The Little Redis Book

by Karl Seguin

The Little Redis Book

Karl Seguin

About This Book

လိုင်စင်

About The Author

Karl Seguin သည် နည်းပညာနှင့်ပတ်သတ်သော ဘာသာရပ်များတွင် အတွေ့အကြုံများစွာရှိသည့် developer တစ်ဦးဖြစ်သည်။ ၎င်းသည် OSS Projects များကို တစ်စိတ်တစ်ပိုင်း contributor တစ်ဦးဖြစ်သည့် အပြင် နည်းပညာအကြောင်း ဟောပြောသူ နှင့် စာရေးသူ တစ်ဦးလည်း ဖြစ်သည်။ ၎င်းသည် redis နှင့်ပတ်သတ်သော article များ၊ tools များကိုလည်း

ရေးသားသူတစ်ဦးဖြစ်သည်။ ၎င်း၏ game developer များအတွက် အခမဲ့ service ကို mogade.com တွင်တွေ့နိုင်ပြီး Redis ကိုအသုံးပြုထားသည်။

Karl သည် The Little MongoDB Book ကိုလည်းရေးသားခဲ့ပါသေးသည်။

သူ၏ blog ကို http://openmymind.net တွင်တွေ့နိုင်ပြီး ၎င်း၏ twitter handle မှာ [@karlseguin](http://twitter.com/karlseguin) ဖြစ်သည်။

ကျေးဇူးတင်လွှာ

သင်၏ မျက်လုံး စိတ်နှင့် ပြင်းထန်သော ဝါသနာများကို ငှားရမ်းမှုအတွက် Perry Neal ကို အထူးပဲကျေးဇူးတင်ရှိရပါတယ်။ သင့်ရဲ့ကူညီမှုက တန်ဖိုးဖြတ်၍မရပါဘူး။ ကျေးဇူးပါ။

Latest Version

စာအုပ်၏ နောက်ဆုံး version ကိုအောက်ပါလင့်တွင် ဖတ်ရှုနိုင်သည်။ http://github.com/karlseguin/the-little-redis-book

နိဒါန်း

ယခုနှစ်ပိုင်းတွင် data များကို တည်ဆောက်ခြင်းနှင့် query ပြုလုပ်သည့် နည်းပညာများသည် အလျင်အမြန်တိုးတက်လာသည်။ ထို့ကြောင့် relational database များသာ အသုံးပြုတော့မည် မဟုတ်ဟု သေချာစွာ ပြောနိုင်သလို data နှင့်ပတ်သတ်သော ecosystem များမှာလည်း အရင်ကတိုင်း တသမတ် တည်းတည်ရှိနေမည် မဟုတ်ပေ။

၎င်း tools အသစ်များနှင့် solution အသစ်များအကြား ကွျန်တော်အတွက်တော့ Redis သည်စိတ်လှုပ်ရှားစရာ အကောင်းဆုံးဖြစ်သည်။ အဘယ့်ကြောင့်ဆိုသော် ၎င်းသည် မယုံကြည်နိုင်လောက်အောင် လေ့လာရလွယ်ကူသည်။ နာရီပိုင်းအတွင်း Redis ကိုအသုံးပြုရသည်မှာ သက်တောင့်သက်တာရှိသည့် အနေအထားတစ်ခုကိုရနိုင်သည်။ ဒုတိယတစ်ခုမှာ ပုံမှန်ဖြစ်နေကြဖြစ်သော ပြဿနာတစ်ချို့တဝက်များကို ရိုးရိုးရှင်းရှင်း ဖြေရှင်းနိုင်သည်။ ဆိုလိုသည်မှာ Redis သည် data နှင့်ပတ်သတ်သည့် အရာအားလုံးကို ဖြေရှင်းရန် ကြိုးစားခြင်းမဟုတ်ပေ။ Redis ကိုသိလာသည့်နှင့်တဖြည်းဖြည်း ၎င်းသည် ဘာနှင့်သက်ဆိုင်ပြီး ဘာနှင့်မဆိုင်ဆိုသည်ကိုပါကွဲပြားလာလိမ့်မည်ဖြစ်ပြီး ထိုသို့ နားလည်ခြင်းသည် developer တစ်ဦးအတွက် ကောင်းမွန်သော အတွေ့အကြုံပင်ဖြစ်သည်။

Redis တစ်ခုတည်းသာ အသုံးပြုပြီး system တစ်ခုလုံးတည်ဆောက်၍ ရသော်လည်း အများစုကတော့ ၎င်းတို့၏ Relational Database ဖြစ်စေ၊ Document အခြေပြု database ဖြစ်စေ တည်ရှိပြီးသား data solution ၏ ဖြည့်ဖက်အဖြစ် အသုံးပြုသည်ကများသည်။ ထိုကြောင့် ၎င်းသည် အထူးပြု features များအတွက် implement ပြုလုပ်ရန်လိုသော solution တစ်ခုဖြစ်သည်။ ထိုသို့ဆိုပါက indexing engine တစ်ခုနှင့် ဆင်တူသည်။ သင့်အနေဖြင့် သင့် application တစ်ခုလုံးကို lucene ပေါ် တင်ထားမည် မဟုတ်။ သို့သော် developer အတွက်ဖြစ်စေ၊ user အတွက်ဖြစ်စေ search အတွက် experience ကောင်းကောင်း လိုအပ်ပါက Redis နှင့် Indexing engines များ၏ ဆင်တူမှုမှာ တခန်းရပ်ပြီဖြစ်သည်။

ယခုစာအုပ်၏ ရည်ရွယ်ချက်မှာ redis ကိုကျွမ်းကျင်ရန် လိုအပ်သော အခြေခံများကို တည်ဆောက်ရန်ဖြစ်သည်။ ကျွန်တော်တို့ Redis ၏ အခြေခံ data structure ငါးခုနှင့် data model ပြုလုပ်သည့် approach များကိုလေ့လာသွားမည်ဖြစ်သည်။ ထိုအပြင် administrative ပြုလုပ်ရာတွင်နှင့် debug ပြုလုပ်ရာတွင် လိုအပ်သော နည်းပညာများကိုပါ ဆက်၍ ပြောပြသွားပါမည်။

အစပျိုး

ကျွန်တော်တို့ တဦးချင်စီ လေ့လာသည် ပုံစံမတူညီကြ။ တချို့မှ စမ်း၍ သင်သည်က အလုပ်ဖြစ်သည်၊ တချို့မှာ video များကြည့်ပြီး သင်ရသည်ကို ကြိုက်သည်။ တချို့မှာ စာဖတ်ခြင်းဖြင့်။ Redis တွင်မူ စမ်းကြည့်သည်က အလုပ်အဖြစ်ဆုံးဖြစ်သည်။ redis သည်သွင်းရသည်မှာ လွယ်ကူပြီး လိုအပ်သည်များကို ဆောင်ရွက်ရန် ရိုးရှင်းသည့် shell interface မှာလည်း ပါဝင်ပြီး ဖြစ်သည်။ ထိုကြောင့် အချိန်အနည်းငယ်ပေးပြီး စက်ထဲ run ၍ရအောင် လုပ်ဆောင်ကြပါစို့။

Windows တွင်

Redis သည် windows ကို တရားဝင် support မလုပ်သော်လည်း အခြားရွေးချယ်စရာနည်းလမ်းများရှိသည်။ ၎င်းတို့ကို production တွင် run ၍ရမည် မဟုတ်သော်လည်း development အတွက် ထူးထူးခြားခြားကန့်သတ်ထားသည်ကို မတွေ့ရပါ။

Microsoft Open Technologies ၏ port တစ်ခုဖြစ်သော repo ကို https://github.com/MSOpenTech/redis တွင်တွေ့ရမည်ဖြစ်ပြီး နောက် solution တစ်ခုကို https://github.com/dmajkic/redis/downloads တွင်တွေ့

ရမည်။ သင့်၏ version ပေါ်မူတည်၍ `64bit` နှင့် `32bit` အကြားရွေးချယ်နိုင်သည်။

MacOSX နှင့် *nix များတွင်

MacOSX နှင့် *nix များအတွက် source မှ build ပြုလုပ်ခြင်းသည် အကောင်းဆုံးဖြစ်သည်။ နောက်ဆုံး version များနှင့် instruction များကို http://redis.io/download တွင်တွေ့နိုင်သည်။ At the time of this writing the latest version is 5.0.7; to install this version we would execute:

```
wget http://download.redis.io/releases/redis-5.0.7.tar.gz
tar xzf redis-5.0.7.tar.gz
cd redis-5.0.7
make
```

ထိုအပြင် Redis သည် package manager အတော်များများတွင်လည်းတွေ့ရှိနိုင်သည်။ ဥပမာ homebrew ရှိသော MacOSX အသုံးပြုသူများအနေဖြင့် `brew install redis` ဟုသွင်းနိုင်သည်။

source မှ build ပြုလုပ်ပါက binary များကို `src` direcotry အောက်တွင်တွေ့ရမည်ဖြစ်သည်။ `src` directory သို့ navigate ပြုလုပ်နိုင်ရန် `cd src` ဟုပြောင်းလိုက်ပါ။

Redis ကိုချိတ်ဆက်ခြင်း

အားလုံး အဆင်ပြေသွားပါက Redis binary များသည် သင့်၏ လက်အောက်တွင်ရှိနေမည် ဖြစ်သည်။ Redis တွင် အသုံးဝင်သည် executable အတော်များများရှိသည်။ ယခုတွင်မူ Redis server နှင့် Command Line Interface ကိုအဓိကထားပြောပြသွားမည်။ ပထမဆုံး server ကိုစတင်ရန် windows ဆိုပါက `redis-server` ကို double click ပြုလုပ်ပါ။ *nix/MacOSX များတွင် `./redis-server` ဟု run လိုက်ပါ။

ပေါ်လာသော message များကိုကြည့်ပါက `redis.conf` ကိုမရှိ ဟုတွေ့ ကောင်းတွေ့ရမည်ဖြစ်သော်လည်း Redis တွင် built-in default ကိုအသုံးပြု မည်ဖြစ်သဖြင့် လက်ရှိအနေဖြင့် အဆင်ပြေပါသည်။

ထိုနောက် windows ဆိုပါက `redis-cli` ကို double click နှိပ်ခြင်းဖြင့် ၊ (*nix/MacOSX) ဆိုပါက `./redis-cli` ဟု run ခြင်းဖြင့် စတင်နိုင်သည်။ ၎င်းသည် local တွင် run နေသည် server ကို default port (6379) မှတဆင့် ချိတ်ဆက်သွားမည်။

အားလုံး အလုပ်သေချာလုပ်မလုပ်ကို `info` ဟုရိုက်ထည့်ပြီး စမ်းသပ် နိုင်သည်။ သင့်အနေဖြင့် server ၏ အခြေအနေများကို ဖော်ပြသော insight များကို key-value pair များအဖြစ်တွေ့ရမည်ဖြစ်မည်။

setup နှင့်ပတ်သတ်ပြီး အခက်အခဲတွေ့ပါက official Redis support group တွင် မေးမြန်းနိုင်သည်။

Redis Driver များ

မကြာမီ လေ့လာရမည့် အတိုင်း Redis ၏ API သည် function များ အနေဖြင့် တည်ရှိသည်။ အလွန်ရိုးရှင်းပြီး Procedural ပုံစံဆန်သည်။ ထို ကြောင့် command line tool များသုံးသည်ဖြစ်စေ ၊ driver မှအသုံးပြုသည် ဖြစ်စေ ခပ်ဆင်ဆင်ပင်ဖြစ်သည်။ ထို့ကြောင့် မည်သည့် programming langauge မှ အလုပ်လုပ်သည် ဖြစ်စေ အခက်အခဲများစွာ တွေ့ရမည် မဟုတ်ပါ။ ၎င်းတို့ကို client page တွင်တွေ့ရှိနိုင်ပြီး အလိုရှိရာ driver များ download ဆွဲနိုင်သည်။

အခန်း (၁) အခြေခံ

Redis သည် ဘာကြောင့် special ဖြစ်တာလဲ ၊ မည်သို့သော ပြဿနာများ ဖြေရှင်းထားသလဲ၊ အသုံးပြုပါက developer များက ဘာကို သတိပြုရ မည်နည်း၊ ထိုမေးခွန်းများကို မဖြေခင် Redis ဆိုသည်မှာ ဘာလဲဆိုသည်ကို နားလည်ရန်လိုသည်။

Redis သည် အများအားဖြင့် in-memory persistent key-value store တစ် ခုအနေဖြင့် သတ်မှတ်ခြင်းခံရသော်လည်း ကျွန်တော်အနေဖြင့် ၎င်းကို မျှတ သည့် သတ်မှတ်ချက်တစ်ခုဟု မထင်။ Redis တွင် data များအားလုံးကို memory တွင် hold ပြုလုပ်ထားနိုင်ပြီး persistence အနေဖြင့် disk ပေါ် တွင် write ပြုလုပ်နိုင်သော်လည်း ၎င်းသည် သာမန် key-value store တစ် ခုထက်ပိုသည်။ ၎င်းထက်ပိုကျော်၍ မတွေးထားပါက redis နှင့်ပတ်သတ် သော သင့်၏ အမြင်နဲ့ ဖြေရှင်းနိုင်သည် များမှာ ကျဉ်းမြောင်းနေမည် ဖြစ်သည်။

လက်တွေ့တွင် Redis သည် မတူညီသော data structure ငါးမျိုးကို ထုတ်ဖော်ပေးပြီး ၎င်းတို့အနက် တစ်မျိုးကသာ ထုံးတမ်းစဉ်လာ key-value structure ဖြစ်သည်။ ၎င်း data structure ငါးမျိုးကို မည်သို့အလုပ်လုပ် သည် မည်သည် method များရှိသည်နှင့် မည်သို့ model များတည်ဆောက် နိုင်သည်ကို နားလည်ခြင်းကသာ Redis ကိုအမှန်တကယ် နားလည်ခြင်း

ဖြစ်သည်။ ပထမဆုံး data structure များ ဖော်ထုတ်သည်ကို တချက်ကြည့်ရအောင်။

အကယ်၍ relational world မှ data structure များကို ပမာပြု၍ ပြောပါက database များသည် data structure တစ်မျိုးတည်သာ ဖော်ထုတ်ပေးသည် ဖြစ်သည်။ ၎င်းမှာ table ဖြစ်သည်။ table များသည် ရှုပ်ထွေးပြီး flexible ဖြစ်သည်။ model လုပ်၍ store ၍ manipulate ပြု၍မရသည် သိပ်များများမရှိ။ သို့သော် ထိုသို့ ဘက်စုံအသုံးပြုနိုင်ခြင်းသည် အားနည်းချက်မရှိသည်သော မဟုတ်။ အထူးသဖြင့် လိုချင်သလောက် မမြန်ခြင်းဖြစ်သည်။ အကယ်၍ အားလုံးသုံး၍ရနိုင်သော data-structure တစ်ခုတည်းထက်စာလျှင် အထူးပြု structure များအသုံးပြုပါက မည်သို့ဖြစ်မည်နည်း။ အကန့်အသတ်ရှိမည်ဖြစ်သော်လည်း ရိုးရှင်းမှုနှင့် speed ကိုရမည်မှာ အမှန်ဖြစ်သည်။

