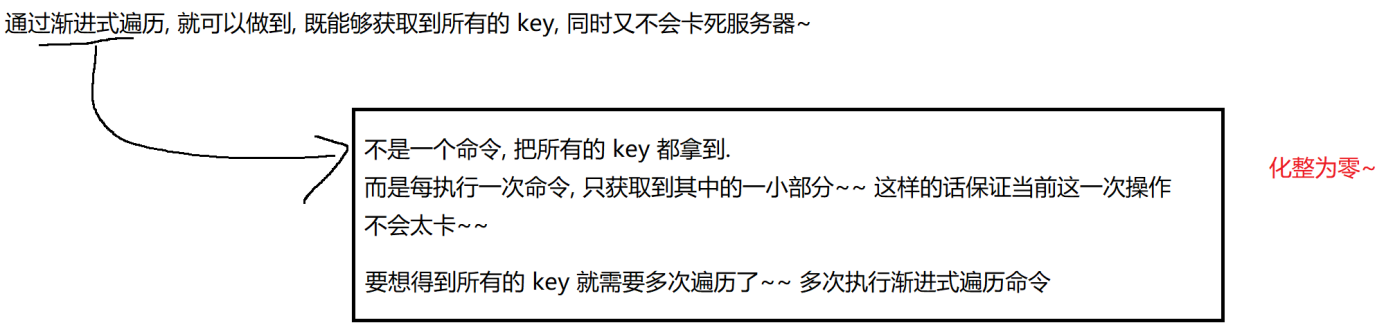
Redis 全量键遍历与数据库特性深度解析（140 ~ 144）

Redis 作为高性能内存数据库，**键空间遍历和数据库管理**是核心基础操作。合理理解和使用这些操作，对于保证 Redis 的稳定性和高性能至关重要。

一、Redis 全量键遍历：`KEYS` 与 `SCAN` 的对比



1. `KEYS` 命令：高风险的全量遍历

- **命令定义**：`KEYS pattern` 返回所有匹配 `pattern` 的键。例如：

```
代码块
1   KEYS user:*
```

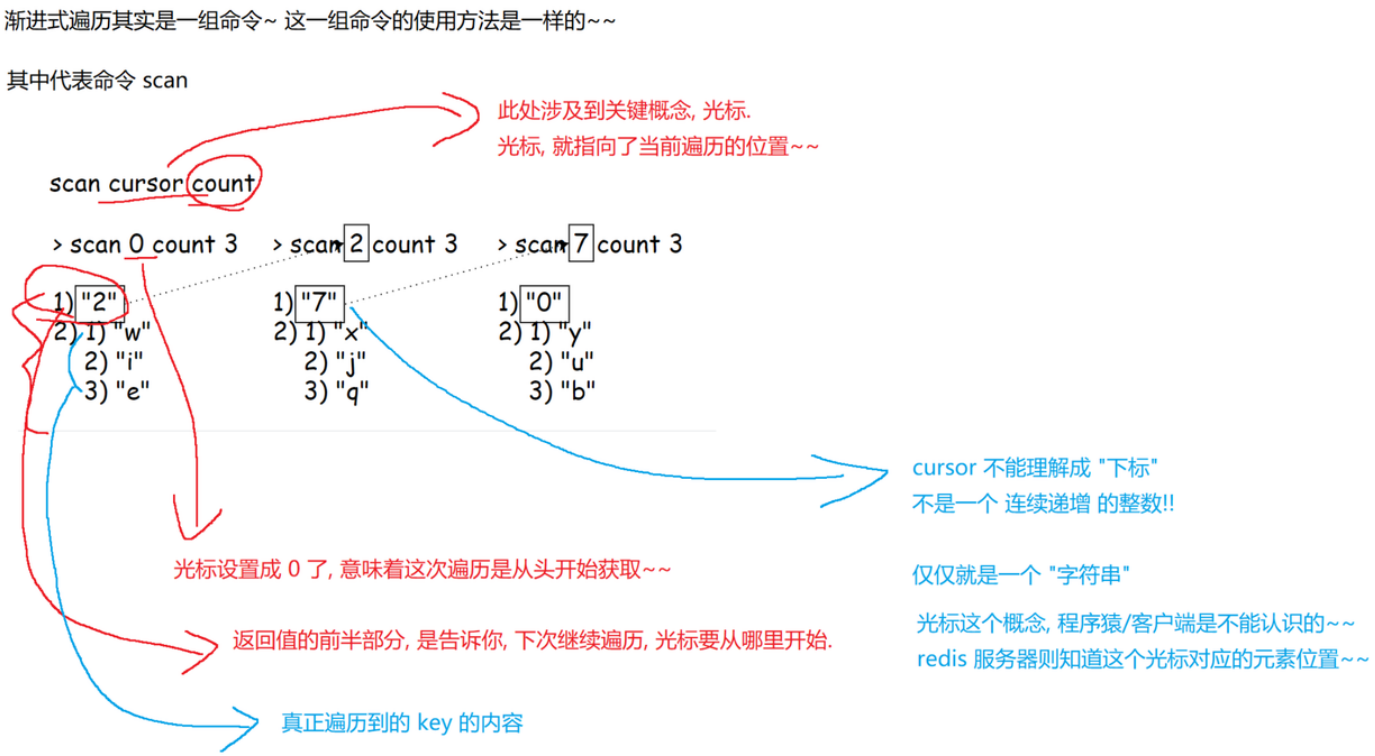
- 会返回所有以 `user:` 开头的键。
- **问题与风险**：
  a. **阻塞风险**
     - Redis 是单线程模型，`KEYS` 会遍历整个键空间（哈希表），时间复杂度为 **O(N)**，N 为数据库中键的总数量。
     - 当键数量非常大（百万级以上）时，`KEYS` 会占用主线程，导致**阻塞 Redis 所有操作**，客户端请求延迟或超时。
  b. **性能隐患**
     - 即使只需要返回少量匹配键，也必须扫描整个键空间，效率低下。
     - 在生产环境中使用 `KEYS` 是**严重的性能杀手**，通常仅用于小型测试环境。

