

Redis Modules TL;DR

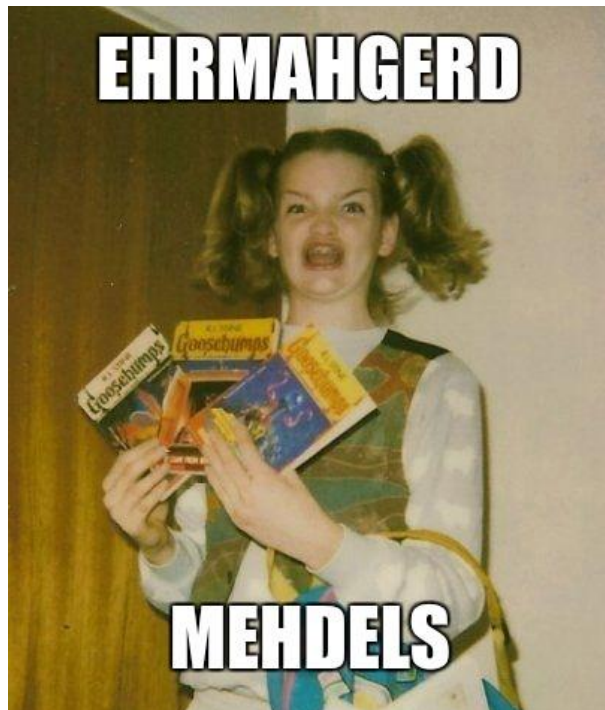
Dvir Volk,
Senior Architect, Redis Labs



redisconf
2016

So... Modules, huh?

- I've been waiting 6 years
- Will change Redis forever IMO
- The key is the API concept



What **Modules** Actually Are

- Dynamically loaded libraries into redis
- Extend redis with new commands
- Written in C (C++ if you really insist)
- Using a C API designed to isolate redis internals

The Race For JS Modules is ON!



Atwood's Law

Any application that *can* be written in JavaScript, *will* eventually be written in JavaScript.

What Modules Can't Do (YET)

- Register new data structures *
- Override existing commands *
- Block
- Not the entire API of redis is accessible directly
- All Coming Soon

** Not completely true!*

Modules in Action

```
127.0.0.1:9875> MODULE LOAD example.so
```

```
OK
```

```
127.0.0.1:9875> HSET foo bar baz
```

```
(integer) 1
```

```
127.0.0.1:9875> HGETSET foo bar w00t!
```

```
"baz"
```

```
127.0.0.1:9875> HGETSET foo bar wat?
```

```
"w00t!"
```

The Main API Layers

High Level API

- Generic RedisCall
- Basically Lua on steroids

Operational API

- Compose responses, examine call replies
- Memory Management

Low Level API

- Selected parts of redis commands (L*, Z*)
- String DMA

Main Building Blocks

RedisModuleString*

Memory managed strings
for the module

Opaque to the developer

Args and replies

RedisModuleCallReply*

Returned from most calls

Like the protocol can be:

- Strings
- Arrays
- Integers
- Errors
- Null

RedisModuleKey*

References to keys we
are working with, mainly
for the lower level API

Acquired with Open

Released automatically
or with Close

RedisModuleCtx *

- Redis' execution context for a command
- Opaque to the module developer
- Passed to most functions
- Behind the scenes manages resources
 - Allocated Objects
 - Opened Keys
 - Automatic free at exit

High Level API

```
RedisModuleCallReply *rep =  
RedisModule_Call(ctx,  
    "HGET",  
    "cl",  
    "foo",  
    1337);
```

ENDLESS POSSIBILITIES

High Level API

- More or less like LUA's API
- Generic **RedisModule_Call**
- Examine replies with **RedisModule_CallReplyType**
- Reply to the client with **RedisModule_ReplyWith***
- Slower than lower level API, but flexible

Low Level API

- MUCH faster



Low Level API

- MUCH faster
- Hash Get/Set

Low Level API

- MUCH faster
- Hash Get/Set
- ZSET operations

Low Level API

- MUCH faster
- Hash Get/Set
- ZSET operations
- ZSET Iterators

Low Level API

- MUCH faster
- Hash Get/Set
- ZSET operations
- ZSET Iterators
- String DMA / truncate

Low Level API

- MUCH faster
- Hash Get/Set
- ZSET operations
- ZSET Iterators
- String DMA / truncate
- List push/pop

Low Level API

- MUCH faster
- Hash Get/Set
- ZSET operations
- ZSET Iterators
- String DMA / truncate
- List push/pop
- Expire

Low Level API

- MUCH faster
- Hash Get/Set
- ZSET operations
- ZSET Iterators
- String DMA / truncate
- List push/pop
- Expire
- More to come!

