Valkey权威手册：从安装到优化

导言：社区驱动数据存储的崛起

本手册旨在为技术从业者提供一份关于Valkey的详尽、权威的指南，内容涵盖从其概念基础到高级运维实践的方方面面。Valkey的诞生不仅是一次技术上的分叉，更代表了一场向社区治理、开放许可的关键基础设施的运动。

Valkey的起源

Valkey的起源可追溯至2024年3月的一个关键事件：Redis Inc. 宣布将其开源许可证从宽容的BSD三句半条款（BSD 3-Clause）更改为更具限制性的源可用许可证（RSAL/SSPL）[1]。这一变化引发了开源社区和行业的广泛关注，因为它对数以万计依赖Redis开源版本的用户和云服务提供商构成了潜在的供应商锁定和许可证不确定性风险[3]。作为回应，一群核心的Redis开源社区维护者、贡献者以及包括AWS、Google Cloud、Oracle、Ericsson和Snap Inc.在内的行业领导者迅速联合起来，在Linux基金会的支持下，创建了Valkey项目[4]。Valkey是Redis OSS 7.2.4版本的直接分支，这是Redis最后一个完全开源的版本，旨在延续其真正的开源精神.[1]

这一行动的意义超越了代码本身。它展示了开源生态系统的一种“免疫反应”：当一个被广泛采用的关键基础设施项目面临被专有控制的风险时，行业内的竞争者们选择搁置分歧，在一个中立的基金会下进行合作，以保护该项目的开放性。这种协作模式体现了一种成熟的共识，即此类基础软件的巨大价值根植于其开放、社区驱动的治理模式，这种模式能比单一供应商模型建立更广泛的信任和生态。Valkey的快速成型和社区的蓬勃发展，为未来类似情况提供了一个强有力的范例和可行的解决方案[8]。

对开源的承诺

Valkey项目的核心是其对开源原则的坚定承诺。项目由中立的Linux基金会托管，这一结构性保

障确保了Valkey将永久保持其宽容的BSD 3-Clause许可证，不会因单一公司的商业决策而发生改变，实现了“永远开源”的承诺[6]。这种治理模式消除了用户的后顾之忧，为企业在关键业务上采用Valkey提供了坚实的信心基础。

Valkey在现代数据栈中的角色

Valkey继承并扩展了其前身的多功能性，在现代数据架构中扮演着不可或缺的角色。它不仅仅是一个缓存系统，更是一个高性能、多用途的数据处理工具。其核心能力包括作为内存键值数据库、消息队列、会话存储，甚至在某些场景下可作为主数据库使用[9]。这种灵活性使其能够广泛应用于电子商务、实时分析、机器学习、富媒体流和地理空间服务等多种场景[10]。Valkey社区正是在这个坚实的基础上，不断推动创新，以满足不断演进的技术需求。

第一节：基础概念与架构

要精通Valkey，首先必须理解其源自Redis并经过验证的核心架构和基本概念。这些构成了后续所有高级功能和优化策略的理论基础。

内存键值范式

Valkey的核心设计理念是“内存优先”。所有数据都存储在主内存（RAM）中，这使得数据访问的延迟极低，通常可以达到亚毫秒级别，这也是Valkey卓越性能的根源[6]。其数据模型非常直观：键（key）是字符串，而值（value）则可以是多种复杂的数据结构。这种模型既简单又强大，能够灵活地应对各种数据建模需求。

丰富的数据结构

Valkey的强大之处不仅在于速度，更在于其对多种原生数据结构的支持。这些数据结构允许开发者在服务器端直接对数据进行高效操作，而无需在客户端进行复杂的数据处理。主要的数据结构包括[9]：

- **Strings (字符串)**: 最基本的数据类型，可以存储文本、序列化对象或二进制数据，最大可达512MB。
- **Hashes (哈希)**: 类似于关联数组，是存储对象字段的理想选择。
- **Lists (列表)**: 按插入顺序排序的字符串元素集合，非常适合实现队列或栈。
- **Sets (集合)**: 无序且唯一的字符串元素集合，支持高效的成员测试和集合运算。
- **Sorted Sets (有序集合)**: 类似于集合，但每个成员都关联一个分数(score)，成员根据分数排序，非常适合实现排行榜、范围查询等功能。
- **Bitmaps (位图)**: 通过字符串进行位操作，能以极高的空间效率存储大量的布尔信息。
- **HyperLogLogs**: 一种概率性数据结构，用于以极小的内存占用估算集合的基数(唯一元素数量)。
- **Streams (流)**: 一种仅支持追加的日志数据结构，是构建消息队列和事件溯源系统的强大工具。
- **Geospatial Indexes (地理空间索引)**: 用于存储和查询地理位置坐标。

核心架构组件

Valkey的架构设计精妙地平衡了性能、简单性和可扩展性。

- 事件循环 **(Event Loop)**: Valkey的核心逻辑处理，如命令执行，传统上在一个单线程的事件循环中完成。这种设计避免了多线程编程中复杂的锁机制和竞态条件，简化了数据一致性的保证，是其高性能和原子性操作的基础[4]。这一设计将在后续章节中与Valkey引入的多线程I/O进行对比。
- 持久化模型 **(Persistence Models)**: 为了防止因服务器重启或故障导致内存数据丢失，Valkey提供了两种主要的持久化机制：RDB(快照)和AOF(仅追加文件)。RDB在指定时间间隔内生成数据集的时间点快照，而AOF则记录每一个写操作命令。这两种机制各有优劣，将在第六节中详细探讨[11]。
- 主从复制 **(Replication)**: Valkey支持主从(Primary-Replica)复制架构。数据从主节点异步复制到一个或多个从节点，用于实现数据冗余、故障转移和读扩展[4]。

可扩展性与可编程性

Valkey提供了强大的服务器端编程能力，允许开发者将复杂的业务逻辑移至数据层执行，从而减少网络开销并保证操作的原子性。

- **Lua**脚本: Valkey内置了Lua脚本引擎，允许用户执行原子性的、服务器端的脚本。通过EVAL等命令，可以将多个操作打包成一个单元，在服务器上一次性执行，避免了多次网络往返的延迟[4]。

- 模块系统 **(Module System)**: 模块API是Valkey最强大的扩展机制之一。它允许开发者使用C、C++或Rust等高性能语言编写动态库，来创建全新的数据类型、命令和功能，从而将Valkey扩展到其核心功能之外[8]。Valkey社区将模块系统视为创新的关键途径，许多新功能（如向量搜索）都是通过模块实现的。

Valkey的架构优势在于其简洁、高效的核心（单线程事件循环）与灵活、强大的扩展机制（数据结构、Lua脚本和模块）的有机结合。这种设计哲学使得Valkey在处理简单任务时能保持极高的性能，同时又能通过模块化扩展来适应复杂和新兴的应用场景。核心处理逻辑的单线程模型保证了单个命令的原子性，并消除了对数据结构加锁的需求，这是一个经过长期验证的高性能模型[4]。然而，为了克服单线程的局限性，Valkey并未使核心复杂化，而是将复杂性推向了边缘：通过丰富的数据结构处理复杂的数据建模，通过Lua脚本处理复杂的事务逻辑，并通过模块系统引入全新的功能[9]。这种分层设计确保了核心的稳定与高速，同时允许创新在模块生态系统中快速发生，而不会影响服务器本身的稳定性。社区对JSON、搜索等模块的重点投入正是这一理念的直接体现[8]。

第二节：Valkey的优势：超越Redis 7.2的创新

本节将深入探讨Valkey自其从Redis 7.2分叉以来所引入的关键新特性和架构改进。这些创新不仅提升了性能和效率，也确立了Valkey作为社区驱动项目的独特价值主张。

架构演进：增强的I/O多线程

Valkey最显著的架构变革之一是在8.0版本中引入的增强型I/O多线程模型[15]。虽然核心的命令执行逻辑（对数据结构的操作）仍然保持单线程以保证原子性和简单性，但Valkey重新设计了其线程模型，将网络I/O操作（从套接字读取客户端命令和向套接字写入响应）卸载到一组专用的工作线程中[4]。

这一改进解决了传统单线程模型在现代多核处理器上的I/O瓶颈问题。在旧模型中，所有的网络通信和命令处理都由同一个线程完成，当并发连接数非常高时，网络I/O会占用大量CPU时间，从而限制了服务器的整体吞吐量。通过将I/O操作并行化，Valkey能够更充分地利用多核CPU资源，显著提升在高并发负载下的性能。社区和行业测试报告显示，这一改进使得Valkey的吞吐量比早期版本提高了约三倍，单节点能够处理超过每秒119万次请求（RPS）[4]。

内存效率的突破

Valkey 8.1版本引入了一项重大的内存效率优化：全新的哈希表实现[17]。这个新设计受到了Google的Swiss Table等现代哈希表实现技术的启发，旨在减少每个键值对的内存开销[16]。

