

加餐（四）-Redis客户端如何与服务器端交换命令和数据？

你好，我是蒋德钧。

在前面的课程中，我们主要学习了Redis服务器端的机制和关键技术，很少涉及到客户端的问题。但是，Redis采用的是典型的client-server（服务器端-客户端）架构，客户端会发送请求给服务器端，服务器端会返回响应给客户端。

如果要对Redis客户端进行二次开发（比如增加新的命令），我们就需要了解请求和响应涉及的命令、数据在客户端和服务端之间传输时，是如何编码的。否则，我们在客户端新增的命令就无法被服务器端识别和处理。

Redis使用RESP（Redis Serialization Protocol）协议定义了客户端和服务端交互的命令、数据的编码格式。在Redis 2.0版本中，RESP协议正式成为客户端和服务端的标准通信协议。从Redis 2.0到Redis 5.0，RESP协议都称为RESP 2协议，从Redis 6.0开始，Redis就采用RESP 3协议了。不过，6.0版本是在今年5月刚推出的，所以，目前我们广泛使用的还是RESP 2协议。

这节课，我就向你重点介绍下RESP 2协议的规范要求，以及RESP 3相对RESP 2的改进之处。

首先，我们先来看下客户端和服务端交互的内容包括哪些，毕竟，交互内容不同，编码形式也不一样。

客户端和服务端交互的内容有哪些？

为了方便你更加清晰地理解，RESP 2协议是如何对命令和数据进行格式编码的，我们可以把交互内容，分成客户端请求和服务端响应两类：

- 在客户端请求中，客户端会给Redis发送命令，以及要写入的键和值；
- 而在服务器端响应中，Redis实例会返回读取的值、OK标识、成功写入的元素个数、错误信息，以及命令（例如Redis Cluster中的MOVE命令）。

其实，这些交互内容还可以再进一步细分成七类，我们再了解下它们。

1. **命令**：这就是针对不同数据类型的操作命令。例如对String类型的SET、GET操作，对Hash类型的HSET、HGET等，这些命令就是代表操作语义的字符串。
2. **键**：键值对中的键，可以直接用字符串表示。
3. **单个值**：对应String类型的数据，数据本身可以是字符串、数值（整数或浮点数），布尔值（True或是False）等。
4. **集合值**：对应List、Hash、Set、Sorted Set类型的数据，不仅包含多个值，而且每个值也可以是字符串、数值或布尔值等。
5. **OK回复**：对应命令操作成功的结果，就是一个字符串的“OK”。
6. **整数回复**：这里有两种情况。一种是，命令操作返回的结果是整数，例如LLEN命令返回列表的长度；另一种是，集合命令成功操作时，实际操作的元素个数，例如SADD命令返回成功添加的元素个数。
7. **错误信息**：命令操作出错时的返回结果，包括“error”标识，以及具体的错误信息。

了解了这7类内容都是什么，下面我再结合三个具体的例子，帮助你进一步地掌握这些交互内容。

```
#成功写入String类型数据，返回OK
127.0.0.1:6379> SET testkey testvalue
OK
```

这里的交互内容就包括了**命令**（SET命令）、**键**（String类型的键testkey）和**单个值**（String类型的值testvalue），而服务器端则直接返回一个**OK**回复。

第二个例子是执行HSET命令：

```
#成功写入Hash类型数据，返回实际写入的集合元素个数
127.0.0.1:6379> HSET testhash a 1 b 2 c 3
(integer) 3
```

这里的交互内容包括三个key-value的Hash**集合值**（a 1 b 2 c 3），而服务器端返回**整数回复**（3），表示操作成功写入的元素个数。

最后一个例子是执行PUT命令，如下所示：

```
#发送的命令不对，报错，并返回错误信息
127.0.0.1:6379> PUT testkey2 testvalue
(error) ERR unknown command 'PUT', with args beginning with: 'testkey', 'testvalue'
```

可以看到，这里的交互内容包括**错误信息**，这是因为，Redis实例本身不支持PUT命令，所以服务器端报错“error”，并返回具体的错误信息，也就是未知的命令“put”。

