Redis 高级应用指南：List、ACL 与阻塞队列（87~93）

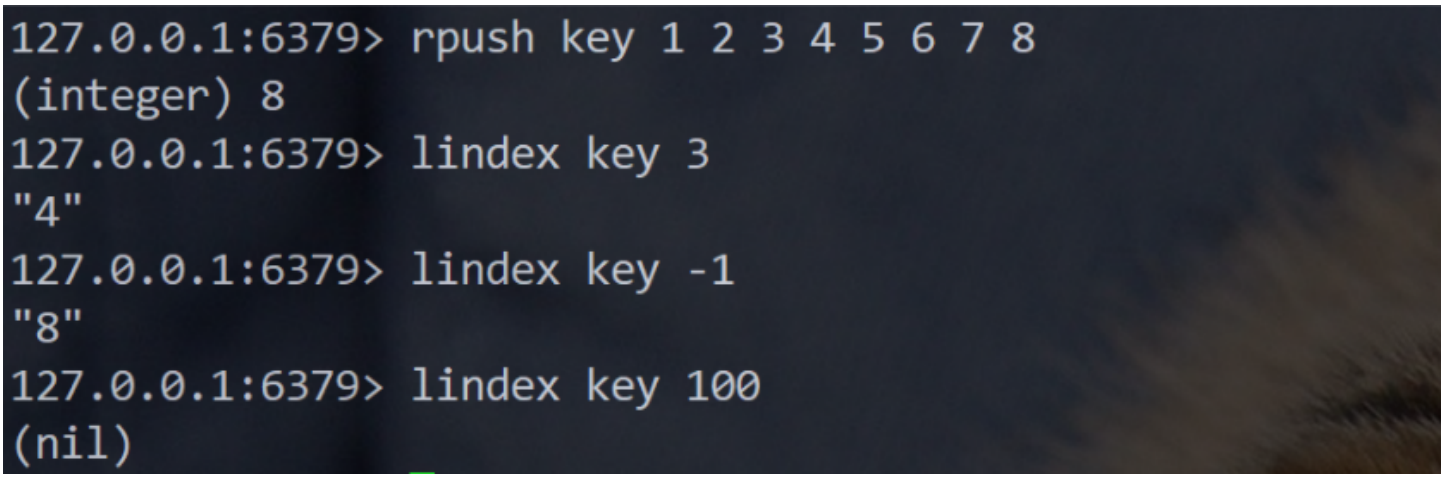
Redis 是一个高性能的内存数据库，其数据结构丰富，使用灵活，尤其是 List（列表）结构，结合阻塞操作和访问控制，可以实现很多高级应用场景。本文将详细介绍 Redis List 进阶命令、访问控制（ACL）、阻塞队列机制及核心设计思想与最佳实践。

一、Redis List 进阶命令详解

Redis 的 List 是一个双端链表（Linked List）结构，可以在两端高效插入和删除元素。除了基础命令（`LPUSH`、`RPUSH`、`LPOP`、`RPOP`），Redis 还提供了一些进阶命令用于元素定位、插入和删除。

1. 元素定位与插入

(1) LINDEX key index



- **功能**：根据索引获取列表中的元素。
- **索引规则**：
  - 正数索引：从 0 开始，0 表示列表头。
  - 负数索引：-1 表示列表尾，-2 表示倒数第二个元素，以此类推。
- **时间复杂度**：O(N)
  因为 Redis 的 List 是链表结构，访问中间或尾部元素需要遍历。
- **示例**：

```
1  RPUSH mylist "a" "b" "c"   # mylist = ["c", "b", "a"]
2  LINDEX mylist 0             # 返回 "c"
3  LINDEX mylist -1            # 返回 "a"
```

- **注意**：
  - 索引超出范围返回 `nil`。
  - 对大列表频繁使用 LINDEX 会影响性能。

---

(2) LINSERT key BEFORE|AFTER pivot element

```
LINSERT key BEFORE | AFTER pivot element
```

```
127.0.0.1:6379> linsert key before 4 100
(integer) 9
```

