

# Redis

برای نصب redis بر روی ویندوز 10 طبق این [لینک](#) پیش روید.

## Redis چیست؟

Redis یک سرور داده-ساختار باز و متن باز است که به عنوان یک پایگاه داده key-value در حافظه عمل می کند.

### 1. ویژگی های Redis

- Redis یک سرور داده-ساختار است که داده ها را در حافظه نگه می دارد.
- این به معنای عملکرد سریع تر نسبت به پایگاه داده های معمولی است که داده ها را در دیسک ذخیره می کنند.
- Redis یک پایگاه داده key-value است که انواع داده مختلفی را پشتیبانی می کند مانند رشته ها، لیست ها، مجموعه ها و غیره.

### 2. استفاده از Redis

- Redis به طور گسترده ای برای کش کردن داده ها، مدیریت جلسه، شمارنده ها و سایر موارد استفاده می شود.
- همچنین می تواند به عنوان یک صف پیام رسان عمل کند.

### 3. کاربردهای Redis

- کش کردن داده ها
- مدیریت جلسه
- شمارنده ها و محدودیت های سرعت
- صف پیام رسان
- پردازش داده های سریع

این [لینک](#) میتونه خیلی مفید باشه.

## کامندهای ساده redis

کامندهای redis به شرح زیر است:

### set(1)

- تابع `set(name, value, ex=None, px=None, nx=False, xx=False, get=False)`
- کاربرد: برای ذخیره یک مقدار جدید در ردیس استفاده میشه.
- پارامترها:
  - `name` کلید مربوط به مقدار مورد نظر
  - `value` مقدار مورد نظر برای ذخیره
  - `ex` زمان انقضای مقدار بر حسب ثانیه (اختیاری)
  - `px` زمان انقضای مقدار بر حسب میلی ثانیه (اختیاری)
  - `Nx` اگر `True` باشه، مقدار فقط در صورتی ذخیره میشه که کلید وجود نداشته باشه (اختیاری)
  - `xx` اگر `True` باشه، مقدار فقط در صورتی ذخیره میشه که کلید قبلا وجود داشته باشه (اختیاری)
  - `get` اگر `True` باشه، مقدار قبلی کلید قبل از ذخیره بازگردونده میشه (اختیاری)

### get(2)

- تابع `get(name, default=None)`
- کاربرد: برای بازخوانی مقدار یک کلید در ردیس استفاده میشه.
- پارامترها:
  - `name` کلید مورد نظر برای بازخوانی مقدار
  - `default` مقدار پیشفرض اگر کلید وجود نداشته باشه (اختیاری)

از این [لینک](#) می توانید انواع کامندهای معروف redis را مشاهده کنید.

### تمرین اول:

فرض کنید سرویس فروشگاه‌ای داریم و برای هر کدام از وندورهای قرار است یک صورت حساب ایجاد کنیم که فرایند پیچیده‌ای دارد و ما نیاز داریم این مقدار را یک روز در کش نگهداری کنیم تا در منابع صرفه جویی کنیم. فرض کنید دیتاها به این شکل هستند:

**Ali => 1000**

**Alex => 2000**

**Hamid => 1356**

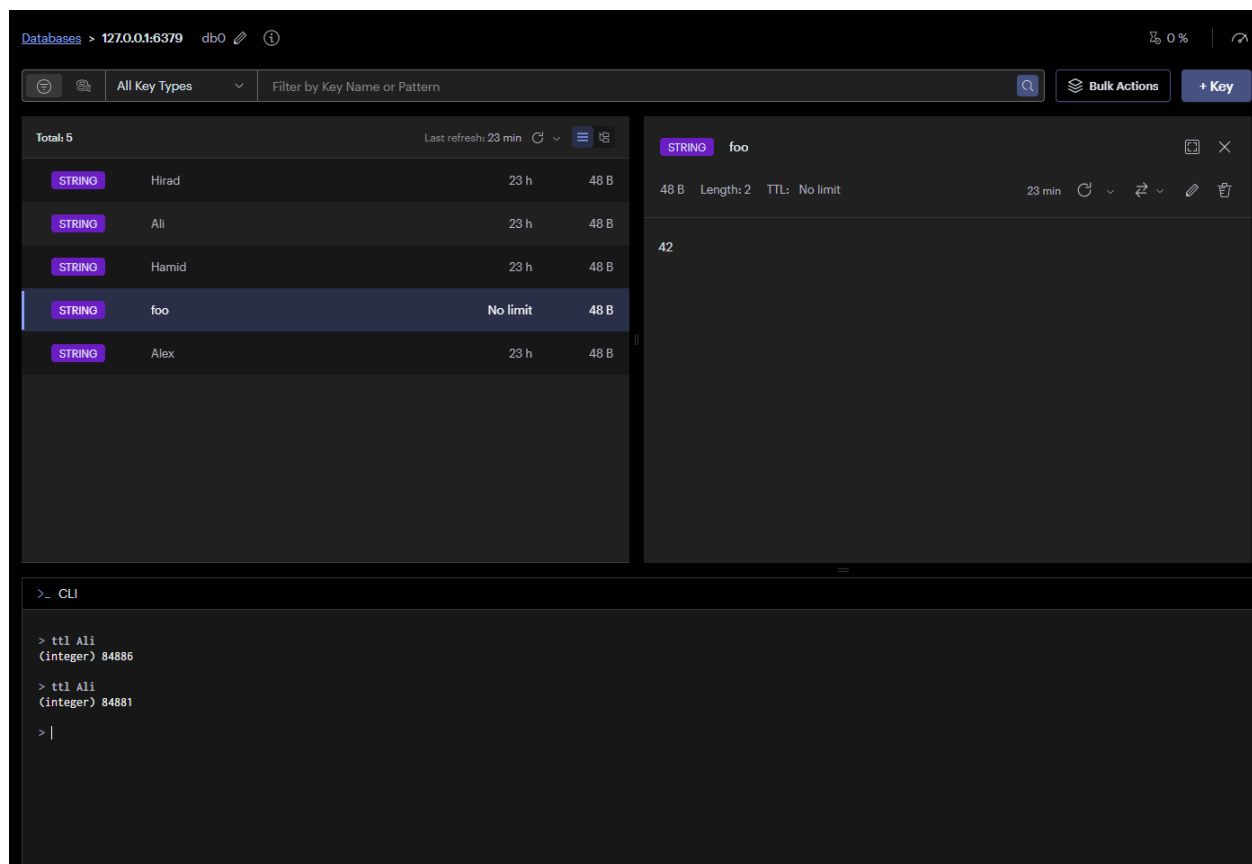
**Hirad => 10180**

این دیتا‌های را با ttl برابر با یک روز روی redis بریزید و بعد سعی کنید مقدارهای ست شده را بخوانید.

پاسخ: در فایل task1.py کد موجود است.

```
import redis
r=redis.Redis(host='127.0.0.1',port='6379',db=0)
data={
    "Ali": 1000,
    "Alex": 2000,
    "Hamid": 1356,
    "Hirad": 10180
}
for key,value in data.items():
    r.set(key,value,ex=86400)
for key in data.keys():
    value=r.get(key)

    if value:
        print(f"{key}:{(int(value))}")
    else:
        print(f"{key}:not found")
```



## نکات جالب درباره redis :

معماری redis کلاینت سرور هست و هم چنین redis یک single thread هست در مقایسه با Memcached که multi thread هست. هم چنین redis TCP را ساپورت میکند ولی UDP را ساپورت نمی کند ولی Memcached هر دو را ساپورت می کند. برای اطلاعات بیشتر [لینک](#) مطالعه کنید.