အထူးပြု data strcuture များကို အသုံးပြု၍ အထူး ပြဿနာများကို ဖြေရှင်းခြင်း သည် code ကောင်းကောင်းမွန်မွန် မရေးထားခြင်းကို ဆိုလိုထားသည်လော ၊ data များအတွက် hashtable များ အသုံးမပြုဘူးလား ၊ scalar variable များအသုံးမပြုဘူးလား ဟု မေးစရာရှိသည်။ ကျွန်တော် အတွက်တော့ Redis ၏ approach သည် scalar များ၊ list များ၊ hash နှင့် set များကို အသုံးပြုပါက ၎င်းတို့အတိုင်း ဘာလို့ မသိမ်းသနည်း ကို မေးခွန်းထုတ်ထားခြင်းဖြစ်သည်။ value တစ်ခုတည်ရှိသည်ကို ဆန်းစစ်လို

ပါက အဘယ်ကြောင့် `exists(key)` ဟုသာမခေါ်ပဲ ရှုပ်ထွေးစွာ query ပြုလုပ်နေသနည်း။

အခြေခံအုတ်မြစ်များ

Database များ

Redis သည် သင် ရင်းနှီးနေသော database များ၏ အခြေခံ concept အတူတူပင်ဖြစ်သည်။ database တစ်ခုတွင် data များပါဝင်ပြီး database တစ်ခု၏ အခြေခံအသုံးမှာ application ၏ data များကိုစုစည်းထားကာ အခြား application တစ်ခုဖြင့်ခွဲခြားထားသည်။

Redis တွင် database များကို နံပါတ်အနေဖြင့် တည်ရှိပြီး default database မှာ `0` ဖြစ်သည်။ အခြား database ကိုပြောင်းလိုပါက `select` ကို အသုံးပြုနိုင်သည်။ command line တွင် `select 1` ဟုရိုက်လိုက်ပါက `OK` ဟု reply ပြန်မည်ဖြစ်ပြီး prompt ၏ title သည် `redis 127.0.0.1:6379[1]>` ဟုပြောင်းသွားလိမ့်မည်။ default database ကိုပြန်ပြောင်းလိုပါက `select 0` ဟုရိုက်ပြီး ပြောင်းနိုင်သည်။

Commands၊ Keys နှင့် Values

redis သည် key-value store ထက်ပိုသော်လည်း ၎င်း၏ အခြေခံဖြစ်သည့် data structure ငါးမျိုးသည် key တစ်ခုနှင့် value တစ်ခုစီပုံစံဖြစ်သည်။ ထို့ကြောင့် အခြားအရာများကိုမလေ့လာခင် key နှင့် value များအကြောင်းကို အရင်ဆုံးလေ့လာသင့်သည်။

Key များသည် data များကို identify ပြုလုပ်ရန်ဖြစ်ပြီး ၎င်းကိုအများဆုံး အသုံးပြုကြသော်လည်း ယခုမူ key များ၏ပုံစံကို `users:leto` ဟု သိထားရန်လိုသည်။ ထိုသို့ဖြင့် user တစ်ဦး၏ အမည်ဖြစ်သော `leto` ကိုသိမ်းထားသည် ဟုသိနိုင်သည်။ ထို column များသည် redis အနေဖြင့် မည်သို့မျှ ထူးခြားသည့် အဓိပ္ပါယ်မရှိသော်လည်း မြင်သာအောင် အသုံးပြုသည့် ပုံစံတစ်ခုဖြစ်သည်။

value များသည် key များညွှန်းဆို အမှန်တကယ်တည်ရှိသည့် data များဖြစ်သည်။ ၎င်းတို့သည် အမျိုးစုံဖြစ်နိုင်သည်။ တခါတရံ string များ ၊ တခါတရံ integer ၊ တခါတရံ serialized object များ (JSON,XML နှင့် အခြားသော format များ) အချိန်တော်တော်များများတွင် redis ၎င်း value များကို byte array အဖြစ်သိမ်းထားမည်ဖြစ်ပြီး ဂရုမစိုက်ပါ။ သတိပြုရန်မှာ မတူညီသော driver များသည် serialization ကို ကွဲပြားစွာ handle လုပ်မည်ဖြစ်ပြီး ထိုကြောင့် ယခုစာအုပ်တွင်မူ string ၊ integer နှင့် JSON များကိုသာဖော်ပြသွားမည်။

စမ်းကြည့်ကြပါစို့။ အောက်ပါအတိုင်းရိုက်ထည့်ကြည့်ပါ။

```
set users:leto '{"name": "leto", "planet": "dune", "likes": ["spice"]}'
```

၎င်းသည် redis command တစ်ခု၏ အခြေခံပုံစံဖြစ်သည်။ ပထမဆုံးတည်ရှိမည်မှာ command ဖြစ်ပြီး ယခုကိစ္စတွင် `set` ဖြစ်သည်။ `set`

command တွင် parameter နှစ်ခုပါဝင်ပြီး set ပြုလုပ်မည့် key များနှင့် value များဖြစ်သည်။ အကုန်မဟုတ်သော်လည်း တချို့ command များတွင် key တစ်ခုကိုယူလေ့ရှိသည်။ (ထိုသို့ပြုလုပ်ပါက ပထမ parameter အနေဖြင့် ဖြစ်သည်) အပေါ်မှ ဥပမာကို မည်သို့ retrive ပြန်လုပ်မည်နည်း။ အောက်ကအတိုင်းဖြစ်သည်။

```
get users:leto
```

တခြားပြောင်း၍ စမ်းကြည့်ကြပါ။ key နှင့် value များသည် အခြေခံ concept များဖြစ်ပြီး `get` နှင့် `set` သည် အရိုးရှင်းဆုံး စမ်းကြည့်၍ရနိုင်သည်။ user တစ်ခုဆောက်ကြည့်၊ အမျိုးမတူသော key များ နှင့် value များကို ထည့်ကြည့်ကြပါ။

Query ပြုလုပ်ခြင်း

ရှေ့ဆက်သွားပါက နှစ်ခုမှာ ပို၍ရှင်းလင်းလာမည်ဖြစ်ပြီး redis အနေဖြင့် key သည်အရာ အားလုံးဖြစ်ပြီး value မှာဘာမှသုံးမရပါ။ တနည်းအားဖြင့် redis တွင် object ၏ value ဖြင့် query ပြုလုပ်၍ရမည်မဟုတ်။ အပေါ်မှ ဥပမာအရ `dune` ဂြိုလ်တွင်နေထိုင်သော user များကိုရှာ၍ရမည်မဟုတ်။

တော်တော်များများ ၎င်းသည် အကျပ်ရိုက်စရာဖြစ်ပါလိမ့်မည်။ ကျွန်တော်တို့အသုံးပြုနေသော data query ပြုလုပ်နိုင်သည်များသည် ပို၍ flexible ဖြစ်ပြီး powerful ဖြစ်သဖြင့် Redis ၏ approach သည် ယုတ္တိမတန်သလို

နှင့်ရှေးကျသလိုဖြစ်နေသည်။ သို့သော် ၎င်းကို စိတ်မပူပါနှင့်။ သတိပြုရ မည်မှာ redis သည် အားလုံးကိုဖြေရှင်းနိုင်သော solution မဟုတ်ပါ။ တချို့ သောအရာများသည် ၎င်းနှင့်သက်ဆိုင်သည်အရာများ မဟုတ် (query လုပ် ရသည်မှာ အကန့်အသတ်ရှိသဖြင့်) ထိုအပြင် အချို့သောအချိန်မှာတွင် သင့် data ကို model ပြုလုပ်ရန် နည်းလမ်းအသစ်များကို ရှာဖွေတွေ့ရှိပါ လိမ့်မည်။

ပို၍ ခိုင်မာသော example များကို နောက်ပိုင်းတွင်တွေ့လာရမည်ဖြစ်ပြီး အဓိက နားလည်ရမည်မှာ ၎င်းသည် redis ၏ ထုံးစံဖြစ်သည်။ ထို့ကြောင့် value များသည် အားလုံးဖြစ်နိုင်ကြောင်း နှင့် redis ကို ဖတ်နိုင် ရန် သိထားရန် မလိုကြောင်း နားလည်ရန်လိုမည်။ ထိုအပြင် ယခု ပုံစံအသစ် အတိုင်း model ပြုလုပ်ရန် စေ့ဆော်ပေးသည်။

Memory နှင့် Persistence

Redis သည် memory အပေါ်ရှိ persistent store တစ်ခုဖြစ်ကြောင်းသိရှိပြီး ဖြစ်သည်။ persistence အကြောင်းဆိုပါက default အနေဖြင့် redis သည် key များမည်မျှ ပြောင်းလဲသည်ကို စောင့်ကြည့်နေပြီး database ကို snapshot ပြုလုပ်သည်ဖြစ်သည်။ သင့်အနေဖြင့် key များမည်မျှပြောင်းလဲ သွားသည်ကို ဖြစ်စေ ၊ ဘယ်လောက်စက္ကန့်အတွင်း snapshot ပြုလုပ်မည် ကို configure ပြုလုပ်နိုင်သည်။ Redis သည် default အနေဖြင့် database

အတွင်း စက္ကန့် ၆၀ အတွင်း key အခုတစ်ထောင်ပို၍ ပြောင်းလဲပါက save မည်ဖြစ်သလို key ၉ ခုအောက် ပြောင်းပါက ၁၅ မိနစ်တစ်ခါ save မည် ဖြစ်သည်။

Snapshot ပြုလုပ်သည့်အပြင် redis သည် append mode အနေဖြင့် လည်း run နိုင်သည်။ key တစ်ခုပြောင်းပါက append ပြုလုပ်ထား သော file ကိုပါ disk ပေါ်တွင် update ပြုလုပ်ပေးသည်။ အချို့အချိန်များ တွင် hardware သို့မဟုတ် software failure များဖြစ်ပါက စက္ကန့် ၆၀ အတွင်း ဖြစ်ပေါ်နေသော data များကို ဆုံးရှုံးသည်ကို လက်ခံနိုင်သည့် အနေအထားရှိပြီး performance ကပို၍ အရေးပါသည်။ အချို့အချိန်များ တွင်မူ ထိုဆုံးရှုံးမှုများကို လက်မခံနိုင်ပါ။ Redis အနေဖြင့် ရွေးချယ်စရာ များပေးထားပြီး အခန်း (၆) တွင်မူ တတိယ နည်းလမ်းဖြစ်သည့် slave ဆီ သို့ persistence offload ပြုလုပ်သည်ကို တွေ့ရမည်။

Memory အနေဖြင့်ကြည့်ပါ Redis တွင်ရှိသမျှ အချက်အလက်အားလုံးကို memory တွင်ထားမည်ဖြစ်သည်။ Redis ကို runခြင်း ၏ သိသာထင်ရှား သော ကုန်ကျစရိတ်မှာ Memory ဖြစ်ပြီး RAM သည် server hardware များအတွင်း ဈေးအကြီးဆုံးဖြစ်သည်။

အချို့ developer များသည် space နည်းနည်းဖြင့် data များကို သိမ်းဆည်းရသည်ကို မေ့လျော့ကုန်သည်ဟုထင်ရသည်။ ဝီလီယံ ရှိတ်စပီ

ယား၏ ဝတ္ထုအားလုံး စုစုပေါင်းသည် 5.5MB ခန့်သာ ကုန်ကျမည် ဖြစ်သည်။ Scaling အတွက်မူ တခြား solution များသည် IO သို့မဟုတ် CPU အပေါ်မူတည်နေသည်။ RAM သို့မဟုတ် IO ၏ limitation များမှာ မည်သည့် data အမျိုးအစားနှင့် မည့်ကဲ့သို့ သိမ်းဆည်းမည်နှင့် query ပြုလုပ်မည့် အပေါ်မူတည်သည်။ Redis အတွင်းတွင် အင်မတန် ကြီးမားသော multimedia file များကို သိမ်းဆည်းခြင်းမှ အပ memory ပြဿနာ ပုံမှန်အားဖြင့် အသေးအဖွဲကများသည်။ တချို့ app များတွင် CPU ကိုမူတည်ခြင်းထက် Memory မူတည်ခြင်းဖြင့် လဲလှယ်ရသည်က ပို အဆင်ပြေသည်။

Redis အနေဖြင့် virtual memory အသုံးပြုမှုကို support ပြုလုပ်ခဲ့ သော်လည်း ထို feature ကို Redis ၏ developer များကိုယ်တိုင် failure အဖြစ်မြင်ခဲ့ကြပြီး ၎င်းအသုံးပြုမှုကို ရပ်တန့်ခဲ့သည်။

(5.5MB ရှိသော Shakspear ၏စာမူအားလုံးကို ချုံ့လိုက်ပါက 2MB အထိ ရနိုင်သည်။ Rdis များ အလိုအလျောက် ချုံ့မပေးသော်လည်း ၎င်း၏ byte တစ်ခုချင်းဆီကို တန်ဖိုးထားသော်ကြောင့် compress နှင့် decompress ပြုလုပ်ခြင်းဖြင့် procession time နှင့် RAM ကိုလဲလှယ် နိုင်သည်)

ပြန်လည်စုစည်းခြင်း

ကျွန်တော်တို့ အပေါ်စီးမှ topic များအကြောင်း စတင်ထိတွေ့ခဲ့ပြီးပါပြီ။ Redis ကို အဓိက မလေ့လာခင် ထို အကြောင်းအရာများကို ပြန်လည် စုစည်းရန်လိုသည်။ အထူးသဖြင့် query ၏ အကန့်အသတ်များ ၊ data structure နှင့် Redis ၏ memory အတွင်းတွင် data သိမ်းဆည်းခြင်းတို့ ဖြစ်သည်။

ထိုအကြောင်းအရာသုံးခုကို စုစည်းလိုက်ပါက သင့်အနေဖြင့် ပိုပြီး ကောင်းမွန်သော ရလဒ် ဖြစ်သည် speed ကိုရမည်ဖြစ်သည်။ တချို့သူများ အနေဖြင့် “Redis ကမြန်မှာပေါ့ Memory ပေါ်မှာပဲ အားလုံးအလုပ်လုပ်တာပဲ” ဟုဆိုကောင်းဆိုနိုင်မည် ဖြစ်သောလည်း ၎င်းသည် အကြောင်းအရာတစ်ခုသာဖြစ်သည်။ တကယ့် Redis သည် အခြား solution များအကြား ထင်ပေါ်နေသည်မှာ ၎င်း၏ အထူးပြုထားသော data structure များကြောင့်ဖြစ်သည်။

ဘယ်လောက်မြန်သလဲ? ၎င်းသည် အရာတော်တော်များများ ပေါ်မူတည်နေသည် မည်သည့် command များကိုအသုံးပြုသည်၊ မည်သည့် data အမျိုးအစား၊ နှင့် အခြားကိစ္စများဖြစ်သည်။ သို့သော် Redis ၏ performance သည် **တစက္ကန့်** ကို သောင်းနဲ့ သိန်းနဲ့ချီပြီး run နိုင်ပြီး သင့်အနေဖြင့် (`redis-server` နှင့် `redis-cli` များရှိသော folder အတွင်းတွင် တည်ရှိသည့်) `redis-benchmark` ကိုအသုံးပြု၍ စမ်းသပ်နိုင်သည်။

