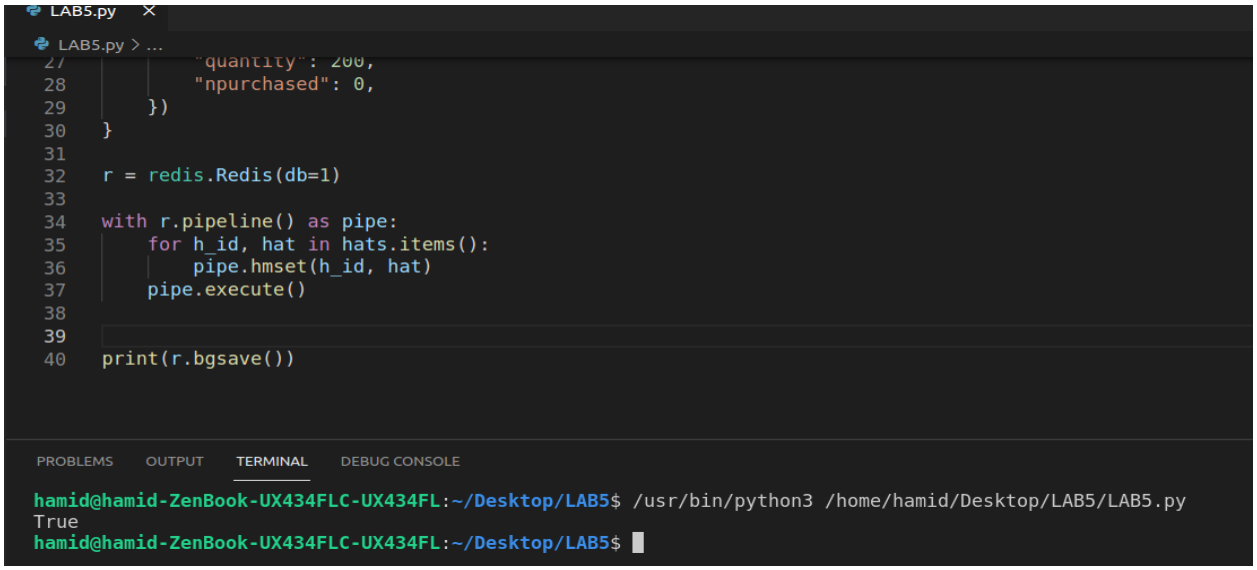
در ابتدا 3 کلاه به صورت ساختار دیکشنری ایجاد می کنیم تا در ادامه از آن استفاده کنیم. این کلاه ها ویژگی هایی مثل color و price و style و quantity و npurchased را در دیکشنری مربوط به خود نگه داری می کنند.

هر کلاه در یک هش Redis از جفت‌های فیلد-مقدار نگهداری می‌شود و هش دارای یک کلید است که یک عدد صحیح تصادفی پیشوندی است، مانند hat:56854717.