旧的哈希表实现依赖于大量的指针，在64位系统中，每个指针占用8个字节，导致了显著的内存开销和因指针追逐（pointer chasing）而产生的CPU缓存未命中。新的哈希表通过将键值数据更紧凑地存储在连续的内存块（64字节缓存行）中，大幅减少了指针的使用。这一改变为每个键值对节省了20到30字节的内存，对于常见的键值模式，内存效率提升超过20%[16]。对于大规模、内存受限的缓存工作负载，这种优化直接转化为显著的成本节约，因为它允许在相同的硬件上存储更多的数据。

性能深度优化

除了宏观的架构改进，Valkey社区还在多个微观层面进行了性能优化。

- 迭代器预取 **(Iterator Prefetching)**: 在Valkey 8.1中，对键空间的迭代操作（例如在主从全量同步或执行KEYS命令时）的速度得到了极大提升。通过采用内存预取技术，当服务器处理当前数据桶时，下一个数据桶及其内容已经被提前加载到CPU缓存中。这使得迭代器速度比之前快了3.5倍，显著缩短了新副本连接时的数据同步时间[17]。
- 复制与**TLS**改进: 延续I/O多线程的思路，Valkey 8.1将TLS连接协商和主从复制流的读写操作也卸载到了I/O线程池。这使得TLS新连接的接受速率提高了约300%，并进一步提升了主副本在启用TLS时的复制效率[17]。
- **SIMD**指令优化: 为了压榨现代CPU的性能，Valkey 8.1开始利用SIMD（单指令多数据流）指令集（如x86平台上的AVX2）。这使得某些计算密集型命令的性能得到指数级提升，例如，对HyperLogLog数据结构进行操作的PFMERGE和PFCOUNT命令速度提高了12倍，而BITCOUNT命令的速度提升高达514%[17]。

增强的可观察性与可靠性

运维的便捷性和系统的可诊断性是生产环境中的关键需求，Valkey在这方面也引入了重要改进。

- 分槽指标 **(Per-Slot Metrics)**: Valkey增加了对集群模式下每个哈希槽（slot）的指标监控。这是一个在Redis开源社区中被长期搁置的功能，它允许管理员轻松识别出集群中的“热点槽”（访问频率极高的槽），从而为集群的负载均衡和优化提供精确的数据支持[4]。
- **COMMANDLOG**: 作为SLOWLOG的增强版，COMMANDLOG是Valkey 8.1中新增的一个强大的可观察性工具。与SLOWLOG只记录慢命令不同，COMMANDLOG可以捕获对系统有影响

的命令的全部细节，包括完整的参数、响应大小以及执行该命令的客户端信息。这为诊断延迟尖峰和性能问题提供了前所未有的深度和上下文[16]。

- 改进的集群算法: Valkey对集群模式的内部算法进行了优化，提升了在节点故障、网络分区等情况下的可靠性，并缩短了故障转移（failover）所需的时间，使集群管理更加稳健[4]。

生态系统的分化：模块优先的策略

Valkey与Redis Inc.在功能扩展的哲学上展现出明显的分歧。Valkey坚定地走社区驱动的模块化路线，将高级功能作为可插拔的模块进行开发和维护。这 fostering 了一个开放和协作的生态系统。目前，社区已经开发出几个关键模块[8]：

- **valkey-search**: 提供先进的查询能力，包括全文搜索和向量相似性搜索。
- **valkey-json**: 为Valkey提供原生的JSON数据类型支持。
- **valkey-bloom**: 实现布隆过滤器等概率性数据结构。

为了简化这些模块的使用，社区还推出了valkey-bundle项目，这是一个将Valkey核心与这些流行模块打包在一起的容器化解决方案，为开发者提供了一站式的部署体验[14]。

相比之下，Redis Inc.选择将JSON、时间序列、向量搜索等功能直接集成到其专有的Redis核心产品中，这些功能在Valkey或Redis的开源版本中是不可用的[4]。

Valkey在分叉后的发展策略，展现出一种清晰的、以提升核心引擎性能和效率为导向的工程哲学。这种策略的成果直接体现在对现代硬件物理限制的突破上，例如通过多线程I/O利用多核CPU，通过优化数据结构减少内存占用。这些都是深入底层的架构性改进，能够为所有用户带来普适性的性能提升和成本降低[4]。

与之相对，Redis Inc.的发展重心似乎更偏向于在其商业产品中增加应用层的功能，如原生的JSON支持、时间序列数据结构和图形化管理界面[19]。这些功能无疑具有很高的应用价值，但它们主要服务于特定的使用场景，代表了一种向“上层应用平台”演进的趋势。

这背后反映了两种不同的发展理念：Valkey致力于优化一个强大、高效的“引擎”，赋能社区在此基础上构建多样化的应用；而Redis Inc.则在构建一个功能集成度更高但相对封闭的“平台”。对于追求开放性、极致性能和成本效益的用户而言，Valkey的“引擎”哲学可能更具吸引力。

表1：**Valkey**与**Redis (>7.2)** 特性与理念对比

特性/方面	Valkey	Redis (>7.2)	分析与启示
许可证	BSD 3-Clause [6]	RSAL/SSPL (商业版), AGPLv3 (开源版) [6]	Valkey提供完全的开源自由，无商业限制；Redis则采用双重许可，限制了商业用途。
治理模式	Linux基金会，社区驱动 [5]	Redis Inc. 公司控制 [18]	Valkey的治理模式保证了项目的长期中立性和开放性，避免了单一公司的商业利益驱动。
线程模型	增强的I/O多线程 [4]	主要为单线程I/O [4]	Valkey能更好地利用多核CPU，在高并发下提供更高的吞吐量。
内存效率	新的哈希表实现，内存开销降低20%+ [16]	传统哈希表实现	Valkey在内存使用上更经济，可显著降低大规模部署的硬件成本。
高级功能	社区驱动的模块 (valkey-bundle) [8]	原生集成的功能 (JSON, Search, Vector) [4]	Valkey保持核心精简，通过开放生态扩展功能；Redis则提供更集成的“开箱即用”体验。
生态系统	快速增长，完全开放 [3]	成熟、广泛，但受商业策略影响 [19]	Valkey的生态系统充满活力和创新潜力；Redis的生态系统更成熟但未来发展方向由公司主导。
兼容性	Redis OSS <= 7.2.4 的直接替代品 [1]	API逐渐分化	Valkey为现有Redis用户提供了无缝的迁移路径，而新版Redis可能需要代码

			修改。

第三节：安装与初始配置

本节将提供在不同环境中安装和运行Valkey的实用分步指南，从理论转向实际操作。

先决条件

在开始安装之前，请确保系统已具备必要的构建工具和依赖项。

- 构建工具: make, gcc, tcl (用于运行测试套件) [7]。
- 可选依赖:
  - **TLS**支持: 需要OpenSSL开发库 (如Debian/Ubuntu上的libssl-dev) [7]。
  - **systemd**支持: 需要systemd开发库 (如Debian/Ubuntu上的libsystemd-dev) [7]。

方法一：在**Linux**虚拟机上从源码编译

从源码编译提供了最大的灵活性，可以启用或禁用特定功能。

1. 获取源码: 从官方GitHub仓库克隆最新的Valkey源码。
   Bash
   ```
   git clone https://github.com/valkey-io/valkey.git
   cd valkey
   ```

2. 编译: 执行make命令进行编译。默认情况下，在Linux上会使用jemalloc内存分配器以获得更好的性能[7]。
   Bash
   ```
   make
   ```

   若要启用特定功能，可以使用构建标志，例如启用TLS支持：
   Bash
   ```
   make BUILD_TLS=yes
   ```

3. 测试 **(推荐)**: 编译完成后，运行测试套件以确保二进制文件的正确性和稳定性。
   Bash
   make test

4. 安装: 将编译好的二进制文件安装到系统中，默认路径为/usr/local/bin。
   Bash
   sudo make install

5. 生产环境配置: Valkey源码的utils目录下提供了一个install_server.sh脚本，可以帮助您快速创建一个生产就绪的服务，包括创建专用用户、配置文件目录 (/etc/valkey)、数据目录 (/var/lib/valkey) 以及systemd或init.d服务脚本[7]。
   Bash
   sudo utils/install_server.sh

方法二：使用系统包管理器

对于大多数主流Linux发行版，使用包管理器是安装Valkey最快捷、最简单的方式。

- **Debian/Ubuntu**: Valkey已进入官方软件源。
  Bash
  sudo apt update
  sudo apt install valkey

  为了方便习惯了Redis命令的用户，可以安装兼容包，它会创建redis-cli等命令的符号链接指向对应的Valkey工具[22]。
  Bash
  sudo apt install valkey-redis-compat

