

Redis协议规范

网络层

请求 - 响应模型

RESP 协议描述

RESP 单行字符串(+ Simple Strings)

示例

RESP 错误信息(- Errors)

示例

RESP 整型数据(: Integers)

示例

RESP 多行字符串(\$ Bulk Strings)

示例

RESP 数组(* Arrays)

示例

发送命令到 Redis 服务端

零声学院 C/C++Linux服务器开发/高级架构师

<https://ke.qq.com/course/420945>

官网地址: <https://redis.io/topics/protocol>

参考测试代码 `redis-protocol.c`

Redis客户端使用名为RESP（Redis序列化协议）的协议与Redis服务器进行通信。虽然该协议是专为Redis设计的，但它可以用于其他CS软件项目的通讯协议。

RESP是以下几方面的考虑：

- 易于实现
- 快速解析
- 可读性高

RESP可以序列化不同的数据类型，如整型，字符串，数组。 还有一种特定的错误类型。 请求将要执行的命令作为字符串数组从Redis客户端发送到Redis服务器。Redis使用特定数据类型的命令进行回复。RESP是二进制安全的，不需要处理从一个进程传输到另一个进程的批量数据，因为它使用前缀长度来传输批量数据。

注意： 此处概述的协议仅用于客户端 – 服务器通信。 Redis Cluster使用不同的二进制协议，以便在节点之间交换消息。

网络层

客户端连接到Redis服务器，是创建TCP连接到端口6379。

虽然RESP在技术上是非TCP特定的，但在Redis的上下文中，协议仅用于TCP连接（或类似的面向流的连接，如Unix套接字）。

请求 – 响应模型

Redis接受由不同参数组成的命令。 收到命令后，将对其进行处理并将回复发送回客户端。

这是最简单的模型，但有两个例外：

- Redis支持流水线操作（本文档稍后介绍）。 因此，客户端可以一次发送多个命令，并等待稍后的回复。
- 当Redis客户端处于 Pub/Sub 时，协议会更改语义并成为推送协议，即客户端不再需要发送命令，因为服务器会在它们接收到命令时发自动向客户端发送新消息。

排除上述两个例外，Redis协议是一个简单的请求 – 响应协议。

RESP 协议描述

RESP协议在Redis 1.2中引入，但它成为与Redis 2.0中的Redis服务器通信的标准方式。 这是每一个Redis客户端中应该实现的协议。

RESP实际上是一个支持以下数据类型的序列化协议：单行字符串，错误信息，整型，多行字符串和数组。

RESP在Redis中用作请求 – 响应协议的方式如下：

- 客户端将命令作为字符串数组发送到Redis服务器。
- 服务器根据命令实现回复一种RESP类型数据。

在 RESP 中，一些数据的类型通过它的第一个字节进行判断：

- 单行(Simple Strings)回复：回复的第一个字节是 "+"
- 错误(Errors)信息：回复的第一个字节是 "-"
- 整形数字(Integers)：回复的第一个字节是 ":"
- 多行字符串(Bulk Strings)：回复的第一个字节是 "\$"
- 数组(Arrays)：回复的第一个字节是 "*"

此外，RESP能够使用稍后指定的Bulk Strings或Array的特殊变体来表示Null值。

在RESP中，协议的不同部分始终以“\r\n”（CRLF）结束。

必须先明确一点，执行同样的命令，错误和正确返回的类型有可能一样，也有可能不一样。

比如：

- SETNX 错误和正确返回的类型一致(Integers)，都是整数的方式返回。
- INCR 错误的时候返回的是错误(Errors)信息， 正常的时候返回的是整形(Integers)。

抓包的时候加上-XX

```
sudo tcpdump -i any dst host 127.0.0.1 and port 6379 -XX
```

RESP 单行字符串(+ Simple Strings)

简单字符串按以下方式编码：+号字符，后跟不能包含CR或LF字符的字符串（不允许换行），由CRLF终止（即“\r\n”，对应十六进制 0x0D, 0x0A）。

Simple Strings用于以最小的开销传输非二进制安全字符串。例如，很多Redis命令成功回复时只有“OK”，因为RESP 单行字符串使用以下5个字节进行编码：

```
1 "+OK\r\n"
```

为了发送二进制安全字符串，使用RESP 多行字符串代替。

当Redis使用Simple String回复时，客户端库应该向调用者返回一个字符串，该字符串由“+”之后的第一个字符组成，直到字符串结尾，不包括最终的CRLF字节。

对应hiredis的返回值为：[REDIS_REPLY_STATUS](#)