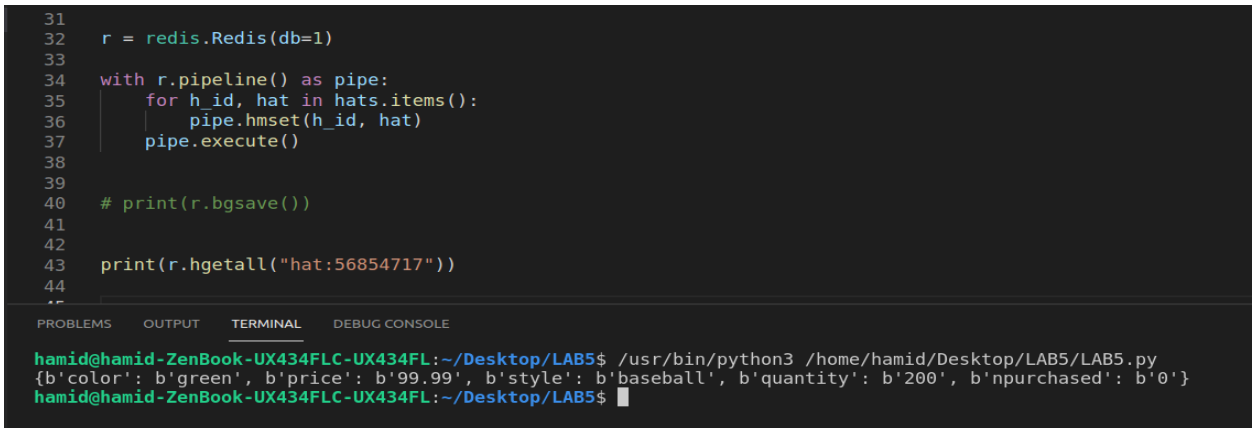
```python
import random

random.seed(444)
hats = {f"hat:{random.getrandbits(32)}": i for i in (
    {
        "color": "black",
        "price": 49.99,
        "style": "fitted",
        "quantity": 1000,
        "npurchased": 0,
    },
    {
        "color": "maroon",
        "price": 59.99,
        "style": "hipster",
        "quantity": 500,
        "npurchased": 0,
    },
    {
        "color": "green",
        "price": 99.99,
        "style": "baseball",
        "quantity": 200,
        "npurchased": 0,
    })
}

print(hats)
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE                                bash

hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/LAB5.py
{'hat:1326692461': {'color': 'black', 'price': 49.99, 'style': 'fitted', 'quantity': 1000, 'npurchased': 0}, 'hat:1236154736': {'color': 'maroon
', 'price': 59.99, 'style': 'hipster', 'quantity': 500, 'npurchased': 0}, 'hat:56854717': {'color': 'green', 'price': 99.99, 'style': 'baseball'
, 'quantity': 200, 'npurchased': 0}}
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

حال در ادامه یک پایگاه داده جدید با شماره آیدی 1 ایجاد می کنیم. حال می خواهیم این محموعه کلاه ها را در دیتابیس ذخیره کنیم. برای نوشتن اولیه این داده ها در Redis، می توانیم از hmset() (هش multi-set) استفاده کنیم و آن را برای هر دیکشنری فراخوانی کنیم. "multi" اشاره ای به تنظیم چند جفت فیلد-مقدار است، جایی که "فیلد" در این مورد با کلید هر یک از دیکشنری ها در کلاه ها مطابقت دارد. می خواهیم از pipe استفاده کنیم و چند دستور را به صورت یک جا اجرا کنیم. برای این کار، کامند های خود را در پایپ اضافه کرده و تمامی دستورات در سمت کلاینت بافر خواهد شد و سپس یک بار متد execute را روی پایپ اجرا می کنیم. با این کار تعداد تراکنش های رفت و برگشتی با سرور ردیس کاهش می یابد.



در ادامه هم برای بررسی صحت کارکرد، با متد hgetall که آرگومانش key یک کلاه است، اطلاعات یک کلاه را بررسی می کنیم.



و در آخر لیست همه کلید ها را چک می کنیم تا مطمئن شویم همه کلاه ها در دیتابیس ذخیره شده اند.

```
31
32  r = redis.Redis(db=1)
33
34  with r.pipeline() as pipe:
35      for h_id, hat in hats.items():
36          pipe.hmset(h_id, hat)
37      pipe.execute()
38
39
40  # print(r.bgsave())
41
42
43  # print(r.hgetall("hat:56854717"))
44
45
46  print(r.keys())
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/LAB5.py
[b'hat:56854717', b'hat:1236154736', b'hat:1326692461']
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

در ادامه، می خواهیم شبیه سازی کنیم که وقتی کاربر روی خرید کلیک می کند چه اتفاقی می افتد و مسئله خریدن کلاه ها را مدیریت کنیم.

بررسی می کنیم که اگر کالا در انبار موجود است، مقدار خرید آن 1 واحد افزایش داده شود و مقدار (موجودی) آن 1 واحد کاهش داده شود. برای این کار از متد hincrby() استفاده می کنیم. با استفاده از این متد مقادیر دو فیلد ذکر شده را تغییر می دهیم و با استفاده از متد hget() صحت آن را می سنجیم.

```
48
49  print(r.hincrby("hat:56854717", "quantity", -1))
50  print(r.hget("hat:56854717", "quantity"))
51
52  print(r.hincrby("hat:56854717", "npurchased", 1))
53  print(r.hget("hat:56854717", "npurchased"))
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/LAB5.py
199
b'199'
1
b'1'
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

برای مدیریت خرید، باید چند بررسی انجام دهیم تا مطمئن شویم همه چیز به درستی رعایت می شود:

مرحله 1: بررسی می کنیم که آیا کالایی که قرار است خریده شود، در انبار موجود است یا خیر. و اگر موجود نبود، یک exception می دهیم. در این مرحله از متد hget() برای بررسی ظرفیت موجودی استفاده می کنیم.

مرحله 2: اگر کالا در انبار موجود است، تراکنش را انجام می دهیم و برای این کار، مقدار ظرفیت کالا را یک واحد کاهش می دهیم و مقدار خریداری شده را یک واحد افزایش می دهیم.

در این مرحله، جفت عملیات های افزایش مقدار خریداری شده و کاهش مقدار موجودی کالا باید به صورت اتمیک اجرا شود؛ یعنی یا هر دو باید با موفقیت اجرا شوند، یا هیچکدام نباید انجام شوند (در صورتی که حداقل یکی از آنها ناموفق باشد).

برای رعایت این مورد در Redis استفاده از transaction block مفید است. در redis-py پایپ لاین به طور پیش فرض یک کلاس پایپ لاین transaction است.

در Redis، یک تراکنش با MULTI شروع می شود و با EXEC به پایان می رسد.

این بدان معنی است که اگر کاهش مقدار موجودی با شکست مواجه شود، پس از آن عملیات افزایش مقدار خریداری شده با شکست مواجه می شود.

مشاهده می کنیم که پس از وارد کردن دستوران HINCRBY خروجی آن ها QUEUED شده و در پایپ لاین قرار گرفته است و پس از وارد کردن EXEC آن ها به صورت اتمیک اجرا شده اند.