好了，到这里，你了解了，Redis客户端和服务端交互的内容。接下来，我们就来看下，RESP 2是按照什么样的格式规范来对这些内容进行编码的。

RESP 2的编码格式规范

RESP 2协议的设计目标是，希望Redis开发人员实现客户端时简单方便，这样就可以减少客户端开发时出现的Bug。而且，当客户端和服务端交互出现问题时，希望开发人员可以通过查看协议交互过程，能快速定位问题，方便调试。为了实现这一目标，RESP 2协议采用了可读性很好的文本形式进行编码，也就是通过一系列的字符串，来表示各种命令和数据。

不过，交互内容有多种，而且，实际传输的命令或数据也会有很多个。针对这两种情况，RESP 2协议在编码时设计了两个基本规范。

1. 为了对不同类型的交互内容进行编码，RESP 2协议实现了5种编码格式类型。同时，为了区分这5种编码类型，RESP 2使用一个专门的字符，作为每种编码类型的开头字符。这样一来，客户端或服务端在对编码后的数据进行解析时，就可以直接通过开头字符知道当前解析的编码类型。

2. RESP 2进行编码时，会按照单个命令或单个数据的粒度进行编码，并在每个编码结果后面增加一个换行符“\r\n”（有时也表示成CRLF），表示一次编码结束。

接下来，我就来分别介绍下这5种编码类型。

1.简单字符串类型（RESP Simple Strings）

这种类型就是用一个字符串来进行编码，比如，请求操作在服务器端成功执行后的OK标识回复，就是用这种类型进行编码的。

当服务器端成功执行一个操作后，返回的OK标识就可以编码如下：

```
+OK\r\n
```

2.长字符串类型（RESP Bulk String）

这种类型是用一个二进制安全的字符串来进行编码。这里的二进制安全，其实是相对于C语言中对字符串的处理方式来说的。我来具体解释一下。

Redis在解析字符串时，不会像C语言那样，使用“\0”判定一个字符串的结尾，Redis会把“\0”解析成正常的0字符，并使用额外的属性值表示字符串的长度。

举个例子，对于“Redis\0Cluster\0”这个字符串来说，C语言会解析为“Redis”，而Redis会解析为“Redis Cluster”，并用len属性表示字符串的真实长度是14字节，如下图所示：

“Redis\0Cluster\0”

C语言解析为“Redis”

Redis SDS结构

len = 14
alloc
buf ("Redis\0Cluster\0")

Redis解析为“Redis Cluster”

这样一来，字符串中即使存储了“\0”字符，也不会导致Redis解析到“\0”时，就认为字符串结束了从而停止解析，这就保证了数据的安全性。和长字符串类型相比，简单字符串就是非二进制安全的。

长字符串类型最大可以达到512MB，所以可以对很大的数据量进行编码，正好可以满足键值对本身的数据量需求，所以，RESP 2就用这种类型对交互内容中的键或值进行编码，并且使用“\$”字符作为开头字符，\$字符后面会紧跟着一个数字，这个数字表示字符串的实际长度。

例如，我们使用GET命令读取一个键（假设键为testkey）的值（假设值为testvalue）时，服务端返回的String值编码结果如下，其中，\$字符后的9，表示数据长度为9个字符。

```
$9 testvalue\r\n
```

3. 整数类型 (RESP Integer)

这种类型也还是一个字符串，但是表示的是一个有符号64位整数。为了和包含数字的简单字符串类型区分开，整数类型使用“:”字符作为开头字符，可以用于对服务器端返回的整数回复进行编码。

例如，在刚才介绍的例子中，我们使用HSET命令设置了testhash的三个元素，服务器端实际返回的编码结果如下：

```
:3\r\n
```

4. 错误类型 (RESP Errors)

它是一个字符串，包括了错误类型和具体的错误信息。Redis服务器端报错响应就是用这种类型进行编码的。RESP 2使用“-”字符作为它的开头字符。

例如，在刚才的例子中，我们在redis-cli执行PUT testkey2 testvalue命令报错，服务器端实际返回给客户端的报错编码结果如下：

```
-ERR unknown command 'PUT', with args beginning with: 'testkey', 'testvalue'
```

其中，ERR就是报错类型，表示是一个通用错误，ERR后面的文字内容就是具体的报错信息。

5. 数组编码类型 (RESP Arrays)

