Redis通用命令（一）（24~30）

一、 软件架构基础与性能评价指标

该部分明确了构建系统时的核心术语和衡量标准：

- **应用 (Application) / 系统 (System)**：指代一个或一组服务器程序。
- **模块 (Module) / 组件 (Component)**：应用内部细分的功能单元。
- **分布式 (Distributed)**：物理层面的概念，指引入多个主机协同配合工作。
- **集群 (Cluster)**：逻辑层面的概念，同样引入多个主机配合，对外表现为一个整体。
- **主 (Master) / 从 (Slave)**：典型的分布式结构，从节点的数据从主节点同步而来。
- **中间件 (Middleware)**：与业务无关的通用服务，如数据库、缓存、消息队列等。
- **可用性 (Availability)**：系统的第一要务，指可用时间占比。
  - **计算公式**：$可用性 = \frac{系统整体可用时间}{总时间}$。
  - **标准**：4 个 9（99.99%）或 5 个 9（99.999%）。
- **响应时长 (Response Time RT)**：衡量服务器性能，数值越小越好。
- **吞吐 (Throughput) vs 并发 (Concurrent)**：衡量系统处理请求能力的方式。

二、 分布式系统演化路径

系统演进通常经历以下七个关键阶段：

1. **单机架构**：应用与数据库同机部署。
2. **数据库和应用分离**：分别部署在不同主机。
3. **负载均衡（集群）**：通过负载均衡器分发请求，提高**可用性**（部分机器挂掉，余者仍可服务）。
4. **读写分离（主从结构）**：主写从读，同步数据。
5. **引入缓存（冷热分离）**：遵循“二八原则”提升性能，但也带来**数据一致性**挑战。
6. **分库分表**：进一步扩展数据库存储空间。
7. **微服务架构**：按业务功能拆分应用服务器，功能更单一小巧。

三、 Redis 核心特性与优势

Redis 作为内存数据结构中间件，具备以下特性：

- **内存数据结构**：非关系型键值对存储。Key 永远是 String，Value 支持 String, Hash, List, Set, Sorted Set, Stream 等。
- **可编程性**：支持简单命令和 **Lua 脚本**。
- **可扩展性**：提供 API 允许使用 C, C++, Rust 编写扩展插件（.so 或 .dll 动态库）。
- **持久化**：内存为主，磁盘为辅，重启后可恢复数据。
- **集群**：支持水平扩展（哈希分片），突破单机内存限制。
- **高可用**：支持主从结构和自动故障转移。

四、 性能分析：为什么 Redis 快?

1. **内存存储**：速度远超硬盘数据库。
2. **逻辑简单**：核心任务是内存数据结构操作，不复杂。
3. **IO 多路复用**：网络层使用 **epoll** 方式，单线程管理多 socket。
4. **单线程模型**：减少线程竞争开销。由于不吃 CPU，单线程足以应对。
5. **C 语言开发**：执行效率高。

对比说明：相比单机系统中的 Hash Map，Redis 因涉及网络传输而更慢，但它解决了**数据共享**和**重启持久化**的问题。

五、 实战应用场景

- **实时数据存储**：用于极高性能需求的场景，如广告搜索。
- **缓存与分布式会话**：
  - **缓存**：拎出热点数据。
  - **Session 存储**：集中管理会话，解决负载均衡导致的会话丢失问题，重启不丢失。
- **消息队列**：实现网络版的生产者-消费者模型，解耦和削峰填谷。

