Redis ZSet（有序集合）全面解析（110 ~ 133）

Redis 的 **ZSet（Sorted Set，有序集合）** 是 Redis 中非常重要的数据类型之一，它结合了 **集合唯一性** 和 **按分数排序** 的特性，在排行榜、任务队列、时间序列等场景中应用广泛。本文将从核心概念、命令详解、底层实现、业务案例到性能优化全面展开。

一、ZSet 核心概念与特性

1. ZSet 的本质

ZSet 是一种**有序集合**，每个元素（`member`）都关联一个浮点数分数（`score`），并按分数进行排序。

- **元素唯一性**：集合内的 `member` 不可重复，重复添加会更新分数。
- **分数可重复**：不同元素可以有相同分数。
- **排序规则**：
  a. 首先按 `score` 从小到大排序。
  b. 若 `score` 相同，则按 `member` 的字典序（lexicographical order）排序，保证顺序确定性。

示例：

代码块

```
ZADD leaderboard 95.5 "Alice"
ZADD leaderboard 97.2 "Bob"
ZADD leaderboard 95.5 "Charlie"

ZRANGE leaderboard 0 -1 WITHSCORES
# 输出顺序: "Alice", 95.5; "Charlie", 95.5; "Bob", 97.2
```

2. 与 List、Set 的核心区别

数据类型	有序性	元素唯一性	排序依据	典型场景
List	插入顺序	允许重复	插入顺序	时间线、消息队列
Set	无序	唯一	无	标签、去重统计
ZSet	分数排序	唯一	分数 + 字典序	排行榜、带权重任务队列、时间序列

总结：ZSet 是 **Set + 排序能力** 的升级版，能同时满足去重和排序需求。

二、ZSet 核心命令与行为细节

1. 基础增删改命令

命令	功能与细节
`ZADD key [NX	XX
ZRANGE key start stop [WITHSCORES]	按分数升序获取指定范围的元素，支持负索引。WITHSCORES 返回元素和分数
ZREVRANGE key start stop [WITHSCORES]	按分数降序获取元素，适合排行榜从高到低展示
ZSCORE key member	获取元素的分数，时间复杂度 O(1)
ZREM key member [member ...]	删除一个或多个元素，返回删除数量
ZCARD key	获取元素总数，时间复杂度 O(1)

2. 极值删除命令

命令	功能	时间复杂度	特点
ZPOPMAX key [count]	移除并返回分数最高的 count 个元素	O(logN * count)	分数相同时按字典序降序删除
ZPOPMIN key [count]	移除并返回分数最低的 count 个元素	O(logN * count)	分数相同时按字典序升序删除
ZREMRANGEBYRANK key start stop	删除指定排名范围的元素	O(logN + K)	支持分页删除大集合
ZREMRANGEBYSCORE key min max	删除指定分数范围的元素	O(logN + K)	可用于清理过期或低价值元素

- **底层逻辑**：
  - 跳表的头节点指向最小元素，尾节点指向最大元素
  - 删除操作需要调整跳表索引 → `O(logN)`
  - 删除 `count` 个元素 → 总时间复杂度 `O(logN * count)`
- **应用示例**：

代码块

```
# 删除排行榜前10名玩家
ZREMRANGEBYRANK leaderboard 0 9

# 删除分数 <= 50 的低分用户
ZREMRANGEBYSCORE leaderboard -inf 50
```

数据结构	是否允许重复元素	是否有序	有序依据	应
列表	是	是	索引下标	时间轴、 消息队列
集合	否	否		标签、社 交
有序集合	否	是	分数	排行榜系 统

命令	功能与特点	时间复杂度
ZCOUNT key min max	统计分数在 [min, max] 区间内的元素数量- 支持闭区间 [min,max] 或开区间 (min,max)- 支持边界 -inf 和 +inf，例如 ZCOUNT key -inf +inf 等同于 ZCARD key	O(logN)
ZRANGE key start stop [WITHSCORES]	按分数升序获取指定排名范围的元素，支持负索引（如 -1 表示最后一个元素）- WITHSCORES 返回元素及分数	O(logN + K)，K 为返回元素数量
ZREVRANGE key start stop [WITHSCORES]	按分数降序获取指定排名范围的元素，适合排行榜从高到低展示	O(logN + K)
ZRANGEBYSCORE key min max [WITHSCORES] [LIMIT offset count]	按分数范围 [min, max] 获取元素，支持分页- LIMIT offset count 用于分页查询- Redis 6.2.0+ 已整合到 ZRANGE + BYSCORE 参数	O(logN + K)