这是一个包含多个元素的数组，其中，元素的类型可以是刚才介绍的这4种编码类型。

在客户端发送请求和服务端返回结果时，数组编码类型都能用得上。客户端在发送请求操作时，一般会同时包括命令和要操作的数据。而数组类型包含了多个元素，所以，就适合用来对发送的命令和数据进行编码。为了和其他类型区分，RESP 2使用“*”字符作为开头字符。

例如，我们执行命令GET testkey，此时，客户端发送给服务器端的命令的编码结果就是使用数组类型编码的，如下所示：

```
*2\r\n$3\r\nGET\r\n$7\r\ntestkey\r\n
```

其中，**第一个*字符标识当前是数组类型的编码结果**，2表示该数组有2个元素，分别对应命令GET和键testkey。命令GET和键testkey，都是使用长字符串类型编码的，所以用\$字符加字符串长度来表示。

类似地，当服务器端返回包含多个元素的集合类型数据时，也会用*字符和元素个数作为标识，并用长字符串类型对返回的集合元素进行编码。

好了，到这里，你了解了RESP 2协议的5种编码类型和相应的开头字符，我在下面的表格里做了小结，你可以看下。

编码类型	简单字符串	长字符串	整数	错误	数组
开头第一个字符	+	\$:	-	*

Redis 6.0中使用了RESP 3协议，对RESP 2.0做了改进，我们来学习下具体都有哪些改进。

RESP 2的不足和RESP 3的改进

虽然我们刚刚说RESP 2协议提供了5种编码类型，看起来很丰富，其实是不够的。毕竟，基本数据类型还包括很多种，例如浮点数、布尔值等。编码类型偏少，会带来两个问题。

一方面，在值的基本数据类型方面，RESP 2只能区分字符串和整数，对于其他的数据类型，客户端使用RESP 2协议时，就需要进行额外的转换操作。例如，当一个浮点数用字符串表示时，客户端需要将字符串中的值和实际数值比较，判断是否为数字值，然后再将字符串转换成实际的浮点数。

另一方面，RESP 2用数组类别编码表示所有的集合类型，但是，Redis的集合类型包括了List、Hash、Set和Sorted Set。当客户端接收到数组类型编码的结果时，还需要根据调用的命令操作接口，来判断返回的数组究竟是哪一种集合类型。

我来举个例子。假设有一个Hash类型的键是testhash，集合元素分别为a:1、b:2、c:3。同时，有一个Sorted Set类型的键是testzset，集合元素分别是a、b、c，它们的分数分别是1、2、3。我们在redis-cli客户端中读取它们的结果时，返回的形式都是一个数组，如下所示：

```
127.0.0.1:6379>HGETALL testhash
1) "a"
2) "1"
3) "b"
4) "2"
5) "c"
6) "3"
```

```
127.0.0.1:6379>ZRANGE testzset 0 3 withscores
```

```
1) "a"  
2) "1"  
3) "b"  
4) "2"  
5) "c"  
6) "3"
```

为了在客户端按照Hash和Sorted Set两种类型处理代码中返回的数据，客户端还需要根据发送的命令操作HGETALL和ZRANGE，来把这两个编码的数组结果转换成相应的Hash集合和有序集合，增加了客户端额外的开销。

从Redis 6.0版本开始，RESP 3协议增加了对多种数据类型的支持，包括空值、浮点数、布尔值、有序的字典集合、无序的集合等。RESP 3也是通过不同的开头字符来区分不同的数据类型，例如，当开头第一个字符是“+”，就表示接下来的编码结果是浮点数。这样一来，客户端就不用再通过额外的字符串比对，来实现数据转换操作了，提升了客户端的效率。

小结

这节课，我们学习了RESP 2协议。这个协议定义了Redis客户端和服务端进行命令和数据交互时的编码格式。RESP 2提供了5种类型的编码格式，包括简单字符串类型、长字符串类型、整数类型、错误类型和数组类型。为了区分这5种类型，RESP 2协议使用了5种不同的字符作为这5种类型编码结果的第一个字符，分别是+、\$、:、-和*。

