Redis Set 深度解析：命令、业务场景与性能优化（100~109）

Redis 是高性能内存数据库，其 **Set（集合）** 是一种存储**无序且唯一元素**的数据结构。集合底层基于哈希表实现，支持高效的增删查操作，并提供强大的集合运算功能。本文将系统讲解 Redis Set 的基础命令、集合运算、典型业务场景以及性能优化策略。

一、Redis Set 基础命令与核心特性

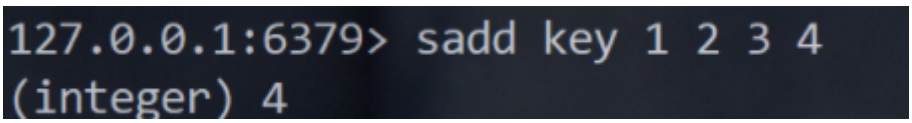
1. 基础增删查命令

Redis Set 的核心特性是 **唯一性、无序性、快速访问**。下面详细介绍每个基础命令的用法、原理和注意事项。

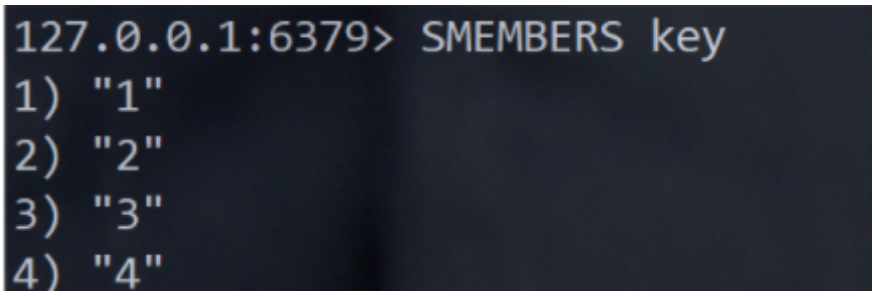
命令	作用与细节
SADD key member [member ...]	向集合中添加一个或多个元素。**唯一性**保证重复元素不会被添加，返回实际新增的元素数量。**示例**：SADD tags "tech" "sports"
SMEMBERS key	返回集合中所有元素，时间复杂度 **O(N)**。大集合会阻塞 Redis 主线程，生产环境建议用 **SSCAN** 分批遍历。
SISMEMBER key member	判断元素是否存在于集合中，时间复杂度 **O(1)**。适合频繁判断操作，例如用户标签检查。
SCARD key	返回集合元素数量，时间复杂度 **O(1)**，读取元数据即可，无需遍

	历。
SPOP key [count]	随机移除并返回一个或多个元素。**注意**：Redis 内部随机是哈希表遍历顺序，并非完全均匀随机。示例：SPOP prizes 1
SMOVE source destination member	原子操作，将元素从 source 集合移动到 destination 集合。若 source 无该元素，返回 0；若 destination 已存在该元素，仅从 source 删除。
SREM key member [member ...]	删除集合中一个或多个元素，返回实际删除数量。时间复杂度 O(1) 单元素。