- **RHEL/CentOS/Fedora**:
  Bash
  # 对于Fedora或启用了EPEL/Remi源的RHEL/CentOS
  sudo dnf install valkey

  同样，也提供了兼容包valkey-compat-redis[22]。

方法三：使用Docker进行部署

Docker提供了环境隔离、可移植和快速部署的优势，是现代应用部署的首选方式之一。

- 官方镜像: Valkey社区维护着官方Docker镜像valkey/valkey。此外，Canonical (ubuntu/valkey) 和Chainguard (cgr.dev/chainguard/valkey) 等组织也提供了经过安全加固的镜像[9]。
- 运行**Valkey**容器: 以下是一个典型的docker run命令示例，它完成了端口映射、配置文件挂载和数据持久化：

```
Bash
docker run -d \
 --name my-valkey-instance \
 -p 6379:6379 \
 -v /path/to/my/valkey.conf:/usr/local/etc/valkey/valkey.conf \
 -v /path/to/my/data:/data \
 valkey/valkey valkey-server /usr/local/etc/valkey/valkey.conf
```

- 使用**valkey-bundle**: 如果需要使用JSON、Search等模块，可以直接使用valkey/valkey-bundle镜像，它预装了这些常用模块，大大简化了部署过程[14]。

初始安全加固

Valkey的默认配置是为了方便开发和测试，绝对不能直接用于生产环境[22]。必须进行以下基础的安全加固：

1. 绑定网络接口: 在valkey.conf中，将bind指令设置为您信任的私有网络IP地址，例如bind 192.168.1.100 127.0.0.1。切勿使用0.0.0.0暴露在公网上。
2. 启用保护模式: 确保protected-mode设置为yes（默认值）。这会阻止未通过bind指令明确绑定的IP地址的客户端连接，除非设置了密码。
3. 设置密码**/ACL**:
   - 简单密码: 使用requirepass指令设置一个强密码。客户端连接时必须使用AUTH命令进行认证。
   - 访问控制列表 **(ACL)**: 这是更推荐的方式。ACL提供了更细粒度的权限控制，可以为不同用户设置不同的命令和键访问权限，是现代安全实践的最佳选择[11]。

Valkey社区在分发和工具链上所做的努力，清晰地表明了其致力于成为Redis OSS“直接替代品”的决心。一个成功的技术分支，不仅需要代码层面的兼容，更需要在运维层面实现无缝衔接。社区迅速为各大Linux发行版建立了官方软件包仓库，并推出了官方Docker镜像，这极大地降低了用户采纳的门槛[22]。更值得注意的是，valkey-redis-compat这类兼容包的推出，体现了社区对用户现有工作流和操作习惯的尊重。它通过创建符号链接，使得依赖redis-cli的自动化脚本和管理员的“肌肉记忆”得以保留[22]。这种在分发、容器化和工具链兼容性上的多方位投入，使得从Redis到

Valkey的转换，对许多用户而言，不再是一项高风险的迁移工程，而更像一次常规的软件升级。

第四节：构建高可用架构

对于任何生产系统，高可用性（High Availability, HA）都是非核心功能之外最重要的考量。Valkey提供了两种成熟的高可用部署模型：Valkey Sentinel（哨兵）和Valkey Cluster（集群）。选择哪种模型取决于应用的具体需求，主要是在数据集规模、写入吞吐量和运维复杂性之间进行权衡。

策略一：使用**Valkey Sentinel**实现主从高可用

Valkey Sentinel是针对单个数据集实现自动故障转移的官方解决方案。它适用于数据集可以完全容纳在单个节点内存中，且写入负载在该节点的处理能力范围内的场景。

架构

Sentinel系统由一组独立的Sentinel进程和一个主从复制（Primary-Replica）的Valkey数据节点集构成。Sentinel进程本身不存储业务数据，也不处理客户端的数据请求。它们的核心职责是[11]：

- 监控 **(Monitoring)**: Sentinel进程持续地向Valkey主节点和从节点发送心跳检测（PING命令），以检查它们是否正常工作。
- 通知 **(Notification)**: 当一个被监控的实例出现问题时，Sentinel可以通过API通知系统管理员或其他程序。
- 自动故障转移 **(Automatic Failover)**: 如果主节点被认定为下线，Sentinel会从其从节点中选举一个新的主节点，并指示其余的从节点转而复制新的主节点。
- 配置提供者 **(Configuration Provider)**: Sentinel还为客户端提供服务发现功能。客户端可以连接到任何一个Sentinel进程，询问当前特定服务（例如，“mymaster”）的主节点地址。当发生故障转移后，Sentinel会向客户端报告新的主节点地址。

设置与仲裁 **(Quorum)**

为了保证Sentinel系统自身的健壮性，避免其成为单点故障，生产环境必须部署一个由奇数个（至

少3个）Sentinel进程组成的集群。
配置一个Sentinel实例非常简单，只需在sentinel.conf文件中指定要监控的主节点信息：

```
sentinel monitor mymaster 192.168.1.1 6379 2
```

这行配置告诉Sentinel监控一个名为mymaster的主节点，其地址为192.168.1.1:6379。最后的数字2是仲裁数 **(quorum)**。

仲裁数是启动一次故障转移所需的最少Sentinel进程同意数。例如，在一个由5个Sentinel组成的集群中，如果仲-裁数设为3，那么必须至少有3个Sentinel进程都认为主节点下线（主观下线 sdown -> 客观下线 odown），故障转移流程才会被触发[30]。这个机制可以有效防止因单个Sentinel节点的网络问题而导致的误判和不必要的故障转移。

故障转移过程

当达到仲裁条件，确认主节点客观下线后，故障转移过程会自动启动[30]：

1. 领导者选举: 剩余的Sentinel进程会进行一次选举，选出一个领导者来负责执行故障转移。
2. 从节点选举: Sentinel领导者会根据一定的规则（如优先级、复制偏移量等）从健康的从节点中选出一个最合适的作为新的主节点。
3. 晋升与重配置: Sentinel向被选中的从节点发送REPLICAOF NO ONE命令，将其提升为新的主节点。然后，向其他从节点发送REPLICAOF命令，让它们开始复制新的主节点。
4. 通知客户端: 客户端通过订阅Sentinel发布的事件或定期查询，可以获知新的主节点地址，从而无缝地切换连接。

策略二：使用Valkey Cluster实现分片与可扩展性

当单个数据集的规模超过了单台服务器的内存容量，或者写入请求的频率超出了单个Valkey实例的处理极限时，就需要Valkey Cluster。Valkey Cluster通过数据分片（sharding）将数据分散到多个节点上，从而实现了水平扩展（horizontal scaling）和高可用性。

架构

Valkey Cluster是一个无中心节点的分布式系统，所有节点都通过一个内部的“集群总线”（cluster bus）进行通信，交换配置信息、进行故障检测和故障转移[31]。

- 哈希槽 **(Hash Slots)**: Valkey Cluster的核心分片机制。整个键空间被逻辑上划分为16384个哈希槽。每个主节点负责处理这些槽的一个子集[31]。当一个客户端需要操作某个键时，Valkey会计算该键的CRC16校验和，然后对16384取模，得到该键所属的哈希槽：slot = CRC16(key) % 16384。客户端（或代理）根据槽位映射信息，将请求直接发送到负责该槽的节点。
- 哈希标签 **(Hash Tags)**: 由于分片机制，涉及多个键的命令（如MSET或事务）默认情况下只能在所有键都属于同一个哈希槽时才能执行。为了解决这个问题，Valkey引入了哈希标签。如果一个键中包含被{}包裹的子字符串，那么只有这个子字符串会被用来计算哈希槽。例如，{user1000}:name和{user1000}:email这两个键会被分配到同一个槽，因为只有user1000部分参与哈希计算[31]。
- 主从复制与故障转移: 在集群模式下，每个主节点都可以配置一个或多个从节点。当某个主节点发生故障时，其从节点之一会自动被集群中的其他节点选举并提升为新的主节点，接管原来主节点负责的哈希槽，从而保证服务的可用性。

集群创建与管理

创建Valkey Cluster通常使用valkey-cli工具。一个最小的、功能完备的集群至少需要3个主节点。一个典型的生产环境设置是6个节点，包括3个主节点和3个从节点（每个主节点一个从节点）。

1. 准备节点: 启动6个Valkey实例，并在它们的配置文件中启用集群模式 (cluster-enabled yes)。
2. 创建集群: 使用valkey-cli的--cluster create命令，并指定--cluster-replicas 1来自动配置主从关系。
   Bash
   ```
   valkey-cli --cluster create 127.0.0.1:7000 127.0.0.1:7001 127.0.0.1:7002 \
   127.0.0.1:7003 127.0.0.1:7004 127.0.0.1:7005 --cluster-replicas 1
   ```
3. 扩展与重新分片: 当需要扩展集群时，可以添加新节点 (--cluster add-node)，然后使用--cluster reshard命令将一部分哈希槽从现有节点迁移到新节点上，以实现负载均衡[31]。