ပုံမှန် project တစ်ခုကို redis သို့ပြောင်းဖူးသည်။ load test တစ်ခုတွင် relational model ကိုအသုံးပြုပါက ၅ မိနစ်ခန့် ကြာသည်။ Redis တွင် ၁၅၀ မီလီစက္ကန့်သာကြာသည်။ ယခုလိုမျိုး အဆများစွာ ကွာခြားမည်ဟု အမြဲတမ်းယုံကြည်နိုင်မည် မဟုတ်သော်လည်း အခြေခံကိုတော့ သဘောပေါက်မည်ဖြစ်သည်။

Redis ၏ နားလည်ရန်လိုသည် အရေးကြီးသည့်အချက်မှာ သင့် interact လုပ်သည့်အပေါ်မူတည်၏ ပြောင်းလဲမည်ဖြစ်သည်။ SQL background မှ လာသော developer များသည် database နှင့် ဆက်သွယ်ရသည့် round trip ကိုတက်နိုင်သမျှ အနည်းဆုံးဖြစ်အောင် လုပ်မည်ဖြစ်သည်။ ၎င်းသည် redis အပါအဝင် မည်သည့် system အတွက်မဆိုကောင်းသော အလေ့အထဖြစ်သော်လည်း မိမိတို့ကိုင်တွယ်ရသည်မှာ ပို၍ရိုးရှင်းသော data structure များဖြစ်သဖြင့် တခါတရံ redis server ကိုကြိမ်ဖန်များစွာ hit ပြီးမှ လိုအပ်သည်ရသည့် အခါမျိုးလည်းရှိသည်။ ထိုသို့သော data access pattern များသည် အစောပိုင်းတွင် အသားကျမည် မဟုတ်သော်လည်း လက်တွေ့တွင် ထိုသို့ပြုလုပ်ခြင်းသည် ပေးဆပ်ရသည်က နည်းလှပြီး ထိုသည်နှင့်ရသည့် performance gain မှာ အဆမတန်ဖြစ်သည်။

ယခု အခန်းတွင်

Redis နှင့် ခဏလေးသာထိတွေ့ရသော်လည်း topic အများအပြားကို ပြောဖြစ်ခဲ့သည်။ query ပြုလုပ်ခြင်းကဲ့သို့ တချို့အရာများကို မသဲကွဲသေးလျင်လည်း စိတ်မပူပါနှင့်။ နောက် အခန်းများတွင် လက်တွေ့ဘက်သွားမည်ဖြစ်ပြီး စိတ်ကူးထားသော မေးခွန်းများအတွက် အဖြေများထွက်လာပါလိမ့်မည်။

ယခုအခန်း၏ အဓိကကျသော အချက်များမှာ

- key များသည် data များကို (value များ) identify ပြုလုပ်ရန် string များဖြစ်သည်
- Value များသည် Redis အနေဖြင့် ဂရုမစိုက်ထားသော byte array များဖြစ်သည်
- Redis အနေဖြင့် အထူးပြု data structure ငါးခုရှိသည်။
- ပေါင်းစပ် အသုံးခြင်းဖြင့် Redis သည် အသုံးပြုရလွယ်ကူပြီး လျင်မြန်သောလည်း အခြေအနေတိုင်းအတွက် သင့်တော်သည် မဟုတ်။

အခန်း ၂ - Data Structure များ

Redis ၏ data structure ၅ မျိုးကို လေ့လာကြပါစို့။ data structure တစ်ခု ချင်းဆီ၏ method များနှင့် feature များကိုရှင်းပြပါမည်။

လက်ရှိအထိ Redis တွင် အတွေ့အများဆုံးမှာ commands များနှင့် key နှင့် value များဖြစ်သည်။ အခုထိ data structure ကြောင်း တိတိပပ မလေ့လာ ရသေးပါ။ `set` command ကိုအသုံးပြုပါက redis အနေဖြင့် မည်သည့် data structure ကိုအသုံးပြုသနည်း? command တစ်ခုတိုင်းသည် data structure တစ်ခုချင်းစီကို အထူးပြုနေသည်။ ဥပမာ `set` ကိုအသုံးပြုပါက value ကို string data structure ထဲသို့ သိမ်းမည်ဖြစ်သည်။ `hset` ဟုသုံးပါ က hash အတွင်းတွင် သိမ်းမည်ဖြစ်သည်။ Redis ၏ vocabulary သည် သိပ်မများလှသဖြင့် အဆင်ပြေသည်ဟု ဆိုရမည်။

[Redis' website](#) သည် အတော် ကောင်းမွန်သော documentation ဖြစ်သည်။ ၎င်းတို့ လုပ်ပြီးသား အလုပ် ကို ထပ်လုပ်ရန် အကြောင်းမရှိပါ။ အခုစာအုပ်တွင် data structure ကို နားလည်ရန် အရေးအကြီးဆုံး command များကိုသာ ညွှန်ပြသွားပါမည်။

၄င်တို့ထက် အရေးကြီးသည်မှာ စမ်းကြည့်ပြီး ပေါ့ပေါ့ပါးပါးလေ့လာနိုင် ရန်ဖြစ်သည်။ သင့်အနေဖြင့် database မှ value အားလုံးကို `flushdb` ဟု

ရိုက်ထည့်ကာ ဖျက်ပစ်နိုင်သည်။ ထို့ကြောင့် ဘာမှ အားမနာပဲ စမ်းကြည့်ကြတာပေါ့!

String များ

String များသည် redis တွင်ပါဝင်သော အခြေခံအကျဆုံး data structure ဖြစ်သည်။ key-value pair တစ်ခုကို တွေးမိပါက string များကို ပထမဦးစွာ တွေးမိမည်ဖြစ်သည်။ သို့သော် နာမည်ကြောင့် မရှုပ်ထွေးပါနဲ့။ ထုံးစံအတိုင်း value များသည် အရာအားလုံးဖြစ်နိုင်သည်။ ကျွန်တော် အနေဖြင့် ဆိုရင် scalar ဟုပင် ခေါ်စေချင်သည်။ ကျွန်တော် အမြင်သက်သက်သာ ဖြစ်သည်။

String များ၏ အသုံးများသော use-case ကိုတွေ့ပြီးဖြစ်ပြီး object များ၏ instance များကို string အနေဖြင့် သိမ်းဆည်းခြင်းဖြစ်သည်။ ၎င်းသည် သင့်အနေဖြင့် အများအားဖြင့်သုံးမည့် အလေ့အထ တစ်ခုဖြစ်သည်။

```
set users:leto '{"name": leto, "planet": dune, "likes": ["spice"]}'
```

ထပ်၍ Redis အနေဖြင့် တခြားသော operation များလည်း ဆောင်ရွက်ခွင့်ပေးသည်။ ဥပမာ `strlen <key>` ဟု အသုံးပြု၍ key ၏ value ၏ အရှည်ကို ရနိုင်သလို ၊ `getrange <key> <start> <end>` ပါက value ၏ အထူးပြုထားသော range ကိုလည်းရနိုင်ပြီး၊ `append <key> <value>` ဆိုပါ

က ရှိပြီးသား value မှ (မရှိသေးပါက အသစ်ပြုလုပ်ပြီး) append ပြုလုပ်ပေးသည်။ ထိုကြောင့် စမ်းကြည့်ပါ ကျွန်တော် စမ်းကြည့်တုံးက ဒီလိုရပါတယ်။

```
> strlen users:leto
(integer) 50

> getrange users:leto 31 48
"\"likes\": [\"spice\"]"

> append users:leto " OVER 9000!!"
(integer) 62
```

မိုက်တော့ မိုက်ပါတယ် ဒါပေမယ့် အဓိပ္ပါယ်မရှိဘူး ဟု တွေးကောင်းတွေးပါလိမ့်မည်။ JSON မှ range တစ်ခုကို pull ဖို့နှင့် value တစ်ခုကို appendဖို့မှာ အဓိပ္ပါယ်မရှိပါ။ မှန်ပါသည်။ ယခု သင်ခန်းစာမှာ အချို့သော command များနှင့် string data structure ကိုအသုံးပြု၍ စမ်းသပ်ကြည့်ခြင်းဖြစ်ပြီး အချို့သော data အမျိုးအစားမှသာ အဓိပ္ပါယ်ရှိနိုင်မည်။

အစောပိုင်းတွင် Redis သည် သင့်၏ value များကို ဂရုမစိုက် ဟုပြောခဲ့ပြီးဖင်းမှာ ကိစ္စအတော်များများအတွက် မှန်ကန်သော်လည်း string command အချို့မှာ အချို့သော အမျိုးအစားများနှင့် value structure များတွင်သာသုံး၍ရသည်။ ဥပမာအနေဖြင့် `append` နှင့် `getrange` များသည် အချို့သော space အသုံးပြု serialization များအတွက် အသုံးဝင်ပါလိမ့်မည်။ ပို၍ခိုင်မာသော ဥပမာအနေဖြင့် `incr`၊ `incrby`၊ `decr` နှင့် `decrby` command

များရှိပြီး ၎င်းသည် string တစ်ခု၏ value ကို တိုးရာ လျှော့ရာတွင် အသုံးပြုသည်။

```
> incr stats:page:about
(integer) 1
> incr stats:page:about
(integer) 2

> incrby ratings:video:12333 5
(integer) 5
> incrby ratings:video:12333 3
(integer) 8
```

ထိုကြောင့် redis string များသည် analytics များအတွက် အသုံးပြုနိုင်သည်။ `users:leto` (integer မဟုတ်သော value) ကိုတိုးကြည့်ပြီး ဘာဖြစ်မလဲဆိုသည်ကို စောင့်ကြည့်ပါ (error ပေါ်လာရပါမည်)

ပို၍ advanced ဖြစ်သော ဥပမာမှာ `setbit` နှင့် `getbit` တို့ဖြစ်သည်။ ၎င်းအတွက် ယနေ့အတွက် သင့် ဘလော့ကို unique visitor ဘယ်နှစ်ယောက်ရှိသလဲဆိုသည်ကို တည်ဆောက်ထားသည် post ကိုဖတ်ကြည့်ပါ။ ၁၂၈ သန်းသော user များအတွက် laptop တစ်ခုမှ စွမ်းဆောင်နိုင်သော အဖြေသည် 50ms အတွင်းပြီးပြီး Memory 16MB မျှသာကုန်သည်။

bitmap များဘယ်လိုအလုပ်လုပ်သည် နှင့် ၎င်းကို ဘယ်လိုအသုံးပြုသွားသလဲဆိုသည်က အရေးမကြီး ထိုက်ထက် Redis ၏ string များသည် မူလက မြင်သည့်ထက်ပို၍ powerful ဖြစ်သည်ကို သိရန်ဖြစ်သည်။ သို့ပင်သော်လည်း အများဆုံးအသုံးပြုသည့် အခြေအနေများမှာ အပေါ်မှ ပြောခဲ့

သလို object များကို သိမ်းဆည်းခြင်း (ရှုပ်ထွေးသည်ဖြစ်စေ) နှင့် counter များအတွက်ဖြစ်သည်။ ထိုအပြင် value တစ်ခုကို key မှရယူခြင်းသည် အလွန်မြန်ဆန်သဖြင့် string များသည် data များကို cache ရန်အသုံးပြုသည်။

Hash များ

Hash များသည် Redis ၏ key-value များသိမ်းဆည်းခြင်းသည် အတိအကျမဖြစ်ကြောင်းကို ပြသနိုင်သည် ဥပမာဖြစ်သည်။ အချိန်တော်တော်များများတွင် hash များသည် string များနှင့်တူညီပြီး အဓိကကွာခြားချက်မှာ field များဖြစ်သည်။ ထိုကြောင့် `set` နှင့် `get` တို့၏ ပုံစံတူမှာ အောက်ပါအတိုင်းဖြစ်သည်။

```
hset users:goku powerlevel 9000
hget users:goku powerlevel
```

field များစွာကို တစ်ခါတည်း set ၍လည်းရသလို field များစွာ တစ်ခါတည်း get ရပြီး field များကို list လုပ်ခြင်းနှင့် field တစ်ခုစီကို delete လုပ်ခြင်းများပြုလုပ်နိုင်သည်။

```
hmset users:goku race saiyan age 737
hmget users:goku race powerlevel
hgetall users:goku
hkeys users:goku
hdel users:goku age
```

သင်မြင်သည့်အတိုင်း hash များသည် ရိုးရိုး string များထက်ပို၍ ထိန်းချုပ်ရလွယ်ကူသည်။ user တစ်ဦးကို serialize value တစ်ခုအနေဖြင့် သိမ်းမည့်အစား hash ဖြင့်သိမ်းဆည်းပါက ပို၍ တိကျသော ညွှန်ပြမှုကို ရရှိနိုင်သည်။ အကျိုးအမြတ်အနေဖြင့် value တစ်ခုလုံးပြန်လည်ရေးသားရန်မလိုပဲ လိုချင်သော data ၏ အပိုင်းအစများကို ဆွဲထုတ်၊ ပြင်ဆင် ၊ ဖျက်နိုင်ခြင်းဖြစ်သည်။

အသေအချာတည်ဆောက်ထားသော object တစ်ခု၏ အမြင်မှ hash များကိုကြည့်ပါက ဥပမာဖြစ်သည့် user တစ်ဦးသည် ၎င်းတို့ကို နားလည်ရန် အဓိက သော့ချက်ဖြစ်သည်။ ထို့အပြင် performance ရှုထောင်ကလည်း ၎င်း၏ တိကျသော ထိန်းချုပ်မှုသည် အသုံးဝင်လောက်သည်။ နောက်အခန်းများတွင် hash များကို အသုံးပြု၍ data များကို မည့်သို့ စီစဉ်ရမည်နှင့် လက်တွေ့တွင် မည့်သို့ query ပြုလုပ်ရမည်ကို ပြောသွားပါ့မည်။ ထို အချက်သည် hash များ၏ အဓိက အားသာချက်ဖြစ်သည်။

List များ

List များသည် key တစ်ခုစီ၏ array value များကို သိမ်းဆည်းပြင်ဆင်နိုင်သည်။ list တွင် value များကိုထည့်နိုင် ၊ ဦးဆုံးနှင့် နောက်ဆုံးနှင့် index အတိုင်း value များကိုရယူပြင်ဆင်နိုင်သည်။ list များသည် အစီအစဉ်အတိုင်းတည်ရှိပြီး index အခြေပြု operation များတွင် အလွန် အသုံးဝင်

သည်။ site ကို register လုပ်သော user အသစ်များကို track ပြုလုပ်ရန် `newusers` ကိုဆောက်နိုင်သည်။

```
lpush newusers goku
ltrim newusers 0 49
```

ရှေးဦးစွာ list ၏ အရှေ့ဆုံးသို့ user အသစ်တစ်ဦးပို့လိုက်ပြီးနောက် အယောက် ငါးဆယ်သာပါရန် trim လုပ်လိုက်ပါသည်။ ၎င်းသည် သုံးနေကြ ပုံစံဖြစ်ပြီး `ltrim` သည် O(N) operation ဖြစ်သဖြင့် N သည် move လုပ်သော value အရေအတွက်ဖြစ်သည်။ ထိုအခြေအနေတွင် insert တစ်ခုလုပ်ပြီးတိုင်း trim ပြုလုပ်ပါက constant performance အနေဖြင့် O(1) ကိုရရှိမည်ဖြစ်သည် (N သည် 1 ဖြင့်အမြဲတူသောကြောင့်)