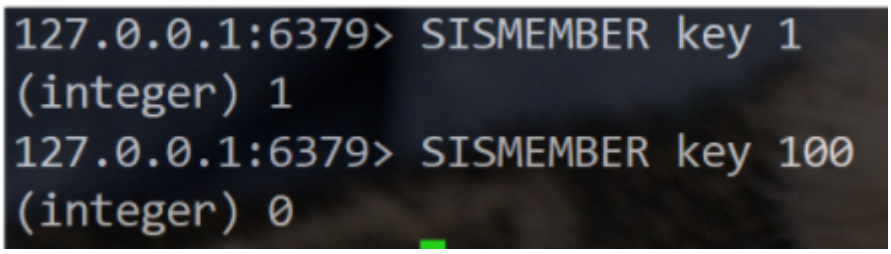
1. sadd



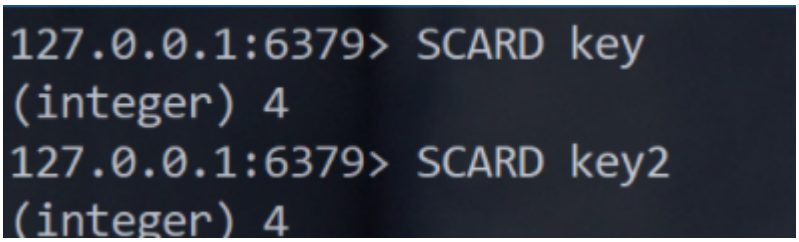
2. Smembers



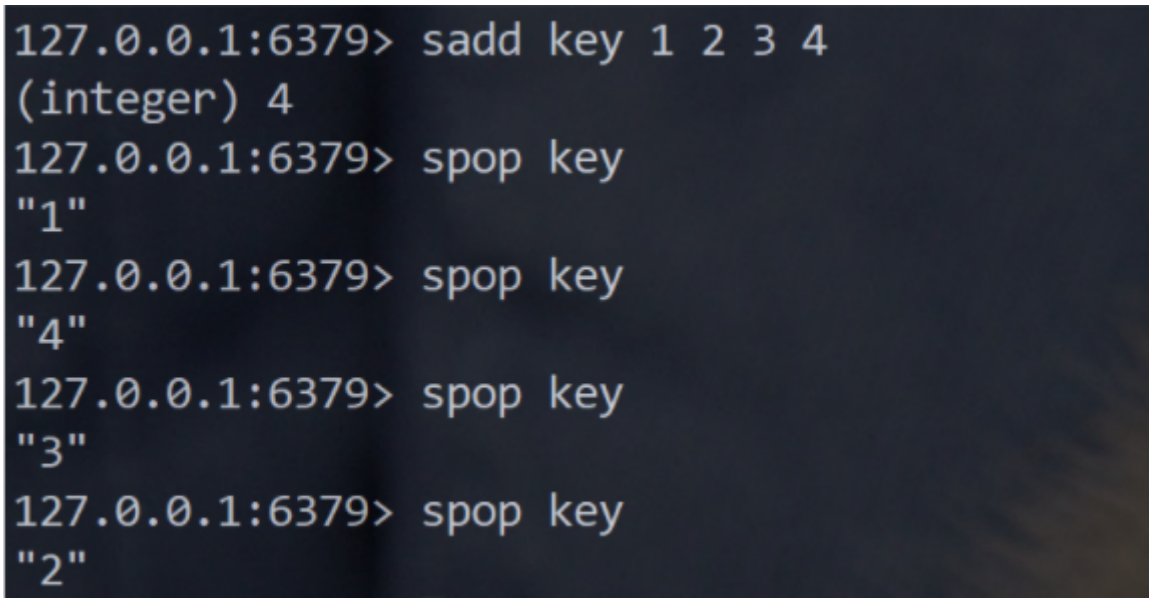
3. Sismember



4. Scard



5. Spop



---

2. 核心特性

1. 唯一性

- 集合自动去重，适合存储不允许重复的元素，例如用户标签、已读消息 ID、任务唯一标识。
- 示例：

代码块

```
SADD user:1:tags "tech"
SADD user:1:tags "tech"  # 不会重复添加
```

2. 无序性

- 元素存储顺序与插入顺序无关。
- 底层采用哈希表，增删查时间复杂度均为 **O(1)**。
- 示例：

代码块

```
SADD set1 1 2 3
```

```
2   SMEMBERS set1  # 返回顺序可能是 2,3,1
```

3. **随机性**
   - `SPOP`、`SRANDMEMBER` 可随机获取元素，适合抽奖、推荐系统。
   - 注意：随机性是哈希表遍历顺序，不是严格均匀随机。

二、Set 集合运算命令

Redis Set 提供**高效的集合运算**，可实现交集、并集和差集操作。

1. 基础集合运算

运算类型	含义	命令示例
交集（Inter）	多个集合的共同元素	SINTER key1 key2
并集（Union）	多个集合的所有元素（去重）	SUNION key1 key2
差集（Diff）	在集合 A 中存在但在其他集合中不存在的元素	SDIFF key1 key2