云托管的高可用方案

值得一提的是，主流云服务提供商如AWS ElastiCache、Google Cloud Memorystore和Oracle Cloud Infrastructure (OCI) Cache都提供了托管的Valkey服务。这些服务极大地简化了高可用架

构的部署和管理，它们通常提供一键式的多可用区（Multi-AZ）部署、自动故障转移、无缝扩展和监控告警等功能，让开发者可以专注于业务逻辑而非底层运维[2]。

Sentinel和Cluster的选择并非孰优孰劣，而是对不同规模和复杂度工作负载的适应性问题。Sentinel专为单个数据集的高可用性而优化，而Cluster则为超越单机内存或写入吞吐量极限的数据集而设计。Sentinel管理的是一个逻辑上统一的数据集，该数据集被完整地复制到主节点及其所有从节点。所有的写操作都指向单一的主节点。这种模型管理简单，但其性能和容量的上限受限于该主节点的物理资源[37]。相比之下，Cluster通过分片将数据集切分到多个主节点上。这使得写操作和总数据量可以通过增加更多节点来进行水平扩展[31]。然而，这种可扩展性也带来了更高的复杂性：开发者必须使用能够感知集群拓扑的客户端，处理跨槽操作的限制，并维护一个更复杂的部署架构[31]。因此，决策路径可以简化为：我的整个数据集和写负载能否被一台性能强大的服务器承载？如果答案是肯定的，Sentinel提供了一个更简单且稳健的高可用方案。如果答案是否定的，那么为了达到所需的规模，承担Cluster带来的运维开销就是必要的。

表2：高可用策略对比 **(Sentinel vs. Cluster)**

方面	Valkey Sentinel	Valkey Cluster
主要目标	高可用性 [30]	高可用性 + 水平扩展 [31]
数据模型	单一、统一的数据集 [37]	分片的数据集，分布在多个节点 [37]
可扩展性	通过从节点扩展读能力；写能力需垂直扩展（更强的服务器）	通过增加分片来水平扩展读和写的能力
运维复杂性	配置和管理相对简单；标准客户端即可工作	拓扑更复杂；需要使用集群感知的客户端
多键操作	支持对任意键的原子操作（如事务、Lua脚本）	仅支持所有键都位于同一哈希槽的操作（需使用哈希标签） [31]
理想用例	数据量能被单节点容纳的应用；需要简单故障转移的场	超大规模数据集；写密集型工作负载；需要线性扩展能

	景	力的应用

第五节：管理与主动监控

有效的管理和监控是从被动响应问题到主动预防问题的关键。Valkey提供了一套丰富且分层的可观察性工具，使管理员能够深入了解其实例的健康状况和性能表现。

INFO 命令：核心诊断工具

INFO命令是Valkey管理员最重要、最常用的工具，它以易于解析的格式返回关于服务器状态和性能的详尽统计信息[38]。通过指定不同的section参数，可以获取特定方面的信息，如memory、stats、replication等。

建立健康基线

判断一个Valkey实例是否“健康”，不仅仅是看它是否在运行，更要看其关键指标是否稳定和可预测。

- 基本连通性检查: 最简单的健康检查是PING命令。如果服务器正常响应PONG，则表示网络连接通畅，且服务器的事件循环没有被长时间阻塞[22]。
- **Sentinel**部署健康检查: 对于使用Sentinel的架构，应定期执行SENTINEL CKQUORUM <primary-name>命令。该命令会检查当前的Sentinel配置是否能够达到法定仲裁数以执行故障转移。这是确保HA系统真正可用的关键检查[30]。
- 应用级健康检查: 一些应用框架，如NestJS，提供了与Valkey集成的健康检查模块（例如nestjs-valkey-health），可以将Valkey的健康状态纳入整个应用的健康监控体系中[40]。

性能问题诊断

当监控指标出现异常（如延迟升高）时，Valkey提供了专门的工具进行深入诊断。

- **SLOWLOG**: SLOWLOG GET命令可以列出执行时间超过指定阈值的命令。这是定位性能瓶颈的“第一站”，能快速识别出应用中可能存在的低效O(N)操作（例如对一个巨大的集合执行SMEMBERS）[16]。
- **COMMANDLOG**: 作为SLOWLOG的增强，Valkey 8.1引入的COMMANDLOG功能可以记录更详细的命令信息，包括参数、响应大小和客户端详情，为复杂问题的根因分析提供了更丰富的上下文[16]。
- 延迟监控器 **(Latency Monitor)**: 这是一个更高级的诊断工具，通过LATENCY系列命令 (LATENCY LATEST, LATENCY HISTORY, LATENCY DOCTOR) 来使用。它能监控并记录Valkey内部不同事件（如command执行、fork系统调用、expire-cycle过期键清理等）的延迟尖峰。当性能问题不是由某个慢命令直接导致，而是由底层系统活动引起时，延迟监控器是找出根本原因的利器[41]。
- **MONITOR**: MONITOR命令会实时输出服务器正在处理的所有命令流，如同一个“消防水管”。这是一个纯粹的调试工具，会极大地影响服务器性能，严禁在生产环境中使用[42]。

常用管理任务

valkey-cli是执行日常管理任务的主要界面。例如：

- 检查详细内存使用情况：MEMORY STATS [43]。
- 列出当前连接的客户端：CLIENT LIST。
- 手动触发后台持久化：BGSAVE。

Valkey提供了一个层次分明的可观察性工具集。有效的管理实践意味着根据问题的性质，从高层级的概览指标（INFO）逐步深入到具体问题的诊断工具（SLOWLOG, LATENCY DOCTOR）。一个健康的实例，不仅是“运行中”的状态，更应该展现出稳定且可预测的性能指标。

监控的起点应聚焦于INFO命令所反映的几个核心信号：延迟（可从命令统计中推断）、流量（instantaneous_ops_per_sec）、错误（rejected_connections）和饱和度（cpu_utilization, used_memory）。这些指标共同构成了系统健康状况的宏观视图[38]。当这些高层级指标显示异常，例如延迟持续攀升时，管理员应转向下一层诊断工具。SLOWLOG能够直接指出是哪些命令执行缓慢，从而将问题范围缩小到特定的应用逻辑[16]。如果SLOWLOG的信息不足以解释问题，例如所有命令本身执行很快但整体响应变慢，那么LATENCY监控器就能发挥作用。它能揭示延迟是否源于命令执行本身，还是由后台的系统事件（如一次耗时较长的fork()调用）所引发[41]。这种分层诊断的方法论，使得问题排查过程高效且专注，避免了滥用如MONITOR这样对性能有严重影响的工具[42]。

表3：关键监控指标及其意义

指标 (Metric)	INFO 部分	意义	关注点
used_memory / used_memory_rss	Memory	Valkey自身分配的内存 vs 操作系统为其分配的物理内存。	mem_fragmentation_ratio（rss/used）过高（如 > 1.5）可能表示内存碎片化严重。
connected_clients	Clients	当前活跃的客户端连接数。	持续增长或突然飙升可能意味着应用程序存在连接泄漏。
instantaneous_ops_per_sec	Stats	服务器当前的命令处理吞吐量（QPS）。	突然下降可能表示服务器或客户端出现瓶颈。
keyspace_hits / keyspace_misses	Stats	键查找的成功次数与失败次数。	命中率（hits / (hits + misses)）持续低于 0.8，表明缓存效果不佳，可能需要增加内存或优化缓存策略[45]。
evicted_keys	Stats	因达到maxmemory限制而被驱逐的键的数量。	持续出现大量驱逐，明确表示内存不足，需要扩容[44]。
latest_fork_usec	Stats	最近一次fork()操作的耗时（微秒）。	该值过高（例如超过几百毫秒）会导致服务器的明显卡顿和延迟尖峰。
master_link_status	Replication	从节点与主节点之间的复制链接状态。	应该始终为up。如果变为down，表示复制中断，需要立即排查。

第六节：数据持久化、备份与恢复

Valkey作为内存数据库，其数据的持久化策略是确保数据安全性和服务可靠性的核心。本节将详细阐述Valkey的两种持久化机制——RDB和AOF，并指导如何根据业务需求选择和配置，以制定稳健的备份与恢复计划。

持久化的权衡

在选择持久化策略时，核心是在数据安全性（即可容忍的数据丢失程度）和性能之间做出权衡。通常，更强的数据持久性保证会带来更高的性能开销。

RDB快照 (RDB Snapshots)