注意点：

- `ZCOUNT` 统计元素数量，但不会返回元素内容；
- `ZRANGE` / `ZRANGEBYSCORE` 可返回元素和分数，适合分页显示或做排名列表。

3. ZCOUNT 的高效性

- **原理：**
  - ZSet 内部使用 **跳表（skiplist）+ 字典（dict）** 结构。
  - 字典用于快速查找 `member` 的 score，跳表用于按 score 排序和范围查询。
  - 跳表节点维护了每个元素的 **层级索引** 和 **跨度信息**（rank），可直接定位任意元素的排名。
- **性能优化：**

- `ZCOUNT key min max` 并非遍历所有元素，而是通过跳表快速找到 `min` 和 `max` 的位置，再计算排名差。
- 举例：

```
代码块
1   min 元素排名 = 100
2   max 元素排名 = 200
3   元素数量 = 200 - 100 + 1 = 101
```

- 时间复杂度仅 **O(logN)**，远快于直观的遍历统计。

4. ZPOPMAX / ZPOPMIN 的底层逻辑

- **原理**：
  - 跳表头节点指向最小元素，尾节点指向最大元素。
  - 获取极值：直接访问头/尾节点 → 时间复杂度 O(1)
  - 删除极值：
    - 跳表需要调整索引结构
    - 每删除一个元素 → O(logN)
    - 删除 `count` 个元素 → O(logN * count)
- **排序规则**：
  - 分数相同的元素按字典序排序（字符串二进制比较）
  - 确保删除顺序确定性，避免极值命令返回结果不稳定

5. 命令演进与版本兼容

- **ZRANGEBYSCORE**：
  - Redis 6.2.0+ 将其整合到 `ZRANGE key start stop BYSCORE [LIMIT]`
  - 统一接口，提高 API 一致性
  - 兼容旧版本，旧命令仍可使用，避免业务代码重构
- **设计思想**：
  - 保持旧命令兼容性
  - 渐进式优化 → 不破坏现有功能的情况下增加新特性

---

6. 命令行为注意点

- **ZADD 返回值：**
  - 默认返回新增元素数量。
  - 使用 `CH` 参数，返回新增 + 更新的元素总数。
- **分数更新对顺序的影响：**
  - 修改 score 后，ZSet 会自动调整元素位置以维持有序性。
  - 调整复杂度 O(logN)（跳表插入和更新）。
- **相同 score 的排序：**
  - 多个元素 score 相同，按 member 的字典序排序。
  - 保证排行榜、分页查询结果稳定。

示例：

代码块

```
ZADD leaderboard NX 98 "David"
ZADD leaderboard XX GT 97.5 "Alice"
ZRANGE leaderboard 0 -1 WITHSCORES
```

三、ZSet 集合运算命令（Redis 6.2+）

1. 基础运算

命令	功能	特点
ZINTER key [key ...] / ZINTERSTORE destination key [key ...]	多 ZSet 交集，默认分数为各集合分数和，可用 AGGREGATE 设置 MIN/MAX	支持存储结果，避免重复计算
ZUNION key [key ...] / ZUNIONSTORE destination key [key ...]	多 ZSet 并集，分数规则同上	适合合并多个排行榜或用户标签
ZDIFF key [key ...] / ZDIFFSTORE destination key [key ...]	多 ZSet 差集，仅保留第一个集合的独有元素	可用于过滤特定集合中的元素

- **注意：**

- Redis 6.2+ 才原生支持
- 早期版本需客户端实现交集/并集逻辑
- 大集合运算成本高 → 建议 `*STORE` 保存结果复用

四、ZSet 核心集合运算命令与用法

ZSet 支持对多个有序集合执行交集、并集和差集运算，并可将结果直接存储到目标集合中，从而避免重复计算，提高性能。

1. `ZINTERSTORE` 交集命令

- **作用**：计算多个 ZSet 的交集，并将结果存储到目标集合。
- **典型用法**：

```
代码块
1   ZINTERSTORE destination 3 key1 key2 key3 WEIGHTS 1 2 1 AGGREGATE SUM
```

