Hello!

I am Esmaeil Kazemi

I'm interested in learning how are you?

You can find me at @eskazemi

NOSQL

vs

SQL

Redis

Redis stands for Remote Dictionary Server

eskazemi

Map

Redis

1- Features

2- application

3- data type

4- Message Queue

5- Transactions

6- Pipelining

7- Lua Scripts

8- Persistence

9- Benchmarks

10- configuration

11- ACLs

12- Redis Cluster

13- Redis vs Memcached

14- Redis vs Hazelcast

15- Redis vs RDBMS

eskazemi


eskazemi
redis
Transactions
Pipelining
Lua Scripts

Transactions

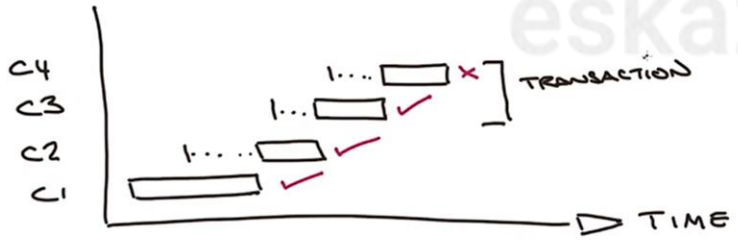
همان طور که می دانیم **transaction** به مجموعه ای از دستورات که قراره پشت سرهم اجرا شوند گفته می شود که که این تراکنش چهار ویژگی دارد :

1. **A**tomic
2. **C**onsistency
3. **i**solation
4. **d**urability

what do transactions mean in a distributed database like Redis?

Redis دستورات را به ترتیبی که از سرور دریافت می کند پردازش می کند هر فرمان دستکاری داده ها atomic است یا به طور کامل انجام می شود یا انجام نمی شود برای تضمین چنین اجرایی Redis دستورات را به صورت single thread اجرا می کند.

Transactions



اگر این اجرا را ترسیم کنیم می توانیم دستورات جدید را هنگام دریافت مشاهده کنیم هچنین انتظار آنها برای اجرا شدن. (منتظر می مانند تا دستور قبلی کامل شود تا آنها اجرا شوند) چندین استثنا داریم برخی از دستورات باعث می شوند که یک فرایند ناهمزمان آغاز شود مثل دستور UNLINK یا دستورات که ممکن است باعث شود یه کلاینت خاص مسدود شود و منتظر بماند با این حال مواردی وجود دارد که نیاز باشد چندین دستور به عنوان یک واحد اجرا شوند به طور مثال دستور 4 و 3 که در نمودار می بینید که دستور 3 اجرا شده است و اما 4 اجرا نشده است که می خواهیم هر دو آنها باهم اجرا شوند و یا هیچ کدام اجرا نشوند در نتیجه این گروه از دستورات باید اتمی باشد. این همان چیزی است که معمولاً به عنوان تراکنش (transaction) شناخته می شود. بنابراین هنگامی که برنامه این دستورات را به عنوان یک عملیات واحد در نظر می گیرد، در این مثال دستور 3 و 4 را در نظر بگیرید. سپس یک تراکنش باید ایجاد شود تا سرور بتواند آنها را به درستی مدیریت کند. این تراکنش مجموعه ای از دستورات را در بر می گیرد که اتمی بودن آنها تضمین شده است.

Transactions

transaction شروع می شود با MULTI <- command و شما باید لیستی از commands هایی که باید اجرا بشوند در تراکنش بهش پاس بدهید و بعد از اتمام آنها با EXEC <- command تراکنش را اجرا کنید.


```
redis 127.0.0.1:6379> MULTI
OK
 List of commands here redis
127.0.0.1:6379> EXEC
```

Transactions

مثال

```
redis 127.0.0.1:6379> MULTI
OK
redis 127.0.0.1:6379> SET tutorial redis
QUEUED
redis 127.0.0.1:6379> GET tutorial
QUEUED
redis 127.0.0.1:6379> INCR visitors
QUEUED
redis 127.0.0.1:6379> EXEC
1) OK
2) "redis"
3) (integer) 1
```