RESP 2协议是文本形式的协议，实现简单，可以减少客户端开发出现的Bug，而且可读性强，便于开发调试。当你需要开发定制化的Redis客户端时，就需要了解和掌握RESP 2协议。

RESP 2协议的一个不足就是支持的类型偏少，所以，Redis 6.0版本使用了RESP 3协议。和RESP 2协议相比，RESP 3协议增加了对浮点数、布尔类型、有序字典集合、无序集合等多种类型数据的支持。不过，这里，有个地方需要你注意，Redis 6.0只支持RESP 3，对RESP 2协议不兼容，所以，如果你使用Redis 6.0版本，需要确认客户端已经支持了RESP 3协议，否则，将无法使用Redis 6.0。

最后，我也给你提供一个小工具。如果你想查看服务器端返回数据的RESP 2编码结果，就可以使用telnet命令和redis实例连接，执行如下命令就行：

```
telnet 实例IP 实例端口
```

接着，你可以给实例发送命令，这样就能看到用RESP 2协议编码后的返回结果了。当然，你也可以在telnet中，向Redis实例发送用RESP 2协议编写的命令操作，实例同样能处理，你可以课后试试看。

每课一问

按照惯例，我给你提个小问题，假设Redis实例中有一个List类型的数据，key为mylist，value是使用LPUSH命令写入List集合的5个元素，依次是1、2、3.3、4、hello，当执行LRANGE mylist 0 4命令时，实例返回给客户端的数组结果是什么样的？

欢迎在留言区写下你的思考和答案，我们一起交流讨论。如果你觉得今天的内容对你有帮助，也欢迎你分享给你的朋友或同事。我们下节课见。

精选留言：

- Kaito 2020-10-23 00:22:41
key为mylist，使用LPUSH写入是1、2、3.3、4、hello，执行LRANGE mylist 0 4命令时，实例返回给客户端的编码结果是怎样的？

测试结果如下，写入命令：

```
127.0.0.1:6479> LPUSH mylist 1 2 3.3 4 hello
(integer) 5
127.0.0.1:6479> LRANGE mylist 0 4
1) "hello"
2) "4"
3) "3.3"
4) "2"
5) "1"
```

使用telnet发送命令，观察结果：

```
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^['.
```

```
LRANGE mylist 0 4
*5
$5
hello
$1
4
$3
3.3
$1
2
$1
1
1
```

Redis设计的RESP 2协议非常简单、易读，优点是对于客户端的开发和生态建设非常友好。但缺点是纯文本，其中还包含很多冗余的回车换行符，相比于二进制协议，这会造成流量的浪费。但作者依旧这么做的原因是Redis是内存数据库，操作逻辑都在内存中进行，速度非常快，性能瓶颈不在于网络流量上，所以设计放在了更加简单、易理解、易实现的层面上。

Redis 6.0重新设计RESP 3，比较重要的原因就是RESP 2的语义能力不足，例如LRANGE/SMEMBERS/HGETALL都返回一个数组，客户端需要根据发送的命令类型，解析响应再封装成合适的对象供业务使用。而RESP 3在响应中就可以明确标识出数组、集合、哈希表，无需再做转换。另外RESP 2没有布尔类型和浮点类型，例如EXISTS返回的是0或1，Sorted Set中返回的score是字符串，这些都需要客户端自己转换处理。而RESP 3增加了布尔、浮点类型，客户端直接可以拿到明确的类型。

的方式有4种：

- 1、固定长度拆分：发送方以固定长度进行发送，接收方按固定长度截取拆分。例如发送方每次发送数据都是5个字节的长度，接收方每次都按5个字节拆分截取数据内容。
- 2、特殊字符拆分：发送方在消息尾部设置一个特殊字符，接收方遇到这个特殊字符就做拆分处理。HTTP协议就是这么做的，以\r\n为分隔符解析协议。
- 3、长度+消息拆分：发送方在每个消息最前面加一个长度字段，接收方先读取到长度字段，再向后读取指定长度即是数据内容。Redis采用的就是这种。
- 4、消息本身包含格式：发送方在消息中就设置了开始和结束标识，接收方根据这个标识截取出中间的数据。例如<start>msg data<end>。

如果我们在设计一个通信协议时，可以作为参考，根据自己的场景进行选择。 [3赞]

拼课微信：699250