String DMA

- Use Redis strings as raw memory
- Zero Copy Overhead
- Resize strings with **RedisModule_Truncate**
- BYODS - Bring Your Own Data Structure

Shameless Plug: RedisModulesSDK

- Basic Module Template
- Complete Documentation
- Automating Boring Stuff
 - Argument Parsing
 - String Manipulation
 - Response Validation
 - Testing

<https://github.com/RedisLabs/RedisModulesSDK>

Let's Make a Module!

```
HGETSET <key> <element> <newval>
```

Step 1: Command Handler

```
#include "redismodule.h"

/* HGETSET <key> <element> <value> */
int HGetSetCmd(RedisModuleCtx *ctx,
               RedisModuleString **argv, int argc) {

    return REDISMODULE_OK;
}
```

Step 2: validate args

```
/* HGETSET <key> <element> <value> */  
int HGetSetCmd(RedisModuleCtx *ctx, RedisModuleString  
**argv, int argc) {  
    if (argc != 4) {  
        return RedisModule_WrongArity(ctx);  
    }  
  
    return REDISMODULE_OK;  
}
```


Step 3: Auto Memory

```
/* HGETSET <key> <element> <value> */  
int HGetSetCmd(RedisModuleCtx *ctx, RedisModuleString  
**argv, int argc) {  
    if (argc != 4) {  
        return RedisModule_WrongArity(ctx);  
    }  
  
    RedisModule_AutoMemory(ctx);  
    return REDISMODULE_OK;  
}
```

Step 4: Making Calls!

```
/* HGETSET <key> <element> <value> */  
int HGetSetCmd(RedisModuleCtx *ctx, RedisModuleString  
**argv, int argc) {  
    ...  
    RedisModuleCallReply *rep = RedisModule_Call(ctx,  
        "HGET", "ss", argv[1], argv[2]);  
  
    RMUTIL_ASSERT_NOERROR(rep)  
    ...  
}
```

Step 4: MOAR CALLZ PLZ

```
/* HGETSET <key> <element> <value> */  
int HGetSetCmd(RedisModuleCtx *ctx, RedisModuleString  
**argv, int argc) {  
    ...  
    RedisModuleCallReply *srep = RedisModule_Call(ctx,  
        "HSET", "sss", argv[1], argv[2], argv[3]);  
    REDIS_ASSERT_NOERROR(srep)  
  
    return REDISMODULE_OK;  
}
```

Step 5: Returning a reply

```
/* HGETSET <key> <element> <value> */  
int HGetSetCmd(RedisModuleCtx *ctx, RedisModuleString  
**argv, int argc) {  
    ...  
  
    RedisModule_ReplyWithCallReply(ctx, rep);  
  
    return REDISMODULE_OK;  
}
```

Step 6: Initializing

```
int RedisModule_OnLoad(RedisModuleCtx *ctx) {  
  
    if (RedisModule_Init(ctx, "EXAMPLE", 1, REDISMODULE_APIVER_1)  
        == REDISMODULE_ERR) return REDISMODULE_ERR;  
  
    if (RedisModule_CreateCmd(ctx, "HGETSET",  
        HGetSetCommand, "write", 1, 1, 1) == REDISMODULE_ERR)  
        return REDISMODULE_ERR;  
  
    return REDISMODULE_OK;  
}
```

Step 6: Initializing

```
int RedisModule_OnLoad(RedisModuleCtx *ctx) {  
  
    if (RedisModule_Init(ctx, "EXAMPLE", 1, REDISMODULE_APIVER_1)  
        == REDISMODULE_ERR) return REDISMODULE_ERR;  
  
    if (RedisModule_CreateCmd(ctx, "HGETSET",  
        HGetSetCommand, "write", 1, 1, 1) == REDISMODULE_ERR)  
        return REDISMODULE_ERR;  
  
    return REDISMODULE_OK;  
}
```

Building and Makefile

- No special Linking required
- Just **#include “redismodule.h”**

```
CFLAGS = -Wall -fPIC
LDFLAGS = -shared -Bsymbolic -lc

module.so: module.o
    $(LD) -o $@ module.o $(LDFLAGS)
```

IT WORKS!

```
127.0.0.1:9875> MODULE LOAD example.so
```

```
OK
```

```
127.0.0.1:9875> HSET foo bar baz
```

```
(integer) 1
```

```
127.0.0.1:9875> HGETSET foo bar w00t!
```

```
"baz"
```

```
127.0.0.1:9875> HGETSET foo bar wat?
```

```
"w00t!"
```