ယခုသည် ပထမဆုံးအကြို key တစ်ခု၏ value သည် အခြားတစ်ခုကို reference Δပုလုပ်သည်ကို မြင်ဖူးခြင်းဖြစ်သည်။ အကယ်၍ နောက်ဆုံးဆယ်ယောက်၏ အသေးစိတ်ကိုသိလိုပါက အောက်ပါအတိုင်း ပြုလုပ်နိုင်သည်။

```
ids = redis.lrange('newusers', 0, 9)
redis.mget(*ids.map {|u| "users:#{u}"})
```

အပေါ်မှ ဥပမာတွင် Ruby အနည်းငယ်ပါဝင်ပြီး ကျွန်တော်တို့ အသုံးပြုနေသည့် roundtrips များအကြောင်းကို ပြလိုက်သလိုပါပဲ။ list များသည် အခြား key များ၏ reference များကိုသိမ်းဆည်းရန်တွင်မကပါ။ value များသည် အမျိုးစုံဖြစ်နိုင်သည်။ log ၏ list များကို သိမ်းဆည်းထားနိုင်သ

လို့ user တစ်ဦး site ထဲဝင်ကြည့်သည့် လမ်းကြောင်းကိုလည်း သိမ်းဆည်းထားနိုင်သည်။ game တစ်ခုကို တည်ဆောက်နေပါက user တစ်ဦး၏ action များကို စောင့်ကြည့်နိုင်သည်။

Set များ

set များသည် unique ဖြစ်သည့် value များကိုသိမ်းဆည်းထားနိုင်ပြီး union ကဲ့သို့သော set အခြေပြု operation များကို အထောက်အပံ့ပေးထားသည်။ set များသည် order လိုက်မဟုတ်သော်လည်း အသုံးဝင်သည့် value အခြေပြု operation များရှိသည်။ friend list များသည် ဥပမာတစ်ခု ဖြစ်သည်။

```
sadd friends:leto ghanima paul chani jessica
sadd friends:duncan paul jessica alia
```

user တစ်ဦးတိုင်းတွင် friend မည့်မျှ ရှိသည်ဖြစ်စေ userX သည် userY ၏ သူငယ်ချင်း ဟုတ်မဟုတ် ကို (O(1)) အနေဖြင့် ပြောပြနိုင်သည်။

```
sismember friends:leto jessica
sismember friends:leto vladimir
```

ထိုအပြင် နှစ်ဦးထက်ပိုသော သူများ ၎င်းတို့၏ သူငယ်ချင်းများ တူကြသလားဆိုသည်ကိုပင် သိနိုင်သည်။

```
sinter friends:leto friends:duncan
```

ထိုအပြင် key အသစ်အဖြစ် သိမ်းဆည်းနိုင်ပါသေးသည်။

```
sinterstore friends:leto_duncan friends:leto friends:duncan
```

Set များသည် value တစ်ခု၏ အခြား property များ ထပ်နေသောအခါ tag နှင့် track ပြုလုပ်ခြင်းများအတွက် ကောင်းမွန်သည်။ (သို့မဟုတ် intersection နှင့် union များပြုလုပ်ရာတွင်)

စီထားသော Set များ

နောက်ဆုံး powerful အဖြစ်ဆုံးသော data structure မှာ sorted set များဖြစ်သည်။ hash သည် string နှင့်တူ၍ field များအပိုပါသည်ဟု ဆိုရပါလျှင် sorted set များသည် set နှင့် အတူတူပင်ဖြစ်သော်လည်း score များပါသည်။ ၎င်း score များသည် sorting နှင့် ranking ပြုလုပ်ရာတွင် အထောက်အကူပြုသည်။ အကယ်၍ သူငယ်ချင်းများ၏ ranked list ကို အလိုရှိပါက အောက်ပါအတိုင်း ဆောင်ရွက်နိုင်သည်။

```
zadd friends:duncan 70 ghanima 95 paul 95 chani 75 jessica 1
vladimir
```

`duncan` ဟုခေါ်သည့် သူသည် score 90 ကျော်သည် သူငယ်ချင်း မည်မျှရှိသနည်း?

```
zcount friends:duncan 90 100
```

ထိုအထဲမှ `chani` ၏ rank ကိုသိချင်ပါက?

```
zrevrank friends:duncan chani
```

`zrank` အစား `zrevrank` ကိုသုံးရသည်မှာ redis ၏ default sort သည် ငယ်ရာမှာကြီးရာဖြစ်သောကြောင့်ဖြစ်သည် (ယခုကိစ္စတွင်မူ ranking မှာ high မှ low သို့ဖြစ်သည်) sorted set ၏ အထင်ရှားဆုံး use case မှာ leaderboard system ဖြစ်သည်။ လက်တွေ့တွင် သင့်အနေဖြင့် integer အသုံးပြု၍ sort လိုသည့်အရာတိုင်းကို အလုပ်ဖြစ်အောင် score အတိုင်း manipulate လုပ်နိုင်သည်က ၎င်း၏ အားသာချက်ဖြစ်သည်။

ယခုအခန်းတွင်

၎င်းသည် redis ၏ data structure ငါးခု၏ အခြေခံအကြောင်းအရာကိုမှာ အထက်ပါအတိုင်းဖြစ်သည်။ Redis ၏ မိုက်သော အချက်မှာ ကိုယ်ထင်ထားသည်ထက်ပို၍ လုပ်နိုင်ခြင်းဖြစ်သည်။ string နှင့် sorted set များကို ကျွန်တော်တို့ မတွေးထားသည့် အတိုင်း သုံးနိုင်သည့် ပုံစံများလည်းရှိဦးမည်။ ပုံမှန်သုံးနေကြပုံစံများသိသည့်တိုင်အောင် သင့်အနေဖြင့် Redis ကိုင်တွယ်နိုင်သော ပြဿနာများကို သတိထားမိပါလိမ့်မည်။ ထိုအပြင် redis အနေဖြင့် ငါးခုကို support လုပ်ထားသောကြောင့် အားလုံးကို သုံးရန်လိုမည်ဟုတ် မထင်ပါနှင့်။ အချို့သော system များသည် command အနည်းငယ်သာသုံးလိုက်ရသည်လည်းရှိပါသေးသည်။

အခန်း (၃) - Data Structure များ အကြောင်းထပ်လောင်း

အရင် အခန်းများတွင် datastructure ၅ ခု အကြောင်း ၊ ဥပမာများနှင့် ၎င်း တို့ဖြေရှင်းနိုင်သည့် ပြဿနာများအကြောင်း ပြောပြီးဖြစ်သည်။ ယခုအခန်း တွင်မူ ပို၍ advanced ဖြစ်ပြီး လက်တွေ့ကျသော topics များနှင့် design pattern များအကြောင်းပြောပါမည်။

Big O Notation

ဒီစာအုပ်တအုပ်လုံးတွင် Big O notation ကို ဥပမာအနေဖြင့် O(n) သို့မဟုတ် O(1) ဟုသာပြောဆိုခဲ့ပြီးဖြစ်သည်။ Big O notation သည် element မည်မျှတွင် မည့်သို့ ဆောင်ရွက်သနည်းကို ရှင်းပြသည့်နေရာတွင် အသုံးဝင်သည်။ Redis တွင်မူ command ၏ အမြန်နှုန်းသည် မည်မျှ မြန်ဆန်သည်ကို ရှင်းပြရန် ပြောသည်။

Redis documentation အနေဖြင့် command တစ်ခုချင်းစီ၏ Big O notation များကို ပြသထားသည်။ ၎င်းသည် performance ကိုပိုမို ကောင်းမွန်စေရန် မည့်သည့် factor များကို အလေးထားရမည်ကို ညွှန်းဆို နေသလိုဖြစ်သည်။ အောက်ပါ ဥပမာများကို ကြည့်ပါ။

O(1) ၏ အမြန်ဆုံးမှာ constant ဖြစ်သည်။ item ၅ ခုနှင့် လုပ်သည်ဖြစ်စေ ၅ သန်းဖြင့် လုပ်သည်ဖြစ်စေ ကြာချိန်မှာ အတူတူဖြစ်သည်။ `sismember` command ဖြင့် value တစ်ခုသည် set တွင် ရှိနေသည် ဟုတ်မဟုတ်ကို ဆန်းစစ်နိုင်သည်။ `sismember` သည် powerful ဖြစ်သော command တစ်ခု ဖြစ်ပြီး ၎င်းသည် performance အနေအထားသည်လည် O(1) ဖြစ်သည်။ Redis command အချို့မှာ O(1) ဖြစ်သည်။

Lograithmic သို့မဟုတ် O(log(N)) သည် ဒုတိယအမြန်ဆုံး ဖြစ်နိုင်ချေတစ်ခုဖြစ်ပြီး ၎င်းသည် ပို၍ပို၍ သေးငယ်သော partitation များကို scan ဖတ်ရ၍ဖြစ်သည်။ ၎င်းကဲ့သို့ divide & conqueror နည်းလမ်းကို အသုံးပြု၍ item အများကြီးကို iteration အနည်းငယ်ဖြင့် run ၍ရအောင် စွမ်းဆောင်ပေးသည်။ `zadd` သည် O(log(N)) ဖြစ်ပြီး N သည် sorted set အတွင်းရှိသော element အရေအတွက်ဖြစ်သည်။

ထိုနောက် linear command ဝါ O(N) ဖြစ်ပြီး ၎င်းသည် index မဟုတ်ထားသော table အတွင်းရှိcolumn တစ်ခုကိုရှာပါက O(N) ဖြစ်သည်။ ထို့ကြောင့် `ltrim` ကဲ့သို့ command များသည် အကျုံးဝင်သည်။ သို့သော် `ltrim` ကိစ္စတွင် N သည် list အတွင်းရှိ element မဟုတ်ပဲ remove ပြုလုပ်လိုက်သော element အရေအတွက်ရှိသည်။ ထိုကြောင့် `ltrim` ကိုအသုံးပြုပြီး တစ်သန်းလောက်ရှိသော list မှ item တစ်ခုကို remove ပြုလုပ်လိုက်ခြင်းသည်းသည် ထောင်နဲ့ချီရှိသော list မှာ item ဆယ်ခုကို remove ပြုလုပ်

ခြင်းထက်ပိုမြန်မည်ဖြစ်သည်။ (နှစ်ခုစလုံးမှာ အလွန်မြန်သဖြင့် ကွာခြားမှုကို မသိနိုင်သော်လည်း)

sorted set တစ်ခု၏ အနည်းဆုံးနှင့် အများဆုံး အကြားရှိ element များကို ဖယ်ရှားပေးသည့် `zremrangebyscore` သည် O(log(N)+M) complexity ရှိသည်။ ထိုကြောင့်အရောဟုဆိုရမည်။ documentation ကိုဖတ်ကြည့်ပါက N သည် set အတွင်းရှိ element အရေအတွက်ဖြစ်ပြီး M သည် remove ပြုလုပ်မည့် element အရေအတွက်ဖြစ်သည်။ တနည်းအားဖြင့် performance အနေဖြင့်ကြည့်ပါက remove ပြုလုပ်မည့် element အရေအတွက်သည်ပို၍ အရေးကြီးသည်ဖြစ်သည်

နောက် အခန်းတွင် ထပ်၍ အသေးစိတ်ဆွေးနွေးမည်ဖြစ်သည့် `sort` command သည် O(N+M*log(M)) complexity ရှိသည်။ performance ရှု ထောင်က သိနိုင်သည်မှာ ၎င်းသည် redis ၏ အရှုပ်ထွေးဆုံး command များဖြစ်သည်။

၎င်းအပြင် အခြားသော complexity များရှိသေးပြီး ကျန်သေးသော နှစ်ခုမှာ O(N^2) နှင့် O(C^N) တို့ဖြစ်ပြီး N ကြီးလာသည်နှင့်အမျှ N သေသော အခါထက် performanceာသည်။ Redis ၏ မည်သည့် command များမှာ ထိုမှ complexity မရှိပါ။

Big O notation သည် အဆိုးဆုံးအခြေအနေများကိုရှင်ဆိုင်ရန်ဖြစ်ပြီး O(N) ဖြစ်သည်ဆိုပါက ၎င်းသည် ပုံမှန်ဖြစ်လျင်ဖြစ်နိုင်သလို နောက်ဆုံးမှလည်း ဖြစ်လျင်ဖြစ်နိုင်သည်။

Pseudo Multi Key Query များ

ပုံမှန်ကြုံတွေ့ရနိုင်သည့် အခြေအနေတစ်ခုမှ တူညီသော value များကို query ပြုလုပ်ခြင်းဖြစ်သည်။ ဥပမာ user ၏ အမည်ကို email ဖြင့် သိမ်းထားပြီး (ပထဆုံးအကြိမ် login ဝင်သောအခါ) ထိုအပြင် id (login ဝင်ပြီးသောအခါ) ဆိုးရွားသော ဆောင်ရွက်မှု ဥပမာ ပုံစံမှာအောက်ပါအတိုင်း နှစ်ခုထပ်တူပွားလိုက်ခြင်းဖြစ်သည်။

```
set users:leto@dune.gov '{"id": 9001, "email": "leto@dune.gov",
...}'
set users:9001 '{"id": 9001, "email": "leto@dune.gov", ...}'
```

၎င်း၏ ဆိုးရွားသည်ဆိုသော အကြောင်းအရင်းမှာ memory နှစ်ဆကို manage လုပ်ရသည်မှာ အိမ်မက်ဆိုးဖြစ်သည်။ Redis အနေဖြင့် key တစ်ခုနှင့်တစ်ခုကို ချိတ်ဆက်ပေးလျှင်ကောင်းမည်ဖြစ်သော်လည်း ထို့သို့ အလုပ်လုပ်သည် မဟုတ် (နောင်လည်း လုပ်မည် မဟုတ်လောက်ပေ) key များကိုအတွင်းပိုင်းမှ ချိတ်ဆက်ပေးခြင်းသည် (key များနှင့်အများကြီးလုပ်လို့ရသည့်အကြောင်းများ မပြောရသေးသော်လည်း) အတွက် redis မှ အခြားသော solution ဖြင့်ဖြေရှင်းထားပြီး ၎င်းမှာ hash များဖြစ်သည်။

hash များကိုအသုံးပြုပြီး duplicate ဖြစ်သည်များကို ဖယ်ရှားနိုင်သည်။

```
set users:9001 '{"id": 9001, "email": "leto@dune.gov", ...}'
hset users:lookup:email leto@dune.gov 9001
```

ယခုပြုလုပ်ခြင်းသည် pesudo secondary index ကိုအသုံးပြုပြီး user object တစ်ခုကို reference ပြုလုပ်ခြင်းဖြစ်သည်။ user တစ်ဦးကို id ဖြင့် လိုချင်ပါက `get` ကိုအသုံးပြုနိုင်သည်။

```
get users:9001
```

user ကို email ဖြင့်လိုချင်ပါက `get` ၏နောက်တွင် `hget` ကိုခေါ်၍ယူရသည်။

```
id = redis.hget('users:lookup:email', 'leto@dune.gov')
user = redis.get("users:#{id}")
```

၎င်းသည် သင်အများဆုံးကြုံရမည့် ပုံစံဖြစ်သည်။ ၎င်းသည် ကျွန်တော်အမြင်အရ hash များ၏ အသုံးဝင်မှုကို ပေါ်လွင်စေသည့် ကာလဖြစ်ပြီး မမြင်ရသေးခင်အထိ သိသာသော use-case တစ်ခုမဟုတ်သေးပါ။

