Redis通用命令（二）(31~37)

一、整体主题概览

- key 是否存在（exists）
- key 的创建、访问、删除
- key 在内存中的存储模型
- key 的过期机制（expire / TTL）
- Redis 如何处理“过期 key”
- Redis 与 MySQL 等持久化数据库在“删除 / 过期”语义上的根本差异
- Redis 为什么要这么设计（性能 & 内存角度）

> **Redis 并不是简单的 KV 存储，而是一个“以内存为核心、以 Key 生命周期为中心”的高性能数据系统**

二、Key 是否存在：`exists`

1️⃣ 基本命令

```
代码块
1   EXISTS key [key ...]
```

- 返回值：
  - `1`：至少有一个 key 存在
  - `0`：所有 key 都不存在

> **exists 只关心“逻辑存在性”，不关心数据类型，也不关心 value 内容**

2️⃣ 隐含语义（图中重点）

- Redis 判断 key 是否存在，本质是：

    - **查字典（hash table）中是否有这个 key 的 entry**
- 并不会：
    - 反序列化 value
    - 扫描 value 内容
    - 关心 key 是否“即将过期”（只要没被删除，就算存在）

三、Key 在 Redis 中是如何存储的

图中有一个 **“Redis 客户端 → Redis 服务端” 的示意图**，这是非常关键的一部分。

1️⃣ Redis 内部模型

- Redis 内部维护一个 **全局字典（dict）**
- 每个 key 对应一个结构体，大致包含：

```
代码块
1   key -> {
2       value 指针
3       type（string / list / hash / set / zset）
4       过期时间（可选）
5       其他元数据
6   }
```

📌 图中用“方框 + 多条线”表示：

- key 本身很轻量
- value 可能是复杂结构
- key 和 value 是通过指针/引用关联的

2️⃣ 非常重要的一点

> **Redis 的“key 是否存在”，并不是“value 是否为空”**

- 即使 value 是空字符串
- 只要 key 在 dict 里
- `exists key` 就返回 1

四、删除 Key：`DEL`

1️⃣ 基本命令

```
代码块
1   DEL key [key ...]
```

- 返回值：
  - 实际被删除的 key 数量

2️⃣ Redis 的“删除”语义（图中红字重点）

这里图中和 MySQL 做了一个**非常重要的对比**：

🔴 MySQL 的删除

```
代码块
1   DELETE FROM table WHERE ...
```

- 是：
  - 修改磁盘数据
  - 涉及日志、事务、索引
  - 成本高、不可忽略

🟢 Redis 的删除

```
代码块
1   DEL key
```

- 本质是：
  - 从内存 dict 中移除 entry
  - 释放 value 对应的内存
- 非常快（O(1) ~ O(n) 与 value 大小有关）

📌 图中强调：

> Redis 的删除 = 内存层面的“断引用 + 回收”

3️⃣ 延伸：惰性删除 vs 立即删除（隐含背景）

虽然图中没直接写 `lazy free`，但它的逻辑基础就是：

- 大对象（如大 list / hash）：
  - Redis 可能采用后台线程释放
- 目的是：
  - 避免阻塞主线程

五、Key 的过期机制：`EXPIRE / TTL`

这是整张图的**核心内容**。

1️⃣ 设置过期时间

```
代码块
1   EXPIRE key seconds
```

- 给 key 设置一个 **生存时间（TTL）**
- 到期后 key 会被“视为无效”

也可以用：

```
代码块
1   PEXPIRE key milliseconds
```

2️⃣ 查看剩余时间：`TTL`

```
代码块
1   TTL key
```

返回值含义（图中重点）：

返回值	含义
>0	剩余秒数
-1	key 存在，但**没有设置过期时间**
-2	key **不存在**（可能被删了或过期了）

📌 图中特别标红了：

> **-1 ≠ -2，这是很多初学者的误区**

3️⃣ 过期 ≠ 立刻删除（非常重要）

图中用大段文字 + 箭头说明了这一点：

> **Redis 的过期是“逻辑过期”，不是“定时器强制删除”**

六、Redis 是如何处理“过期 Key”的

图中这一部分信息密度非常高，我帮你完整拆解。

1️⃣ 惰性删除（Lazy Expiration）

- 当客户端访问一个 key 时：
  - Redis 先检查：
    - key 是否设置了过期时间
    - 当前时间是否超过 TTL
- 如果过期：
  - **立即删除**
  - 返回“不存在”