Transactions

دستورات مربوط به آن Transaction و توضیحات آنها

	دستورها	توضیخات
1	DISCARD	Discards all commands issued after MULTI (Throws away queued up, pending commands in the Transaction)
2	EXEC	اجرا کردن همه دستور ها بعد از دستور MULTI نوشته شده است
3	MULTI	شروع یک بلوک transaction را نشان می دهد
4	UNWATCH	Forgets about all watched keys
5	WATCH key [key ...]	کلید ها با استفاده از دستور watch قرنطینه شوند در این حالت اگر کسی از خارج از اون transaction اقدام به تغییر اطلاعات کند transaction متوقف می شود

Transactions

Redis در مورد transaction چگونه فکر می کند ؟

Redis از تراکنش های تو در تو پشتیبانی نمی کند.

تمام دستوارتی که در transaction کپسوله می شوند به صورت متوالی اجرا می شوند و تضمین می کند که تمام دستورات در یک عملیات اجرا می شوند به عبارتی اتمی بودن transaction را تضمین می کند یعنی اگر یکی از دستورات اجرا نشود (اتصال قطع شود) برخلاف تراکنش (transaction) در یک پایگاه داده رابطه ای rollback اتفاق نمی افتد در واقع اگر خطایی در یکی از دستورها رخ دهد، ۲ احتمال وجود دارد:

1- Programing errors

- **Syntax**
- **Operation on incorrect data type**

2- System errors

Transactions

1. اگر خطای نحوی باشد **(System errors)** مانند اشتباه در تعداد آرگومان ها، هنگام صف بندی دستورها تشخیص داده می شود و تراکنش اجرا نخواهد شد.

2. اگر یک خطای معنایی باشد (مانند یک عملیات بر روی نوع داده اشتباه)، تنها در حین اجرای تراکنش تشخیص داده می شود و خطا در داخل لیست پاسخ ها، به عنوان پاسخ برای دستور خاص بازگردانده خواهد شد. اما دستورهای بعدی در صف به صورت عادی اجرا خواهند شد و تراکنش قطع نخواهد شد. این بدان معنی است که Redis مکانیزم rollback مانند RDBMS سنتی ندارد.(برای مثال فرض کنید یک INCR روی کلیدی انجام می دهید که شامل یک نوع داده لیست است. دستور INCR از نظر نحوی صحیح است، فقط زمانی که دستور صف اجرا می شود باعث شکست می شود زیرا نمی تواند بر خلاف نوع داده لیست عمل کند. در این مورد Redis همچنان به اجرای دستورات صف بعدی در تراکنش ادامه خواهد داد.) این به Redis اجازه می دهد تا با حداکثر توان عملیاتی و حداقل تاخیر کار کند.

Transactions

Nested Transactions

Other Databases:

- Commands applied immediately
- System resources consumed

Redis:

- Commands queued
- Applied in a single operation
- No nesting of Transactions

Transactions

محدودیت

با استفاده از transaction ها ما توانایی استفاده از مقادیر میانی برای دستورهای بعدی را نداریم. ما تنها لیست کامل پاسخ ها را در پایان دریافت می کنیم.

شما باید استفاده کنید از transaction ها اگر :

- **نیاز به اجرای atomic دستور ها دارید**
- **برای نوشتن دستورات بعدی به مقادیر میانی نیاز ندارید**

Transactions: Optimistic Concurrency Control

مدیریت تراکنش ها در Redis بسیار ساده است.
Redis اغلب برای انجام صدها هزار یا حتی میلیون ها نوشتن در ثانیه استفاده می شود.
اگر دستورات قبل از اجرای تراکنش در صف قرار گیرند، چه تضمینی دارید که آیا کلید خاصی توسط فرآیند یا برنامه دیگری تغییر کرده است؟ قفل خوش‌بینانه (optimistic locking) یا کنترل همزمان خوش‌بینانه مکانیزمی است که به شما امکان می‌دهد علاقه‌ای به یک شی را مشخص کنید و در صورت تغییر آن شی، اعلان دریافت کنید.

