# **二、系统设计**

**1. 可靠性需求：**

1.1 数据持久化：

* Redis + MySQL：结合 Redis和 MySQL，通过异步队列同步数据。Redis 用于实时读写，MySQL 作为最终一致性的备份‌。
* 多副本与分片：
  + Redis 使用主从复制 + Sentinel 哨兵模式实现高可用。
  + MySQL 采用主从复制 + 分库分表策略，按玩家 ID 哈希分片，避免单点故障。

1.2 事务与一致性：

* MySQL事务：排行榜中的核心数据（如排名与玩家信息等）使用MySQL事务的特性来保证安全。
* Redis 最终一致性：
  + 通过 AOF 日志 + RDB 快照组合持久化策略。
  + 采用 Write-Ahead Log（WAL） 确保缓存操作可追溯。

1.3 监控与数据容灾：

* 多活架构：多服务器部署 Redis Cluster 和 MySQL 集群，支持区域化容灾。
* 定期备份：MySQL 每日全量备份 + 增量备份，MongoDB 用于存储操作日志和审计数据。
* 健康检查：通过 Prometheus + Grafana 监控数据库负载和同步延迟，触发自动告警。

**2. 性能要求**

2.1 读写分离与分层存储：

* ‌实时读写层‌：
  + Redis 使用 ‌Sorted Set‌ 存储玩家实时排名，支持 O(1) 复杂度更新和查询‌。
  + 分片策略：按排行榜类型拆分 Redis Key，降低单节点压力‌。
* ‌持久化层‌：
  + MySQL存储玩家基础数据（playerId等），通过覆盖索引‌优化查询效率‌。

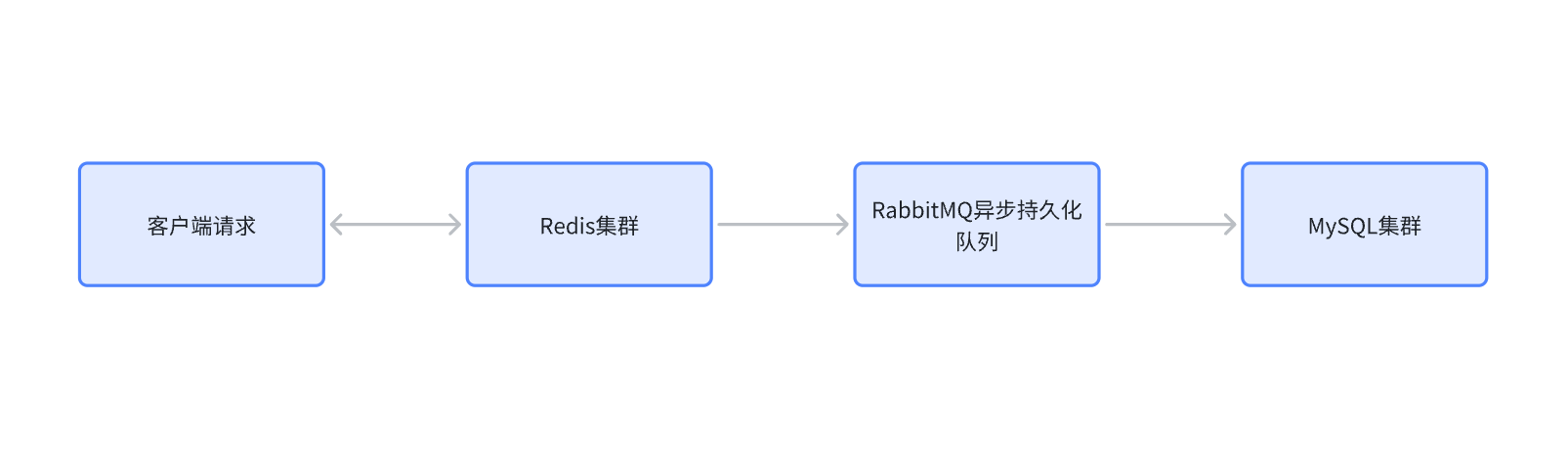
2.2 高并发优化

* ‌冷热分离‌：
  + 活跃玩家数据保留在 Redis，非活跃玩家归档到MySQL。
  + MySQL 使用内存表（Memory Engine）缓存近期排名变化‌。
* 异步队列‌：通过 RabbitMQ 异步处理非实时操作（如数据统计、日志记录）。

2.3 扩展性与负载均衡

* 水平扩展‌：
  + Redis Cluster 自动分片，支持百万级玩家 ZRANGE/ZREVRANGE 操作‌。
  + MySQL 通过 ProxySQL 或MyCat实现读写分离。

数据流程图如下所示：



# **三、游戏需求更改**

大概思路如下：

1. 数据结构调整：

TreeSet<Integer> uniqueScores：存储所有不同分数（降序），用于快速确定分数排名。

Map<Integer, Integer> scoreCounts：记录每个分数的玩家数量。

Map<Integer, Integer> scoreToRank：缓存每个分数对应的密集排名，避免重复计算。

1. 关键操作调整：

更新分数时：维护 uniqueScores 和 scoreCounts，并刷新 scoreToRank 缓存。

查询排名时：直接通过 scoreToRank 获取缓存的密集排名。