```
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ redis-cli
127.0.0.1:6379> MULTI
OK
127.0.0.1:6379> HINCRBY 56854717 quantity -1
QUEUED
127.0.0.1:6379> HINCRBY 56854717 npurchased 1
QUEUED
127.0.0.1:6379> EXEC
1) (integer) -1
2) (integer) 1
127.0.0.1:6379>
```

مرحله 3: حواسمان باید باید که اگر فردی موجودی کالا را گرفت، آن را رزرو کنیم تا فرد دیگری در این مدت آن را خریداری نکند و مشکلی ایجاد نشود. در این مرحله، از یکی از امکانات Redis که قفل optimistic locking نامیده می‌شود، استفاده می کنیم. این قفل به این معنی است که به صورت خوش بینانه، فرض می کنیم همچین مشکل و تداخلی پیش نیامده است. سرور ردیس، در طول زمانی که قفل نگه

داشته است، بر روی تغییرات داده هایی که در حال نوشتن روی آن ها هستیم، نظارت می کند و کار را پیش می برد. اگر در این مدت تداخلی وجود داشته باشد، به سادگی کل فرآیند را دوباره از ابتدا امتحان می کند.

برای این کار می توانیم با استفاده از دستور WATCH و در redis-py از دستور watch() این قفل را اعمال کنیم.

حال یک قطعه کد را بررسی می کنیم.

```python
import logging
import redis

logging.basicConfig()

class OutOfStockError(Exception):
    """Raised when PyHats.com is all out of today's hottest hat"""

def buyitem(r: redis.Redis, itemid: int) -> None:
    with r.pipeline() as pipe:
        error_count = 0
        while True:
            try:
                # Get available inventory, watching for changes
                # related to this itemid before the transaction
                pipe.watch(itemid)
                nleft: bytes = r.hget(itemid, "quantity")
                if nleft > b"0":
                    pipe.multi()
                    pipe.hincrby(itemid, "quantity", -1)
                    pipe.hincrby(itemid, "npurchased", 1)
                    pipe.execute()
                    break
                else:
                    # Stop watching the itemid and raise to break out
                    pipe.unwatch()
                    raise OutOfStockError(
                        f"Sorry, {itemid} is out of stock!"
                    )
            except redis.WatchError:
                # Log total num. of errors by this user to buy this item,
                # then try the same process again of WATCH/HGET/MULTI/EXEC
                error_count += 1
                logging.warning(
                    "WatchError #%d: %s; retrying",
                    error_count, itemid
                )
    return None
```

در این قطعه کد یک اکسپن تعریف می کنیم تا اگر موجودی ظرفیت کالا صفر بود، از آن استفاده شود.

در خط 16 با pipe.watch(itemid) استفاده شده است که به Redis می گوید آیتم داده شده را برای هر گونه تغییر در مقدار آن نظارت کند.

برنامه موجودی را از طریق صدا زدن r.hget(itemid, "quantity")، در خط 17 بررسی می کند.

اگر موجودی در این فاصله زمانی که کاربر موجودی کالا را بررسی می‌کند و سعی می‌کند آن را بخرد کم شود، Redis یک خطا را برمی‌گرداند و redis-py یک WatchError (خط 30) ایجاد می‌کند.

یعنی، اگر هر یک از هش های اشاره شده توسط itemid بعد از فراخوانی hget() اما قبل از فراخوانی های hincrby() بعدی در خطوط 20 و 21 تغییر کند، آنگاه کل فرآیند را دوباره اجرا می کنیم.

ما برای استفاده از پایپ، همه دستورات را در یک بافر قرار می دهیم و سپس آن ها را در یک درخواست به سرور ارسال می‌کنیم. ما نمی‌توانیم نتایج را بلادرنگ پس از آن که دستورات که در transactional pipeline اضافه می کنیم، دریافت کنیم.

در نهایت، اگر موجودی در صفر باشد، شناسه کالا را حذف می‌کنیم و یک OutOfStockError (خط 27) ایجاد می‌کنیم. و watch را از روی پایپ بر می داریم.

در ادامه می خواهیم این متد پیاده سازی شده را تست کنیم.

برای این کار 3 خرید از یک کلاه انجام می دهیم. و مشاهده می کنیم یکی از ظرفیت کلاه ها کاسته شده و یکی به تعداد فروش رفته آن کلاه افزوده شده است.