Transactions: Optimistic Concurrency Control

WATCH باید قبل از شروع تراکنش فراخوانی شود، بنابراین باید از قبل درباره کلیدهایی که باید دنبال شوند و تغییرات آن ها را دنبال کنید تصمیم بگیرید.

چند دستور WATCH را می توان قبل از MULTI اجرا کرد. دستورات بعدی WATCH کلیدهای قبلی در حال مشاهده را لغو نمی کند. WATCH به صورت local می باشد و مختص clinet و اتصال فعلی هستند.

یعنی اگر client چندین کلید را watch کرده باشد و شروع به اجرای یه transaction کرده باشد و کلید ها تغییر کرده باشند قبل از اجرا دستور EXEC تراکنش شکست خواهد خورد .

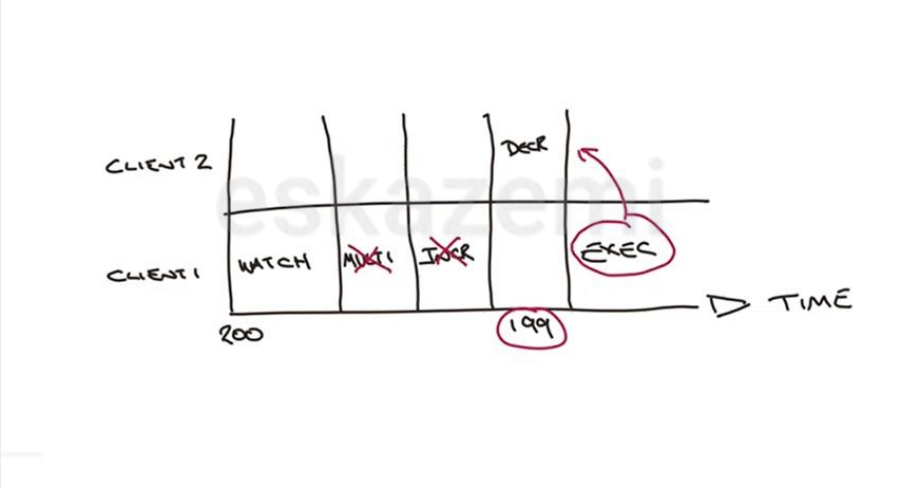
اگر تراکنش به طور متوالی با EXECتکمیل شود، تمام کلیدها به طور خودکار مشاهده نمی شوند. پس از انجام یک تراکنش موفق، نیازی به unwatched کردن کلید ها ندارید به طور خودکار کلیدها unwatched می شوند.

نکته نمی توانید داخل یک تراکنش یک کلید را watch کنید

Transactions: Optimistic Concurrency Control

UNWATCH برای حذف تمام کلیدهای که watched شده اند

Transactions: Optimistic Concurrency Control



همان طور که می بینید client-1 یک کلید را watch کرده است زمانی که مقدار آن 200 می باشد clinet-1 شروع به اجرای یک تراکنش می کند و دستور INCR در صف قرار می دهد اما همزمان client-2 دستور DECR اجرا کرده بر روی کلیدی که watching شده بود، با اجرای تراکنش توسط client-1 تراکنش شکست می خورد چون کلیدی که watching شده بود تغییر کرده است

Transactions: Optimistic Concurrency Control

WHAT IS REDIS PIPELINE

pipeline

کلاینت ها و سرورهای Redis با استفاده از پروتکلی به نام REP مبتنی بر TCP است، با یکدیگر ارتباط برقرار می کنند. در پروتکل مبتنی بر TCP، سرور و کلاینت با یک مدل درخواست / پاسخ ارتباط برقرار می کنند.
کلاینت یک درخواستی به سرور می فرستد، و توسط سرور از طریق سوکت خوانده می شود، سرور که در حال پردازش command (request) می باشد client منتظر پاسخ به روش مسدود کننده است. سرور بعد از پردازش پاسخ را به client بر می گرداند.