RDB是Valkey的默认持久化方式。它在特定的时间点将内存中的整个数据集生成一个紧凑的二进制快照文件（默认为dump.rdb）。

工作机制

当满足触发条件时，Valkey会执行fork()系统调用，创建一个子进程。父进程继续处理客户端请求，而子进程则负责将内存中的数据写入一个临时的RDB文件。写入完成后，子进程用这个新文件原子性地替换旧的RDB文件。这个过程利用了操作系统的写时复制（Copy-on-Write）机制，对父进程的性能影响较小[13]。

配置

通过valkey.conf中的save指令来配置自动快照的触发条件。例如：

```
save 900 1      # 900秒内至少有1个key被修改
save 300 10     # 300秒内至少有10个key被修改
save 60 10000   # 60秒内至少有10000个key被修改
```

也可以通过BGSAVE命令手动触发一次后台快照。若要完全禁用RDB，可以将所有save指令注释掉，并添加save ""[13]。

优缺点

- 优点:
  - 文件紧凑: RDB文件是经过压缩的二进制格式，非常适合用于备份和灾难恢复[13]。
  - 恢复速度快: 加载单个RDB文件来恢复数据通常比重放AOF日志要快得多，尤其是在数据集很大的情况下[13]。
  - 性能影响小: 持久化工作由子进程完成，父进程基本不受影响[13]。
- 缺点:
  - 数据丢失风险: RDB是基于时间点的快照，如果Valkey在两次快照之间发生故障，那么这期间的所有数据变更都将丢失。数据丢失的窗口可能长达数分钟[13]。

AOF日志 (Append-Only File)

AOF通过记录服务器接收到的每一个写操作命令来持久化数据。当Valkey重启时，它会按顺序重新执行AOF文件中的所有命令，从而恢复数据状态。

工作机制

启用AOF后，所有写命令都会被追加到一个日志文件中。随着时间的推移，AOF文件会越来越大。为了解决这个问题，Valkey提供了**AOF重写（rewrite）**机制。当AOF文件达到一定大小时，Valkey会在后台启动一个子进程，根据当前内存中的数据状态，生成一个包含最少命令集的新AOF文件来替代旧文件，从而实现日志压缩[13]。Valkey 7.0之后采用了多部分AOF机制，将AOF文件拆分为基础文件和增量文件，由一个清单文件管理，这使得重写过程更加安全和高效[13]。

配置与持久性级别

在valkey.conf中设置appendonly yes来启用AOF。AOF的安全性关键在于appendfsync策略，它决定了数据写入AOF文件后多久被同步到磁盘：

- always: 每个写命令都立即fsync到磁盘。最安全，但性能最差。
- everysec: 每秒fsync一次。这是推荐的默认值，在性能和数据安全性之间取得了极佳的平衡，最多只会丢失1秒的数据[13]。
- no: 完全依赖操作系统的缓存刷新策略。性能最好，但数据丢失风险最高。

优缺点

- 优点：
  - 高持久性: 使用everysec策略时，数据丢失的风险极小[13]。
  - 日志可读性: AOF文件是协议文本格式，易于理解和修复。
  - 安全性: 即使文件末尾有损坏的命令，也可以使用valkey-check-aof工具轻松修复[13]。
- 缺点：
  - 文件体积: 通常情况下，AOF文件比相同数据集的RDB文件要大[13]。
  - 恢复速度: 数据恢复速度可能慢于RDB，因为它需要重放所有写命令[13]。

制定稳健的备份与恢复计划

策略选择

- 对于纯缓存场景: 可以完全禁用持久化。
- 对于需要高持久性的生产系统: 强烈建议同时启用**RDB**和**AOF**[12]。在这种配置下，AOF用于日常的数据恢复和保证高持久性，而RDB则用于定期备份、灾难恢复和快速迁移。当Valkey重启时，如果同时存在AOF和RDB文件，它会优先使用AOF文件来恢复数据，因为AOF的数据通常更完整[13]。

备份流程

1. 定期备份**RDB**文件: 设置一个cron job或其他调度任务，定期将dump.rdb文件复制到安全的备份位置（如另一台服务器或云存储）。
2. 安全备份**AOF**文件: 备份AOF文件目录时需要特别注意，因为AOF重写过程是动态的。安全的做法是先通过CONFIG SET auto-aof-rewrite-percentage 0临时禁用自动重写，确认没有正在进行的重写任务后，再复制整个AOF目录，完成后再重新启用自动重写[13]。

恢复流程

1. 标准恢复: 将备份的RDB或AOF文件（或目录）放回Valkey的数据目录，然后启动Valkey服务即可。
2. **AOF**文件损坏修复: 如果Valkey因AOF文件损坏而无法启动，可以使用valkey-check-aof --fix <filename>命令尝试修复[13]。

特定键的备份与恢复

对于只需要备份或迁移部分数据的场景，可以使用valkey-cli的--raw dump和restore命令组合，对单个键进行序列化转储和恢复，这为测试和精细化数据操作提供了便利[48]。

持久化策略的选择，本质上是一个关于可接受数据丢失风险的业务决策，而非纯粹的技术决策。Valkey的默认RDB设置优先考虑性能而非持久性，这对于那些将Valkey视为主要数据库而未进行适当配置的用户来说，可能是一个危险的陷阱。一个典型的默认配置（如save 900 1）意味着在发生故障时，系统可能会丢失长达15分钟的数据[13]。对于简单的缓存应用，这是可以接受的；但对于会话存储或记录系统而言，这绝对是不可容忍的。因此，在生产部署中，最关键的第一步就是审慎评估appendonly和appendfsync配置。如果应用无法承受分钟级别的数据丢失，就必须启用AOF。RDB + AOF的组合配置[13]是最安全的选择，它既提供了细粒度的高持久性保障，又保留了RDB带来的便捷的时间点备份能力。

表4：持久化策略对比 **(RDB vs. AOF vs. RDB+AOF)**

方面	仅 RDB	仅 AOF	RDB + AOF

数据丢失窗口	分钟级（可配置）	约1秒（使用everysec策略）	约1秒
性能影响	较低（fork开销）	中等（后台I/O）	中等
磁盘空间	紧凑	较大，但会自动重写压缩	较大
恢复速度	较快	较慢（需重放所有命令）	较慢（优先使用AOF）
备份友好性	极佳（单一紧凑文件）	较复杂（文件目录）	良好（可直接使用RDB文件进行备份）
推荐用例	缓存、非关键数据	高持久性要求的场景	关键业务生产系统

第七节：高级性能调优

要将Valkey的性能发挥到极致，需要进行全面的调优，涵盖内存管理、网络、CPU以及客户端行为等多个层面。本节将探讨超越基础配置的高级技术，帮助您最大化Valkey的吞吐量并降低延迟。

内存优化技术

内存是Valkey性能的核心，高效的内存管理至关重要。

- 利用数据结构内部编码：Valkey对小型的聚合数据类型（如Hashes, Lists, Sets）使用了特殊的内存优化编码。当这些结构中的元素数量和大小在一定阈值内时，它们会以一种比标准实现节省多达5到10倍内存的方式存储[49]。因此，一个重要的实践是：用多个小哈希（**Hash**）来代替大量的顶级键（**top-level keys**）。例如，将一个对象的所有属性存储在一个哈希中，而不是为每个属性创建一个单独的键。
- 设置**maxmemory**与驱逐策略：在生产环境中，必须设置maxmemory参数来限制Valkey的最大内存使用量，以防止其耗尽系统内存。当达到内存上限时，maxmemory-policy参数决定了Valkey的行为。常用的策略包括[49]：

    - allkeys-lru: 从所有键中移除最近最少使用的键(LRU)，适用于通用缓存。
    - volatile-lru: 仅从设置了过期时间的键中移除最近最少使用的。
    - allkeys-lfu: 从所有键中移除最不常用的键(LFU)，在Valkey 4.0后引入，对某些访问模式更有效。
    - noeviction: 不驱逐任何键，当内存满时，写操作会返回错误。适用于不希望数据丢失的场景。
- 主动内存碎片整理: 长时间运行的Valkey实例可能会产生内存碎片，导致used_memory_rss(操作系统报告的内存)远大于used_memory(Valkey报告的内存)。可以启用activedefrag yes来让Valkey在后台进行主动的内存碎片整理。这会消耗一定的CPU资源，但可以有效地降低内存碎片率，回收内存。