Reference နှင့် Index များ

value တစ်ခုမှ အခြားတစ်ခုသော reference ပြုလုပ်သော ဥပမာအတော်များများတွေ့ရပြီးဖြစ်သည်။ ပထမဆုံးအနေဖြင့် list ဥပမာတွင်တွေ့ရပြီးဖြစ်ပြီး ထိုနောက် hash များကိုအသုံးပြု၍ query ပြုလုပ်ရသည်ကို ပိုမိုလွယ်ကူအောင် အသုံးချခဲ့သည်။ index ကို ကိုယ့်ဖာသာ manage ပြုလုပ်

ပြီး value များအကြား reference ပြုလုပ်ခြင်းဖြစ်သည်။ အမှန်အတိုင်း ပြောရလျှင် နည်းနည်းပင်ချာသည် ဟုဆိုရမည် အထူးသဖြင့် manage ၊ update နှင့် delete များပြုလုပ်ပါက ထို reference များကိုပါပြင်ရမည်ဖြစ်သဖြင့် ဖြစ်သည်။ ၎င်းပြဿနာများကိုဖြေရှင်းရန် magic solution မရှိပါ။

ကျွန်တော်တို့ set များကို ယခုကဲ့သို့သော manual index ကို implement ပြုလုပ်ရန်အသုံးချသည်ကို မြင်တွေ့ပြီးဖြစ်သည်။

```
sadd friends:leto ghanima paul chani jessica
```

set တစ်ခု၏ member တိုင်းသည် user တစ်ဦး detail ကိုဖော်ပြထားသည် string value ၏ reference ဖြစ်သည်။ အကယ်၍ `chani` သည် နာမည်ပြောင်းလိုက်သည် သို့မဟုတ် account ကို delete လုပ်လိုက်ပါက ဘယ်လိုလုပ်မလဲ။ ထိုကြောင့် relationship ၏ပြောင်းပြန်ကိုပါ trackဖို့ လိုအပ်သည်။

```
sadd friends_of:chani leto paul
```

Maintenance လုပ်ရသည်မှာ အလုပ်ပိုသည်အပြင် သင်သည် ကျွန်တော်လို ဖြစ်ပါက index အသစ်ပြုလုပ်၍ process လုပ်ရသည်နှင့် memory ပိုသုံးသဖြင့် စိတ်တိုမိမည်မှာ အမှန်ပါ။ နောက် တစ်ပိုင်းတွင် extra round trips နှင့်ပတ်သတ်၍ performance cost ပိုကောင်းအောင် ဘယ်လိုလုပ်မလဲဆိုသည်ကို ပြောပါမည်။

ထိုကဲသို့ တွေးပါက relational database များသည် တူညီသော overhead များရှိသည်။ idnex များသည် memory တွင်ယူပြီး လိုအပ်သော record များအတွင် scan ပြုလုပ်ရသည် ကောက်ယူသည်က အတူတူပင်။ ထို overhead များသည် သေသပ်စွာ ဖယ်ထုတ်ထားသည် ဖြစ်သည် (process ပြုလုပ်ရာတွင် optimization အများအပြားပြုလုပ်ထားသည်)

ထိုအပြင် redis တွင် manual အနေဖြင့် reference များကိုကိုင်တွယ်ရသည်က အဆင်မပြေလှ။ သို့သော် အစောပိုင်းအနေဖြင့် memory နှင့် performance များနှင့်ပတ်သတ်၍ သံသယဝင်ပါက စမ်းသပ်ကြည့်သင့်သည်။ ကျွန်တော်အနေဖြင့် သင့် သိပ်မများလှဟု တွေ့မည်ဟုထင်ပါသည်။

Round Trip များနှင့် Pipeline ပြုလုပ်ခြင်း

Redis တွင် server သို့ အကြိမ်ကြိမ်စေခိုင်းခြင်းသည် ပုံမှန်ဖြစ်သည်ဟု ပြောပြီးဖြစ်သည်။ ထိုသိုကြိမ်ဖန်များစွာပြုလုပ်ရမည်ဖြစ်သည့် မိမိတို့ အသုံးချနိုင်မည့် feature များကို အနီးကပ်ကြည့်ရှုကြည့်ကြပါစို့။

ရှေးဦးစွာ command အများစု သည် တစ်ခုနှင့်တစ်ခုထက်ပိုသော parameter များကို လက်ခံနိုင်သည် သို့မဟုတ် အခြားသော လက်အောက်ခံ command မှတဆင့် parameter ပေါင်းများစွာလက်ခံနိုင်သည်။ ခုနမှ `mget` သည် key များစွာကို လက်ခံနိုင်ပြီး value ကို return ပြန်ပေးသည်။

```
ids = redis.lrange('newusers', 0, 9)
redis.mget(*ids.map {|u| "users:#{u}"})
```

သို့မဟုတ် `sadd` command သည် တစ်ခုနှင့် တစ်ခုထက်ပိုသော member များကို set အတွင်းသို့ထည့်နိုင်သည်။

```
sadd friends:vladimir piter
sadd friends:paul jessica leto "leto II" chani
```

Redis တွင် pipeline ပြုလုပ်ခြင်းကိုလည်း support လုပ်ပါသည်။ ပုံမှန်အားဖြင့် client မှ request ပြုလုပ်သောအခါ နောက် request မလာသေးခင် ၎င်း၏ reply ကိုစောင့်ဆိုင်းသော်လည်း pipeline ပြုလုပ်ပါက ၎င်း၏ response များကိုစောင့်ဆိုင်းစရာမလိုပဲ request များစွာကိုပို့၍ရသည်။ ထိုသို့ဖြင့် network overload များကိုလျော့ချနိုင်ပြီး သိသာသော performance ကောင်းမွန်မှုကို ကြုံရမည်ဖြစ်သည်။

Redis သည် queue ပြုလုပ်ထားသော command များအတွက် memory ကိုအသုံးပြုမည်ကို မှတ်သားရန်လိုပြီး ၎င်းကို batch ပြုလုပ်ရန်လိုသည်။ မည်မျှအထိ batch ကိုအသုံးပြုမည်ဆိုသည်က မိမိတို့အသုံးပြုမည့် command များအပေါ်မူတည်ပြီး အထူးသဖြင့် parameter မည့်မျှကြီးမည်စသဖြင့်ဖြစ်သည်။ သို့သော် သင့်အနေဖြင့် character ၅၀ ခန့်မျှရှိသော key များဖြစ်ပါက သောင်းနဲ့ချီ batch ပြုလုပ်၍ရသည်။

ထိုသို့ pipeline အတွင်း command များ execute ပြုလုပ်ခြင်းသည် driver တစ်ခုနှင့်တစ်ခုပေါ်မူတည်၍ကွဲပြားမည်ဖြစ်သည်။ Ruby တွင် `pipelined`

method ကို အသုံးပြုနိုင်သည်။

```
redis.pipelined do
  9001.times do
    redis.incr('powerlevel')
  end
end
```

သင့်အနေဖြင့် မှန်းဆထားသည့်အတိုင်း pipeline ပြုလုပ်ခြင်းဖြင့် batch import များကိုပို၍ လျှင်မြန်အောင် ဆောင်ရွက်နိုင်သည်။

Transaction များ

Redis command တိုင်းသည် အရာများစွာပြုလုပ်သည့်အရာများပင် atomic ဖြစ်သည်။ ထိုအပြင် redis သည် command များစွာကို အသုံးပြုရာတွင် transaction များကိုအသုံးပြုနိုင်သည်။

သို့သော် redis သည် single thread ဖြင့် အလုပ်လုပ်ပြီး ထို့ကြောင့် command တစ်ခုတိုင်းသည် atomic ဖြစ်သည်ဟု အာမခံထားခြင်းဖြစ်သည်။ command တစ်ခု execute ပြုလုပ်ချိန်တွင် အခြား command များ run မည်မဟုတ်။ (scale ပြုလုပ်သည်နှင့်ပတ်သတ်၍ နောက်ပိုင်းအခန်းများတွင် အကြမ်းဖြင်းပြောသွားမည်ဖြစ်သည်) အချို့ command များသည် အရာများစွာဆောင်ရွက်သည်ကို သိရှိထားပါက ထိုအချက်သည် အလွန်အသုံးဝင်သည်ဟု ဆိုရမည်။ ဥပမာ

`incr` သည် `get` ပြုလုပ်ပြီး `set` ပြုလုပ်ခြင်းဖြစ်သည်။

`getset` သည် value အသစ်တစ်ခုကို set ပြီး မူလတန်ဖိုးကို return ပြုလုပ်ခြင်းဖြစ်သည်။

`setnx` သည် key တစ်ခုရှိသည်မရှိသည်ကို စစ်ဆေးပြီး မရှိပါက value ကို set ပြုလုပ်ခြင်းဖြစ်သည်။

၎င်း command များသည် အသုံးဝင်သော်လည်း သင့်အနေဖြင့် command ပေါင်းများစွာကို အုပ်စုလိုက် atomic အနေဖြင့် run ရမည်ဖြစ်သည်။ ထို့ကြောင့် `multi` command ကိုအသုံးပြုပြီး execute ပြုလုပ်သော command များကို ထည့်သွင်းခြင်းပြီး transaction အဖြစ် တည်ဆောက်နိုင်သည်။ ထို့နောက် `exec` command ဖြင့် execute ပြုလုပ်နိုင်သလို `discard` ဖြင့် ပယ်ဖျက်နိုင်သည်။ Redis တွင် transaction အတွက်အာမခံချက်ပေးထားသည်များမှာ

- command များသည် အစီအစဉ်အတိုင်း execute ပြုလုပ်သွားခြင်း
- command များသည် single atomic operation အနေဖြင့် execute ပြုလုပ်သွားခြင်း (climent command မှာ တဝက်တပျက် execute ပြုလုပ်သွားသည့်တိုင်)
- ၎င်းသည် command များအာလုံးကို execute လုပ်သည့်နှင့် တစ်ခုမှ မပြုလုပ်သည့် အနေအထားသာဖြစ်မည်

သင့်အနေဖြင့် commandline interface တွင် စမ်းသပ်နိုင်ပြီး စမ်းလည်း စမ်းသပ်သင့်သည်။ ထို့အပြင် သင့်အနေဖြင့် pipeline ကိုပါ ပေါင်းစပ်၍ အသုံးပြုနိုင်သည်။

```
multi
hincrby groups:1percent balance -9000000000
hincrby groups:99percent balance 9000000000
exec
```

နောက်ဆုံးတွင် Redis သည် key တစ်ခု (သို့မဟုတ် တခုထက်ပို၍) ကို စောင့်ကြည့်နိုင်ပြီး key များပြောင်းလဲသွားပါက transaction ကို apply ပြုလုပ်နိုင်သည်။ ၎င်းသည် transaction များအတွင်း သင့်အနေဖြင့် value ကိုယူပြီး execute ပြုလုပ်ရသော အခြေအနေများတွင် အသုံးပြုသည်။ အပေါ်မှ code ဆိုပါက `exec` ပြီးပါက အတူတကွ execute ပြုလုပ်သွားမည်ဖြစ်၍ ကိုယ့်ဖာသာ `incr` command ကို implement ပြုလုပ်နိုင်မည်မဟုတ်။

```
redis.multi()
current = redis.get('powerlevel')
redis.set('powerlevel', current + 1)
redis.exec()
```

Redis ၏ transaction များသည် ထိုကဲ့သို့ အလုပ်လုပ်သည်မဟုတ် သော်လည်း `watch` အနေဖြင့် `powerlevel` ကိုadd လိုက်ပါက လုပ်ဆောင်နိုင်သည်။

```
redis.watch('powerlevel')
current = redis.get('powerlevel')
redis.multi()
```

```
redis.set('powerlevel', current + 1)
redis.exec()
```

သင့်၏ အခြား client မှာ `powerlevel` ၏ တန်ဖိုးကို ပြောင်းလိုက်ပါက `watch` ကိုခေါ်ထားသဖြင့် transaction သည် ဆောင်ရွက်နိုင်မည် မဟုတ်ပေ။ အခြား client မှ မပြောင်းလဲထားပါက ၎င်း set သည်အလုပ်လုပ်မည် ဖြစ်သည်။ ထိုသို့အလုပ်မလုပ်မချင်း code ကို loop အတွင်း execute ပြုလုပ်နိုင်သည်။

Key များ၏ Anti-Pattern

နောက် အခန်းတွင် data structure များနှင့် မသက်ဆိုင်သော command များကို ပြောသွားမည်ဖြစ်သည်။ အချို့သည် adminstrative ပြုလုပ်ရာတွင် သော်လည်းကောင်း အချို့သည် debuging tool များဖြစ်သည်။ သို့သော် ကျွန်တော်ပြောလိုသော အရာတစ်ခုမှာ `keys` command ဖြစ်သည်။ ထို command သည် pattern တစ်ခုကိုလက်ခံပြီး match ဖြစ်သော key များကို ရှာဖွေပေးသည် ၎င်းသည် command သည်အချို့သော အရာများတွင် သင့်တော်သော်လည်း production တွင်လုံးဝမသုံးသင့်ပေ။ အဘယ်ကြောင့်ဆိုသော် ၎င်းသည် linear အရ key များအားလုံးကို scan ပြုလုပ်သွားသောကြောင့် တနည်းအားဖြင့် နှေးသောကြောင့်ဖြစ်သည်။

ဘယ်လိုအသုံးပြုကြလို့လဲ ဟုမေးပါက bug tracking service တစ်ခုတည်ဆောက်သည်ဟု ဆိုပါက account တိုင်းတွင် `id` တစ်ခုရှိမည်

ဖြစ်သည်။ `bug:account_id:bug_id` နဲ့ကိုက်ညီသော bug များကို string value store လုပ်ချင်သည်ဟုဆိုပါစို့။ သင့်အနေဖြင့် account တိုင်း၏ bugs များကိုရှာရမည်ဖြစ်၍ (ပြဿရန်ဖြစ်စေ ၊ ဖျက်ရန်ဖြစ်စေ) သင့်အနေဖြင့် `keys` command ကိုသုံးချင်မည် ဖြစ်သည်။

```
keys bug:1233:*
```

ပို၍ကောင်းမွန်သည် solution သည် hash ကိုအသုံးပြုရန်ဖြစ်ပြီး ၎င်းကိုအသုံးပြုခြင်းဖြင့် secondary index များကို expose ပြုလုပ်နိုင်ပြီး data များကို organize ပြုလုပ်နိုင်သည်။

```
hset bugs:1233 1 '{"id":1, "account": 1233, "subject": "..."}'
hset bugs:1233 2 '{"id":2, "account": 1233, "subject": "..."}'
```

bug ids များကိုလိုချင်ပါက `hkeys bugs:1233` ဟုအလွယ်တကူခေါ်နိုင်သည်။ ဖျက်ချင်ပါက`hdel bugs:1233 2` ဟုခေါ်ယူပြီး key ကို `del bugs:1233` ဟုဖျက်နိုင်သည်။

ယခုအခန်းတွင်

ယခင်အခန်းနှင့်ပေါင်းစပ်ပြီး redis ၏ အဓိက feature များနှင့်ပတ်သတ်၍ မည်သို့အသုံးပြုရမည်နည်းကို ရှင်းပြသွားပါသည်။ အမျိုးစုံတည်ဆောက်ရန် အခြားသော pattern များကိုပေါင်းစပ်အသုံးပြုနိုင်ပြီး အဓိကအချက်မှာ အခြေခံ data structure များနှင့် ၎င်းတို့ကို မည်သို့အသုံးချမည်နည်းကို နားလည်ရန်လိုသည်။