هر یک درخواست به Redis ، 250 میلی ثانیه طول می کشد در نهایت در یک ثانیه 4 درخواست توسط Redis پردازش می شود در صورتی که یک سرور Redis توانایی آن را دارد در یک ثانیه 100000 هزار درخواست را پرادزش کند. در نتیجه باید از pipeline Redis استفاده کرد pipeline ها در Redis به شما اجازه میدن که چندین درخواست رو داخل یک پکیج یک جا بفرستید و Redis هم چندین پاسخ را داخل یک پکیج برای شما بفرسته

توجه اگر از pipeline redis استفاده نکنیم دیگر نمی توان از حداکثر توان سرور Redis استفاده کرد و سرعت مون کاهش پیدا می کند.

یکی از ویژگی های کلیدی Redis پشتیبانی آن از pipeline است که اجازه می دهد چندین دستور Redis در یک شبکه رفت و برگشت به سرور ارسال شود. این می تواند عملکرد برنامه را با کاهش سربار ارتباطات شبکه و پردازش سرور، به میزان زیادی بهبود بخشد.

مثال

pipeline

```
$(echo -en "PING\r\n SET tutorial redis\r\nGET tutorial\r\nINCR
visitor\r\nINCR visitor\r\nINCR visitor\r\n"; sleep 10) | nc localhost 6379
+PONG
+OK
redis
:1
:2
:3
```


به طور مثال اینجا یه نمونه pipeline نشان داده شده است در اینجا چندین command رو همزمان اجرا کردیم با استفاده از pipeline ، command هایی همچون ping ، ایجاد یک کلید با مقدار redis و بعد همه کلید ها رو گرفتیم و ...
همان طور که می بینید همه command ها یکبار submit میشه و Redis خروجی همه command ها در یک مرحله بر گردانده است.

مثال در پایتون

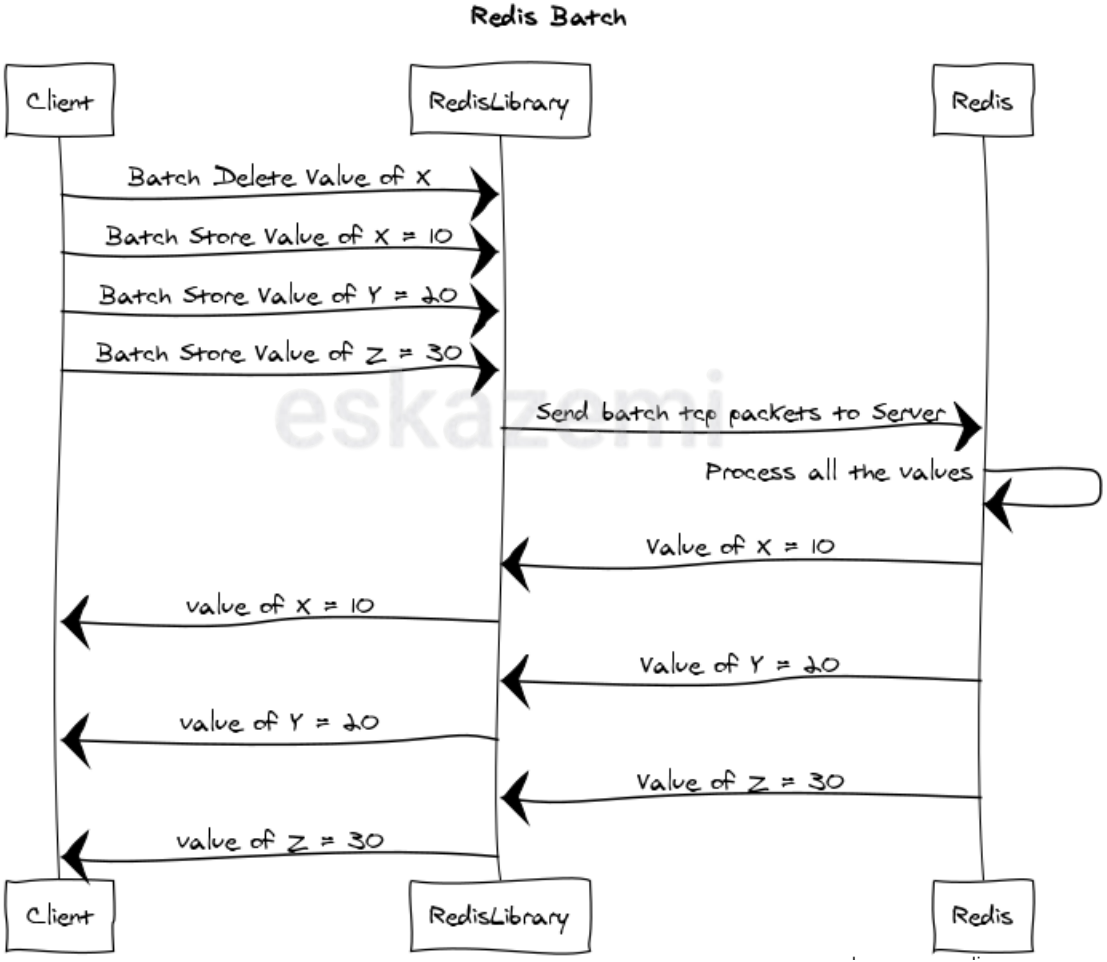
pipeline

```
import redis

conn = redis.Redis()
pipeline = conn.pipeline(transaction=False)
pipeline.get('foo')
pipeline.get('bar')
pipeline.execute()
```