- **集合运算原理**：
  - Redis 使用底层哈希表快速查找元素。
  - 时间复杂度通常为 O(M*N)，M 为最小集合大小，N 为集合数量。

2. 存储型运算命令

- `SINTERSTORE`、`SUNIONSTORE`、`SDIFFSTORE`：
  - 将运算结果存储到目标集合，避免重复计算。
  - 示例：

代码块

```
1   SINTERSTORE common:friends user:1:friends user:2:friends
```

- **适用场景**：
  - 频繁复用运算结果。
  - 减少重复集合计算的性能开销。

三、典型业务场景

1. 标签系统

- **应用示例**：

```
代码块
1   SADD user:1:tags "tech" "sports"
2   SISMEMBER user:1:tags "tech"  # true
3   SMEMBERS user:1:tags           # 返回所有标签
```

- **推荐系统**：
  - 计算共同兴趣：

```
代码块
1   SINTER user:1:tags user:2:tags
```

  - 交集越大表示兴趣越相似，可用于好友推荐或内容推荐。

2. 抽奖/随机推荐

- **随机抽奖**：

```
代码块
1   SPOP prizes 1  # 从奖品集合随机抽取一名获奖者
```

- **随机推荐**：
  - 若需要保留原集合，可用 `SRANDMEMBER`：

```
代码块
1   SRANDMEMBER products 5  # 随机获取 5 个商品
```

3. 好友关系与共同关注

- **好友关系存储**：

```
代码块
```

```
SADD user:1:friends 2 3 4
SADD user:2:friends 3 4 5
```

- **计算共同好友**：

```
代码块
SINTER user:1:friends user:2:friends  # 返回 3,4
```

- **分析单向关注**：

```
代码块
SDIFF user:1:friends user:2:friends  # 返回 user:1 关注但 user:2 未关注的好友
```

四、性能注意事项与最佳实践

1. **避免全量遍历**
   - `SMEMBERS` 返回所有元素，若集合过大，会阻塞 Redis 主线程。
   - **推荐**使用 `SSCAN` 进行增量扫描：

```
代码块
SSCAN key 0 MATCH pattern COUNT 100
```

2. **集合运算的性能风险**
   - 大集合频繁执行 `SINTER`、`SUNION` 会增加 CPU 开销。
   - **优化策略**：
     - 预计算结果并存储到新集合中，减少重复计算。
     - 示例：

```
代码块
SINTERSTORE cached:common_tags user:1:tags user:2:tags
```

3. **随机命令的特性**

- `SPOP` 和 `SRANDMEMBER` 的随机性基于哈希表内部结构。
- 对于严格均匀随机抽样，可结合业务逻辑二次处理。

4. **数据量适配**
   - Redis 会自动选择集合编码：
     - **intset**：元素为整数且数量小，内存紧凑。
     - **hashtable**：元素较多或包含非整数，支持 O(1) 增删查。
   - 开发者无需干预，但理解编码切换有助于分析性能瓶颈。
5. **集合操作最佳实践**
   - 小集合：可直接使用 `SMEMBERS` 和集合运算命令。
   - 大集合：使用 `SSCAN` 和 `*STORE` 命令减少性能压力。
   - 避免频繁随机抽取大量元素，推荐先缓存或分片处理。

五、Redis Set 集合运算命令与底层编码

4. 核心集合运算命令

Redis 提供的集合运算命令可以对多个集合进行交集、并集和差集计算，同时支持直接存储结果，避免重复计算。

命令	功能与细节	示例
SUNION key [key ...]	返回多个集合的**并集**，即所有集合的元素去重后的结果。时间复杂度 **O(N)**，N 为参与集合的总元素数。	SUNION tags:male tags:sports
SUNIONSTORE destination key [key ...]	计算多个集合的并集并存储到 destination 集合中，避免重复计算。	SUNIONSTORE tags:combined tags:male tags:sports
SDIFF key [key ...]	返回多个集合的**差集**，即在第一个集合中存在、但在其他集合中不存在的元素。	SDIFF friends:user:1 friends:user:2
SDIFFSTORE destination key [key ...]	计算差集并存储到目标集合，减少重复运算开销。	SDIFFSTORE friends:unique friends:user:1 friends:user:2
SINTER key [key ...]	返回多个集合的**交集**，即所有集合的共同元素。时间复杂度 **O(M*K)**（M 为最小集合大小，K 为集合数量），大集合场景需谨慎使用。	SINTER tags:male tags:sports
SINTERSTORE destination key [key ...]	计算交集并存储到目标集合，适合频繁复用交集结果的业务场景。	SINTERSTORE tags:active_male tags:male tags:active

`SUNION key [key ...]`

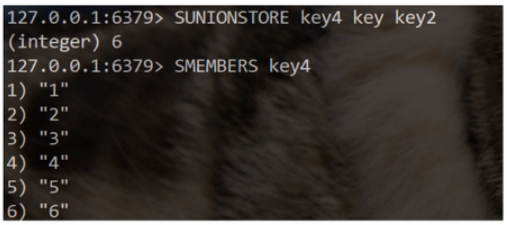
sunion 返回的就是并集的结果数据.