  - `destination`：存储运算结果的目标 ZSet
  - `numkeys`：参与运算的源集合数量
  - `WEIGHTS`：为每个源集合设置权重
  - `AGGREGATE`：
    - `SUM`：默认，交集元素的分数取加和
    - `MIN`：取所有集合中最小分数
    - `MAX`：取所有集合中最大分数
- **业务示例**：
  - 用户标签交集：找同时喜欢 A、B 两种兴趣的用户
  - 多维度积分：计算用户在不同活动中的综合得分

---

2. `ZUNIONSTORE` 并集命令

- **作用**：计算多个 ZSet 的并集，并将结果存储到目标集合。
- **典型用法**：

```
代码块
```

```
ZUNIONSTORE destination 2 keyA keyB WEIGHTS 0.7 0.3 AGGREGATE MAX
```

- 可以用不同权重调整各源集合的影响力
- `AGGREGATE` 决定并集元素的最终分数计算方式

- **业务场景**：
  - 综合排行榜：按活跃度、消费能力、内容贡献计算综合排名
  - 用户行为评分：将不同行为指标加权后生成统一得分

---

3. 时间复杂度分析与优化思路

- **复杂度公式**：

代码块

```
O(N*M) + O(K*logK)
```

- `N`：参与运算的最小集合元素数
- `M`：参与运算的集合数量
- `K`：结果集合的元素数

- **原理**：
  - Redis 遍历最小集合元素，检查其他集合是否包含该元素（交集）或累加分数（并集）
  - 最小集合驱动 → 降低不必要的遍历
- **优化思路**：
  - 尽量让最小集合驱动运算
  - 对结果集合可使用 `STORE` 保存，避免重复计算
- **工程实践**：
  - 对大集合或高频运算，推荐预计算或按时间/维度分片计算，减少实时压力

五、ZSet 底层编码与内存优化

ZSet 底层有两种编码方式，Redis 会根据集合大小和元素大小自动切换，以兼顾 **性能与内存**。

1. 编码切换规则

编码类型	条件	优势	劣势
ziplist（压缩列表）	元素数量 ⩽ zset-max-ziplist-entries（默认 128），单元素大小 ⩽ zset-max-ziplist-value（默认 64 字节）	内存紧凑、访问开销小	大数据量插入/删除性能低
skiplist + dict（跳表+字典）	超过 ziplist 阈值	支持高效插入、删除、范围查询	内存占用比压缩列表高

- **跳表 + 字典组合**：
  - 字典（dict）：O(1) 查询元素分数
  - 跳表（skiplist）：O(logN) 插入/删除/范围查询
  - 两者结合，既保证有序性，又保证快速单元素访问

---

2. 内存占用估算

- 假设游戏排行榜场景：
  - 每个元素包含：
    - `userid`（4 字节整数）
    - `score`（8 字节浮点数）
    - 元数据约 12 字节/元素
- 对 1200 万用户：

```
代码块
1   内存 ≈ 1200万 × 12字节 ≈ 144MB
```

- **结论**：
  - 适合实时排行榜，百万级用户级别数据可控
  - Redis 会自动切换编码以优化内存和性能

六、底层实现与性能特征

1. 底层数据结构

ZSet 使用 **跳表（skiplist）+ 字典（dict）** 的组合实现：

1. **字典（dict）**
   - key: member
   - value: score
   - 用于 **快速 O(1) 查找** 单个元素的 score。
2. **跳表（skiplist）**
   - 维护 **有序序列**，支持按 score 排序。
   - 插入、删除、范围查询时间复杂度 O(logN)。
   - 分层索引设计，快速定位元素范围。

