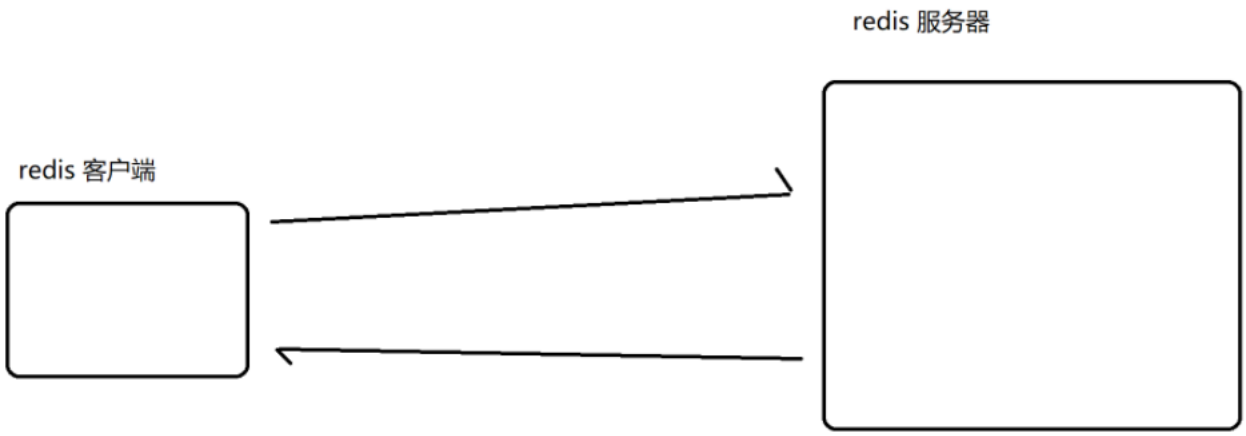
Redis - 客户端与服务端通信详解与自定义实现（146 ~ 147）

Redis 作为高性能内存数据库，其客户端与服务端之间的通信是核心环节。理解 Redis 通信模型、RESP 协议及自定义客户端实现方法，可以帮助开发者实现高性能、稳定可靠的业务逻辑，并在特殊业务场景下进行深度定制。

一、Redis 客户端与服务端的通信模型



1. 为什么需要自定义 Redis 客户端

虽然 Redis 官方提供了多种客户端，但在某些场景下仍需开发自定义客户端：

1. **第三方客户端局限性**：
   - 官方或社区客户端可能无法完全满足特定业务需求，例如：
     - 自定义序列化方式（JSON、ProtoBuf、MsgPack 等）
     - 高性能优化（批量发送、异步非阻塞）
     - 特殊安全策略（加密传输、ACL 集成）
   - 有些客户端是第三方维护的，稳定性和更新频率可能无法保证。
2. **理解协议与性能优化**：
   - 深度理解 Redis 协议（RESP）和网络通信流程，可以实现：
     - 自定义高效通信逻辑
     - 低延迟的数据传输

    - 对高并发场景进行优化

3. **类比理解**：
    - 类似于开发 QQ、微信客户端：
        - 官方提供 SDK，但如果你要实现高级定制（如特殊 UI、消息加密、优化网络协议），就必须理解服务端通信协议。
    - Redis 客户端开发也是如此：核心在于实现 RESP 协议，而非依赖 SDK。

---

2. 网络通信层级

Redis 客户端通信涉及多层网络体系，但开发者主要关注应用层：

层级	作用	开发者关注点
物理层 / 数据链路层	网卡、光纤、交换机、网线	无需关心，由 OS 和网络设备处理
网络层（IP）	数据包寻址与路由	无需关心，OS TCP/IP 栈处理
传输层（TCP）	可靠传输、顺序控制、重传	客户端需建立 TCP Socket 连接，但无需实现 TCP 逻辑
应用层（RESP）	Redis 定义的协议	客户端必须实现请求编码与响应解析，这是核心