ENDLESS POSSIBILITIES

The Low Level API

ENDLESS POSSIBILITIES



redisconf
2016

Low Level API: Example

```
/* ZSUMRANGE <key> <startscore> <endscore> */  
int ZsumRange_RedisCommand(RedisModuleCtx *ctx,  
RedisModuleString **argv, int argc) {  
  
    return REDISMODULE_OK;  
  
}
```

Low Level API: Read Arguments

```
int ZsumRange_RedisCommand(...) {  
    double min, max;  
    double sum = 0;  
  
    if (MUtil_ParseArgs(argv, argc, 2, "dd", &min, &max)  
        != REDISMODULE_OK) {  
        RedisModule_WrongArity(ctx);  
    }  
    ...  
}
```

Low Level API: Open The Key

```
int ZsumRange_RedisCommand(...) {  
    ...  
    RedisModuleKey *k =  
        RedisModule_OpenKey(ctx, argv[1], REDISMODULE_READ);  
  
    if (RedisModule_KeyType(k) != REDISMODULE_KEYTYPE_ZSET) {  
        return RedisModule_ReplyWithError(...);  
    }  
  
}
```

Low Level API: Iterate ZSET

```
int ZsumRange_RedisCommand(...) {  
    ...  
    // Open The iterator  
    RedisModule_ZsetFirstInScoreRange(k, min, max, 0, 0);  
  
    while(!RedisModule_ZsetRangeEndReached(k)) {  
        ...  
        RedisModule_ZsetRangeNext(k);  
    }  
    RedisModule_ZsetRangeStop(k);  
}
```

Low Level API: Read Iterator Values

```
while (!RedisModule_ZsetRangeEndReached(k)) {  
  
    double score;  
    RedisModuleString *ele =  
        RedisModule_ZsetRangeCurrentElement(k, &score);  
  
    RedisModule_FreeString(ctx, ele);  
    sum += score;  
    RedisModule_ZsetRangeNext(k);  
  
}
```

Low Level API: Return the value

```
/* ZSUMRANGE key startscore endscore */
int ZsumRange_RedisCommand(RedisModuleCtx *ctx,
    RedisModuleString **argv, int argc) {
    ...

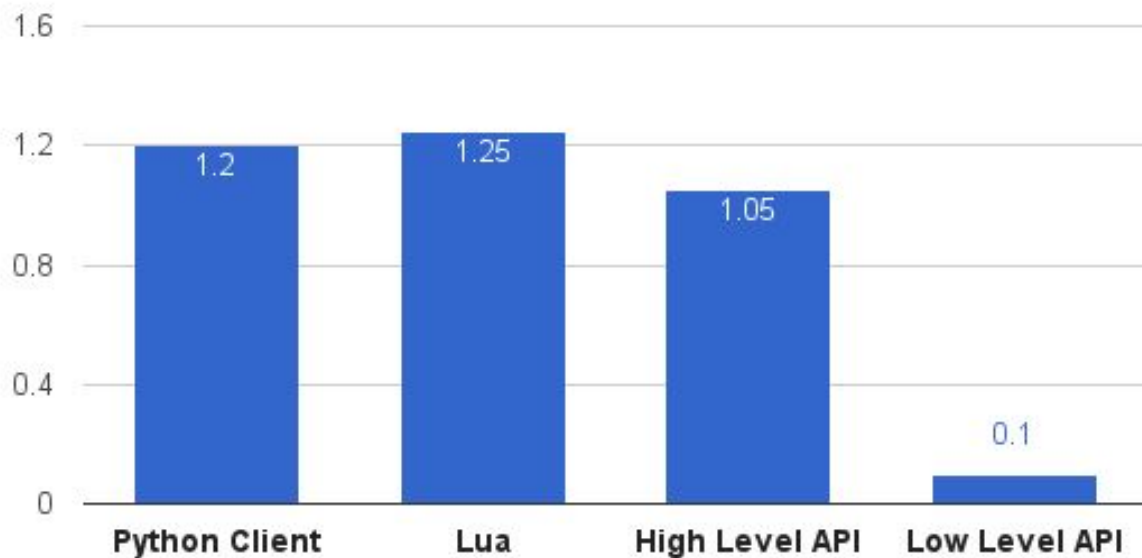
    RedisModule_CloseKey(key);
    RedisModule_ReplyWithDouble(ctx, sum);

    return REDISMODULE_OK;
}
```

A Little Benchmark

- Sum the scores of 1,000,000 ZSET Elements
- Python client, Lua script, High/Low Level Modules
- Low Level API uses iterators
- The rest use ZRANGEBYSCORE

Benchmark: Results (seconds)



KTHXBAI!



redisconf
2016