အခန်း ၄ - data structure များလွန်၍

data structure ငါးခုသည် redis ၏ အခြေခံကိုဖော်ဆောင်ပေးချိန်တွင် data structure အခြေပြုမဟုတ်သော အခြားသော command များလည်းရှိသေးသည်။ မြင်ဖူးပြီးသော အရာများဖြစ်သည့် `info` ၊ `select`၊ `flushdb`၊ `multi`၊ `exec`၊ `discard`၊ `watch` နှင့် `keys` တို့ပါဝင်သည်။ ယခုအခန်းတွင် အခြားသောအရေးကြီးသည်များကို ဖော်ပြသွားမည်။

Expiration

Redis တွင် key တစ်ခုကို expiration သတ်မှတ်နိုင်သည်။ unix timestamp တစ်ခုအနေဖြင့် (၁၉၇၀ ခုနှစ် ဇန်နဝါရီလ ၁ ရက်နေ့မှစ၍) ဖြစ်စေ ရှိနေမည့် စက္ကန့်အနေဖြင့် ဖြစ်စေ ပေးထားနိုင်သည်။ ၎င်းသည် key အခြေပြု command ဖြစ်သဖြင့် မည့်သည့် data structure မဆိုသုံးနိုင်သည်။

```
expire pages:about 30
expireat pages:about 1356933600
```

ပထမ command တွင် စက္ကန့် ၃၀ ကျော်ပါက key နှင့် ၎င်း၏ value ကို ဖျက်ပစ်မည်ဖြစ်ပြီး ဒုတိယတစ်ခုသည် 12:00 a.m. December 31st, 2012 တွင်ဖျက်ပစ်မည်ဖြစ်သည်။

၎င်း feature သည် redis ကို caching engine တစ်ခုအဖြစ်စွမ်းဆောင်နိုင်သည်။ သင့်အနေဖြင့် item တစ်ခု မည်မျှ အသက်ရှင်ရန် ကျန်သေးသည်ကို

`ttl` command ကိုအသုံးပြုပြီး သိနိုင်ပြီး `persist` command ကို အသုံးပြု၍ expiration ဖြစ်မှုကို ဖယ်ရှားနိုင်သည်။

```
ttl pages:about
persist pages:about
```

နောက်ဆုံး အနေဖြင့် `setex` ဟုခေါ်သည့် အထူး string command သည် string တစ်ခုကို set ပြုလုပ်ပြီး အချိန်အတိုင်းအတာ တစ်ခုအထိသာ တည်ရှိစေပါမည်။

```
setex pages:about 30 '<h1>about us</h1>....'
```

Publication နှင့် Subscription များ

Redis list များတွင် တစ်ခုတည်းကျန်သည့်အထိ ပထမဆုံး (သို့မဟုတ် နောက်ဆုံး) element ကို return ပြန်ပြီး remove ပြုလုပ်သော command များမရှိသည်။ ၎င်းကို simple queue အနေဖြင့် အသုံးပြုနိုင်သည်။

ထိုအပြင် Redis တွင် message များကို publish ပြုလုပ်ခြင်းနှင့် channel များကို subscribe ပြုလုပ်ရာတွင် အထောက်အကူအပြုသည်။ သင့်အနေဖြင့် `redis-cli` window ကိုဖွင့်၍စမ်းနိုင်သည်။ ပထမဆုံး channel ကို subscribe ပြုလုပ်ပါ။ (`warning` ဟုအမည်ပေးပါမည်)

```
subscribe warnings
```

Subscription ဖြစ်သွားပြီဖြစ်ကြောင်းကို reply လုပ်ပါမည်။ ထိုနောက် နောက်ထပ် window တစ်ခုတွင် `warning` channel သို့ message တစ်ခု publish ပြုလုပ်ပါ။

```
publish warnings "it's over 9000!"
```

ပထမ window သို့ ပြန်သွားပါက `warning` channel အတွင်းတွင် message ရောက်နေသည်ကိုတွေ့ရမည်ဖြစ်သည်.။

သင့်အနေဖြင့် channel ပေါင်းများစွာ (`subscribe channel1 channel2 ...`) ကို subscribe ပြုလုပ်နိုင်ပြီး Pattern အနေဖြင့်လည်း (`psubscribe warnings:*`) အသုံးပြုနိုင်သည့်အပြင် `unsubscribe` နှင့် `punsubscribe` ကို အသုံးပြု၍ တစ်ခုနှင့် တစ်ခုထက်ပိုသော channel များကို unsub ပြုလုပ်နိုင်သည်။

ထိုနောက် `publish` command သည် value 1 ကိုပြန်မည်ကို သတိထားမိမည်ဖြစ်ပြီး ၎င်းသည် လက်ခံရရှိသော client အရေအတွက်ကို ညွှန်းဆိုခြင်းဖြစ်သည်။

Monitor နှင့် Slow Log

`monitor` command သည် Redis ဘာဖြစ်နေသည်ကို ပြသော command ဖြစ်သည်။ သင့် application နှင့် Redis ချိတ်ဆက် အလုပ်လုပ်လုပ်နေ

သည့် insight ကိုပြသပေးသော debugging tool တစ်ခုဖြစ်သည်။ ခုနက redis-cli window နှစ်ခုအနက် တစ်ခုတွင် (တစ်ခုက sub ပြုလုပ်ထားပါက `unsubscribe` command ပြုလုပ်၍ဖြစ်စေ ထွက်ပြီး window အသစ်မှ ပြန်ဝင်၍ဖြစ်စေ) `monitor` command ကိုရိုက်လိုက်ပါ။ နောက်တစ်ခုတွင် အခြား command များ (ဥပမာ `get` သို့မဟုတ် `set`) ဆောင်ရွက်ကြည့်ပါ။ Parameter ကအစ ၎င်း command များကိုပါ ပထမ window တွင်တွေ့ရမည်ဖြစ်သည်။

production တွင် monitor မ run မိစေရန် သတိပြုရမည်ဖြစ်သည်။ ၎င်းသည် debug နှင့် development ပြုရာတွင် အထောက်အကူပြုသော်လည်း ထိုမှအပ အခြား ပြောရန်သိပ်မရှိပေ။ အလွန်အသုံးဝင်သော tool ဖြစ်သည်။

`monitor` နှင့်အတူ Redis တွင် `slowlog` ဟုခေါ်သည် profiling tool တစ်ခုရှိသည်။ ၎င်းသည် **micro**second အတိုင်းအတာ တစ်ခုထိ ကြာသော မည့်သည့် command မဆို log ပြုလုပ်ပေးသည်။ နောက်တစ်ဖြတ်တွင် Redis တွင်မည်သို့ configure ပြုလုပ်သင့်သည်ကို ရှင်းပြသွားမည် ဖြစ်ပြီး ယခုတွင်မူ command အားလုံးကို log လိုပါက အောက်ပါအတိုင်း configure ပြုလုပ်နိုင်သည်။

```
config set slowlog-log-slower-than 0
```

ထိုနောက် အောက်ပါ command များကို ထပ်ထည့်ပါ။ နောက်ဆုံးတွင် log များအားလုံး သို့မဟုတ် နောက်ဆုံး logs များကို retrieve လုပ်လိုပါက

```
slowlog get
slowlog get 10
```

slow log တွင်ရှိသော item အရေအတွက်ကို `slowlog len` ရိုက်ရင်းသိနိုင်သည်။

Command တိုင်းအတွက် သင့်အနေဖြင့် parameter လေးမျိုးကိုတွေ့ရမည်ဖြစ်ပြီး :

- auto-increment ဖြစ်သော id
- Command ဖြစ်ပေါ်သောအချိန်၏ Unix timestamp
- ထို command ကို run ရန်ကြာချိန် (microsecond ဖြင့်)
- ထို command နှင့် ၎င်း၏ parameter

The slow log is maintained in memory, so running it in production, even with a low threshold, shouldn't be a problem. By default it will track the last 1024 logs.

Sort

Redis ၏ powerful အဖြစ်ဆုံး command တစ်ခုမှာ `sort` ဖြစ်သည်။ ၎င်းသည် list ၊ set သို့မဟုတ် sorted set (sorted set များသည် score အရ

order ပြုလုပ်ထားခြင်းဖြစ်ပြီး member အလိုက် မဟုတ်ပါ) အတွင်းရှိ value များကို sort ပြုလုပ်နိုင်သည်။ အရိုးရှင်းဆုံးပုံစံအနေဖြင့် အောက်ပါ အတိုင်း ဆောင်ရွက်နိုင်သည်။

```
rpush users:leto:guesses 5 9 10 2 4 10 19 2
sort users:leto:guesses
```

၎င်း value များကို အနည်းဆုံးမှ အများဆုံးသို့ sort သွားမည်ဖြစ်သည်။ အောက်တွင် နောက်ဥပမာ တစ်ခုကိုကြည့်နိုင်ပါသည်။

```
sadd friends:ghanima leto paul chani jessica alia duncan
sort friends:ghanima limit 0 3 desc alpha
```

အပေါ်က command သည် sort ပြုလုပ်ထားသော records များကို limit မှ တဆင့် paging ပြုလုပ်ပုံကို ပြထားခြင်းဖြစ်သည်။ `desc` ကိုအသုံးပြု၍ descending order အတိုင်းစီနိုင်ပြီး နံပါတ်စဉ်တိုင်းမဟုတ်ပဲ စာသားအရစီလိုချင် `alpha` ကိုအသုံးပြုနိုင်သည်။

`sort` ၏ တကယ့် power မှာ referenced object ကိုအခြေပြု၍ sort နိုင်ခြင်းဖြစ်သည်။ အစောပိုင်းတွင် list၊ set နှင့် sorted set များသည် အခြား Redis Object များကို reference ပြုလုပ်ရန် အသုံးပြုကြကြောင်း ပြောပြီးပြီ ဖြစ်သည်။ `sort` command သည် ၎င်း relation များကို dereference ပြုလုပ်ပြီး ၎င်း value များဖြင့် sort ပြုလုပ်နိုင်သည်။ ဥပမာ issue များကို user များက watch ပြုလုပ်သည့် bug tracker စနစ်တစ်ခုရှိ

သည်ဆိုပါစို့ watch ပြုလုပ်သော issue များကို track လုပ်နိုင်ရန် အောက်ပါအတိုင်း set ရပါမည်။

```
sadd watch:leto 12339 1382 338 9338
```

၎င်းကို id ဖြင့် sort ပြုလုပ်ခြင်း (default အနေဖြင့်ပြုလုပ်မည်ဖြစ်ပြီး) ပုံမှန်ဖြစ်သော်လည်း ကျွန်တော်တို့အနေဖြင့် severity ဖြင့်စီလိုသည်။ ထို့သို့ပြုလုပ်ရန် Redis ကိုမည်သည့် pattern ဖြင့် sort မည်ကိုပြောရန်လိုသည်။ ရှေးဦးစွာ ပို၍အဓိပ္ပါယ်ရှိစေရန် data များထည့်ကြပါစို့။

```
set severity:12339 3
set severity:1382 2
set severity:338 5
set severity:9338 4
```

အမြင့်ဆုံးမှ အနိမ့်ဆုံးသို့ bug များကို severity ဖြင့်စီရန် အောက်ပါအတိုင်း

```
sort watch:leto by severity:* desc
```

Redis တွင် list များ၊ set များနှင့် sorted set များအတွင်းရှိ value များကို * pattern အတိုင်း အစားထိုးမည်ဖြစ်သည်။ (`by` ဖြင့် identify ပြုလုပ်ရပြီး) ၎င်းသည် Redis ၏ အစစ်အမှန် value ကို sortပြီး key များကို ထုတ်ပေးမည်ဖြစ်သည်။

Redis ထဲတွင် key များသန်းနဲ့ချီရှိနိုင်သော်လည်း ရှုပ်နေမည်ဖြစ်သည်။ ထို့ကြောင့် `sort` ကိုအသုံးပြုနိုင်ပြီး ၎င်းသည် hash များနှင့် ၎င်း၏ field များ

တွင်လည်း အလုပ်လုပ်သည်။ top level key အများကြီးထားမည့် အစား hash များကို အောက်ပါအတိုင်း ဆောင်ရွက်နိုင်သည်။

```
hset bug:12339 severity 3
hset bug:12339 priority 1
hset bug:12339 details '{"id": 12339, ....}'

hset bug:1382 severity 2
hset bug:1382 priority 2
hset bug:1382 details '{"id": 1382, ....}'

hset bug:338 severity 5
hset bug:338 priority 3
hset bug:338 details '{"id": 338, ....}'

hset bug:9338 severity 4
hset bug:9338 priority 2
hset bug:9338 details '{"id": 9338, ....}'
```

အားလုံးမှာ ပို၍ organize ဖြစ်သလို `severity` သို့မဟုတ် `priority` ဖြင့်လည်း စီနိုင်သည့် အပြင် မည်သည် field ဖြင့် `sort` ပြုလုပ်သည်ကိုပါ ပြောနိုင်သည်။

```
sort watch:leto by bug:*->priority get bug:*->details
```

value substution ပေါ်လာပါသော်လည်း redis အနေဖြင့် `->` ကို နားလည်သည်ဖြစ်၍ ၎င်းနှင့် ဆက်စပ်နေသော field ကို hash ထဲမှ သွားရှာပါ့မည်။ substitute နှင့် lookupပြုလုပ်ရန် field ကို `get` parameter မှ ထည့်သွင်းလိုက်ပြီး ၎င်းမှ bug detail ကိုရမည်ဖြစ်သည်။

ပမာဏများသော set များတွင်မူ `sort` သည်နှေးနိုင်သည်။ သတင်းကောင်းတစ်ခုမှ `sort` ၏ ရလဒ်ကို store လုပ်နိုင်ခြင်းဖြစ်သည်။

```
sort watch:leto by bug:*->priority get bug:*->details store
watch_by_priority:leto
```

`store` ပြုလုပ်နိုင်ခြင်းနှင့် ရှေ့မှ တွေ့ထားခဲ့သော expiration command နှစ်ခုကို ပေါင်းစပ်အသုံးပြုနိုင်သည်။

Scan

ယခင်အပိုင်းများတွင် အသုံးဝင်သော်လည်း production တွင် မသုံးသင့်သော `keys` အလုပ်လုပ်ပုံကိုသိပြီးဖြစ်သည်။ Redis 2.8 တွင် production-safe ဖြစ်သည့် `scan` ကိုမိတ်ဆက်ပေးခဲ့သည်။ `scan` သည် `keys` ၏ပုံစံအတိုင်း သုံးရန်ရည်ရွယ်ထားသော်လည်း သိသာထင်ရှားသည့် ကွဲပြားခြားနားချက် များရှိသည်။ ထိုကွဲပြားချက်များမှာ မသိသာဟုထင်ရသော်လည်း ၎င်းသည် အဓိကကြသော အချက်ဖြစ်သည်။

ရှေးဦးစွာ `scan` ဟု တစ်ခုတည်းခေါ်ရုံဖြင့် match ဖြစ်သော result အကုန်လုံးကို ထုတ်ပေးမည် မဟုတ်ပေ။ paged ပြုလုပ်ထားသော result များမှာ ထူးဆန်းလှသည်မဟုတ်သော်လည်း `scan` အနေဖြင့် အသေအချာ control လုပ်၍မရသည့် အစီအစဉ်ဖြင့် result များကို ထုတ်ပေးသည်။ `count` hint အနေဖြင့် default အနေဖြင့် 10 ဖြစ်မည်ဖြစ်သော်လည်း အနည်းအများကို ချိန်နိုင်သည်။