示例

```
1 void testSimpleString(redisContext *context)
2 {
3     // Set Key Value
4     const char *key = "str";
5     const char *val = "Hello World";
6     /*SET key value */
7     redisReply *reply = (redisReply *)redisCommand(context, "SET %s
  %s", key, val);
8     printf("reply->type: %d\n", reply->type);
9     if (reply->type == REDIS_REPLY_STATUS)
10    {
11        /*SET str Hello World*/
12        printf("SET %s %s\n", key, val);
13    }
```

```

14     printf("reply->str: %s\n", reply->str);
15
16     freeReplyObject(reply);
17 }

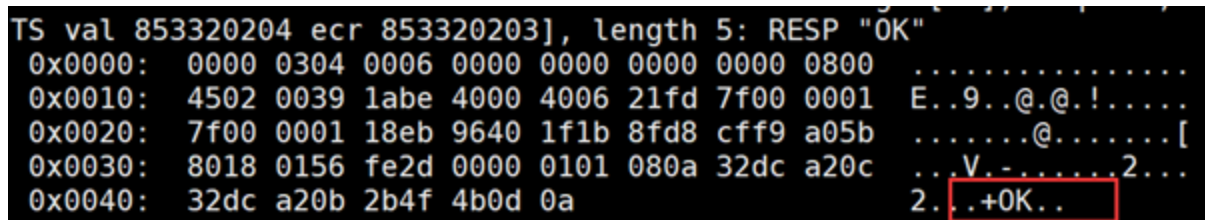
```

打印:

reply->type: 5

SET str Hello World

reply->str: OK



```

TS val 853320204 ecr 853320203], length 5: RESP "OK"
0x0000: 0000 0304 0006 0000 0000 0000 0000 0800 .....
0x0010: 4502 0039 1abe 4000 4006 21fd 7f00 0001 E..9..@.@.!....
0x0020: 7f00 0001 18eb 9640 1f1b 8fd8 cff9 a05b .....@.....[
0x0030: 8018 0156 fe2d 0000 0101 080a 32dc a20c ...V.-.....2...
0x0040: 32dc a20b 2b4f 4b0d 0a 2. .+OK..

```

从抓包可以看出来具体的情况。

如果使用SETNX 去测试, 成功返回1, 错误返回0, 该命令和SET的返回是不一样的。

RESP 错误信息(- Errors)

RESP具有错误的特定数据类型。实际上错误与RESP 单行字符串完全相同, 但第一个字符是减号'-'字符而不是+号。

RESP中单行字符串和错误之间的真正区别在于客户端将错误视为异常, 组成错误类型的字符串是错误消息本身。

基本格式如下:

```

1 "-Error message\r\n"

```

错误回复仅在发生错误时发送, 例如, 如果您尝试对错误的数据类型执行操作, 或者命令不存在等等。收到错误回复时, 客户端应将异常抛出。

以下是错误回复的示例:

```

1 -ERR unknown command 'foobar'
2 -WRONGTYPE Operation against a key holding the wrong kind of value

```

“-”之后的第一个单词，直到第一个空格或换行符，表示返回的错误类型。这只是Redis使用的约定，不是RESP错误格式的一部分。

例如，ERR是一般错误，而WRONGTYPE是一个更具体的错误，意味着客户端尝试对错误的数据类型执行操作。这称为错误前缀，是一种允许客户端理解服务器返回的错误类型的方法，而不依赖于给定的确切消息，这可能随时间而变化。

客户端实现可以针对不同的错误返回不同类型的异常，或者可以通过直接将错误名称作为字符串提供给调用者来提供捕获错误的通用方法。

但是，这样的功能不应该被认为是至关重要的，因为它很少有用，并且有限的客户端实现可能只返回通用的错误条件，例如false。

示例

```
1 127.0.0.1:6379> SET teacher darren
2 OK
3 127.0.0.1:6379> INCR teacher
4 (error) ERR value is not an integer or out of range
```