```
39
40  r = redis.Redis(db=1)
41
42  buyitem(r, "hat:56854717")
43  buyitem(r, "hat:56854717")
44  buyitem(r, "hat:56854717")
45  print(r.hmget("hat:56854717", "quantity", "npurchased"))  # Hash multi-get
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ python3 LAB5-1.py
[b'187', b'13']
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

در این بخش با یک حلقه کل کلاه های از یک نوع را خریداری می کنیم. که مشاهده می شود مقدار موجودی آن کلاه برابر صفر شده است و ظرفیت فروش رفته آن کلاه برابر ظرفیت موجودی اولیه آن شده است.

```
47
48  # Buy remaining 196 hats for item 56854717 and deplete stock to 0
49  for _ in range(196):
50      buyitem(r, "hat:56854717")
51  print(r.hmget("hat:56854717", "quantity", "npurchased"))
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

/usr/bin/python3 /home/hamid/Desktop/LAB5/LAB5-1.py
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/LAB5-1.py
[b'0', b'200']
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

حال اگر یک بار دیگر آن کلاه را بخریم با اکسپشن زیر رو به رو می شویم چون موجودی آن کلاه، به پایان رسیده بود.

```
53
54  buyitem(r, "hat:56854717")
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

/usr/bin/python3 /home/hamid/Desktop/LAB5/LAB5-1.py
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/LAB5-1.py
Traceback (most recent call last):
  File "/home/hamid/Desktop/LAB5/LAB5-1.py", line 54, in <module>
    buyitem(r, "hat:56854717")
  File "/home/hamid/Desktop/LAB5/LAB5-1.py", line 27, in buyitem
    raise OutOfStockError(
__main__.OutOfStockError: Sorry, hat:56854717 is out of stock!
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

حال می خواهیم انقضای کلید را در ردیس بررسی کنیم.

هنگامی که یک کلید را منقضی می کنیم، آن کلید و مقدار مربوط به آن به طور خودکار پس از چند ثانیه یا در یک زمان مشخص از پایگاه داده حذف می شود. در redis-py، یکی از راه‌هایی که می‌توانیم این کار را انجام دهیم از طریق setex() است.

در ردیس، روش‌هایی برای به دست آوردن طول عمر باقی‌مانده (زمان تا زنده بودن) کلیدی که تنظیم کرده‌ایم تا منقضی شود، وجود دارد. مانند دو متد ttl() و pttl().

با متد expire() می توانیم در لحظه کلید را منقضی کنیم. و متد get() با آرگومان "runner"، اگر دیگر مقداری را بر نگرداند، آن گاه کلید منقضی شده است. و متد exists() با آرگومان "runner" اگر 0 برگرداند، یعنی کلید منقضی شده است.

```
hamid@hamid-ZenBook-UX434FLC-UX434FL:~$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>> r = redis.Redis(db=1)
>>> from datetime import timedelta
>>> r.setex("runner", timedelta(minutes=1), value="now you see me, now you don't")
True
>>> r.ttl("runner")
40
>>> r.pttl("runner")
30036
>>> r.get("runner")  # Not expired yet
b"now you see me, now you don't"
>>> r.expire("runner", timedelta(seconds=3))  # Set new expire window
False
>>> r.get("runner")
>>> r.exists("runner")  # Key & value are both gone (expired)
0
>>>
```

حال از این قابلیت تمرین شده، در ادامه استفاده خواهیم کرد.

برخی از کاربران ممکن است با استفاده از ربات ها، برای خرید چندین کالا در عرض چند ثانیه، تعداد بسیار زیادی درخواست ارسال کنند؛ که برای سلامت طولانی مدت تجارت خوب نیست. پس باید از آن جلوگیری کنیم.