Paging များကို implement ပြုလုပ်ရာတွင် `limit` နှင့် `offset` များအစား `cursor` ကိုအသုံးပြုသည်။ ပထဆုံး `scan` ကိုခေါ်ပါက cursor မှာ `0` ဖြစ်သည်။ အောက်တွင် pattern match (optional) နှင့် count (optional) ကို အသုံးပြု၍ implement လုပ်ပြထားသည်။

```
scan 0 match bugs:* count 20
```

၎င်း reply ၏ တစိတ်တစ်ပိုင်းအနေဖြင့် `scan` သည် နောက်ထပ်အသုံးပြုနိုင်သည့် cursor တစ်ခုထည့်ပေးထားသည်။ သိရမည်မှာ နောက်ထပ် cursor value မှာ random ဖြစ်သည်။

ပုံမှန် လုပ်ဆောင်ပုံမှာ အောက်ပါအတိုင်းဖြစ်သည်:

```
scan 0 match bugs:* count 2
> 1) "3"
> 2) 1) "bugs:125"
scan 3 match bugs:* count 2
> 1) "0"
> 2) 1) "bugs:124"
>    2) "bugs:123"
```

ပထမဆုံး call သည် နောက်ထပ် cursor တစ်ခုဖြစ်သည့် (3) နှင့် result တစ်ခု return ပြန်ပေးမည်ဖြစ်သည်။ နောက်ထပ် call တွင် ပထမ cursor ကိုအသုံးပြုထားပြီး end cursor ဖြစ်သည့် (0) နှင့် နောက်ဆုံး result နှစ်ခုကို ထုတ်ပေးထားသည်။ အပေါ်မှ ပုံစံသည် ပုံမှန်လုပ်ဆောင်ပုံဖြစ်သည်။ `count` မှာ hint သက်သက်သာဖြစ်၍ `scan` တွင် 0 မဟုတ်သော `cursor` နှင့် တကယ့် result များမဟုတ်သည်ကို return ၍ရသည်။ တနည်းအားဖြင့်

empty result အနေဖြင့် result အသစ်ရှိမရှိကို မသိနိုင်ပါ။ 0 cursor ကသာ နောက်ထပ် result မရှိသည်ကို သိရှိနိုင်သည်။

`scan` သည် redis ဘက်မှကြည့်လျင် stateless ဖြစ်နေသော်ကြောင့် cursor ကို close ပြုလုပ်ရန်မလိုသလို အစအဆုံး မ read ပဲပြတ်ကျသွားလျင်လည်း ပြဿနာမဟုတ်ပေ။ valid ဖြစ်ပြီး နောက်ထပ် cursor ကို return ပြန်နေသည့်တိုင်အောင် iterate ပြုလုပ်သည်ကို ရပ်တန့်နိုင်သည်။

သို့သော် အချက်နှစ်ချက်ကို သတိထားရန်လိုပြီး ပထမဆုံး `scan` သည် တူညီသော key ကို ကြိမ်ဖန်များစွာ return ပြန်နိုင်ပြီး သင့်အနေဖြင့် ၎င်းကို ဘယ်လိုဖြေရှင်းမည်ဆိုသည်က (ရှိပြီးသား value များကို set တစ်ခုတည်း ထည့်ထားခြင်းဖြစ်စေ) တစ်ခုဖြစ်ပြီး ဒုတိယအနေဖြင့် `scan` သည် iteration ကာလအတွင်းသာ value များကိုသာ အာမခံနိုင်သည်။ အကယ်၍ သင့်အနေဖြင့် iterate ပြုလုပ်နေချိန် value များကို ထပ်ထည့်ခြင်း ဖျက်ပစ်ခြင်း ပြုလုပ်ပါက ထို value များသည် return ပြန်မည် မပြန်မည်က မသေချာပေ။ ထိုအပြင် ၎င်းသည် `scan` သည် state အနေဖြင့်မရှိသည်ဖြစ်၍ ရှိနေသည့် value ကို snapshot အဖြစ်မသိမ်းထားမည့်အစား (database တော်တော်များများ consistency နှင့်ပတ်သတ်ပါက အခိုင်အမာ အာမခံကြသော်လည်း) memory space အတွင်းသာ iterate ပြုလုပ်ထားခြင်းကြောင့် modified ပြုလုပ်ခြင်းခံရမည် ရှိမရှိက မသေချာပေ။

`scan` အပြင် `hscan`၊ `sscan` နှင့် `zscan` တို့လည်းရှိသေးပြီး ၎င်းတို့ဖြင့် hash ၊ set နှင့် sorted set များအတွင်း iterate ပြုလုပ်နိုင်မည်။ `keys` သည်အခြား caller များကို block ပြုလုပ်သကဲ့သို့ hash command `hgetall` နှင့် set command `smember` တို့သည်လည်း အတူတူပင်ဖြစ်သည်။ သင့်အနေဖြင့် ပမာဏများသည့် hash သို့မဟုတ် set မှ iterate ပြုလုပ်ပါက ထို command များကိုအသုံးပြုရန်စဉ်းစားသင့်သည်။ `zrangebyscore` နှင့် `zrangebyrank` တို့သည် နဂိုကတည်းကပါရှိသဖြင့် `zscan` သည် သိပ် အသုံးဝင်သည့်ပုံမရသော်လည်း သင့်အနေဖြင့် sorted set အကြီးမှ အစ အဆုံး iterate ပြုလုပ်လိုပါက `zscan` ကိုအသုံးပြုနိုင်သည်။

ယခုအခန်းတွင်

ယခုအခန်းတွင် data structure နှင့်မသက်ဆိုင်သော command များကို အာရုံစိုက်ထားသည်။ အရာအားလုံးကဲသို့ပင် ၎င်း၏ အသုံးမှာ အခြေအနေ အရဖြစ်သည်။ ထိုကြောင့် expiration ၊ publication၊ subscription နှင့် sorting ကို ကိုအသုံးမပြုသော app သို့မဟုတ် feature များသည် ဆန်းသည် မဟုတ်ပေ။ သို့သော် ၎င်းတို့ရှိသည်ကို သိထားရန်သလိုသည်။ ထိုအပြင် command ထဲမှာ အချို့ကိုသာ ရှင်းသွားသည်ဖြစ်ပြီး မပြောပဲချန်လှန်ထားသော ommand များလည်းရှိသေးပြီး ၎င်းတို့ကို [ယခုလင့်ကို] နှိပ်၍ကြည့်နိုင်သည်။

အခန်း (၆) - Lua Script ရေးခြင်း

Redis 2.6 တွင် Lua interpreter ပါဝင်ပြီး ၎င်းကို အသုံးပြု၍ Redis အတွင်းတွင် ပို၍အဆင့်မြင် query များကို execute ပြုလုပ်နိုင်သည်။ relational database များတွင် ပါဝင်သော stored procedure များနှင့် ဆင်တူသည်ဟု မှတ်ယူနိုင်သည်။

၎င်း feature ကိုကျွမ်းကျင်စွာအသုံးပြုနိုင်ရန် အခက်အခဲမှာ Lua ကိုသင်ယူရခြင်းဖြစ်သည်။ သို့သော် Lua သည် အခြားသော general purpose language များနှင့်ဆင်တူပြီး documentation ကောင်းကောင်းလည်းရှိသည့်အပြင် ၎င်း community သည်လည်း active ဖြစ်ကာ Redis တွင် script ပြုလုပ်ခြင်းသာမက အခြားနေရာများတွင်လည်း အသုံးဝင်သည်။ ယခုအခန်းတွင် Lua အကြောင်းအသေးစိတ်ပြောမည်မဟုတ်ပဲ အစပျိုးမိတ်ဆက်ရန် ဥပမာအနည်းငယ်များကိုသာ ထိတွေ့သွားပါမည်။

Why?

Lua Scripting ကိုအသုံးမပြုမီ အဘယ်ကြောင့် သုံးသင့်သည်ကို စဉ်းစားကောင်းစဉ်းစားပါလိမ့်မည်။ developer အတော်များများသည် ထုံးတမ်းစဉ်လာ stored procedure များကို သဘောမကျသဖြင့် ယခုဟာသည်လည်း ဘာကွာသနည်းဟု မေးစရာပေါ်လာသည်။ တိုတိုဖြေရပါက

“ဟင့်အင်း” ဟုဖြေရန်ရှိသည်။ ကောင်းကောင်းမွန်မွန် အသုံးမချတတ်ပါက Redis ၏ lua scripting သည် Code test ပြုလုပ်ရသည်မှာ ခက်ခဲသည့် အပြင် business logic နှင့် data access မှာ tightly couple ဖြစ်ရုံသာမက logic duplication ပါဖြစ်နိုင်သည်။

သို့သော် မှန်ကန်စွာ အသုံးချပါက ရိုးရိုးရှင်းရှင်း ရေးနိုင်ပြီး performance ကို ပိုမိုကောင်းမွန်စေသည်။ ၎င်း အကျိုးအမြတ်များသည် command များစွာကို group ပြုလုပ်ရာပြီး function အဖြစ်အသုံးပြုရာတွင် သိသာလှသည်။ Code မှာ ရိုးရှင်းသဖြင့် Lua script ၏ invocation တိုင်းသည် အတားအဆီးမရှိပဲ atomic command တိုင်းကို သေသေသပ်သပ် ဖန်တီးနိုင်သည်။ (`watch` command လိုအပ်ခြင်းကို ချေဖျက်နိုင်ခြင်းဖြင့်) ချက်ချင်း result ကို return မပြန်ပဲ နောက်ဆုံး calculated လုပ်ထားသည်ကိုသာ return ပြန်ခြင်းဖြင့် performance ပိုမိုကောင်းမွန်စေနိုင်သည်။

အောက်က ဥပမာများသည် ၎င်းအချက်များကို သိသာစေနိုင် တည်ဆောက်ထားခြင်းဖြစ်သည်။

Eval

`eval` command သည် Lua script ကို string အနေဖြင့် လက်ခံပြီး မိမိတို့ operating လုပ်မည့် key များနှင့် pass ပြုလုပ်မည့် argument များကို

optional အနေဖြင့် ပါဝင်သည်။ Ruby မှ execute ပြုလုပ်လိုက်သော အောက်က ဥပမာကိုကြည့်ပါ။

```
script = <<-eos
  local friend_names = redis.call('smembers', KEYS[1])
  local friends = {}
  for i = 1, #friend_names do
    local friend_key = 'user:' .. friend_names[i]
    local gender = redis.call('hget', friend_key, 'gender')
    if gender == ARGV[1] then
          table.insert(friends, redis.call('hget', friend_key,
'details'))
    end
  end
  return friends
eos
Redis.new.eval(script, ['friends:leto'], ['m'])
```

အပေါ်မှ code သည် Leto ၏ ယောကျ်ားလေး သူငယ်ချင်းအားလုံး၏ detail အားလုံးကို ရရှိနိုင်သည်။ Script အတွင်းရှိ Redis Command များကို `redis.call("command", ARG1, ARG2, ...)` ဟုသော method ကို အသုံးပြုသည်ကို သတိထားမိမည်ဖြစ်သည်။

Lua နှင့်မရင်းနှီးပါက တစ်လိုင်းချင်းစီ ဖတ်ကြည့်သင့်သည်။ `{}` သည် empty `table` တစ်ခု ဖန်တီးပေးပြီး (array သို့မဟုတ် dictionary ကဲ့သို့ အလုပ်လုပ်သည်) `#TABLE` ဖြင့် TABLE အတွင်းရှိ element များကို ရယူပြီး `..` ဖြင့် string များကို concat လုပ်ရာတွင် အသုံးပြသည်။

`eval` သည် တတယ်တော့ parameter ၄ ခုလက်ခံပါသည်။ ဒုတိယ parameter သည် key အရေအတွက်ဖြစ်သင့်သော်လည်း Ruby driver မိမိ

တို့အတွက် အလိုအလျောက် create ပေးသည်။ အဘယ်ကြောင့်လုပ်ပေးသနည်း။ အောက်ကအတိုင်း CLI မှ execute ပြုလုပ်သည်ဟု ယူဆပါ။

```
eval "....." "friends:leto" "m"
vs
eval "....." 1 "friends:leto" "m"
```

ပထမတစ်ကြောင်း (အမှား) တွင် Redis အနေဖြင့် မည်သည်က key မည်သည်က value ဆိုသည်ကို သိနိုင်မည်နည်း။ ဒုတိယတစ်ကြောင်းတွင်မူ ထိုသို့ ပြဿနာများမရှိပါ။

ထိုကြောင့် မေးစရာနောက်တစ်ခုပေါ်လာသည်။ အဘယ်ကြောင့် keys များကို list လုပ်ပေးရန်လိုသနည်း။ Redis တွင်ရှိသော command တိုင်း execution time တွင် မည်သည် keys များလိုအပ်သည်ကို သိရှိရန်လိုပါသည်။ ထိုမှသာ Redis Cluster ကဲ့သို့သော tools များတွင် Redis server များအတွက် request များကို ခွဲဝေပေးနိုင်မည်ဖြစ်သည်။ အပေါ်မှ ဥပမာတွင် key များကို `eval` တွင် pass ပြုလုပ်စရာမလိုပဲ dynamic အနေဖြင့် ဆောင်ရွက်သွားသည်ကို တွေ့ရမည်ဖြစ်သည်။ `hget` သည် Leto ၏ ယောင်္ကျားလေး သူငယ်ချင်းအားလုံးကို ရယူပေးမည်ဖြစ်သည်။ အဘယ်ကြောင့်ဆိုသော် key များကိုကြို၍ list ပြုလုပ်ခြင်းသည် တင်းကျပ်သော စည်းမျဉ်းတစ်ခုဖြစ်သည်။ အပေါ်မှ code သည်ါ single-instance အနေဖြင့်သာမက replication တွင် ကောင်းကောင်း run မည်ဖြစ်သော်လည်း Redis Cluster တွင်အလုပ်လုပ်မည်မဟုတ်။

Script Management

`eval` မှ run သော script များသည် Redis မှ cache ပြုလုပ်သေးသော်ည လး်ံ လိုချင်သည့်အခါတိုင်း body ကို sent ရချင်သည် အလုပ်မတွင်လှ ချေ။ ထိုကြောင့် Redis တွင် ထို script ကိုကြေညာပြီး key များကိုသာ execute ပြုလုပ်နိုင်သည်။ ထိုသို့ပြုလုပ်ရန် `script load` command ကို အသုံးပြုနိုင်ပြီး ၎င်းမှာ script ၏ SHA1 digest ကို return ပြန်ပေးသည်။

```
redis = Redis.new
script_key = redis.script(:load, "THE_SCRIPT")
```

script ကို load ပြီးသည်နှင့် `evalsha` ကိုအသုံးပြု၍ execute ပြုလုပ် နိုင်သည်။

```
redis.evalsha(script_key, ['friends:leto'], ['m'])
```

`script kill`၊ `script flush` နှင့် `script exists` တို့သည် Lua script များ ကို manage ပြုလုပ်နိုင်သော command များဖြစ်ပြီး script များ run ရာ ၊ internal cache ထဲမှာ script များ remove လုပ်ရာတွင်၊ cache အတွင်းနှင့် script ရှိမရှိ ဆန်းစစ်ရာတွင် အသုံးပြုသည်။