网络与CPU调优

在许多高吞吐量场景中，瓶颈往往出现在网络而非CPU。

- 网络带宽与延迟: 性能调优的第一步是确保客户端与服务器之间的网络具有高带宽和低延迟。在基准测试前，使用ping检查网络延迟是基本操作[50]。
- **CPU亲和性 (CPU Pinning)**: 在对延迟极度敏感的高性能环境中，可以通过taskset命令或Docker的--cpuset-cpus参数，将Valkey的主进程和I/O线程绑定到特定的CPU核心上。这可以减少操作系统调度器带来的上下文切换开销，并提高CPU缓存的命中率，从而降低尾部延迟(tail latencies)[51]。
- 管道 **(Pipelining)**: 这是最重要的客户端性能优化技术之一。客户端可以将多个命令一次性发送给服务器，而无需等待每个命令的响应，服务器处理完所有命令后再一次性返回所有结果。这极大地减少了因网络往返时间(RTT)造成的延迟累积，能将吞吐量提升一个数量级[11]。

客户端最佳实践

客户端的行为对整个系统的性能有决定性影响。

- 连接池 **(Connection Pooling)**: 应用程序不应为每个请求都创建和销毁TCP连接。这会带来巨大的性能开销。必须使用客户端库提供的连接池功能，复用已建立的连接，并限制并发连接的总数，以防止耗尽服务器的连接资源[52]。
- 超时与退避策略 **(Timeouts and Backoff)**: 客户端应配置合理的连接和读写超时时间。当连接失败或超时时，重试逻辑必须实现带抖动(jitter)的指数退避算法。这可以防止在服务器恢复或网络抖动时，所有客户端同时发起重连，从而避免"惊群效应"(thundering herd)压垮服务器[52]。

避免性能陷阱

错误的用法会轻易抵消掉所有硬件和配置上的优化。

- 警惕**O(N)**复杂度的命令: 在生产环境中，必须严格避免对大数据集使用时间复杂度为O(N)或更高的命令，因为它们会长时间阻塞单线程的Valkey服务器，导致所有其他客户端的请求被延迟。常见的"危险"命令包括:
  - KEYS *: 遍历整个键空间，是性能杀手之首。应使用SCAN命令进行安全的迭代。
  - HGETALL, SMEMBERS, LRANGE 0 -1 (on large collections): 一次性返回一个大集合的所有成员，可能导致巨大的网络流量和服务器内存分配压力[52]。
- 避免存储过大的键值: 在单个键中存储数兆字节(MB)甚至更大的值是一种反模式。这会给内存分配、网络传输和持久化带来巨大压力。正确的做法是将大对象拆分成多个较小的字段，存储在哈希中，然后按需获取[52]。

Valkey的性能调优是一个系统工程，涉及服务器、网络和客户端应用的协同优化。通常，最大的性能提升并非来自对服务器配置的精细调整，而是源于客户端的优化实践(如管道和连接池)以及明智的数据建模选择(如避免大键和O(N)命令)。Valkey核心操作的执行速度以微秒计[10]，而网络往返时间(RTT)则以毫秒计，两者相差几个数量级。因此，通过管道技术将多次网络往返的成本摊销到一批命令上，是提升高吞吐量工作负载效率的最有效手段[11]。同样，不当的数据模型，例如用GET获取一个10MB的大键，或用SMEMBERS拉取一个包含数万成员的集合，会长时间阻塞单线程的服务器，对所有其他客户端造成延迟尖峰[52]。因此，构建一个"高性能"的Valkey系统，其关键不在于调整内核参数，而在于设计与Valkey高效、"友好"的应用交互模式。

第八节:迁移与版本升级

本节为管理员提供两项关键运维任务的清晰、可操作的指南:从现有的Redis实例迁移到Valkey，以及在Valkey内部进行版本升级。

从**Redis**迁移到**Valkey**

由于Valkey是Redis OSS 7.2.4的直接分支，从兼容的Redis版本迁移过来通常是一个平滑的过程。

- 兼容性检查: 首先确认您的Redis实例版本为7.2.x或更早的开源版本。Valkey是这些版本的直

接替代品。需要确保您的应用没有使用Redis 7.4及以后版本引入的专有命令[53]。

- 方法一：离线迁移 **(RDB文件)**: 这是最简单直接的方法，适用于可以接受服务中断的场景。
  1. 在Redis实例上执行一次最终的BGSAVE，确保RDB文件是最新状态。
  2. 停止Redis服务。
  3. 将dump.rdb文件从Redis的数据目录复制到Valkey的数据目录。
  4. 启动Valkey服务。Valkey会自动检测并加载这个RDB文件，数据迁移完成[54]。
- 方法二：在线迁移 **(主从复制)**: 此方法可以最大限度地减少或避免服务停机，是生产环境迁移的首选。
  1. 部署一个新的、空的Valkey实例。
  2. 在该Valkey实例上执行REPLICAOF <redis_host> <redis_port>命令，将其配置为现有Redis主节点的从节点[56]。
  3. Valkey将开始从Redis进行全量数据同步，然后进入增量复制阶段。通过INFO replication命令监控复制状态，等待master_link_status变为up且复制延迟接近于零。
  4. 在应用层，将流量逐步或一次性地从Redis切换到Valkey实例。
  5. 确认所有流量都已指向Valkey后，在Valkey实例上执行REPLICAOF NO ONE命令，将其提升为独立的主节点，断开与Redis的复制关系。
- 方法三：增量迁移 **(MIGRATE命令)**: MIGRATE命令可以原子性地将一个或多个键从源实例迁移到目标实例。这对于需要分阶段、精细化控制迁移过程的复杂场景非常有用。该命令支持COPY（不删除源键）和REPLACE（覆盖目标键）等选项[54]。

Valkey版本升级

在Valkey内部进行版本升级是常规的维护任务，遵循正确的流程可以确保服务的连续性。

- 理解版本号: Valkey遵循主版本.次版本.补丁版本（Major.Minor.Patch）的语义化版本规范。
  - 补丁版本: 仅包含向后兼容的bug修复，升级非常安全。
  - 次版本: 增加向后兼容的新功能，升级通常是安全的。
  - 主版本: 可能包含不向后兼容的重大变更，升级前必须仔细阅读发布说明[57]。
- 安全的滚动升级流程: 对于主从或集群架构，推荐采用以下滚动升级流程，以实现零停机或最小停机升级。
  1. 先升级从节点 **(Replicas First)**: 逐个升级所有的从节点。Valkey的复制协议设计保证了从较旧版本的主节点到较新版本的从节点的复制是安全的。反之则不一定[57]。
  2. 执行受控的故障转移: 当所有从节点都升级到新版本后，执行一次手动的、受控的故障转移。在Sentinel架构中，可以触发一次failover；在Cluster架构中，可以使用CLUSTER FAILOVER命令。这将一个已升级的从节点提升为新的主节点[31]。
  3. 升级旧的主节点: 原来的主节点现在已经降级为从节点，并连接到新的主节点。此时可以安全地对其进行升级。升级完成后，它会以新版本重新加入，作为新主节点的从节点。
- 云服务商的升级: 托管Valkey服务（如AWS ElastiCache、Google Memorystore、OCI Cache）通常提供了简化的、甚至是一键式的版本升级功能。平台会自动在后台执行安全的滚动升级

流程，极大地减轻了运维负担[58]。

Valkey复制协议的设计是实现安全、零停机滚动升级的关键。该协议特意保证了从旧版本主节点到新版本从节点的数据复制具有向后兼容性，这是一个为生产运维而深思熟虑的关键特性[57]。这一保证使得一个明确且安全的升级序列成为可能：首先升级从节点，然后进行故障转移，最后升级旧的主节点。通过先升级从节点，可以确保它们能够理解旧主节点发送的任何复制数据。在故障转移之后，新的主节点运行的是新版本，而旧的主节点（现在已成为从节点）也能理解来自新主节点的复制流。这种精心的设计，将一个潜在的高风险操作转变为一项常规、低影响的维护任务。

第九节：Valkey生态系统：命令与开发者工具

本节将作为一份实用的参考指南，为开发者和管理员提供Valkey常用命令的分类概览，并介绍可用于与Valkey交互的各种工具和客户端库。

综合命令参考

Valkey拥有超过200个命令，覆盖了从基本数据操作到高级系统管理的所有方面。这些命令可以按其操作的数据类型或功能进行分类[62]。

- 按数据类型分类:
  - **String**: SET, GET, INCR, APPEND, MGET
  - **List**: LPUSH, RPOP, LRANGE, BLPOP (阻塞式)
  - **Hash**: HSET, HGET, HGETALL, HINCRBY
  - **Set**: SADD, SISMEMBER, SMEMBERS, SINTER
  - **Sorted Set**: ZADD, ZRANGE, ZRANK, ZREVRANGEBYSCORE
  - **Stream**: XADD, XREAD, XREADGROUP, XACK
- 按功能分类:
  - 通用 **(Generic)**: DEL, EXISTS, EXPIRE, KEYS (慎用), SCAN
  - 连接 **(Connection)**: AUTH, PING, SELECT, QUIT
  - 服务器 **(Server)**: INFO, CONFIG GET/SET, BGSAVE, CLIENT LIST
  - 集群 **(Cluster)**: CLUSTER NODES, CLUSTER MEET, CLUSTER FAILOVER
  - 脚本 **(Scripting)**: EVAL, EVALSHA, SCRIPT LOAD