شروع یک redis pipeline و گرفتن مقدار ها در کتابخانه redis مربوط به زبان پایتون

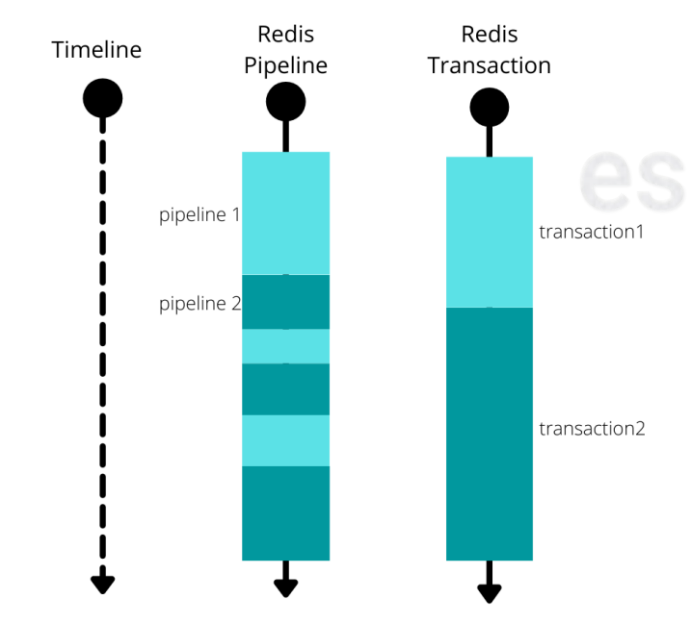
نحوه ی عملکرد pipeline با جزئیات



Transactions vs Pipeline

pipeline

ما دیده ایم که pipeline راهی برای ارسال دسته جمعی دستور ها از سمت کلاینت به سرور است ، همین موضع در مورد Transaction نیر صدق می کند، تفاوت آنها در چیه؟



	Transaction	Pipelines
atomic	✓	✗

به این معنی که 2 تا transaction نمی توانند همزمان اجرا شوند در حالی که چندین pipeline می توانند اجرا شوند به صورت همزمان توسط Redis-server و پایان یابند.