INCR teacher的返回错误

```
0x0000: 0000 0304 0006 0000 0000 0000 0000 0800 .....
0x0010: 4502 0062 910b 4000 4006 ab86 7f00 0001 E..b..@.@.....
0x0020: 7f00 0001 18eb 965c fb90 66a6 2d42 daa7 .....\.f.-B..
0x0030: 8018 0156 fe56 0000 0101 080a 32ef 1729 ...V.V.....2..)
0x0040: 32ef 1728 2d45 5252 2076 616c 7565 2069 2..(-ERR)value.i
0x0050: 7320 6e6f 7420 616e 2069 6e74 6567 6572 s.not.an.integer
0x0060: 206f 7220 6f75 7420 6f66 2072 616e 6765 .or.out.of.range
0x0070: 0d0a ..
```

hiredis实现

```
1 void testError(redisContext *context)
2 {
3     // Set Key Value
4     printf("SET\n");
5     const char *key = "teacher";
6     const char *val = "darren";
7     /*SET key value */
8     redisReply *reply = (redisReply *)redisCommand(context, "SET %s
    %s", key, val);
9     printf("reply->type: %d\n", reply->type);
10    if (reply->type == REDIS_REPLY_STATUS)
```

```

11     {
12         /*SET str Hello World*/
13         printf("SET %s %s\n", key, val);
14     }
15     printf("reply->str: %s\n", reply->str);
16
17     freeReplyObject(reply);
18
19     // INCR Key
20     printf("\nINCR\n");
21     /*INCR key value */
22     reply = (redisReply *)redisCommand(context, "INCR %s", key);
23     printf("reply->type: %d\n", reply->type);
24     if (reply->type == REDIS_REPLY_ERROR)
25     {
26         /*INCR key*/
27         printf("INCR %s failed:%s\n", key, reply->str);
28     } else if (reply->type == REDIS_REPLY_INTEGER) {
29         printf("INCR %s = %lld\n", key, reply->integer);
30     }
31
32     freeReplyObject(reply);
33 }

```

打印

SET

reply->type: 5 ([REDIS_REPLY_STATUS](#))

SET teacher darren

reply->str: OK

INCR

reply->type: 6 ([REDIS_REPLY_ERROR](#))

INCR teacher failed:ERR value is not an integer or out of range (此时如果incr的是一个本身不在的key则返回1)

RESP 整型数据(: Integers)

此类型只是一个CRLF终止的字符串，表示一个以“:”字节为前缀的整数。例如“:0\r\n”或“:1000\r\n”是整数回复。

许多Redis命令返回RESP 整型，如INCR, LLEN 和LASTSAVE。

返回的整数没有特殊含义，它只是INCR的增量值，LASTSAVE的UNIX时间等等。但是，返回的整数应保证在有符号的64位整数范围内。

整数回复也被广泛使用，以便返回真或假。例如，EXISTS或SISMEMBER之类的命令将返回1表示true，0表示false。

如果实际执行操作，其他命令（如SADD，SREM和SETNX）将返回1，否则返回0。

以下命令将回复整数回复：SETNX，DEL，EXISTS，INCR，INCRBY，DECR，DECRBY，DBSIZE，LASTSAVE，RENAMENX，MOVE，LLEN，SADD，SREM，SISMEMBER，SCARD。

示例

SET count 10

INCR count

```
1 void testIntegers(redisContext *context)
2 {
3     // Set Key Value
4     printf("SET\n");
5     const char *key = "count";
6     const char *val = "10";
7     /*SET key value */
8     redisReply *reply = (redisReply *)redisCommand(context, "SET %s
    %s", key, val);
9     printf("reply->type: %d\n", reply->type);
10    if (reply->type == REDIS_REPLY_STATUS)
11    {
12        /*SET str Hello World*/
13        printf("SET %s %s\n", key, val);
14    }
15    printf("reply->str: %s\n", reply->str);
16
17    freeReplyObject(reply);
18
19    // INCR Key
20    printf("\nINCR\n");
```

```

21      /*INCR key value */
22      reply = (redisReply *)redisCommand(context, "INCR %s", key);
23      printf("reply->type: %d\n", reply->type);
24      if (reply->type == REDIS_REPLY_ERROR)
25      {
26          /*INCR key*/
27          printf("INCR %s failed:%s\n", key, reply->str);
28      } else if (reply->type == REDIS_REPLY_INTEGER) {
29          printf("INCR %s = %lld\n", key, reply->integer);
30      }
31
32      freeReplyObject(reply);
33 }

```

打印

SET

reply->type: 5

SET count 10

reply->str: OK

INCR

reply->type: 3 (REDIS_REPLY_INTEGER)

INCR count = 11

```

b,Ts val 859942757 ecr 859942757], length 5: RESP "11"
0x0000: 0000 0304 0006 0000 0000 0000 0000 0800 .....
0x0010: 4502 0039 6993 4000 4006 d327 7f00 0001 E..9i. @. @.. '....
0x0020: 7f00 0001 18eb 9696 e66e 9f00 cf43 063c .....n...C.<
0x0030: 8018 0156 fe2d 0000 0101 080a 3341 af65 ...V.-.....3A.e
0x0040: 3341 af65 3a31 310d 0a 3A.e:11..