时间复杂度, O(N)　　N 指的是总的元素个数.

`SUNIONSTORE destination key [key ...]`

直接把并集的结果存储到 destination 对应的 集合 中.

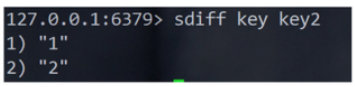
返回值并集的元素个数.

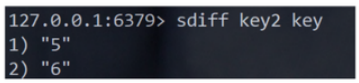


`SDIFF key [key ...]`

返回值是差集的结果.

时间复杂度: O(N)

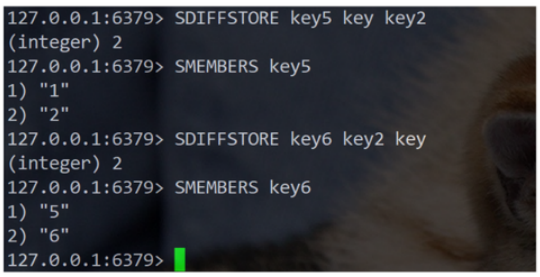




`SDIFFSTORE destination key [key ...]`

返回值就是差集的元素个数

时间复杂度: O(N)



使用示例：

代码块

```
# 用户标签集合
SADD user:1:tags "sports" "tech" "music"
SADD user:2:tags "tech" "movies" "music"

# 并集
SUNION user:1:tags user:2:tags
# 输出: "sports", "tech", "music", "movies"

# 交集
SINTER user:1:tags user:2:tags
# 输出: "tech", "music"

# 差集
SDIFF user:1:tags user:2:tags
# 输出: "sports"
```

2. 底层编码与优化

Redis Set 底层根据元素类型和数量使用 **两种编码**：

1. **intset（整数集合）**
   - 当集合元素为整数且数量较少时使用。
   - 内存占用小，存储在紧凑的数组中。
   - 增删查复杂度为 **O(N)**（小集合，性能影响可忽略）。
   - 自动切换阈值：
     - 集合大小或整数范围超过阈值，自动切换为 **hashtable**。
2. **hashtable（哈希表）**
   - 元素为字符串或集合较大时使用。
   - 增删查复杂度为 **O(1)**。
   - 内存消耗比 intset 高，但可支持大集合和高并发访问。

开发者优化思路：

- 对小集合存储整数元素，可充分利用 intset 的内存优势。
- 对大集合或混合类型集合，使用 hashtable，确保 O(1) 的操作性能。
- 编码切换是 Redis 自动完成的，无需手动干预，但理解机制有助于优化内存与性能。

六、Set 的典型业务场景

Redis Set 的核心优势在于 **唯一性、无序性、集合运算能力**，可广泛应用于用户画像、社交关系、去重统计等场景。

1. 用户画像与标签系统

- **场景说明**：
  - 每个用户打标签（性别、兴趣、消费行为、活跃度等）。
  - 通过集合运算分析用户特征，实现精准营销和推荐。
- **实现示例**：

代码块

```
# 用户标签
SADD user:1:tags "female" "25-30" "sports" "premium"
SADD user:2:tags "female" "25-30" "sports"

# 交集 - 查找 25-30 岁女性喜欢运动的用户
```

```
SINTER user:1:tags user:2:tags
# 输出: "female", "25-30", "sports"

# 并集 - 合并多个标签用户群体
SUNION user:1:tags user:2:tags
# 输出: "female", "25-30", "sports", "premium"
```

- **商业价值**：
  - 精准营销：通过交集获取目标群体。
  - 用户分层：利用标签组合区分高价值用户与潜在用户。
  - 个性化推荐：基于标签交集计算兴趣相似度。