انتخاب اندازه مناسب pipeline

هنگام استفاده از pipeline، مهم است که اندازه pipeline مناسب را برای تعادل عملکرد و استفاده از حافظه انتخاب کنید. اندازه pipeline به تعداد فرمان هایی اشاره دارد که در یک درخواست pipeline به هم متصل شده و به سرور فرستاده می شوند.

Big size : اندازه خط لوله بزرگ تر می تواند با کاهش سربار ارتباط شبکه و پردازش سرور منجر به عملکرد بهتر شود. با این حال، اندازه های خط لوله بزرگ تر همچنین به حافظه بیشتری در سمت client برای نگهداری درخواست های در حال انتظار نیاز دارد و می تواند احتمال از دست رفتن داده ها را در صورت شکست یک درخواست افزایش دهد.

Small size : اندازه خط لوله کوچک تر می تواند استفاده از حافظه را کاهش دهد و قابلیت اطمینان را بهبود بخشد، اما همچنین می تواند به دلیل افزایش سربار ارتباطات شبکه و پردازش سرور منجر به عملکرد کم تر شود.

pipeline

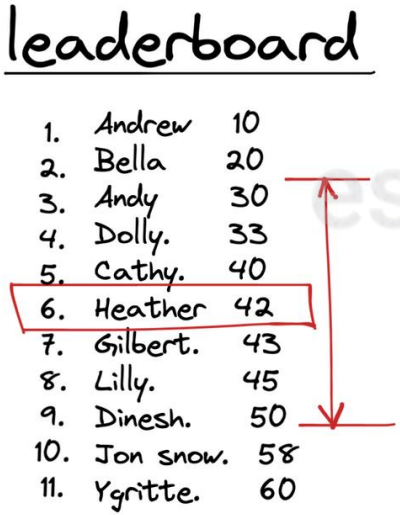
چه زمانی نمی توانیم استفاده کنیم از pipeline

1 – زمانی که نوشتنی به دنبال خواندن باشد امکان پذیر نیست در یک pipeline چون که نتایج همه command ها را باهم در انتها می بینیم. به عبارتی دیگه ابتدا مقدار کلیدی را بخوانیم و بخواهیم از مقدار این کلید یک کلید دیگه با همین مقدار ایجاد کنیم.

2 - زمانی که داریم استفاده می کنیم از pipeline و همچنین از Redis در حالت خوشه ای (cluster) استفاده می کنیم باید مطمئن شویم که کلیدها باید در یک hash slot باشند و نه فقط در همان گره چرا که در حالت خوشه ای Redis اگر تمام کلیدها به یک slot یک سان نگاشت نشوند، Redis ارور برخواهد گردانید (CROSSSLOT Keys in request don't hash to the same slot) با این حال این خطا را می توان با استفاده از hash-tags کنترل کرد. با hash-tags ها می توانیم کلیدها را مجبور کنیم که روی یک same hash slot نگاشت شوند.

Redis Lua Scripting

Lua Scripts



فرض بگیرید که ما یک تابلوی امتیاز داریم که در ZSET نگهداری می شود ، اکنون چیزی که دنبال آن هستیم رتبه یک کاربر می خواهیم به علاوه همسایگان آن کاربر در تابلوی امتیازات . در ظاهر بیان مسئله بسیار آسان به نظر می رسد ، بیایید تابلوی امتیازات خود را تجسم کنیم و ببینیم چگونه می توان این مسئله را حل کرد.

Lua Scripts

راه حل ساده به نظر می رسد اجرا ی دو دستور پشت سرهم است!