组合优势：

- O(1) 单元素查找 + O(logN) 排序插入
- 兼顾性能和有序性
- 适合排行榜、优先队列、时间序列等高并发场景

---

2. 性能分析

操作	时间复杂度	说明
插入/删除/更新	O(logN)	依赖跳表结构，保持有序
范围查询 (ZRANGE, ZRANGEBYSCORE)	O(logN + K)	K 为返回元素数量，适合分页查询
单元素查找 (ZSCORE)	O(1)	直接通过 dict 查询
元素总数 (ZCARD)	O(1)	元数据直接维护

优化策略：

- 分页获取范围元素，避免全量查询阻塞服务端。
- 对频繁更新排行榜，分数变化较小的场景，使用 `GT` / `LT` 参数减少不必要的调整。
- 对小量元素的 ZSet，Redis 会使用 `ziplist` 编码节省内存。

七、典型业务场景

1. 排行榜系统

- 游戏段位、商品销量、用户贡献值等。
- 使用 `ZADD` 更新分数， `ZREVRANGE` 获取前 N 名， `ZRANK` / `ZREVRANK` 获取用户排名。

示例:

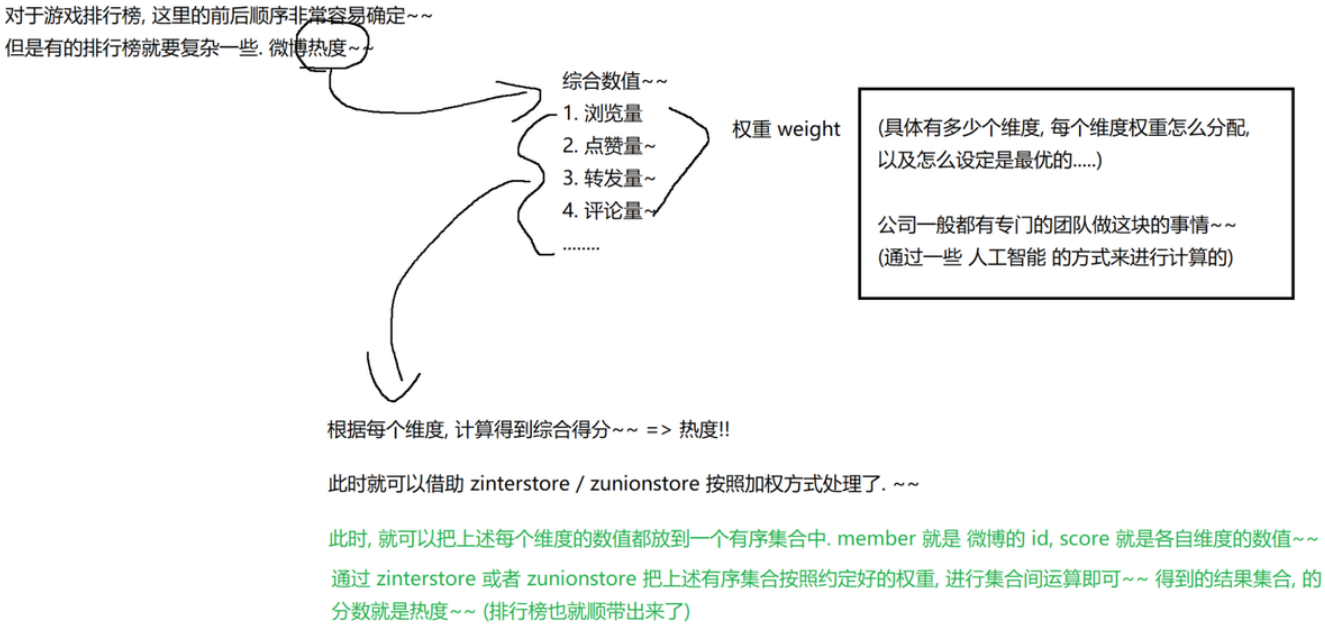
代码块

```
ZADD game:rank 97.8 "关羽"
ZADD game:rank 96.5 "张飞"
ZADD game:rank 99.1 "刘备"

ZREVRANGE game:rank 0 4 WITHSCORES
# 输出: 刘备 99.1, 关羽 97.8, 张飞 96.5

ZRANK game:rank "关羽"
# 输出: 1 (按升序排名)
ZREVRANK game:rank "关羽"
# 输出: 1 (按降序排名)
```