- **核心结论**：开发自定义 Redis 客户端，本质是**实现 RESP 协议的编码与解码逻辑**，其余网络细节由操作系统处理。

二、RESP（Redis Serialization Protocol）协议详解

RESP 是 Redis 官方定义的通信协议，是客户端与服务端交互的基础。

网络通信过程中, 会用到很多的 "协议"

应用层:
传输层: TCP / UDP 协议
网络层: IP 协议
数据链路层: 以太网
物理层

虽然业界有很多成熟的应用层协议, HTTP 等.....
但是此处更多的时候, 都会 "自定义" 应用层协议~
Redis 此处应用层协议, 就是自定义的协议~~

这些协议是固定好的, 是在系统内核或者驱动程序中实现的~~
咱们程序猿只能选择, 不能修改~~

(传输层还是基于 TCP)

客户端按照这里的应用层协议, 发送请求
服务器按照这个协议进行解析
再按照这个协议构造响应
客户端再解析这个响应

之所以上述通信能够完成
是因为开发客户端的人, 和开发服务器的人 都清楚的知道 协议的细节 ~~

咱们作为第三方, 要想开发一个 redis 客户端
也就需要知道 redis 的应用层协议~~

暗号~~ (redis 自定义的应用层协议, 就是这个暗号)

能不能知道?
能!!! redis 官网上就有~~

redis 自定义的应用层协议的名字 => resp

RESP protocol spec

Redis serialization protocol (RESP) specification

序列化

协议~

RESP 协议

优点:
1. 简单好实现
2. 快速进行解析
3. 肉眼可读

传输层这里基于 TCP, 但是和 TCP 又没有强耦合~

请求和响应之间的通信模型是一问一答的形式~~

客户端给服务器发一个请求, 服务器返回一个响应~

前面学过的 redis 命令

Redis uses RESP as a request-response protocol in the following way:

- Clients send commands to a Redis server as a RESP Array of Bulk Strings.
- The server replies with one of the RESP types according to the command implementation.

客户端给服务器发送的是 redis 命令. (bulk string 数组的形式发送的)

不同的命令, 返回结果不一样~~
有的命令, 直接返回个 ok
有的命令, 可能返回个整数.
有的命令, 可能返回个数组.....

1. 协议设计特点

1. **简单易实现**：
   - 基于文本，使用 `\r\n` 分隔。
   - 几乎所有语言都能快速实现解析器。
2. **高效解析**：
   - 通过首字符即可判断数据类型：
     - `+` → 简单字符串
     - `-` → 错误
     - `:` → 整数
     - `$` → 批量字符串
     - `*` → 数组
   - 无需复杂的语法分析器，解析效率高。
3. **二进制安全**：
   - 批量字符串可传输任意二进制数据，包括图片、序列化对象。
   - 避免了编码转换开销。
4. **兼容性与可扩展性**：
   - 新命令或新数据类型无需修改协议结构。
   - 保证客户端与服务端的长期兼容。

---

2. 核心数据类型与格式

数据类型	首字符	格式示例	说明
简单字符串	+	+OK\r\n	返回成功状态或简短文本，不支持换行
错误	-	-ERR invalid command\r\n	返回错误信息
整数	:	:100\r\n	返回整数结果，如 INCR 或 LLEN
批量字符串	$	$5\r\nhello\r\n	支持任意二进制数据，长度前缀 $5，-1 表示 nil
数组	*	*2\r\n$3\r\nfoo\r\n$3\r\nbar\r\n	用于返回多个元素，如 MGET，数组内可嵌套其他 RESP 类型

- **批量字符串解析注意点**：
  - 长度为 `-1` → `nil`。
  - 长度 ⩾ 0 → 精确读取指定长度的内容。
- **数组解析注意点**：
  - 元素数量可能为 `0`（空数组）或 `-1`（nil 数组）。
  - 数组可嵌套数组或混合数据类型。

---

3. 请求-响应模型

Redis 客户端与服务端通信基于**请求-响应模型**：

1. **客户端发送请求**：
   - 所有命令都封装为 RESP 数组格式。
   - 示例：`SET key value`

```
代码块
1   *3\r\n$3\r\nSET\r\n$3\r\nkey\r\n$5\r\nvalue\r\n
```