## پیاده سازی لاک با redis :

هدف لاک این است که اطمینان حاصل شود که در بین چند اینستنس کد که ممکن است سعی کنند کار مشابهی را انجام دهند، فقط یکی از آنها آن کار را در واقع انجام دهد حداقل فقط یکی در یک زمان. این کار ممکن است نوشتن برخی از داده ها در یک سیستم ذخیره سازی مشترک، انجام برخی محاسبات، فراخوانی برخی از API های خارجی یا موارد مشابه باشد. از طرفی باید این نکته را در نظر داشت که ردیس یک دیتابیس core single است و در هر لحظه فقط میتواند یک کامند رو اجرا کند. ما میخواهیم از این ویژگی ردیس برای پیاده سازی تمرین زیر استفاده کنیم.

## تمرین دوم:

فرض کنید یک سرویس داریم که اینستنسهای مختلفی از آن در حال اجراست. یک درخواست مشابه به همه ی پادها ارسال میشود که دیتابیس رو آپدیت کنند، حال میخواهیم با استفاده از ردیس کاری کنیم که اگر پادها درخواست های مشابه ایی همزمان دریافت کردند فقط یکی از پاد کار را انجام دهد. با جستوجو وب راه حل مناسب رو پیدا کنید.

## پاسخ:

راهش استفاده از distributed locker هست. می تونیم از یک کلید redis به عنوان lock استفاده کنیم که هر پاد باید قبل از اینکه هر پاد عملیات آپدیت دیتابیس را انجام دهد، باید این قفل را بگیرد. اگر قفل در دست یکی دیگر از پادها باشد، آن پاد باید عملیات آپدیت را نمی تونه انجام بده. دو لینک زیر اطلاعات خوبی درباره distributed lock میدن:

<https://reintech.io/blog/implementing-distributed-lock-redis>

<https://www.youtube.com/watch?v=2hlxBJa23M0>

## ساختمان داده های مختلف در redis

داده ساختارهای مختلفی در redis داریم که تو تمرین یک از string که ساده ترینش هست استفاده کردیم. مثلا می تونیم از داده ساختار list استفاده کنیم که به صورت زیر است:

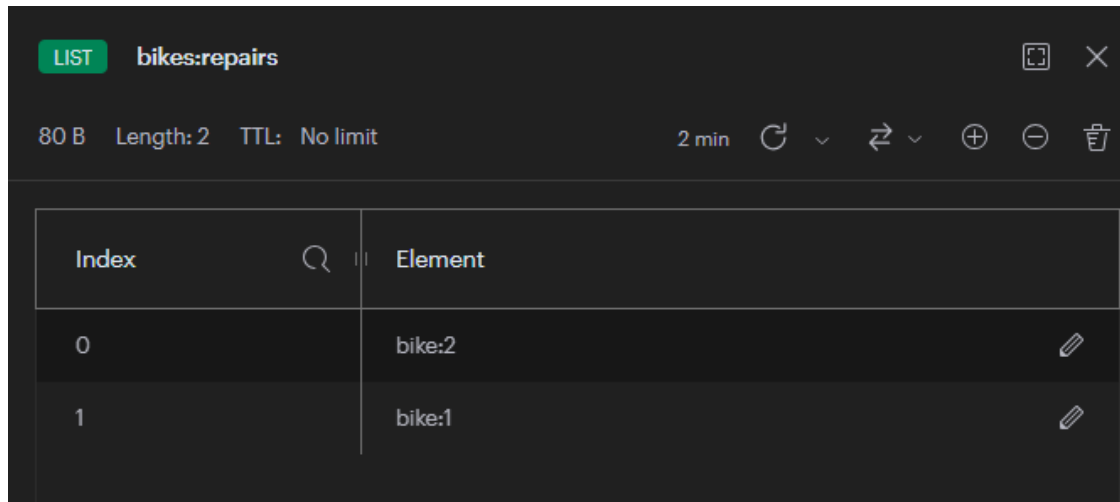
```
import redis

r = redis.StrictRedis(host='127.0.0.1',port='6379',db=0)

res1 = r.lpush("bikes:repairs", "bike:1")
print(res1) # >>> 1

res2 = r.lpush("bikes:repairs", "bike:2")
print(res2) # >>> 2
```

و خروجی نیز به صورت زیر است:



Index	Element
0	bike:2
1	bike:1

انواع داده ساختارهای دیگری هم موجود است:

Sort,sortset,hashes,...