```

RESP 多行字符串(\$ Bulk Strings)

多行字符串用于表示长度最大为512 MB的单个二进制安全字符串。

多行字符串按以下方式编码：

- 一个“\$”字节后跟组成字符串的字节数（一个前缀长度），由CRLF终止。
- 字符串数据。
- 最终的CRLF。

所以字符串“foobar”的编码如下：

```

1 "$6\r\nfoobar\r\n"

```


当只是一个空字符串时：

```
1 "$0\r\n\r\n"
```

RESP 多行字符串也可用于使用用于表示Null值的特殊格式来表示值的不存在。在这种特殊格式中，长度为-1，并且没有数据，因此Null表示为：

```
1 "$-1\r\n"
```

当服务器使用Null 多行字符串回复时，客户端库API不应返回空字符串，而应返回nil对象。例如，Ruby库应返回'nil'，而C库应返回NULL（或在reply对象中设置特殊标志），依此类推。

示例

```
1 127.0.0.1:6379> MSET king redis darren avmedia
2 OK
3 127.0.0.1:6379> MGET king darren
4 1) "redis"
5 2) "avmedia"
```

```
[nop,nop,TS val 862108240 ecr 862108240], length 28: RESP "redis" "avmedia"
0x0000: 0000 0304 0006 0000 0000 0000 0000 0800 .....
0x0010: 4502 0050 fded 4000 4006 3eb6 7f00 0001 E..P..@.>.....
0x0020: 7f00 0001 18eb 96a6 96f8 2744 3e81 47a7 ..... 'D>.G.
0x0030: 8018 0156 fe44 0000 0101 080a 3362 ba50 ...V.D.....3b.P
0x0040: 3362 ba50 2a32 0d0a 2435 0d0a 7265 6469 3b.P*2..$5..redi
0x0050: 730d 0a24 370d 0a61 766d 6564 6961 0d0a s..$7..avmedia..
```

hiredis编码

```
1 void testBulkStrings(redisContext *context)
2 {
3     printf("MSET\n");
4     /*MSET key value [key value ...]*/
5     redisReply *reply = (redisReply *)redisCommand(context, "MSET ki
    ng redis darren avmedia");
6     if (reply->type == REDIS_REPLY_STATUS) {
7         printf("MSET king redis darren avmedia\n");
8     }
9     freeReplyObject(reply);
10 }
```

```

11     printf("\nMGET\n");
12     /*MGET key [key ...]*/
13     reply = (redisReply *)redisCommand(context, "MGET king darren");
14     printf("reply->type: %d\n", reply->type);
15     if (reply->type == REDIS_REPLY_ARRAY) {
16         printf("MGET king darren\n");
17         redisReply **pReply = reply->element;
18         int i = 0;
19         size_t len = reply->elements;
20         //hello world good
21         for (; i < len; ++i) {
22             printf("%s \n", pReply[i]->str); // 如果没有数据时为null
23         }
24         printf("\n");
25     }
26     freeReplyObject(reply);
27 }

```

打印

MSET

MSET king redis darren avmedia

MGET

reply->type: 2 (REDIS_REPLY_ARRAY)

MGET king darren

redis

(null)

RESP 数组(* Arrays)

客户端使用RESP 数组将命令发送到Redis服务器。类似地，某些Redis命令将元素集合返回给客户端使用RESP 数组是回复类型。一个例子是LRANGE命令，它返回列表的元素。

RESP数组使用以下格式发送：

- *字符作为第一个字节，后跟数组中的元素个数作为十进制数，后跟CRLF。
- 数组的每个元素的附加RESP类型。

所以空数组就是以下内容：

```
1 "*0\r\n"
```

那么两个RESP批量字符串“foo”和“bar”的数组编码为：

```
1 "*2\r\n$3\r\nfoo\r\n$3\r\nbar\r\n"
```

正如您在数组前面加上* CRLF部分之后所看到的那样，组成数组的其他数据类型将一个接一个地连接起来。例如，三个整数的数组编码如下：

```
1 "*3\r\n:1\r\n:2\r\n:3\r\n"
```

数组可以包含混合类型，元素不必具有相同的类型。例如，四个整数和批量字符串的列表可以编码如下：

```
1 *5\r\n
2 :1\r\n
3 :2\r\n
4 :3\r\n
5 :4\r\n
6 $6\r\n
7 foobar\r\n
```

服务器发送的第一行是* 5 \ r \ n，以指定将跟随五个回复。然后发送构成多重回复项目的每个回复。Null 数组的概念也存在，并且是指定Null值的替代方法（通常使用Null 多行字符串，但由于历史原因，我们有两种格式）。