2. **服务端返回响应**：
   - 根据命令类型返回不同 RESP 类型。
   - 示例：成功返回简单字符串：

```
代码块
1   +OK\r\n
```

   - 错误返回：

```
代码块
1   -ERR wrong number of arguments for 'set' command\r\n
```

3. **解析逻辑**：
   - 客户端接收原始字节流，根据首字符判断类型。
   - 批量字符串按长度读取，数组递归解析。

三、自定义客户端的实现步骤

实现自定义客户端的核心在于**编码请求 → TCP 传输 → 解码响应**。

1. 核心流程

1. **建立 TCP 连接**：
   - 默认端口 6379，可使用 `IP:Port` 或 Unix Socket。
   - 可选择阻塞/非阻塞模式。
2. **编码请求**：
   - 将命令和参数转换为 RESP 数组格式。
   - 示例（Java）：

代码块

```
String command = "*3\r\n$3\r\nSET\r\n$3\r\nkey\r\n$5\r\nvalue\r\n";
byte[] request = command.getBytes(StandardCharsets.UTF_8);
```

3. **发送请求**：
   - 写入 TCP Socket。
   - 对于高并发，通常使用缓冲区或异步写入。
4. **读取响应**：
   - 从 TCP Socket 读取数据。
   - 需处理半包、粘包问题（TCP 特性）。
5. **解码响应**：
   - 根据首字符判断类型，解析长度、内容。
   - 转换为客户端语言的原生数据类型（String、int、List）。
6. **关闭连接或复用**：
   - 可直接关闭，也可放入连接池复用，避免频繁建立连接开销。

---

2. 关键注意事项

1. **二进制安全**：
   - 批量字符串长度前缀 `$` 必须精确。
   - 传输二进制数据时，严格按照长度读取，不可依赖 `\0` 或换行符。
2. **空值处理**：

- 批量字符串长度为 `-1` → `nil`。
- 数组长度为 `-1` → nil 数组；长度为 `0` →空数组。

3. **数组解析**：
   - 数组元素可嵌套数组或混合类型。
   - 客户端需支持递归解析。
4. **性能优化**：
   - 连接池复用 TCP 连接，减少建立连接开销。
   - 批量命令或管道（pipeline）减少往返延迟。
   - 异步非阻塞 I/O 提升并发吞吐量。

四、核心设计思想与价值

1. **协议与业务解耦**：
   - RESP 协议只定义通信格式，与 Redis 命令逻辑解耦。
   - 新命令或数据结构无需修改协议，保证可扩展性。
2. **简单性优先**：
   - 设计理念：简单易实现 → 支持 50+ 种语言客户端。
   - 降低客户端开发门槛，提高生态普适性。
3. **性能与兼容性平衡**：
   - 二进制安全 → 可传输任意数据。
   - 高效解析 → 快速响应，满足高并发场景。
   - 协议兼容性 → 新命令、数组嵌套、二进制数据都可平滑支持。
4. **客户端价值**：
   - 深入理解协议 → 可定制高性能客户端。
   - 可实现特殊业务需求（自定义序列化、分布式功能）。
   - 对运维和性能优化提供工具基础。

五、实践与扩展

1. **自定义客户端实践建议**：
   - 先熟悉 Redis 命令和数据结构。
   - 实现最基础的请求-响应通信（如 GET/SET）。

- 扩展数组、管道、事务命令的支持。
- 加入连接池和异步支持。

2. **高级扩展方向**：
   - 支持订阅/发布（Pub/Sub）模式。
   - 支持 Cluster 集群通信（Slot 路由）。
   - 实现客户端重试、故障转移和健康检查机制。
   - 二进制协议加密（TLS/SSL）或压缩优化。

✅ **总结**：

- 开发自定义 Redis 客户端，本质是实现 **RESP 协议编码解码 + TCP 通信**。
- RESP 协议设计简单、高效、二进制安全，支持数组嵌套和批量传输。
- 自定义客户端有利于 **性能优化、业务定制和特殊场景扩展**。
- 理解协议、请求-响应模型和 TCP I/O 是自定义客户端开发的核心。