ما قصد داریم یک کلاینت Redis ایجاد کنیم که به عنوان یک ناظر عمل می کند و لیستی از آدرس های IP ورودی را پردازش می کند، که به نوبه خود ممکن است از چندین اتصال HTTPS به سرور وب سایت باشد.

هدف ناظر این است که جریانی از آدرس‌های IP را از منابع متعدد نظارت کند و در مدت زمان بسیار کوتاهی به دنبال پیدا کردن جریان درخواست‌های زیاد از یک آدرس واحد باشد تا آن را بلاک کند.

یک دیتابیس با آیدی 5 ایجاد می کنیم. و یک لیست ips در این دیتابیس ایجاد می کنیم تا ip های ورودی را ذخیره کند. و حداکثر تعداد درخواست مجاز از یک آدرس واحد را 15 قرار می دهیم.

```
>>> r = redis.Redis(db=5)
>>> r.lpush("ips", "51.218.112.236")
1
>>> r.lpush("ips", "90.213.45.98")
2
>>> r.lpush("ips", "115.215.230.176")
3
>>> r.lpush("ips", "51.218.112.236")
4
>>>
```

حال با قطعه کد زیر سعی می کنیم افراد مزاحم را تشخیص دهیم.

در ابتدا در یک حلقه بی نهایت، به صورت blocking از لیست ips ها با blpop می خوانیم. استفاده از blpop() (یا دستور BLPOP) بدین صورت است که تا زمانی که یک مورد در لیست موجود باشد مسدود می شود و اگر موردی در لیست بود آن گاه پاسخ می دهد.

ipwatcher مانند یک مصرف‌کننده عمل می‌کند، و منتظر است تا IPهای جدید در لیست «ips Redis» قرار بگیرند. آنها را به عنوان بایت دریافت می کند، مانند b"51.218.112.236" و آنها را به یک آدرس مناسب تر با متد ipaddress تبدیل می کند.

سپس با استفاده از آدرس و دقیقه و ساعتی که ipwatcher آدرس ip را دید، یک کلید استرینگ Redis تشکیل می‌دهیم، و تعداد مشاهده این ip را 1 واحد افزایش می‌دهیم. برای این کلید، زمان انقضای 60 ثانیه در نظر می گیریم. چون به اطلاعات آن کلید در آن دقیقه نیاز داریم و پس از آن باید از دیتابیس پاک شود.

اگر تعداد مشاهده آن آدرس بیشتر از MAXVISITS شده باشد، به نظر می رسد که ما یک وب اسکراپر داریم که باید ip آن را بلاک کنیم.

خروجی را در پایان عکس مشاهده می کنیم.

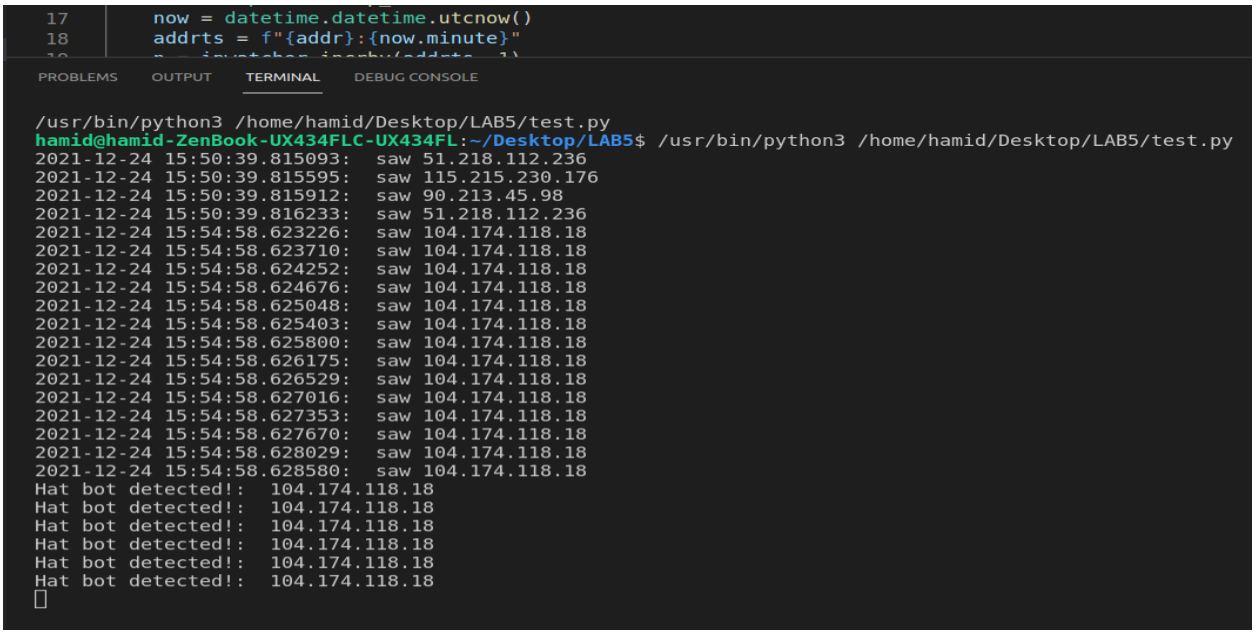
```python
# New shell window or tab

import datetime
import ipaddress

import redis

# Where we put all the bad egg IP addresses
blacklist = set()
MAXVISITS = 15

ipwatcher = redis.Redis(db=5)

while True:
    _, addr = ipwatcher.blpop("ips")
    addr = ipaddress.ip_address(addr.decode("utf-8"))
    now = datetime.datetime.utcnow()
    addrts = f"{addr}:{now.minute}"
    n = ipwatcher.incrby(addrts, 1)
    if n >= MAXVISITS:
        print(f"Hat bot detected!:  {addr}")
        blacklist.add(addr)
    else:
        print(f"{now}:  saw {addr}")
    _ = ipwatcher.expire(addrts, 60)
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

/usr/bin/python3 /home/hamid/Desktop/LAB5/test.py
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/test.py
2021-12-24 15:50:39.815093:  saw 51.218.112.236
2021-12-24 15:50:39.815595:  saw 115.215.230.176
2021-12-24 15:50:39.815912:  saw 90.213.45.98
2021-12-24 15:50:39.816233:  saw 51.218.112.236
```

