通信协议（protocol）

本文档翻译自： http://redis.io/topics/protocol 。

Redis 协议在以下三个目标之间进行折中：

- 易于实现
- 可以高效地被计算机分析（parse）
- 可以很容易地被人类读懂

网络层

客户端和服务器通过 TCP 连接来进行数据交互， 服务器默认的端口号为 6379 。

客户端和服务器发送的命令或数据一律以 `\r\n` （CRLF）结尾。

请求

Redis 服务器接受命令以及命令的参数。

服务器会在接到命令之后，对命令进行处理，并将命令的回复传送回客户端。

新版统一请求协议

新版统一请求协议在 Redis 1.2 版本中引入， 并最终在 Redis 2.0 版本成为 Redis 服务器通信的标准方式。

你的 Redis 客户端应该按照这个新版协议来进行实现。

在这个协议中， 所有发送至 Redis 服务器的参数都是二进制安全（binary safe）的。

以下是这个协议的一般形式：

```
*<参数数量> CR LF
$<参数 1 的字节数量> CR LF
<参数 1 的数据> CR LF
...
$<参数 N 的字节数量> CR LF
<参数 N 的数据> CR LF
```

译注：命令本身也作为协议的其中一个参数来发送。

举个例子， 以下是一个命令协议的打印版本：

```
*3
$3
SET
$5
mykey
$7
myvalue
```

这个命令的实际协议值如下：

```
"*3\r\n$3\r\nSET\r\n$5\r\nmykey\r\n$7\r\nmyvalue\r\n"
```

稍后我们会看到， 这种格式除了用作命令请求协议之外， 也用在命令的回复协议中： 这种只有一个参数的回复格式被称为**批量回复（Bulk Reply）**。

统一协议请求原本是用在回复协议中， 用于将列表的多个项返回给客户端的， 这种回复格式被称为**多条批量回复（Multi Bulk Reply）**。

一个多条批量回复以 `*<argc>\r\n` 为前缀， 后跟多条不同的批量回复， 其中 `argc` 为这些批量回复的数量。

回复

Redis 命令会返回多种不同类型的回复。

通过检查服务器发回数据的第一个字节， 可以确定这个回复是什么类型：

- 状态回复（status reply）的第一个字节是 `"+"`
- 错误回复（error reply）的第一个字节是 `"-"`
- 整数回复（integer reply）的第一个字节是 `":"`
- 批量回复（bulk reply）的第一个字节是 `"$"`
- 多条批量回复（multi bulk reply）的第一个字节是 `"*"`

状态回复

一个状态回复（或者单行回复，single line reply）是一段以 `"+"` 开始、 `"\r\n"` 结尾的单行字符串。

以下是一个状态回复的例子：

```
+OK
```

客户端库应该返回 `"+"` 号之后的所有内容。 比如在在上面的这个例子中， 客户端就应该返回字符串 `"OK"`。

状态回复通常由那些不需要返回数据的命令返回，这种回复不是二进制安全的，它也不能包含新行。

状态回复的额外开销非常少，只需要三个字节（开头的 `"+"` 和结尾的 CRLF）。

错误回复

错误回复和状态回复非常相似， 它们之间的唯一区别是， 错误回复的第一个字节是 `"-"`， 而状态回复的第一个字节是 `"+"`。

错误回复只在某些地方出现问题时发送： 比如说， 当用户对不正确的数据类型执行命令， 或者执行一个不存在的命令， 等等。

一个客户端库应该在收到错误回复时产生一个异常。

以下是两个错误回复的例子：

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

在 `"-"` 之后，直到遇到第一个空格或新行为止，这中间的内容表示所返回错误的类型。

ERR 是一个通用错误，而 WRONGTYPE 则是一个更特定的错误。 一个客户端实现可以为不同类型的错误产生不同类型的异常， 或者提供一种通用的方式， 让调用者可以通过提供字符串形式的错误名来捕捉（trap）不同的错误。

不过这些特性用得并不多， 所以并不是特别重要， 一个受限的（limited）客户端可以通过简单地返回一个逻辑假（false）来表示一个通用的错误条件。

整数回复

整数回复就是一个以 ":" 开头， CRLF 结尾的字符串表示的整数。

比如说， ":0\r\n" 和 ":1000\r\n" 都是整数回复。

返回整数回复的其中两个命令是 INCR 和 LASTSAVE 。 被返回的整数没有什么特殊的含义， INCR 返回键的一个自增后的整数值， 而 LASTSAVE 则返回一个 UNIX 时间戳， 返回值的唯一限制是这些数必须能够用 64 位有符号整数表示。

整数回复也被广泛地用于表示逻辑真和逻辑假： 比如 EXISTS 和 SISMEMBER 都用返回值 1 表示真， 0 表示假。

其他一些命令， 比如 SADD 、 SREM 和 SETNX ， 只在操作真正被执行了的时候， 才返回 1 ， 否则返回 0 。

以下命令都返回整数回复： SETNX 、 DEL 、 EXISTS 、 INCR 、 INCRBY 、 DECR 、 DECRBY 、 DBSIZE 、 LASTSAVE 、 RENAMENX 、 MOVE 、 LLEN 、 SADD 、 SREM 、 SISMEMBER 、 SCARD 。

批量回复

服务器使用批量回复来返回二进制安全的字符串，字符串的最大长度为 512 MB 。

```
客户端：GET mykey
服务器：foobar
```

服务器发送的内容中：

- 第一字节为 "$" 符号
- 接下来跟着的是表示实际回复长度的数字值
- 之后跟着一个 CRLF
- 再后面跟着的是实际回复数据
- 最末尾是另一个 CRLF