چندتا از داده ساختارها کدش در فایل `datastructure_redis.py` موجود است.

مثلا کاربرد `sorted set` به این صورت هست که ما در یک مسابقه که ممکن هست میلیون ها کاربر داشته باشد و هر دفعه که میخوایم امتیازها رو اپدیت کنیم ممکن هست کلی طول بکشه علاوه بر این ممکن هست بخوایم امتیازات سورت شده داشته باشیم که میدانیم بهترین اردر الگوریتم های سورت برابر با  $O(n \log n)$  است ولی تو `sorted set` هر وقت که المان جدیدی اضافه می کنیم خودش در پوزیشن درست با اردر زمانی  $O(\log n)$  قرار میگیره یا مثلا می خوایم کاربرانی که امتیازاشون بین دو محدوده خاص هست را داشته باشیم جدول زیر زیر به خوبی پیچیدگی زمانی هر کدام از عملیات را توضیح داده است.

	List	HashSet	SortedSet
Iteration	$O(n)$	$O(n)$	$O(n)$
Search	$O(n)$	$O(1)$	$O(n)$
Add	$O(n)$	$O(1)$	$O(\log n)$
Remove	$O(n)$	$O(1)$	$O(\log n)$
Enumerating in Sorted Order	$O(n \log n)$	$O(n \log n)$	$O(n)$
Allow Duplicates	YES	NO	NO

این [لینک](#) خیلی خوب این موارد را توضیح داده است.

Geospatial هم خیلی برای داده های جغرافیایی مناسب هست مثلا می خوایم ادرس هایی در 5 کیلومتری اون محدوده آدرسی ما هست پیدا کنیم. در نوتبوک `restaurant_geospatial.ipynb` من با استفاده از flask و پایتون کد زدم که در ابتدا حدود 50000 مختصات رندوم با فواصل مختلف نسبت به تهران رو در redis اد میکنه و بعد میاد یک نقطه رندوم بهش میدم و بعد 50 تا از نزدیک ترین آدرس های به اون رستوران رو میده.

```
1)b'42575' 51.45257145166397, 35.7531836813866
2)b'45880' 51.45281285047531, 35.75326978190602
3)b'17202' 51.453247368335724, 35.75318867082892
4)b'41345' 51.453488767147064, 35.75327992079065
5)b'21276' 51.45257145166397, 35.75242825448112
6)b'26218' 51.45253390073776, 35.75377672613768
7)b'41034' 51.45191162824631, 35.752699469645165
8)b'30707' 51.45185261964798, 35.75344467766601
9)b'33831' 51.45376235246658, 35.753105025030656
10)b'33573' 51.45180433988571, 35.75267665715473
11)b'27574' 51.45286113023758, 35.75214943515359
12)b'43765' 51.451702415943146, 35.7525600599814
13)b'47335' 51.45159512758255, 35.75266144882777
14)b'42216' 51.45378917455673, 35.7524916225101
15)b'29027' 51.453950107007626, 35.75345988599297
16)b'80' 51.452946960926056, 35.75419242040882
17)b'19095' 51.45209401845932, 35.75205818519185
18)b'2625' 51.45181506872177, 35.753944017734405
19)b'3312' 51.45386427640915, 35.7523695589445
20)b'35236' 51.452700197696686, 35.75190103247997
21)b'47726' 51.452989876270294, 35.75424564955237
22)b'12123' 51.451471745967865, 35.75371082338774
23)b'9750' 51.45415931940079, 35.753371170752395
24)b'7799' 51.451219618320465, 35.75317092778081
25)b'27718' 51.453236639499664, 35.75423297594657
26)b'32962' 51.45385354757309, 35.75215196987475
27)b'40419' 51.454352438449086, 35.752803393212695
28)b'11863' 51.45105332136154, 35.75286422652052
29)b'27861' 51.45329028367996, 35.75176162281621
30)b'24411' 51.454432904720306, 35.75324443469442
31)b'46571' 51.45102649927139, 35.75280085849154
32)b'33516' 51.45289868116379, 35.751600953954665
33)b'28940' 51.454352438449086, 35.75235221284634
34)b'29615' 51.45428270101547, 35.75224321983648
35)b'21668' 51.45183652639389, 35.75455235081264
36)b'17816' 51.45183116197586, 35.751536032633034
37)b'47281' 51.454738676548004, 35.752537274749097
38)b'15554' 51.45066171884537, 35.75260315024111
39)b'45857' 51.45426660776138, 35.754339434235256
40)b'40530' 51.45087093114853, 35.752108879615044
41)b'32892' 51.45053297281265, 35.75282620570314
42)b'10124' 51.45491570234299, 35.752557525260244
43)b'38490' 51.45339757204056, 35.75129776884405
44)b'7654' 51.45208865404129, 35.75121665776695
45)b'16888' 51.45076900720596, 35.754172142638744
46)b'4179' 51.45476549863815, 35.75204297686489
47)b'15792' 51.451230347156525, 35.751536032633034
48)b'42213' 51.45400911569595, 35.754737385457275
49)b'44513' 51.45392328500748, 35.75136620631535
50)b'13020' 51.451992094516754, 35.75119384527652
```