- **总结**：
  `KEYS` 简单但危险，适合调试或小规模数据，生产环境应严格避免。

---

2. `SCAN` 命令：渐进式、安全遍历

为解决 `KEYS` 的阻塞问题，Redis 提供了 **SCAN 系列命令**，支持**渐进式遍历**。



- **命令格式**：

```
代码块
1   SCAN cursor [MATCH pattern] [COUNT count] [TYPE type]
```

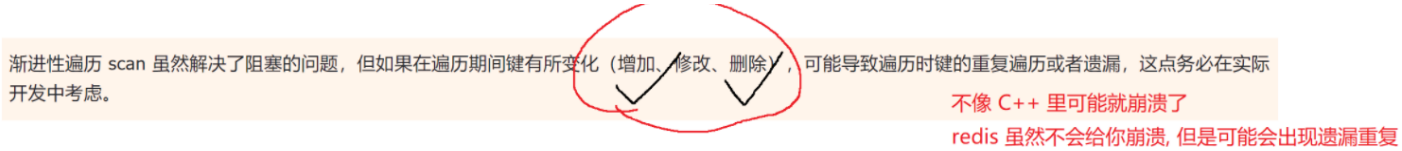
  - `cursor`：游标，第一次调用传入 `0`，之后传入上一次返回的游标值，直到返回 `0` 表示遍历完成。
  - `MATCH pattern`：可选，匹配键的正则模式（与 `KEYS` 相同）。
  - `COUNT count`：可选，提示 Redis 每次返回的键数量（默认 10）。注意，它是 **提示值**，并非严格限制返回数量。
  - `TYPE type`：可选，仅返回指定类型的键（如 `string`、`hash`）。

- **关键特性**：

  a. **非阻塞**：每次迭代只返回部分键，不会占用过多 CPU。

  b. **游标驱动**：通过游标记录遍历位置，可断点续传。

c. **重复与遗漏**：遍历过程中可能有重复键，也可能遗漏某些键，客户端需处理去重。



- **示例执行流程**：

```
SCAN 0 MATCH user:* COUNT 3 → 游标 3，返回键 [user:1, user:2]
SCAN 3 MATCH user:* COUNT 3 → 游标 5，返回键 [user:3]
SCAN 5 MATCH user:* COUNT 3 → 游标 0，返回键 [user:4]
```