对于前面的 GET 命令，服务器实际发送的内容为：

```
"$6\r\nfoobar\r\n"
```

如果被请求的值不存在， 那么批量回复会将特殊值 -1 用作回复的长度值， 就像这样：

```
客户端：GET non-existing-key
服务器：$-1
```

这种回复称为空批量回复（NULL Bulk Reply）。

当请求对象不存在时，客户端应该返回空对象，而不是空字符串： 比如 Ruby 库应该返回 nil ， 而 C 库应该返回 NULL （或者在回复对象中设置一个特殊标志）， 诸如此类。

多条批量回复

像 LRANGE 这样的命令需要返回多个值， 这一目标可以通过多条批量回复来完成。

多条批量回复是由多个回复组成的数组， 数组中的每个元素都可以是任意类型的回复， 包括多条批量回复本身。

多条批量回复的第一个字节为 "*"， 后跟一个字符串表示的整数值， 这个值记录了多条批量回复所包含的回复数量， 再后面是一个 CRLF 。

```
客户端： LRANGE mylist 0 3
服务器： *4
服务器： $3
服务器： foo
服务器： $3
服务器： bar
服务器： $5
服务器： Hello
服务器： $5
服务器： World
```

在上面的示例中，服务器发送的所有字符串都由 CRLF 结尾。

正如你所见到的那样， 多条批量回复所使用的格式， 和客户端发送命令时使用的统一请求协议的格式一模一样。 它们之间的唯一区别是：

- 统一请求协议只发送批量回复。
- 而服务器应答命令时所发送的多条批量回复，则可以包含任意类型的回复。

以下例子展示了一个多条批量回复， 回复中包含四个整数值， 以及一个二进制安全字符串：

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

在回复的第一行， 服务器发送 `*5\r\n` ， 表示这个多条批量回复包含 5 条回复， 再后面跟着的则是 5 条回复的正文。

多条批量回复也可以是空白的（empty）， 就像这样：

```
客户端： LRANGE nokey 0 1
服务器： *0\r\n
```

无内容的多条批量回复（null multi bulk reply）也是存在的， 比如当 BLPOP 命令的阻塞时间超过最大时限时， 它就返回一个无内容的多条批量回复， 这个回复的计数值为 `-1` ：

```
客户端： BLPOP key 1
服务器： *-1\r\n
```

客户端库应该区别对待空白多条回复和无内容多条回复： 当 Redis 返回一个无内容多条回复时， 客户端库应该返回一个 null 对象， 而不是一个空数组。

多条批量回复中的空元素

多条批量回复中的元素可以将自身的长度设置为 `-1` ， 从而表示该元素不存在， 并且也不是一个空白字符串（empty string）。

当 SORT 命令使用 `GET pattern` 选项对一个不存在的键进行操作时， 就会发生多条批量回复中带有空白元素的情况。

以下例子展示了一个包含空元素的多重批量回复：

```
服务器： *3
服务器： $3
服务器： foo
服务器： $-1
```

```
服务器： $3
服务器： bar
```

其中， 回复中的第二个元素为空。

对于这个回复， 客户端库应该返回类似于这样的回复：

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

多命令和流水线

客户端可以通过流水线， 在一次写入操作中发送多个命令：

- 在发送新命令之前， 无须阅读前一个命令的回复。
- 多个命令的回复会在最后一并返回。

内联命令

当你需要和 Redis 服务器进行沟通， 但又找不到 `redis-cli` ， 而手上只有 `telnet` 的时候， 你可以通过 Redis 特别为这种情形而设的内联命令格式来发送命令。

以下是一个客户端和服务器使用内联命令来进行交互的例子：

```
客户端： PING
服务器： +PONG
```

以下另一个返回整数值的内联命令的例子：

```
客户端： EXISTS somekey
服务器： :0
```

因为没有了统一请求协议中的 "*" 项来声明参数的数量， 所以在 `telnet` 会话输入命令的时候， 必须使用空格来分割各个参数， 服务器在接收到数据之后， 会按空格对用户的输入进行分析（parse）， 并获取其中的命令参数。

高性能 Redis 协议分析器

尽管 Redis 的协议非常利于人类阅读， 定义也很简单， 但这个协议的实现性能仍然可以和二进制协议一样快。

因为 Redis 协议将数据的长度放在数据正文之前， 所以程序无须像 JSON 那样， 为了寻找某个特殊字符而扫描整个 payload ， 也无须对发送至服务器的 payload 进行转义（quote）。

程序可以在对协议文本中的各个字符进行处理的同时， 查找 CR 字符， 并计算出批量回复或多条批量回复的长度， 就像这样：

```c
#include <stdio.h>

int main(void) {
    unsigned char *p = "$123\r\n";
    int len = 0;

    p++;
    while(*p != '\r') {
        len = (len*10)+(*p - '0');
        p++;
    }

    /* Now p points at '\r', and the len is in bulk_len. */
    printf("%d\n", len);
```

```
    return 0;
}
```

得到了批量回复或多条批量回复的长度之后， 程序只需调用一次 `read` 函数， 就可以将回复的正文数据全部读入到内存中， 而无须对这些数据做任何的处理。

在回复最末尾的 CR 和 LF 不作处理，丢弃它们。

Redis 协议的实现性能可以和二进制协议的实现性能相媲美， 并且由于 Redis 协议的简单性， 大部分高级语言都可以轻易地实现这个协议， 这使得客户端软件的 bug 数量大大减少。