## lua scripts & Pipeline , transactions

برای مطالعه این [لینک](#) مطالعه کنید

### pipeline:

Pipeline به این صورت هست که در ابتدا چندین دستور را در لیست قرار میدیم و به صورت یکجا ارسال میکنیم و بعد redis دستورات را به صورت متوالی اجرا میکنه و در نهایت همه رو با هم بر میگردونه. مزیت اصلی هم این هست که کاهش ping بین کلاینت و سرور redis رو به همراه داره. ولی باید حواسمون باشه که response دستور قبلی به دستور بعدی وابسته نباشه.

### transactions

فرقی که با pipeline داره این هست که تمام دستورات به صورت یکپارچه اجرا میشن ولی در pipeline به صورت مجزا اجرا می شوند. اگر در حین اجرای transaction ، خطایی رخ دهد، تمام تغییرات لغو می شوند و داده ها در وضعیت قبلی باقی می مانند.

## Lua scripts

Redis در Lua Scripts یک قابلیت قدرتمند است که برای اجرای کدهای پیچیده تر در سمت سرور استفاده می شود و به توسعه دهندگان این امکان را می دهد که منطق پیچیده تری را در بستر Redis پیاده سازی کنند.

## تمرین سوم:

فرض کنید می خواهیم ۱۰۰۰ کاربر جدید در ثانیه در ردیس بسازیم. برای این کار اگر به صورت عادی این کار را انجام دهیم به ازای هر ریکوئست ۱ میلی ثانیه زمان صرف می شود و در کل این کار ۱۰ ثانیه طول خواهد کشید اما اگر این کار را به یکباره و با transaction یا pipeline ها انجام دهیم زمان بسیار کمتری صرف خواهد شد. به این صورت که این بار لیست کامند ها را به ردیس می دهیم و ردیس کامند ها را به ترتیب انجام می دهد. فرض کنید ساختار کاربران به این شکل است و سعی کنید کد این برنامه را پیاده کنید.

User-12: hash (

Name: Ali

LastName: Alavi

)

من در نوتبوک task3.ipynb ابتدا بدون pipeline و سپس با استفاده از pipeline یوزرها را اد کردم اگه نگاه کنید بدون pipeline 0.3 ثانیه طول کشیده است و با pipeline 0.1 ثانیه طول کشیده. با transaction نیز اینکار انجام دادم و همین 0.1 ثانیه طول کشید.

## مطالعه:

برای مطالعه ی این موضوع که چطور pipeline پرفورمنس رو بهتر میکند این [لینک](#) رو مشاهده کنید.

برای مطالعه بیشتر نیز درباره امنیت از این [لینک](#) بخونید.