در ادامه سعی می کنیم که از یک ip یکسان تعداد 20 درخواست به سرور فرستاده شود تا عملکرد آن را بسنجیم.

```
hamid@hamid-ZenBook-UX434FLC-UX434FL:~$ python3
Python 3.8.10 (default, Nov 26 2021, 20:14:08)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import redis
>>> r = redis.Redis(db=5)
>>> for _ in range(20):
...     r.lpush("ips", "104.174.118.18")
...
1
1
2
2
3
4
4
5
6
7
7
8
9
9
10
11
11
12
12
13
>>>
```

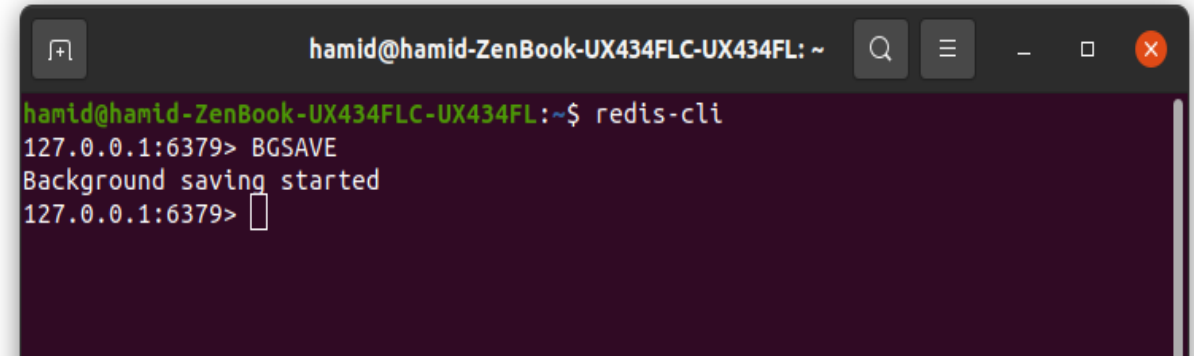
که مشاهده می کنیم پس از 15 درخواست، آن آدرس را ربات تشخیص داده ایم.