例如，当BLPOP命令超时，它返回一个计数为-1的Null数组，如下例所示：

```
1 "*-1\r\n"
```

当Redis使用Null数组回复时，客户端库API应返回空对象而不是空数组。这是区分空列表和不同条件（例如BLPOP命令的超时条件）所必需的。

RESP中可以使用数组中嵌套数组。例如，两个数组的数组编码如下：

```
1 *2\r\n
2 *3\r\n
3 :1\r\n
4 :2\r\n
5 :3\r\n
6 *2\r\n
7 +Foo\r\n
8 -Bar\r\n
```

第二个元素是Null。客户端库应返回如下内容：

```
1 ["foo",nil,"bar"]
```

注意，这不是前面部分中所述的例外，而只是进一步指定协议的示例。

示例

```
1 127.0.0.1:6379> LPUSH list darren qiuxiang king milo
2 (integer) 4
3 127.0.0.1:6379> LRANGE list 0 2
4 1) "milo"
5 2) "king"
6 3) "qiuxiang"
7 127.0.0.1:6379> LRANGE list 0 -1
8 1) "milo"
9 2) "king"
10 3) "qiuxiang"
11 4) "darren"
```

LRANGE list 0 2

```
,TS val 862928188 ecr 862928188], length 38: RESP "milo" "king" "qiuxiang"
0x0000: 0000 0304 0006 0000 0000 0000 0000 0800 .....
0x0010: 4502 005a fe2a 4000 4006 3e6f 7f00 0001 E..Z.*@.>...
0x0020: 7f00 0001 18eb 96a6 96f8 27ca 3e81 488c .....>.H.
0x0030: 8018 0156 fe4e 0000 0101 080a 336f 3d3c ...V.N.....3o=<
0x0040: 336f 3d3c 2a33 0d0a 2434 0d0a 6d69 6c6f 3o=<[*3]..$4..milo
0x0050: 0d0a 2434 0d0a 6b69 6e67 0d0a 2438 0d0a ..$4..king..$8..
0x0060: 7169 7578 6961 6e67 0d0a .....qiuxiang..
```

LRANGE list 0 -1

```
,TS val 862938054 ecr 862938053], length 50: RESP "milo" "king" "qiuxiang" "darren"
0x0000: 0000 0304 0006 0000 0000 0000 0000 0800 .....
0x0010: 4502 0066 fe2b 4000 4006 3e62 7f00 0001 E..f.+@.>b...
0x0020: 7f00 0001 18eb 96a6 96f8 27f0 3e81 48b5 .....>.H.
0x0030: 8018 0156 fe5a 0000 0101 080a 336f 63c6 ...V.Z.....3oc.
0x0040: 336f 63c5 2a34 0d0a 2434 0d0a 6d69 6c6f 3oc[*4]..$4..milo
0x0050: 0d0a 2434 0d0a 6b69 6e67 0d0a 2438 0d0a ..$4..king..$8..
0x0060: 7169 7578 6961 6e67 0d0a 2436 0d0a 6461 qiuxiang..$6..da
0x0070: 7272 656e 0d0a .....rren..
```

hiredis编程

LPUSH

reply->type: 3

LRANGE

reply->type: 2

```
LRange list 0 2
```

```
miLo
```

```
king
```

```
qiuxiang
```

```
reply->type: 2 (REDIS_REPLY_ARRAY)
```

```
LRange list 0 -1
```

```
miLo
```

```
king
```

```
qiuxiang
```

```
darren
```

发送命令到 Redis 服务端

既然熟悉RESP序列化格式，那么编写Redis客户端库的实现将很容易。我们可以进一步讲述客户端和服务端之间的交互如何工作：

- 客户端向Redis服务器发送仅由Bulk Strings组成的RESP阵列。
- Redis服务器回复发送任何有效RESP数据类型作为回复的客户端。

因此，例如，典型的交互可以是以下所示。

客户端发送命令LLEN mylist以获取存储在密钥mylist中的列表长度，服务器回复一个Integer回复，如下例所示（C：是客户端，S：服务器）。

```
1 C: *2\r\n
2 C: $4\r\n
3 C: LLEN\r\n
4 C: $6\r\n
5 C: mylist\r\n
6 S: :48293\r\n
```

通常我们将协议的不同部分与换行符分开以简化，但实际的交互是客户端发送* 2 \ r \ n \$ 4 \ r \ n LLEN \ r \ n \$ 6 \ r \ n mylist \ r \ n整体。