📌 优点：

- 不浪费 CPU
- 只在“被访问时”处理

📌 缺点：

- 如果 key 一直没人访问：
  - 会一直占内存

2️⃣ 定期删除（Active Expiration）

图中画了“周期扫描”的概念。

- Redis 后台会：
  - 定期随机抽取一部分设置了过期时间的 key
  - 检查是否过期
  - 过期就删

📌 注意：

- 不是全量扫描
- 是 **概率性、渐进式** 清理

有点类似于TCP的拥塞控制

3️⃣ 为什么不全量扫描？

原因只有一个：

> **性能**

- Redis 是单线程主逻辑
- 全量扫描会：
  - 阻塞命令
  - 影响 QPS
- 所以采用：
  - 惰性删除 + 定期抽样删除

七、过期 Key 与内存问题

图中最后一大块红字，本质是在强调一个现实问题：

> **过期 Key 并不等于“内存立刻释放干净”**

1️⃣ 可能出现的问题

- 大量 key：
  - 设置了过期时间
  - 但访问频率极低
- 结果：
  - 内存占用上升
  - 实际“无效数据”变多

2️⃣ Redis 的设计取舍

Redis 并不追求：

> “每一个过期 key 都第一时间删除”

而是追求：

> **在性能、延迟、内存之间的平衡**

八、Redis Key 生命周期完整总结

我把整张图压缩成一条**完整生命周期链路**：

代码块

```
key 创建
 ↓
key 存在于 dict
 ↓
（可选）设置 TTL
 ↓
正常访问（exists / get / ttl）
 ↓
到期后：
   ├─ 被访问 → 惰性删除
   └─ 被后台抽查 → 定期删除
 ↓
DEL / 过期删除
 ↓
内存回收
```

九、总结：Redis的key的过期策略是怎么实现的 【经典面试题】

> 一个redis 中可能同时存在很多很多key.这些key 中可能有很大一部分都有过期时间.
>
> 此时，redis 服务器咋知道哪些key 已经过期要被删除，哪些key还没过期??

如果直接遍历所有的key，显然是行不通的.效率非常低~~

Redis的整体策略是

1. 定期删除

> 也需要结合定期删除的操作

每次抽取一部分，进行验证过期时间。保证这个抽取检查的过程，足够快

为啥这里对于定期删除的时间，有明确的要求呢?

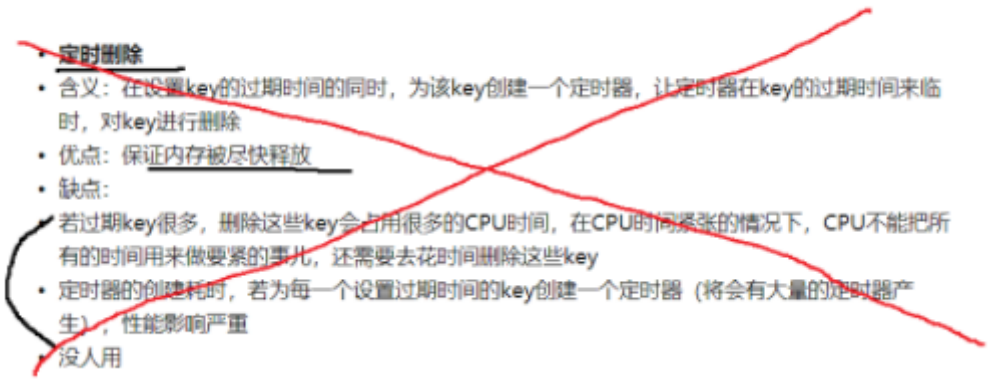
因为redis是单线程的程序.主要的任务(处理每个命令的任务，刚才扫描过期key....)如果扫描过期key 消耗的时间太多了，就可能导致正常处理请求命令就被阻塞了.(产生了类似于执行keys*这样的效果)

2. 惰性删除

假设这个key 已经到过期时间了，但是暂时还没删它，key还存在.紧接着，后面又一次访问，正好用到了这个key，于是这次访问就会让redis服务器触发删除key的操作，同时再返回一个nil

虽然有了上述两种策略结合，整体的效果一般仍然可能会有很多过期的key被残留了，没有及时删除掉

redis为了对上述进行补充，还提供了一系列的内存淘汰策略~~

> 
>

> 1. redis 中并没有采取定时器的方式来实现过期key 删除.
> 2. 如果有多个key过期，也可以通过一个定时器来高效/节省cpu的前提下来处理多个key
>
> 基于优先级队列 或者 基于 时间轮都可以实现比较高效的定时器

> 为啥redis没有采取这种定时器的方式呢?很难考证为啥.个人的猜测:基于定时器实现，势必就要引入多线程了.redis早期版本就是奠定了单线程的基调引入多线程就打破了作者的初衷