六、 安装、配置与架构

- **架构**：经典的 Client-Server 结构。
- **客户端**：包括命令行客户端（redis-cli）、图形化界面以及基于 API（C, JDBC 等）的自定义开发端。
- **安装 (Ubuntu 20.04)**：

a. `su` 切换 root 用户。

b. `apt install redis` 安装。

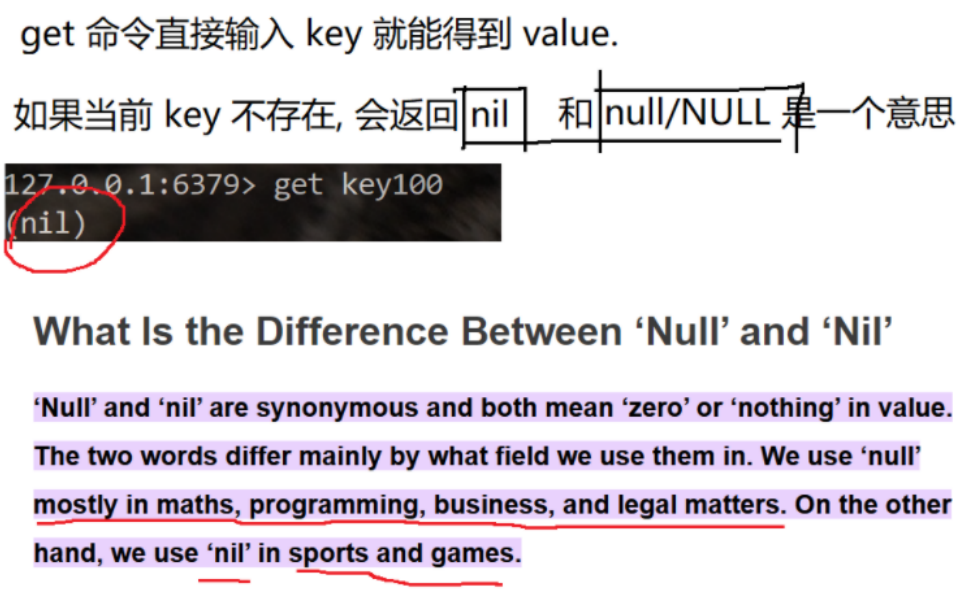
c. 修改 `redis.conf`：将 `bind` 改为 `0.0.0.0`，并设置 `protected-mode no`。

d. `service redis-server restart` 重启。

- **CentOS 安装**：使用 `yum`，注意版本差异（CentOS 7 默认版本较低，需 scl 源）。
- **运行方式**：通常以**守护进程 (Daemon)** 在后台运行。

七、 Redis 常用命令与全局操作

- **核心命令**：`SET key value` 存入数据，`GET key` 读取数据。



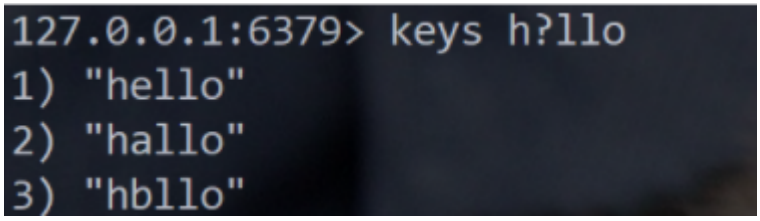
- **概念辨析**：Redis 返回 `(nil)`，等同于 `Null` 或 `NULL`（空值）。
- **全局命令 (keys)**：支持通配符查询。

> 用来查询当前服务器上匹配的key
>
> 通过一些特殊符号(通配符)来描述key的模样，匹配上述模样的key 就能被查询出来.