客户端库的选择

由于Valkey与Redis协议的高度兼容性，绝大多数现有的Redis客户端库都可以直接用于连接Valkey实例[2]。然而，为了获得最佳的兼容性和对Valkey新功能的支持，推荐使用由Valkey社区官方维护或支持的客户端库。

- 官方客户端: Valkey社区正在积极维护从redis-py、go-redis等流行库分支出并重命名的官方客户端，例如valkey-py、valkey-go和valkey-java[63]。

深入了解：GLIDE项目

GLIDE (General Language Independent Driver for the Enterprise) 是Valkey社区的一项战略性投资，旨在为Valkey打造下一代的官方客户端库[63]。

- 架构: GLIDE采用了一种创新的架构：其核心逻辑（包括网络通信、协议解析、连接管理、集群拓扑发现等）使用高性能的Rust语言编写成一个统一的核心库。然后，针对不同的编程语言（如Python, Java, Go, Node.js等）开发轻量级的、符合该语言习惯用法的包装器（wrapper）[63]。
- 设计目标:
  - 性能: 通过Rust核心和高效的连接多路复用、管道化技术，提供卓越的性能[67]。
  - 可靠性: 将复杂的底层逻辑集中在经过严格测试的Rust核心中，减少了在各个语言实现中重复引入bug的可能性。
  - 一致性: 为不同语言的开发者提供统一、一致的功能集和行为，简化了在多语言（polyglot）环境下的开发和运维[63]。

GLIDE项目是Valkey社区的一项关键战略举措。它旨在解决长期以来困扰Redis生态系统的一个问题：不同语言的客户端库在质量、功能支持和性能特性上参差不齐[63]。这种不一致性给开发者带来了碎片化的体验，也使得在跨语言团队中推行最佳实践变得困难。GLIDE通过将复杂的、性能敏感的逻辑（如连接管理、协议解析、集群拓扑）抽象到一个高度优化的Rust核心中来解决这个问题[66]。各语言的包装器只需提供符合其语言习惯的API，而无需重新实现核心逻辑。这意味着对Rust核心的一次bug修复或性能改进，将立即惠及所有支持的语言。这是Valkey生态系统走向成熟的重要一步，超越了Redis开源社区原有的模式。

表5：Valkey客户端库特性对比 (部分)

名称	语言	集群感知	客户端缓存	连接池	Pub/Sub恢复	AZ感知路由
valkey-glide	Python	✓	✗	✗	✓	✓
valkey-py	Python	✓	✗	✓	✗	✗
valkey-glide	Java	✓	✗	✗	✓	✓
valkey-java	Java	✓	✗	✓	✗	✓
valkey-glide	Go	✓	✗	✗	✓	✓
valkey-go	Go	✓	✓	✓	✓	✓
iovalkey	Node.js	✓	✗	✓	✗	✗

数据基于*Valkey*官方文档，特性支持可能随版本更新而变化。[63]

核心工具与实用程序

Valkey发行版自带了一套功能强大的命令行工具，是管理和测试Valkey实例不可或缺的。

- **valkey-cli**: 交互式的命令行客户端，用于执行命令、调试和日常管理[11]。
- **valkey-benchmark**: 性能基准测试工具，用于模拟不同负载下的服务器性能，是性能调优和容量规划的重要工具[11]。
- **valkey-check-aof / valkey-check-rdb**: 用于检查和修复持久化文件的完整性[11]。
- 图形用户界面 **(GUI)** 工具: 许多现有的Redis GUI工具（如RedisInsight, TablePlus等）在基本功能上与Valkey兼容。然而，对于Valkey特有的新功能（如COMMANDLOG），可能需要专门的工具。社区中已经开始出现一些为Valkey量身定做的开源GUI项目，例如Valkey Insight[68]。

结论：Valkey的未来

Valkey的诞生和快速发展，是开源社区力量的一次有力证明。它不仅为Redis OSS用户提供了一条可靠的、无缝的迁移路径，更在一个开放、协作的治理模式下开启了新的创新篇章。

核心优势总结

Valkey的核心价值主张清晰而强大：

- 真正的开源：在Linux基金会的支持下，Valkey保证了其BSD许可证的永久性，为用户提供了免受供应商锁定的自由。
- 卓越的性能：通过I/O多线程、内存效率优化等底层创新，Valkey在现代多核硬件上展现出超越其前身的性能。
- 社区驱动的创新：其开放的治理模式和模块化的扩展策略，吸引了广泛的社区和行业参与，形成了一个充满活力的创新生态系统。

社区路线图展望

Valkey社区正积极规划其未来发展。即将到来的Valkey 9.0版本预计将带来更多激动人心的新功能，例如在哈希数据结构中实现字段级别的过期时间（Hash Field Expirations），以及在集群模式下支持多数据库（Numbered Databases），这些都将进一步增强Valkey的灵活性和应用场景[69]。同时，对模块生态系统和GLIDE客户端的持续投入，将确保Valkey在功能广度和开发者体验上不断进步。

总而言之，Valkey已经证明了自己不仅是一个可靠的替代品，更是一个面向未来的、充满创新活力的高性能数据存储解决方案。它建立在Redis经过数十年验证的坚实基础上，并由开源社区的集体智慧推动向前，是现代数据架构中一个值得信赖和投资的选择。

引用的著作

1. AWS Elasticache: A Performance and Cost Analysis of Redis 7.1 vs. Valkey 7.2, 访问时间为 十月 15, 2025, https://builder.aws.com/content/33pPyndP8elcjWJprmCQaRiDf70/aws-elasticache-a-performance-and-cost-analysis-of-redis-7-1-vs-valkey-7-2

2. Memorystore launches Valkey 8.0 on Google Cloud, 访问时间为 十月 15, 2025, https://cloud.google.com/blog/products/databases/memorystore-launches-valkey-8-0-on-google-cloud
3. Redis vs Valkey Which In-Memory Database Is Best in 2025 - SQLFlash, 访问时间为 十月 15, 2025, https://sqlflash.ai/article/20250924_redis_vs_valkey/
4. Valkey vs Redis: How to Choose in 2025 | Better Stack Community, 访问时间为 十月 15, 2025, https://betterstack.com/community/comparisons/redis-vs-valkey/
5. Linux Foundation Launches Open Source Valkey Community, 访问时间为 十月 15, 2025, https://www.linuxfoundation.org/press/linux-foundation-launches-open-source-valkey-community
6. Redis OSS vs. Valkey - Difference Between Caches - AWS, 访问时间为 十月 15, 2025, https://aws.amazon.com/elasticache/redis/
7. valkey-io/valkey: A flexible distributed key-value database that is optimized for caching and other realtime workloads. - GitHub, 访问时间为 十月 15, 2025, https://github.com/valkey-io/valkey
8. Forking Ahead: A Year of Valkey - Linux Foundation, 访问时间为 十月 15, 2025, https://www.linuxfoundation.org/blog/a-year-of-valkey
9. Valkey, 访问时间为 十月 15, 2025, https://valkey.io/
10. What is Valkey? - Valkey Explained - AWS, 访问时间为 十月 15, 2025, https://aws.amazon.com/cn/elasticache/what-is-valkey/
11. Valkey · Topics, 访问时间为 十月 15, 2025, https://valkey.io/topics/
12. Configure Valkey | Adobe Commerce - Experience League, 访问时间为 十月 15, 2025, https://experienceleague.adobe.com/en/docs/commerce-operations/configuration-guide/cache/valkey/config-valkey
13. Valkey Documentation · Persistence, 访问时间为 十月 15, 2025, https://valkey.io/topics/persistence/
14. valkey-bundle: One stop shop for real-time applications, 访问时间为 十月 15, 2025, https://valkey.io/blog/valkey-bundle-one-stop-shop-for-low-latency-modern-applications/
15. Valkey: What's New and What's Next? - The New Stack, 访问时间为 十月 15, 2025, https://thenewstack.io/valkey-whats-new-and-whats-next/
16. Year One of Valkey: Open-Source Innovations and ElastiCache version 8.1 for Valkey - AWS, 访问时间为 十月 15, 2025, https://aws.amazon.com/blogs/database/year-one-of-valkey-open-source-innovations-and-elasticache-version-8-1-for-valkey/
17. Valkey 8.1: Continuing to Deliver Enhanced Performance and Reliability, 访问时间为 十月 15, 2025, https://valkey.io/blog/valkey-8-1-0-ga/
18. A year on, Valkey charts path to v9 after break from Redis - The Register, 访问时间为 十月 15, 2025, https://www.theregister.com/2025/05/15/a_year_of_valkey/
19. Redis vs. Valkey, 访问时间为 十月 15, 2025, https://redis.io/compare/valkey/
20. Understanding Redis 7.4+ and ValKey (fork from 7.3) : r/devops - Reddit, 访问时间为 十月 15, 2025,