Library များ

Redis ၏ Lua implementationတွင် အသုံးဝင်လှသည့် library များပါ ပါဝင်သည်။ `table.lib`, `string.lib` နှင့် `math.lib` တို့မှာ အလွန်အသုံးဝင်

သော်လည်း ကျွန်တော်အကြိုက်ဆုံးမှာ `cjson.lib` ဖြစ်သည်။ သင့်အနေဖြင့် multiple argument များအနေဖြင့်script ကို pass ပြုလုပ်ရပါက JSON အနေဖြင့်ဆို ပို၍ clean ဖြစ်မည်ဖြစ်သည်။

```
redis.evalsha ".....", [KEY1], [JSON.fast_generate({gender: 'm', ghola: true})]
```

ထိုနောက် Lua Script အတွင်းအောက်ပါအတိုင်း deserialize ပြုလုပ်နိုင်သည်။

```
local arguments = cjson.decode(ARGV[1])
```

JSON library သည် Redis အတွင်းရှိ value များကို parse လုပ်ရာတွင်လည်း အသုံးဝင်သည်။ အပေါ်မှဥပမာကို အောက်ပါအတိုင်း ပြန်ပြင်ရေး၍ရသည်။

```
  local friend_names = redis.call('smembers', KEYS[1])
  local friends = {}
  for i = 1, #friend_names do
        local friend_raw = redis.call('get', 'user:' .. friend_names[i])
    local friend_parsed = cjson.decode(friend_raw)
    if friend_parsed.gender == ARGV[1] then
      table.insert(friends, friend_raw)
    end
  end
  return friends
```

gender ကို hash field မှ ယူမည့်အစား friend data မှယူ၍လည်းရသည်။ (၎င်းသည်ပို၍ နှေးပြီး မူလပုံစံကို ပို၍သဘောကျသော်လည်း ဖြစ်နိုင်သည်ကိုပြခြင်းဖြစ်သည်)

Atomic

Redis သည် thread တစ်ခုတည်းဖြစ်သဖြင့် Lua script တစ်ခုကို နောက် Redis command တစ်ခုမှ ကြားဖြတ်မည်ကို စိုးရိမ်ရန်မလိုပေ။ ၎င်း၏ အထင်ရှားဆုံးအားသာချက်တစ်ခုမှ key များ၏ TTL သည် execution ပြုလုပ်နေချိန်တွင် expire ဖြစ်မည်မဟုတ်ပါ။ အကယ်၍ key တစ်ခုသည် script ၏အစတွင်ရှိပါက မဖျက်သေးသရွေ့ရှိနေမည်ဖြစ်သည်။

Administration

နောက်အခန်းတွင် Redis administration နှင့် configuration အကြောင်း အသေးစိတ်၍ပြောပါမည်။ ယခုတွင်တော့ `lua-time-limit` သည် Lua script terminate မပြုလုပ်ခင် execution ကြာချိန်ကို limit လုပ်ခြင်း ဖြစ်သည်။ default မှာ ၅ စက္ကန့်ဖြစ်ပြီး ထိုထက်ပို၍လျှော့နိုင်သည်။

ယခုအခန်းတွင်

ယခုအခန်းတွင် Redis ၏ Lua scripting လုပ်နိုင်စွမ်းကို မိတ်ဆက်ပေးခဲ့ပြီ ဖြစ်သည်။ ၎င်း feature အသုံးမတက်ပါက ဘေးဖြစ်နိုင်သော်လည်း မိမိ တို့၏ custom command နှင့် အထူးပြု command များဖန်တီးနိုင်ပြီး code များကိုရှင်းလင်းရုံသာမက performance လည်ပိုမိုကောင်းပါသည်။ Lua scripting သည် အခြား redis command များကဲ့သို့ပင် အစောပိုင်းတွင် အခက်အခဲရှိမည်ဖြစ်ပြီး နောက်ပိုင်းတဖြည်းဖြည်းရင်းနှီးလာပါလိမ့်မည်။

အခန်း (၆) - Administration

အခုနောက်ဆုံး အခန်းသည် Redis ကို run ရာတွင် administration လုပ်သည့် အပိုင်းကို အဓိကထားပြီး ပြောသွားမည်ဖြစ်သည်။ သို့သော် ၎င်းသည် Redis adminstration အတွက် ပြီးပြည့်စုံသွားမည် ဟု ဆိုလိုသည် မဟုတ်ပဲ Redis ကိုစတင် အသုံးပြုသူအတွက် အခြေခံမေးခွန်းများကို ဖြေနိုင်မည်ဖြစ်သည်။

Configuration

ပထမဆုံး Redis Server ကို launch လုပ်လိုက်ချိန်တွင် `redis.conf` ဟုသော file ကိုမတွေ့ဟု warn လုပ်လိမ့်မည်ဖြစ်သည်။ ၎င်း file သည် Redis ၏ အရာတော်တော်များများကို config ပြုလုပ်ရာတွင် အသုံးပြုသည်။ အသေအချာ document ပြုလုပ်ထားသော `redis.conf` သည် Redis version တိုင်းတွင်ပါရှိပြီး default config option များ ပါဝင်သဖြင့် မည်သည့် setting များက ဘာလုပ်သည်နှင့် default value များကိုလေ့လာနိုင်သည်။ ၎င်းကို http://download.redis.io/redis-stable/redis.conf တွင် တွေ့နိုင်သည်။

ထို file သည် well-document ဖြစ်သဖြင့် setting ကိုထပ်၍မပြောတော့ပေ။

`redis.conf` အပြင် Redis တွင် `config set` ဟု value တစ်ခုချင်းစီကို set လုပ်နိုင်သည့် command လည်းရှိသေးသည်။ `slowlog-log-slower-than` ကို 0 အဖြစ်သတ်မှတ်စဉ်က အသုံးပြုခဲ့ပြီးဖြစ်သည်။

ထိုအပြင် setting value ကိုဖော်ပြသည့် `config get` လည်းရှိပါသေးသည်။ ထို command သည် pattern matching ကို support လုပ်သည်။ အကယ်၍ logging နှင့်ပတ်သတ်သည့်အရာအားလုံးကိုသိလိုပါက အောက်ပါအတိုင်း လုပ်နိုင်သည်။

```
config get *log*
```

Authentication

Redis ကို password ဖြင့် config လုပ်နိုင်သည်။ ၎င်းသည် `redis.conf` မှ ဖြစ်စေ၊ `config set` ဖြစ်စေ `requirepass` ကိုအသုံးပြုခြင်းဖြင့် set နိုင်သည်။ `reqiurepass` သည် value ကို set လိုက်ပါက client များသည် `auth password` ကိုအသုံးပြုရန်လိုသည်။

Client မှ authenticate ဖြစ်သည်နှင့် database အတွင်းရှိ command တိုင်းကို run နိုင်သည်။ ထိုကြောင့် database အားလုံးမှ key အားလုံးကို ဖျက်ပစ်သော `flushall` ကဲ့သို့သော key များလည် ပါဝင်သည်။ configuration ကို အသုံးပြုပြီး ပိုမိုရှုပ်ထွေးသော နာမည်ပေးခြင်းဖြင့် ရှောင်ရှားနိုင်သည်။

```
rename-command CONFIG 5ec4db169f9d4dddacbfb0c26ea7e5ef
rename-command FLUSHALL 1041285018a942a4922cbf76623b741e
```

သို့မဟုတ် နာမည်အသစ်ကို empty string ပေးခြင်းဖြင့်လည်း disable ပြုလုပ်နိုင်သည်။

Size Limitations

Redis ကိုစတင်အသုံးပြုနေစဉ်ကတည်းက တွေးမိနိုင်ပါသည်။ “ငါ key ဘယ်နှစ်ခုလောက်ထည့်၍ရမလဲ”၊ “သင့် data ကို organize ပြုလုပ်ရာတွင် အထူးသဖြင့် hash တစ်ခုတွင် field ဘယ်နှစ်ခုပါ၍ရနိုင်နည်း”၊ “list တစ်ခု ဝါ set တစ်ခုတွင် element မည်မျှအထိ ထည့်၍ရသနည်း”။ တကယ်လက်တွေ့တွင် ၎င်း limit များသည် သန်းရာနဲ့ချီအထိ ဖြစ်သည်။

Replication

Redis တွင် replication ကို support ပြုလုပ်သဖြင့် Redis instance တစ်ခု (master) ကို write ခြင်းဖြင့် တစ်ခုထက်ပိုသော အခြားသော instance (slave) များ ကို အလိုအလျောက် up-to-date ဖြစ်အောင် config လုပ်၍ရသည်။ slave အနေဖြင့် config လုပ်ရန် `slaveof` ကို setting file တွင်၎င်း ၊ command အနေဖြင့်၎င်း အသုံးပြုနိုင်သည်။ (ထို config မပါပဲသုံးပါက master ဖြစ်နေမည်ဖြစ်သည်။

Replication သည် မတူညီသော server များတွင် သင့် data များကို ပွားထားခြင်းဖြင့် protect လုပ်ပေးမည်ဖြစ်ပြီး Replication သည် slave များမှ

read နိုင်သောကြောင့် performance လည်းပိုကောင်းစေနိုင်သည်။ ၎င်းသည် အနည်းငယ် နောက်ကျနေသော data များကို respond လုပ်မည်ဖြစ်သော်လည်း app အတော်များများအတွက် ထိုက်တန်သော tradeoff ဖြစ်သည်။

သို့သော် Redis replication သည် automate failover မပါရှိသဖြင့် master သေသွားပါက slave သည် manually အနေဖြင့် promote ပြုလုပ်ရန်လို သည်။ ထုံတမ်းစဉ်လာ heartbeat monitoring tools များနှင့် script များကို အသုံးပြု၍ စောင့်ကြည့်ရခြင်းသည် လက်ရှိတော့ ခေါင်းကိုက်စရာဖြစ်နေ ပါသေးသည်။

Backups

Redis တွင် backup ပြုလုပ်ခြင်းသည် မည်သည့်နေရာကမဆို (S3၊ FTP…) Redis snapshot ကို copy လုပ်ခြင်းသာဖြစ်သည်။ Default အနေဖြင့် Redis သည် snapshot ကို `dump.rdb` ဟု အမည်ဖြင့် save လေ့ရှိ ပြီး မည်သည့်အခြေအနေမဆို သင့်အနေဖြင့် `scp`၊ `ftp` သို့မဟုတ် `cp` နှင့် အခြားသော command များအသုံးပြု၍ ကူးယူနိုင်သည်။

master အတွက် snapshot ပြုလုပ်ခြင်းနှင့် append ပြုလုပ်ခြင်း (aof) ကို disable လုပ်လေ့လုပ်ထရှိပြီး slave တွင်သာ on ထားတက်သည်။ ထိုသို့

ဖြင့် master တွင် load ကို လျှော့ချနိုင်ပြီး ပို၍ parameter များကို slave တွင် save နိုင်ပြီး system ၏ responsiveness ကိုထိခိုက်ခြင်းမရှိလှပါ။

Scaling နှင့် Redis Cluster

Replication သည် site တစ်ခု grow ဖြစ်လာသောအခါ ပထမဆုံးအသုံးပြုနိုင်သည့် tool ဖြစ်သည်။ အချို့သော command များသည် အခြားသော command များထက်ပို၍ expensive ဖြစ်ပြီး (ဥပမာ `sort` ကဲ့သို့) ၎င်း execution များကို slave များကို ခွဲပေးလိုက်ခြင်းဖြင့် ထပ်၍လာသော query များအတွက် ပို၍ responsive ဖြစ်သော system ဖြစ်ပါလိမ့်မည်။

ထိုမှကျော်လွန်၍ Redis ကို တကယ် scale ပြုလုပ်ခြင်းသည် Redis instance များစွာတွင် သို့ keys များစွာကို distribute ပြုလုပ်ထားခြင်းဖြစ်သည်။ (Redis သည် single thread ဖြစ်သဖြင့် Box တစ်ခုထဲတွင် run သည်လည်းဖြစ်နိုင်သည်) ထိုအချိန်ကာလအတွင်း သင်အနေဖြင့် ကိုင်တွယ်စရာတစ်ခုရှိပါသည် (Redis driver အတော်များများ support ပြုလုပ်ထားသော်လည်း) Horizontal distribution သည် ဒီစာအုပ်ထဲတွင်မပါ။ သင့်အနေဖြင့် အချိန်ကာလတော်တော်များများအထိ ပူစရာမလိုသော်လည်း ဘယ် solution ကိုသုံးသည်မဆို အချိန်ကြာလှသည့်အမျှ စဉ်းစားရမည့်အချက်ဖြစ်သည်။

သတင်းကောင်းမှာ ထိုအလုပ်သည် Redis Cluster တွင်ပါရှိပြီး ၎င်းသည် horizontal scaling အတွက် support လုပ်ရုံသာမက reblancing အတွက်ပါ ပါဝင်ပြီး high availabity ဖြစ်ရန် အလိုအလျောက် ပြန်စသည် စနစ်ပါရှိ မည်ဖြစ်သည်။

သင့်အနေဖြင့် အချိန်နှင့် အားစိုက်မှု ပါရှိပါက High availability နှင့် scaling သည် ရရှိနိုင်သည့် ရလဒ်တစ်ခုဖြစ်ပြီး ထိုထက်ပို၍ Redis Cluster များသုံး ခြင်းသည် ပို၍လွယ်ကူသည်။

ယခုအခန်းတွင်

Redis ဖြင့် projects များ၊ sites များ အသုံးပြုပြီးကာလ Redis သည် production ready ဖြစ်သည်ကို အငြင်းပွားရန်မရှိပေ။ သို့သော် အချို့သော tool များ အထူးသဖြင့် security နှင့် availabilty အတွက် tools များသည် သိပ်မကြာလှသေးပေ။ Redis Cluster သည် လက်ရှိ management challenge များကို ဖြေရှင်း ပေးမည်ဟု မျှော်လင့်ရသည်။

နိဂုံး

Redis သည် data များကို ကိုင်တွယ်ရတာတွင် ရိုးရှင်းမှု၏ သင်္ကေတ တစ်ခု ဖြစ်သည်။ အခြား system များတွင်ပါရှိတက်သော complexity နှင့် abstraction များကို ခွဲခြားပေးနိုင်သည်။ ထိုကြောင့် Redis ကို မှားယွင်းစွာ အသုံးပြုမိလေ့ရှိသည်။ အချို့ ကိစ္စများတွင်မူ Redis သည် သင့် data နှင့် အံကိုက်ဖြစ်လိမ့်မည်။

အနှစ်ချုပ်ရပါလျှင် Redis သည် သင်ကြားရလွယ်ကူသည်။ တခြား နည်းပညာအသစ်များစွာရှိပြီး ၎င်းတို့ကို လေ့လာရန် ထိုက်တန်ခြင်းရှိမရှိကို ဆန်းစစ်ရသည်မှာ လွယ်သည်မဟုတ်။ Redis ၏ တကယ့် အကျိုးကျေးဇူးနှင့် ၎င်းရိုးရှင်းမှုတို့ကို နှိုင်းစာပါလျှင် သင့်နှင့် သင်၏ team အတွက် ထိုက်တန်သော ရင်းနှီးမြုပ်နှံမှုတစ်ခုဟု မြင်မိပါသည်။