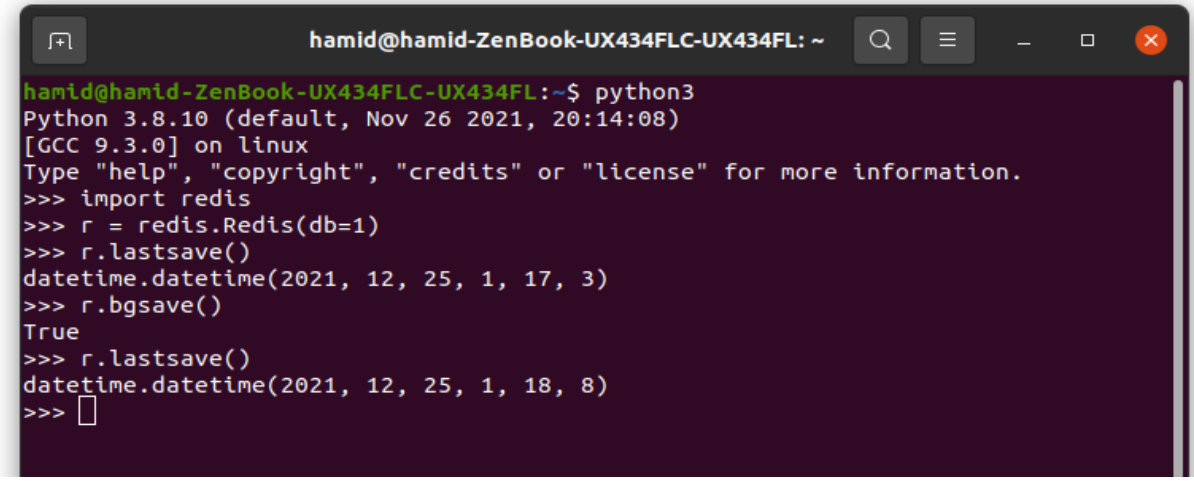
یکی از دلایلی که Redis در عملیات خواندن و نوشتن بسیار سریع است این است که پایگاه داده در حافظه (RAM) روی سرور نگهداری می شود. با این حال، پایگاه داده Redis همچنین می تواند در فرآیندی به نام (Snapshotting) روی دیسک ذخیره شود. این کار باعث نگه داشتن یک نسخه پشتیبان فیزیکی در فرمت باینری است تا بتوان داده ها را بازسازی کرد و در صورت نیاز، مانند هنگام راه اندازی سرور، دوباره در حافظه قرار داد.

فرمت ذخیره سازی به صورت <changes> <seconds> است. ما به Redis می‌گوییم که اگر حداقل یک عملیات تغییر و نوشتن در آن بازه زمانی 60 ثانیه‌ای رخ داد، هر 60 ثانیه پایگاه داده را روی دیسک ذخیره کند.

می توانیم با دستور ردیس BGSAVE یک رکورد را به صورت دستی فراخوانی کنیم. و "BG" در BGSAVE نشان می دهد که عملیات ذخیره سازی در پس زمینه انجام می شود.



در ادامه یک مثال انجام می دهیم. در این جا lastsave() در Redis، آخرین رکورد دیتابیس را برمی‌گرداند که پایتون آن را به عنوان یک شی تاریخ به شما برمی‌گرداند. مشاهده می کنیم که خروجی r.lastsave() در نتیجه ی تغییر r.bgsave() تغییر می کند.



در ادامه می خواهیم به مفهوم سریال سازی بپردازیم.

فرض کنید می خواهیم مقدار هش ردیس را معادل یک جیسون قرار بدهیم. ردیس این مورد را به طور مستقیم نمی تواند مدیریت کند.

در عکس اول مشاهده می کنید که با خطا مواجه شده ایم.

دو روش برای استفاده از داده های تو در تو در ردیس وجود دارد. یا اینکه مقادیر را در یک استرینگ با متدی مانند json.dumps() سریال کنیم. یا اینکه از یک جداکننده در استرینگ های کلید استفاده کنیم.

```
LAB5.py   test1.py   LAB5-1.py   test.py
test1.py > test1.py > ...
1   import redis
2
3   import warnings
4   warnings.filterwarnings("ignore")
5
6   restaurant_484272 = {
7       "name": "Ravagh",
8       "type": "Persian",
9       "address": {
10          "street": {
11              "line1": "11 E 30th St",
12              "line2": "APT 1",
13          },
14          "city": "New York",
15          "state": "NY",
16          "zip": 10016,
17      }
18  }
19
20  r = redis.Redis(db=3)
21
22  r.hmset(484272, restaurant_484272)
```

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

  File "/home/hamid/.local/lib/python3.8/site-packages/redis/retry.py", line 32, in call_with_retry
    return do()
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/client.py", line 1072, in <lambda>
    lambda: self._send_command_parse_response(conn,
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/client.py", line 1050, in _send_command_parse_response
    conn.send_command(*args)
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/connection.py", line 735, in send_command
    self.send_packed_command(self.pack_command(*args),
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/connection.py", line 784, in pack_command
    for arg in map(self.encoder.encode, args):
  File "/home/hamid/.local/lib/python3.8/site-packages/redis/connection.py", line 109, in encode
    raise DataError("Invalid input of type: '%s'. Convert to a "
redis.exceptions.DataError: Invalid input of type: 'dict'. Convert to a bytes, string, int or float first.
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