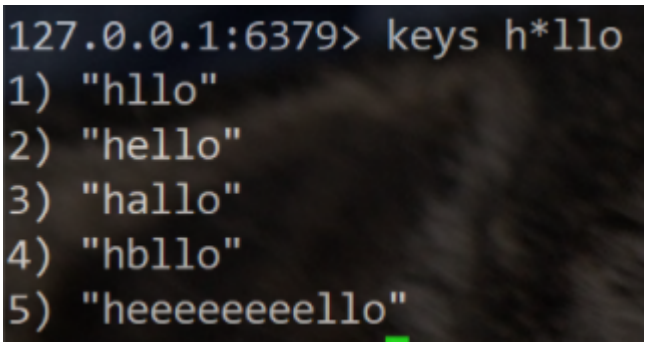
pattern

> 包含特殊符号的字符串.有的地方翻译成"样式”或者"模式"重点去认识这个英文术语
>
> 存在的意义，是去描述另外的字符串 长啥样的

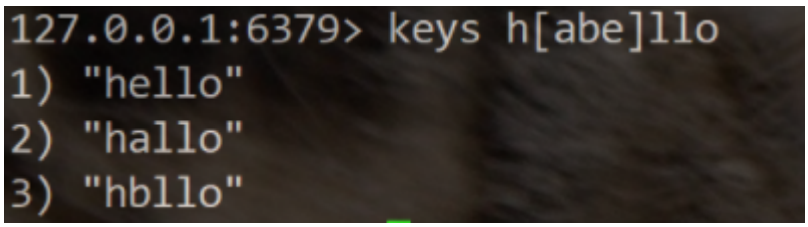
- `?`：匹配任意一个字符。



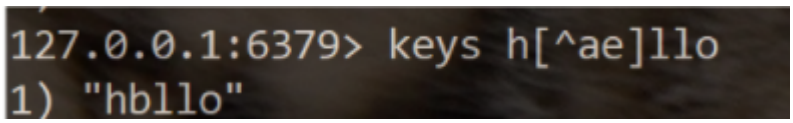
- `*`：匹配 0 个或多个字符。



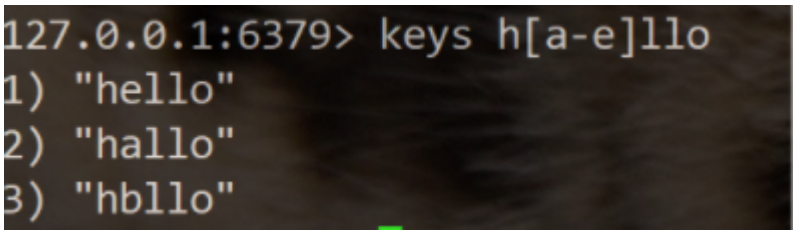
- `[abc]`：只能匹配括号内的 a、b 或 c。



- `[^e]`：排除 e，匹配除 e 以外的字符。



- `[a-b]`：匹配指定范围内的字符。



八、 生产环境警告与局限

- **高危操作**：`keys *` 命令在生产环境下**严禁使用**！其时间复杂度为 $O(N)$，在数据量大时会阻塞单线程服务器，导致服务瘫痪。
- **环境区分**：明确办公环境、开发环境、测试环境与线上/生产环境的区别。
- **局限性**：Redis 不适合存储超大规模数据。
- **学习建议**：注重业务场景，理清技术来龙去脉，通过作业、博客和项目深入理解。

九、未来在工作中会涉及到的几个环境:

1.办公环境(入职公司之后，公司给你发个电脑)

笔记本(windows, mac)/台式机，现在办公电脑，一般内存8C16G，硬盘512G

2.开发环境 有的时候，开发环境和办公环境是一个

有的时候，开发环境是单独的服务器.28C128G 4T

做前端/做客户端，一般来说，开发环境就是办公环境了.后端来说，很可能是单独的服务器

有的后端程序，会比较复杂~~

1. 编译一次时间特别久(C++)=>C++23才会引入module(#include要接锅)

使用高性能的服务器，进行编译~~

2. 有的程序一启动要消耗很多的cpu和内存资源.办公电脑难以支撑

商业搜索的服务器通常启动起来要吃100G的内存

3. 有的程序比较依赖linux，在windows环境搭不起来

3.测试环境(测试工程师使用的)

28C128G4T

4.线上环境/生产环境

(办公环境，开发环境，测试环境，也统称为线下环境，外界用户无法访问到的)

线上环境则是外界用户能够访问到的.

一旦生产环境上出问题，一定会对于用户的使用产生影响!!

直接的影响到公司营收!!!

很多公司的营收都是靠广告.广告一般是按照展示/点击次数来计费的