1- گرفتن رتبه کاربر با استفاده از دستور زیر و برگرداندن رتبه کاربر (رتبه 5)

```
zrank leaderboard Heather
```

2- برای بدست آوردن همسایگان کاربر می توان 3 کاربر بالایی و پایینی را با دستور بدست آورد

```
zrange leaderboard 2 8
```

Lua Scripts

اجرای دو دستور در Redis client

```
127.0.0.1:6379> zrange leaderboard 0 -1
 1) "Andrew"
 2) "Bella"
 3) "Andy"
 4) "Dolly"
 5) "Cathy"
 6) "Heather"
 7) "Gilbert"
 8) "Dinesh"
 9) "Lilly"
10) "Ygritte"
11) "Jon snow"
127.0.0.1:6379> zrank leaderboard Heather
(integer) 5
127.0.0.1:6379> zrange leaderboard 2 8
1) "Andy"
2) "Dolly"
3) "Cathy"
4) "Heather"
5) "Gilbert"
6) "Dinesh"
7) "Lilly"
127.0.0.1:6379>
```

گرچه در ظاهر راه حل خوبی است ، اما باگ است، زیرا تابلوی امتیازات می تواند می تواند بین صدا زدن zrank و zrange تغییر کند.

راه حلی که می تواند به ذهن خطور کند استفاده از Redis pipeline است ، با این حال ، pipeline را نمی متوان استفاده کرد چون خروجی zrank به عنوان ورودی zrange استفاده کرد، چون که نمی توان هردو را در یک pipeline استفاده کرد .

Lua Scripts

یکی از راه حل هایی که در واقع جواب می دهد، استفاده از نوعی قفل قبل از به دست آوردن رتبه کاربر و سپس استفاده از zrange و سپس برداشتن قفل است. و قبل از نوشتن در تابلوی امتیازات باید قفل را چک کنیم، و اگر قفل وجود دارد، دوباره امتحان کنیم تا قفل برداشته شود. روش قفل کردن این است که مطمئن شوید هیچ نوشتن بین zrank و zrang اتفاق نمی افتد. در حالی که این روش جواب می دهد، ما به وضوح پیچیدگی کل عملیات را افزایش داده ایم.

while reading from redis.

```
take_lock()
    get_rank()
    get_neighbours()
remove_lock()
```

while writing to redis.

```
locked = get_lock();
if not locked:
    write_to_leaderboard()
else:
    keep_retrying()
```

Lua Scripts

یک راه عالی برای حل این مسائل استفاده از **Lua Scripts** است. Redis به ما اجازه می دهد **Lua Scripts** را روی سرور آپلود و اجرا کنیم و از آنجا که اسکریپت ها روی سرور اجرا می شوند، خواندن و نوشتن داده از طریق اسکریپت ها بسیار کارآمد است.

Redis اجرای atomic ← **Lua Scripts** ← تضمین می کند.

در حین اجرای اسکریپت، تمام فعالیت های سرور در طول زمان اجرای آن مسدود می شوند.

بیایید با یک مثال ساده شروع کنیم :

```
> EVAL "return 'Hello, world!'" 0 "Hello, world!"
```

با نوشتن یک Lue Scripts ساده مشکل را حل می کنیم.

Lua Scripts

```
local rank = redis.call('zrank', KEYS[1], ARGV[1]); local min =
math.max(rank - ARGV[2], 0);
local max = rank + ARGV[2];
local ldb = redis.call('zrange', KEYS[1], min, max); return {rank+1, ldb};
```

بیایید خط به خط اسکریپت را بشکنیم:

در خط اول ما گرفتیم رتبه کاربر که KEYS[1] نام تابلوی امتیازات است و ARGV[1] هست username برای اینکه اطلاعات همسایگان کاربر را بدست آوردیم.

خطوط 2 و 3 فقط برای بدست آوردن کران پایین و بالایی برای محدوده cmd هستند. در ARGV[2]، ما در حال دریافت offset برای تابلوی امتیازات هستیم . برای کمرتین مقدار یک کردیم که زیر 0 نرود.