دو متد json.dumps() و json.loads() معکوس یکدیگر هستند، و به ترتیب برای سریال سازی و جداسازی داده ها استفاده می شوند.

```
6
7   restaurant_484272 = {
8       "name": "Ravagh",
9       "type": "Persian",
10      "address": {
11          "street": {
12              "line1": "11 E 30th St",
13              "line2": "APT 1",
14          },
15          "city": "New York",
16          "state": "NY",
17          "zip": 10016,
18      }
19  }
20
21  r = redis.Redis(db=3)
22
23  print(r.set(484272, json.dumps(restaurant_484272)))
```

```
PROBLEMS   OUTPUT   TERMINAL   DEBUG CONSOLE

/usr/bin/python3 /home/hamid/Desktop/LAB5/test1.py/test1.py
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/test1.py/test1.py
True
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

```
27   from pprint import pprint
28   pprint(json.loads(r.get(484272)))

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

/usr/bin/python3 /home/hamid/Desktop/LAB5/test1.py/test1.py
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/test1.py/test1.py
{'address': {'city': 'New York',
             'state': 'NY',
             'street': {'line1': '11 E 30th St', 'line2': 'APT 1'},
             'zip': 10016},
 'name': 'Ravagh',
 'type': 'Persian'}
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

یکی دیگر از پروتکل های رایج سریال سازی yaml است. ما به صورت کلی یک شی پایتون را به یک بایتستینگ تبدیل می کنیم که در چندین زبان قابل شناسایی و مبادله است.

```
31   import yaml  # python -m pip install PyYAML
32   print(yaml.dump(restaurant_484272))

PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

/usr/bin/python3 /home/hamid/Desktop/LAB5/test1.py/test1.py
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/test1.py/test1.py
address:
  city: New York
  state: NY
  street:
    line1: 11 E 30th St
    line2: APT 1
  zip: 10016
name: Ravagh
type: Persian

hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

حال روش دوم را بررسی می کنیم.

```
22   from collections.abc import MutableMapping
23
24   def setflat_skeys(
25       r: redis.Redis,
26       obj: dict,
27       prefix: str,
28       delim: str = ":",
29       *,
30       _autopfix=""
31   ) -> None:
32       """Flatten `obj` and set resulting field-value pairs into `r`.
33
34       Calls `.set()` to write to Redis instance inplace and returns None.
35
36       `prefix` is an optional str that prefixes all keys.
37       `delim` is the delimiter that separates the joined, flattened keys.
38       `_autopfix` is used in recursive calls to created de-nested keys.
39
40       The deepest-nested keys must be str, bytes, float, or int.
41       Otherwise a TypeError is raised.
42       """
43       allowed_vtypes = (str, bytes, float, int)
44       for key, value in obj.items():
45           key = _autopfix + key
46           if isinstance(value, allowed_vtypes):
47               r.set(f"{prefix}{delim}{key}", value)
48           elif isinstance(value, MutableMapping):
49               setflat_skeys(
50                   r, value, prefix, delim, _autopfix=f"{key}{delim}"
51               )
52           else:
53               raise TypeError(f"Unsupported value type: {type(value)}")
54
55
56   r.flushdb()  # Flush database: clear old entries
57   setflat_skeys(r, restaurant_484272, 484272)
```

این روش، به مسطح کردن دیکشنری تودرتو از طریق روش بازگشتی می پردازد، به طوری که کلید هر رشته به صورت استرینگی به هم پیوسته از کلیدهای مقادیر تودرتو است.

```
58
59  for key in sorted(r.keys("484272*")):  # Filter to this pattern
60      print(f"{repr(key):35}{repr(r.get(key)):15}")
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

/usr/bin/python3 /home/hamid/Desktop/LAB5/test1.py/test3.py
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/test1.py/test3.py
b'484272:address:city'              b'New York'
b'484272:address:state'             b'NY'
b'484272:address:street:line1'      b'11 E 30th St'
b'484272:address:street:line2'      b'APT 1'
b'484272:address:zip'               b'10016'
b'484272:name'                      b'Ravagh'
b'484272:type'                      b'Persian'
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```

```
61
62  print(r.get("484272:address:street:line1"))
```

```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE

/usr/bin/python3 /home/hamid/Desktop/LAB5/test1.py/test3.py
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$ /usr/bin/python3 /home/hamid/Desktop/LAB5/test1.py/test3.py
b'11 E 30th St'
hamid@hamid-ZenBook-UX434FLC-UX434FL:~/Desktop/LAB5$
```