---

2. 社交关系计算

- **好友关系**：

代码块

```
SADD user:1:friends 2 3 4
SADD user:2:friends 3 4 5

# 共同好友
SINTER user:1:friends user:2:friends
# 输出: 3, 4

# 单向关注分析
SDIFF user:1:friends user:2:friends
# 输出: 2
```

- **应用场景**：
  - 社交平台共同好友推荐。
  - 关系图分析，辅助好友推荐算法。
  - 单向关注统计，如分析未互粉用户。

---

3. 数据去重与统计

- **PV / UV 统计**：
  - **PV（页面浏览量）**：每次访问增加计数即可，用 String。

- **UV（独立访客数）**：需去重，用 Set 存储访问用户 ID，`SCARD` 获取集合大小。

```
# UV 统计
SADD uv:20260126 user:1001
SADD uv:20260126 user:1002
SCARD uv:20260126
# 输出: 2
```

- **优势**：
  - 去重天然支持。
  - 增删查时间复杂度 O(1)，适合高并发环境。
- **时间窗口管理**：
  - 为日/周/月统计，可通过 key 前缀区分：

```
uv:20260126  # 日 UV
uv:2026w4    # 周 UV
uv:2026m1    # 月 UV
```

  - 便于定期清理旧集合，避免无限增长。

七、延伸思考：互联网大厂的业务与技术逻辑

1. 用户分层的商业价值

- 用户分层是精准营销和流量变现的核心。
- 通过 Set 存储的用户标签，可以将用户按兴趣、消费行为、活跃度分层。
- 示例：
  - 高价值用户 → 定向推送优惠券。
  - 潜在用户 → 推送低价引流商品。
- 核心逻辑：**通过集合运算快速划分群体，实现业务目标**。

2. 技术与业务协同

- 大厂技术方案本质上是为业务目标服务。
- Set 的集合运算能力为“用户画像、社交关系、去重统计”等场景提供高效解决方案。

- 技术选型原则：
  a. 解决业务痛点优先。
  b. 性能优化要结合业务特性，如 UV 统计用 Set 而非 String。

八、性能注意事项与最佳实践

1. **大集合运算风险**
   - `SINTER`、`SUNION` 对大集合性能影响大。
   - **优化策略**：
     - 使用 `*STORE` 命令预计算并存储结果。
     - 示例：

```
代码块
1   SINTERSTORE tags:active_female tags:female tags:active
```

2. **编码感知优化**
   - 小整数集合 → `intset`，节省内存。
   - 字符串或大集合 → `hashtable`，保证 O(1) 访问。
   - 了解编码切换有助于调优内存和性能。
3. **时间窗口管理**
   - UV/活跃用户统计等需按日/周/月管理。
   - 避免集合无限增长，定期清理历史 key。
4. **渐进式遍历替代全量查询**
   - 大集合避免使用 `SMEMBERS`。
   - **推荐**使用 `SSCAN`：

```
代码块
1   SSCAN uv:20260126 0 MATCH user:* COUNT 100
```

   - 可分批遍历集合元素，降低阻塞风险。
5. **随机命令使用注意**
   - `SPOP`、`SRANDMEMBER` 随机性基于哈希表遍历顺序。

- 若业务需要严格均匀随机，可结合应用逻辑进行二次处理。

✅ 总结

- **Redis Set 核心特性**：
  - **唯一性**：天然去重，适合 UV 统计、用户标签。
  - **无序性**：基于哈希表，增删查 O(1)。
  - **集合运算能力**：交集、并集、差集，高效支撑用户画像、社交关系计算。
- **命令体系**：
  - 基础增删查：`SADD`、`SREM`、`SISMEMBER`、`SCARD`、`SPOP`。
  - 集合运算：`SINTER`、`SUNION`、`SDIFF` 及对应 `*STORE` 命令。
- **典型场景**：
  a. 用户画像与标签系统 → 精准营销。
  b. 社交关系分析 → 好友推荐、单向关注统计。
  c. 数据去重与统计 → UV、独立用户计数。
- **性能优化**：
  - 大集合使用 `SSCAN`，避免阻塞。
  - 高频集合运算使用 `*STORE` 缓存结果。
  - 注意编码切换，利用 intset 内存紧凑优势。
  - 时间窗口管理，避免集合无限增长。