در خط چهارم ما در حال گرفتن leaderboard هستیم

در خط پنجم یک آرایه با رتبه کاربر و همسایگان مقداردهی اولیه کرده و آن را بر گدانده ایم

Lua Scripts

```
127.0.0.1:6379> eval "local rank = redis.call('zrank', KEYS[1], ARGV[1]); loca
l min = math.max(rank - ARGV[2], 0); local max = rank + ARGV[2]; local ldb = r
edis.call('zrange', KEYS[1], min, max); return {rank, ldb};" 1 leaderboard Hea
ther 3
1) (integer) 5
2) 1) "Andy"
   2) "Dolly"
   3) "Cathy"
   4) "Heather"
   5) "Gilbert"
   6) "Dinesh"
   7) "Lilly"
127.0.0.1:6379>
127.0.0.1:6379>
```

Lua Scripts

در حالی که موارد بالا کار می کند، ارسال هر بار این اسکریپت بزرگ از طرف client کارآمد نیست

خوشبختانه Redis داری API هایی است که با آن می توانیم اسکریپت ها روی سرور ذخیره کنیم و به جای اینکه هر بار از سمت client این اسکریپت های بزرگ را دریافت کنیم از آنها استفاده کنیم

لطفا توجه داشته باشید که حافظه پنهان اسکریپت Redis همیشه ناپایدار است و پایدار نیست. حافظه نهان ممکن است هنگام راه اندازی مجدد سرور، در طول شکست زمانی که یک کپی (replica) نقش اصلی را بر عهده می گیرد، یا به صراحت توسط SCRIPT FLUSH این بدان معناست که اسکریپت های حافظه نهان زودگذر هستند و متحویات حافظه نهان (کش) می توانند در هر زمانی از بین برود.

Lua Scripts

دستور زیر برای ذخیره اسکریپت که بر می گرداند یک کلید SHA1 که اختصاصی برای اسکریپت است و بعد از آن از دستور دوم می توان برای فراخوانی اسکریپت استفاده کرد.

```
> SCRIPT LOAD <script>
```

```
> EVALSHA <sha1>
```

بیایید نگاهی بیندازیم به اجرای دو دستور:

Lua Scripts

```
127.0.0.1:6379> script load "local rank = redis.call('zrank', KEYS[1], ARGV[1]
); local min = math.max(rank - ARGV[2], 0); local max = rank + ARGV[2]; local
ldb = redis.call('zrange', KEYS[1], min, max); return {rank, ldb};"
"ce79631a2e3d6313be47a8cd3160a8b8165e4bf1"
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379>
127.0.0.1:6379> evalsha "ce79631a2e3d6313be47a8cd3160a8b8165e4bf1" 1 leaderboa
rd Heather 3
1) (integer) 5
2) 1) "Andy"
   2) "Dolly"
   3) "Cathy"
   4) "Heather"
   5) "Gilbert"
   6) "Dinesh"
   7) "Lilly"
127.0.0.1:6379>
```

Lua Scripts

جمع بندی:

- ✓ از آنجایی که اسکریپت ها در سرور اجرا می شوند ، خواندن و نوشتن داده ها از طریق اسکریپت ها بسیار کارآمد است.محلی بودن داده ها تاخیر کلی را کاهش می دهد و منابع شبکه را ذخیره می کند.
- ✓ **Lua Scripts** را می توان استفاده کنیم در جاهایی که به atomicity نیاز است و از pipeline و Transaction نمی توان استفاده کرد، چرا که Redis ، atomicity بودن **Lua Scripts** را تضمین می کند.
- ✓ دستور EVAL برای اجرای اسکریپت ها استفاده می شود اما ارسال اسگریپت های طولانی کارآمد نیست، در این صورت می توانیم اسکریپت را با استفاده از SCRIPT LOAD <script> روی سرور ذخیره کنیم و سپس از دستور
- ✓ EVALSHA <SHA> برای اجرای اسکریپت استفاده کنیم.
- ✓ حافظه پنهان اسکریپت Redis همیشه فرار است و ثبات ندارد.

How does Redis Lua scripting improve performance?

Redis Lua scripting improves performance by reducing the number of round-trips between the client and the server, as well as by allowing for complex operations to be executed on the server-side.

Thanks!

Any questions?

You can find me at:

- @eskazemi
- m.esmaeilkazemi@gmail.com