  - 游标返回 `0` 表示遍历结束
  - 每次返回的键数量不固定，但整体遍历完所有键空间

- **工程实践**：
  - 用 `SCAN` 遍历大量键，配合客户端去重集合（HashSet）保证准确性。
  - 可以在后台批处理任务中使用，避免阻塞主线程。

二、`SCAN` 系列命令与底层机制

1. 系列命令

Redis 针对不同数据类型提供了专门的 SCAN 命令：

命令	遍历对象	描述
SCAN	全库键	遍历整个数据库的键空间
HSCAN key cursor	Hash	遍历 hash 的 field-value 对
SSCAN key cursor	Set	遍历 set 元素
ZSCAN key cursor	ZSet	遍历 zset 元素及分数

- 设计目标都是**渐进式非阻塞**，适合生产环境大数据量遍历。

---

2. 底层实现机制

- Redis 键空间基于 **哈希表**，每个 key 都落在哈希桶上。
- **SCAN** 的实现：
  - 游标实际上是哈希桶索引
  - 每次迭代只扫描部分哈希桶，返回对应键
  - 当哈希表动态扩容或缩容时，某些键可能被重新分配到不同桶，导致：
    - **重复返回**：键可能出现多次
    - **可能遗漏**：动态新增或删除键可能被跳过
- **客户端去重**：建议维护 HashSet 或其他数据结构，保证遍历结果唯一。
- **线程安全**：
  - Redis 单线程模型保证命令原子性
  - SCAN 本身是无状态的，服务端不保存游标，客户端负责游标管理

三、Redis 数据库（Database）特性

1. 数据库概念

- Redis 默认支持 16 个数据库（编号 0-15），每个数据库有独立键空间。
- 数据库切换：

```
代码块
1   SELECT 1
```

- 数据库之间的键互不干扰，但共享 Redis 实例的内存和持久化策略。

---

2. 与 MySQL 数据库对比

特性	Redis 数据库	MySQL 数据库
隔离性	轻量级，只有键空间隔离	完整隔离，包含表、索引、事务等
持久化	所有数据库共享同一个 RDB/AOF 文件	每个数据库独立文件
使用场景	键前缀隔离业务	完整逻辑隔离业务

- **实践建议**：
  - 生产环境只用数据库 0
  - 通过键前缀实现业务隔离（如 `user:1`、`order:1`）
  - 避免使用 `SELECT` 来切换数据库

---

3. 数据库操作命令

命令	描述	注意事项
SELECT dbindex	切换数据库	建议仅开发或测试使用
FLUSHDB	清空当前数据库所有键	高危命令，生产环境需禁止
FLUSHALL	清空所有数据库所有键	禁止在生产环境使用，风险极高

四、核心设计思想与最佳实践

1. 非阻塞遍历设计原则

- `SCAN` 通过**渐进式迭代**平衡遍历效率和性能影响
- 生产环境中：
  - **全量遍历** → 必须使用 `SCAN`，不要用 `KEYS`
  - **客户端处理** → 去重、断点续传、异常重试

---

2. 数据库隔离最佳实践

- **业务隔离**：键前缀比切换数据库更灵活
- **限制高危操作**：通过 ACL 或运维策略禁止 `FLUSHDB` / `FLUSHALL`
- **监控**：生产环境定期检查数据库键数量和内存占用

---

3. 客户端实现注意事项

- **游标管理**：
  - 每次调用 `SCAN` 需传入上次游标

    - 遍历结束游标为 `0`
- **去重机制**：
    - 避免重复键影响业务逻辑
    - 可使用集合或 HashSet 存储已返回的键
- **容错与恢复**：
    - 遍历过程中 Redis 异常 → 可从游标 `0` 重新开始
    - 可结合日志或批处理系统实现增量恢复

五、总结与工程实践

1. **不要使用 KEYS**：生产环境高风险
2. **使用 SCAN 系列命令**：非阻塞、渐进式、支持大数据量
3. **客户端去重 + 游标管理**：保证遍历完整性
4. **数据库隔离**：通过键前缀而非切换数据库
5. **高危命令限制**：`FLUSHDB` / `FLUSHALL` 需 ACL 控制