https://www.reddit.com/r/devops/comments/1i51uiz/understanding_redis_74_and_valkey_fork_from_73/
21. Valkey Installation + Setup Guide - Dragonfly, 访问时间为 十月 15, 2025, https://www.dragonflydb.io/guides/valkey-installation-setup
22. Documentation: Installation - Valkey, 访问时间为 十月 15, 2025, https://valkey.io/topics/installation/
23. Valkey version 8.1 - Remi's RPM repository - Blog, 访问时间为 十月 15, 2025, https://blog.remirepo.net/post/2025/08/01/Valkey-version-8.1
24. Valkey container image – securely designed, compliant, and long term supported (LTS), 访问时间为 十月 15, 2025, https://ubuntu.com/blog/valkey-container-image
25. valkey Secure-by-Default Container Image | Chainguard, 访问时间为 十月 15, 2025, https://images.chainguard.dev/directory/image/valkey/overview
26. valkey/valkey - Docker Image - Docker Hub, 访问时间为 十月 15, 2025, https://hub.docker.com/r/valkey/valkey/
27. valkey - Docker Hub, 访问时间为 十月 15, 2025, https://hub.docker.com/u/valkey
28. valkey-container - GitHub, 访问时间为 十月 15, 2025, https://github.com/valkey-io/valkey-container
29. valkey package : Ubuntu - Launchpad, 访问时间为 十月 15, 2025, https://launchpad.net/ubuntu/+source/valkey
30. Documentation: High availability with Valkey Sentinel, 访问时间为 十月 15, 2025, https://valkey.io/topics/sentinel/
31. Valkey Documentation · Cluster tutorial, 访问时间为 十月 15, 2025, https://valkey.io/topics/cluster-tutorial/
32. Announcing general availability of Memorystore for Valkey | Google Cloud Blog, 访问时间为 十月 15, 2025, https://cloud.google.com/blog/products/databases/announcing-general-availability-of-memorystore-for-valkey
33. How to create a Valkey™ cluster | Yandex Cloud - Documentation, 访问时间为 十月 15, 2025, https://yandex.cloud/en/docs/managed-redis/operations/cluster-create
34. How to Create Valkey Clusters | DigitalOcean Documentation, 访问时间为 十月 15, 2025, https://docs.digitalocean.com/products/databases/valkey/how-to/create/
35. Creating a cluster for Valkey or Redis OSS - Amazon ElastiCache - AWS Documentation, 访问时间为 十月 15, 2025, https://docs.aws.amazon.com/AmazonElastiCache/latest/dg/Clusters.Create.html
36. High availability and replicas | Memorystore for Valkey - Google Cloud, 访问时间为 十月 15, 2025, https://cloud.google.com/memorystore/docs/valkey/ha-and-replicas
37. Valkey - cluster mode vs sentinel - Percona Forum, 访问时间为 十月 15, 2025, https://forums.percona.com/t/valkey-cluster-mode-vs-sentinel/37338
38. Valkey Command · INFO, 访问时间为 十月 15, 2025, https://valkey.io/commands/info/
39. Valkey Command · INFO, 访问时间为 十月 15, 2025, https://valkey.io/commands/info
40. toxicoder/nestjs-valkey-health - NPM, 访问时间为 十月 15, 2025, https://www.npmjs.com/package/@toxicoder/nestjs-valkey-health

41. Documentation: Latency monitoring - Valkey, 访问时间为 十月 15, 2025, https://valkey.io/topics/latency-monitor/
42. MONITOR - Valkey Command, 访问时间为 十月 15, 2025, https://valkey.io/commands/monitor/
43. Valkey Command · MEMORY STATS, 访问时间为 十月 15, 2025, https://valkey.io/commands/memory-stats/
44. Supported monitoring metrics | Memorystore for Valkey - Google Cloud, 访问时间为 十月 15, 2025, https://cloud.google.com/memorystore/docs/valkey/supported-monitoring-metrics
45. Metrics for Valkey and Redis OSS - Amazon ElastiCache - AWS Documentation, 访问时间为 十月 15, 2025, https://docs.aws.amazon.com/AmazonElastiCache/latest/dg/CacheMetrics.Redis.html
46. How to Monitor Valkey Database Performance | DigitalOcean Documentation, 访问时间为 十月 15, 2025, https://docs.digitalocean.com/products/databases/valkey/how-to/monitor-databases/
47. About AOF persistence | Memorystore for Valkey - Google Cloud, 访问时间为 十月 15, 2025, https://cloud.google.com/memorystore/docs/valkey/about-aof-persistence
48. Valkey/Redis Key-Specific Dump, Restore, and Migration - Percona, 访问时间为 十月 15, 2025, https://www.percona.com/blog/valkey-redis-key-specific-dump-restore-and-migration/
49. Valkey Documentation · Memory optimization, 访问时间为 十月 15, 2025, https://valkey.io/topics/memory-optimization
50. Documentation: Benchmarking tool - Valkey, 访问时间为 十月 15, 2025, https://valkey.io/topics/benchmark/
51. Valkey Turns One: How the Community Fork Left Redis in the Dust - Momento, 访问时间为 十月 15, 2025, https://www.gomomento.com/blog/valkey-turns-one-how-the-community-fork-left-redis-in-the-dust/
52. Best practices: Valkey/Redis OSS clients and Amazon ElastiCache | AWS Database Blog, 访问时间为 十月 15, 2025, https://aws.amazon.com/blogs/database/best-practices-valkey-redis-oss-clients-and-amazon-elasticache/
53. Cut AWS ElastiCache Costs by Migrating Redis to Valkey - Pump, 访问时间为 十月 15, 2025, https://www.pump.co/blog/aws-migrate-redis-to-valkey
54. Redis to Valkey migration guide - Percona Documentation, 访问时间为 十月 15, 2025, https://docs.percona.com/valkey/migration.html
55. Valkey Documentation · Migration from Redis to Valkey, 访问时间为 十月 15, 2025, https://valkey.io/topics/migration/
56. Redis to ValkeyDB migration guide - Kloia, 访问时间为 十月 15, 2025, https://www.kloia.com/blog/redis-to-valkeydb-migration-guide

57. Documentation: Releases and versioning schema - Valkey, 访问时间为 十月 15, 2025，https://valkey.io/topics/releases/
58. Upgrade the Valkey version of an instance - Google Cloud, 访问时间为 十月 15, 2025，https://cloud.google.com/memorystore/docs/valkey/upgrade-valkey-version
59. Valkey™ version upgrade | Yandex Cloud - Documentation, 访问时间为 十月 15, 2025，https://yandex.cloud/en/docs/managed-redis/operations/cluster-version-update
60. Upgrading engine versions including cross engine upgrades - Amazon ElastiCache, 访问时间为 十月 15, 2025，https://docs.aws.amazon.com/AmazonElastiCache/latest/dg/VersionManagement.HowTo.html
61. Upgrade OCI Cache with Redis to Valkey | by Phantompete | Aug, 2025 - Medium, 访问时间为 十月 15, 2025，https://medium.com/@devpiotrekk/upgrade-oci-cache-with-redis-to-valkey-d3c01deb8733
62. Valkey · Commands, 访问时间为 十月 15, 2025，https://valkey.io/commands/
63. Valkey · Client Libraries, 访问时间为 十月 15, 2025，https://valkey.io/clients/
64. valkey-io repositories - GitHub, 访问时间为 十月 15, 2025，https://github.com/orgs/valkey-io/repositories
65. Client library code samples | Memorystore for Valkey - Google Cloud, 访问时间为 十月 15, 2025，https://cloud.google.com/memorystore/docs/valkey/client-library-code-samples
66. valkey-io/valkey-glide: An open source Valkey client library that supports Valkey, and Redis open source 6.2, 7.0 and 7.2. Valkey GLIDE is designed for reliability, optimized performance, and high-availability, for Valkey and Redis OSS based applications. GLIDE is a multi language client library, written in - GitHub, 访问时间为 十月 15, 2025，https://github.com/valkey-io/valkey-glide
67. Introducing the Valkey Glide Go Client: Now in Public Preview!, 访问时间为 十月 15, 2025，https://valkey.io/blog/2025-03-4-go-client-in-public-preview/
68. Valkey Insight, a fast and open source GUI for Valkey : r/opensource - Reddit, 访问时间为 十月 15, 2025，https://www.reddit.com/r/opensource/comments/1o4shux/valkey_insight_a_fast_and_open_source_gui_for/
69. Valkey · Blog, 访问时间为 十月 15, 2025，https://valkey.io/blog/