返回值是插入之后, 得到的新的 list 的长度

- **功能**：在指定元素 `pivot` 之前或之后插入新元素。
- **工作原理**：
  a. Redis 从左到右遍历列表，查找第一个匹配 `pivot` 的元素。
  b. 找到后在其前或后插入新元素。
- **时间复杂度**：O(N)
- **示例**：

代码块

```
1  RPUSH mylist "x" "y" "z"   # mylist = ["x", "y", "z"]
2  LINSERT mylist BEFORE "y" "a" # mylist = ["x", "a", "y", "z"]
3  LINSERT mylist AFTER "z" "b"  # mylist = ["x", "a", "y", "z", "b"]
```

- **应用场景**：
  - 用于在有序任务列表中插入紧急任务。
- **性能注意**：
  - 对大列表使用 LINSERT 会导致主线程阻塞，生产环境需谨慎。

---

(3) LSET key index element

- **功能**：将列表指定索引位置的元素替换为新值。
- **时间复杂度**：O(N)
- **示例**：

```
代码块
1   LSET mylist 1 "new_value"    # 将索引 1 的元素替换为 "new_value"
```

- **注意**：
  - 索引超出范围会返回错误。
  - 常用于修改某条任务状态或更新缓存数据。

---

2. 元素删除

(1) LREM key count element

- **功能**：删除列表中与 `element` 相等的元素。
- **参数说明**：
  - `count > 0`：从左到右删除 `count` 个匹配元素。
  - `count < 0`：从右到左删除 `|count|` 个匹配元素。
  - `count = 0`：删除列表中所有匹配元素。
- **时间复杂度**：O(N)
- **示例**：

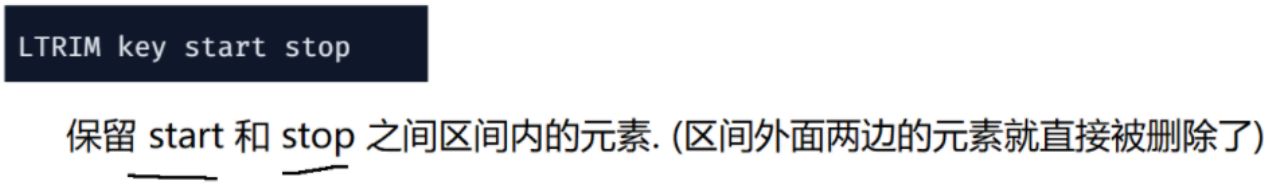
```
代码块
```

```
RPUSH mylist "a" "b" "a" "c" "a"   # mylist = ["a","b","a","c","a"]
LREM mylist 2 "a"                    # 删除左侧两个 "a"，结果 ["b","c","a"]
```

- **应用场景**：
  - 删除重复的任务或日志条目。
  - 数据清理时移除特定标记元素。

---

(2) LTRIM key start stop



- **功能**：修剪列表，只保留指定范围的元素。
- **时间复杂度**：O(N)
- **示例**：

代码块

```
LTRIM mylist 0 2   # 保留索引 0~2 的元素
```

- **应用场景**：
  - 限制时间线或日志的最大长度，防止列表无限增长。
  - 保留最近 N 条记录，如最近 100 条通知。
- **注意**：
  - 不在范围内的元素会被直接删除。
  - 结合 `LPUSH` 使用可以实现“固定长度队列”。

二、Redis 访问控制（ACL）

Redis 6.0 引入 ACL（Access Control List，访问控制列表），提供对不同用户的**命令和 key 级别的权限控制**。

1. 核心功能

- **用户管理**：

- 可以创建多个 Redis 用户，每个用户有独立密码。
- 允许针对每个用户限制命令和 key 的访问。

- **命令权限控制**：
  - 白名单方式：指定允许执行的命令集合。
  - 黑名单方式：禁止执行某些危险命令，如 `FLUSHDB`、`CONFIG`。
- **动态管理**：
  - 可通过 `ACL SETUSER username on >password ~pattern +command` 动态调整权限。
  - 可查看用户权限：`ACL LIST`、`ACL GETUSER username`。