使用 zset 来完成上述操作, 就非常简单~~

比如游戏天梯排行,
只需要把玩家信息和对应的分数给放到有序集合中即可.
自动就形成了一个排行榜~~
随时可以按照 排行 (下标), 按照分数 进行范围查询~~

随着分数发生改变, 也可以比较方便的, zincrby 修改分数. 排行顺序也能自动调整 (logN)



- **场景价值**：
  - 支持前 N 名展示
  - 快速获取用户排名
  - 支持分页和定期更新

---

2. 带权重的任务队列

- **任务权重** → score
- 高分数任务优先处理，低分任务后处理
- 使用 `ZADD` 加入队列， `ZREVRANGE` 或 `ZRANGEBYSCORE` 获取优先级任务

示例：

代码块

```
ZADD task_queue 10 "task1"
ZADD task_queue 20 "task2"
ZREVRANGE task_queue 0 0
# 输出: "task2" (优先级最高)
```

- 应用场景：
  - 异步订单处理
  - 紧急告警推送
  - 任务调度

---

3. 时间序列数据存储

- score = 时间戳
- member = 业务数据（事件、日志、监控指标）
- 支持按时间范围查询

示例：

```
代码块
1   ZADD logs:20260126 1737888000 "user:1 login"
2   ZADD logs:20260126 1737888600 "user:2 login"
3   ZRANGEBYSCORE logs:20260126 1737888000 1737888600
4   # 输出两条登录记录
```

- 适合日志分析、监控告警、统计事件等

4. 实时排行榜

- `ZINCRBY` 更新分数
- `ZREVRANGE` 获取前 N 名
- `ZREVRANK` 查询用户排名
- 示例：

```
代码块
1   ZINCRBY leaderboard 10 "user:1"
2   ZREVRANGE leaderboard 0 9 WITHSCORES
```

5. 用户积分系统

- 累计积分：`ZINCRBY`
- 查询积分：`ZSCORE`
- 清理低分/过期积分：`ZREMRANGEBYSCORE`
- 示例：

```
代码块
1   ZREMRANGEBYSCORE leaderboard -inf 50
```

6. 多维度数据统计

- 使用 `ZINTERSTORE` 计算用户标签交集
- 可实现精准分层/推荐
- 示例：

```
代码块
1   ZINTERSTORE active_sports_users 2 age_25_30 sports_fans
```

八、最佳实践与注意事项

1. 区间查询边界

- 明确闭区间 `[min,max]` 与开区间 `(min,max)`，避免统计/分页错误
- 示例：

```
代码块
1   ZCOUNT key 10 20   # 包含 10 和 20
2   ZCOUNT key (10 20  # 不包含 10，包含 20
```

2. 大集合分页

- 避免一次性查询大量元素：

```
代码块
1   ZRANGE key 0 999
```

  - 对大集合可能阻塞 Redis

- 使用 `LIMIT offset count` 分页获取

```
代码块
1   ZRANGEBYSCORE key 0 100 BYSCORE LIMIT 0 100
```

3. 极值命令适用场景

- `ZPOPMAX` → 高优先级任务队列

- `ZPOPMIN` → 低分元素清理、过期策略

4. **版本兼容性**
   - Redis 6.2+ 优先使用：

```
代码块
1   ZRANGE key 0 -1 BYSCORE
```

   - 保持 API 一致性，减少旧命令使用
5. **结合业务场景优化**
   - 排行榜系统 → 分页 + 缓存
   - 任务队列 → score 表示优先级 → 极值命令消费
   - 时间序列数据 → score 表示时间戳 → 范围查询 + 分页

✅ 总结

- **ZSet 核心优势**：
  a. 唯一元素 + 分数排序
  b. 支持排行榜、优先队列、时间序列
  c. 跳表+字典结构保证 O(logN) 插入与范围查询 + O(1) 单元素查询
- **典型场景**：
  - 游戏/商品排行榜
  - 带权重任务队列
  - 时间序列事件存储
- **优化策略**：
  - 避免全量查询，大集合分页
  - 使用参数优化命令行为（NX、XX、GT、LT、CH）
  - 分数精度放大为整数，减少浮点误差
  - 内存优化，理解 ziplist 与跳表切换