---

2. 典型应用场景

- **多租户环境**：
  - 不同业务系统共享 Redis 实例。
  - 为分析用户配置只读权限，禁止修改数据。
- **安全防护**：
  - 阻止普通用户执行破坏性命令（如删除所有数据）。
- **细粒度控制**：
  - 针对 key 模式限制访问，如仅允许访问 `cache:*` 前缀。

示例：

```
代码块
1   ACL SETUSER analyst on >password ~cache:* +get +mget
```

- `analyst` 用户只能访问 key 前缀为 `cache:` 的数据，并只能执行 `GET` 和 `MGET` 命令。

三、阻塞式列表命令与阻塞队列

Redis 的阻塞列表命令使 List 结构可以实现轻量级阻塞队列，常用于任务调度和异步处理。

1. 阻塞式弹出命令

(1) BLPOP key [key ...] timeout

- **功能**：从列表头部弹出元素，如果列表为空则阻塞。

- **参数**：
  - `timeout`：阻塞时间（秒），0 表示永久阻塞。
- **特点**：
  a. 阻塞机制：空列表时客户端会挂起，等待新元素。
  b. 多键监听：可同时监听多个列表，返回第一个非空列表元素。
  c. 公平性：多个客户端阻塞时，先到先得。
  d. 非阻塞兼容：列表非空时立即返回。
- **示例**：

```
代码块
1   BLPOP task_queue 5   # 阻塞 5 秒等待 task_queue 的元素
```

(2) BRPOP key [key ...] timeout

- 与 BLPOP 类似，但从列表尾部弹出元素。

---

2. 阻塞队列的应用

结合 List 的阻塞操作可以实现轻量级任务队列模式。

生产者

```
代码块
1   LPUSH task_queue "task1"   # 插入新任务
```

消费者

```
代码块
1   BLPOP task_queue 0         # 永久阻塞，等待任务
```

注意事项

1. **无消息确认机制**：
   - 阻塞队列消费失败会导致消息丢失。
   - 不适合强可靠性场景。

2. **适用场景**：
   - 日志收集、通知推送、轻量异步任务处理。
3. **高可靠方案**：
   - 对可靠性要求高，建议使用 Kafka、RabbitMQ 等专业消息队列。

四、核心设计思想与最佳实践

1. **命令复杂度权衡**：
   - 高复杂度命令（`LINSERT`、`LREM`）在大列表中可能阻塞主线程。
   - 优先使用 O(1) 命令（`LPUSH`、`RPUSH`、`LPOP`、`RPOP`）。
2. **阻塞队列适用场景**：
   - 轻量级任务队列。
   - 避免空轮询，提高 CPU 利用率。
3. **ACL 安全规范**：
   - 生产环境必须开启 ACL。
   - 为不同角色配置最小权限。
   - 防止误操作或恶意攻击。
4. **列表长度控制**：
   - 使用 `LTRIM` 修剪列表。
   - 保留最近 N 条记录，防止内存无限增长。
5. **结合实际业务场景**：
   - 阻塞队列可用于异步任务处理，但需结合日志、备份等策略确保可靠性。
   - ACL 和 List 命令结合可实现安全、稳定的消息队列服务。

✅ 总结

- **Redis List**：支持双端操作、定位、插入和删除操作，但复杂命令需注意性能。
- **ACL**：提供细粒度用户权限控制，保障多租户和安全场景。
- **阻塞队列**：BLPOP/BRPOP 能实现轻量级异步任务处理，但不保证可靠性。
- **最佳实践**：
  - 优先选择 O(1) 操作。
  - 阻塞队列用于低可靠性场景。

- ACL 限权保护生产环境。
- 定期 LTRIM 控制列表长度。
