

# 目录 Contents

- ◆ 过期数据
- ◆ 数据删除策略
- ◆ 逐出算法

## 过期数据

### Redis中的数据特征

- Redis是一种内存级数据库，所有数据均存放在内存中，内存中的数据可以通过TTL指令获取其状态
  - XX：具有时效性的数据
  - -1：永久有效的数据
  - -2：已经过期的数据或被删除的数据或未定义的数据



过期的数据真的删除了吗？

## 过期数据



### Redis中的数据特征



## 过期数据

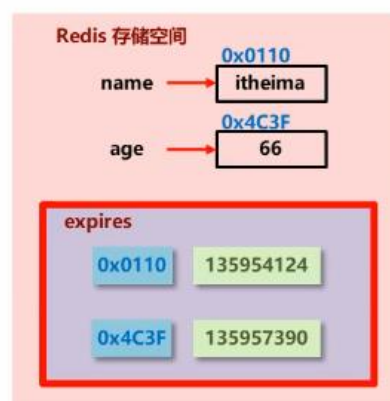
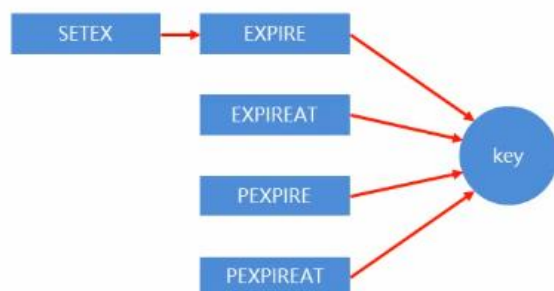
### 数据删除策略

1. 定时删除
2. 惰性删除
3. 定期删除

## 数据删除策略



### 时效性数据的存储结构



## 数据删除策略

### 数据删除策略的目标

在内存占用与CPU占用之间寻找一种平衡，顾此失彼都会造成整体redis性能的下降，甚至引发服务器宕机或内存泄露

## ■ 数据删除策略

### 定时删除

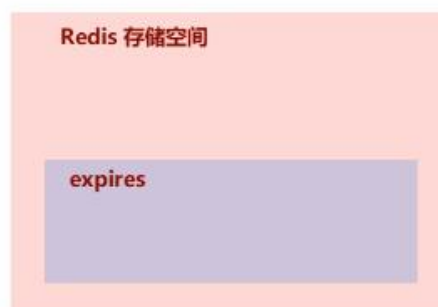
- 创建一个定时器，当key设置有过期时间，且过期时间到达时，由定时器任务立即执行对键的删除操作



## ■ 数据删除策略

### 定时删除

- 创建一个定时器，当key设置有过期时间，且过期时间到达时，由定时器任务立即执行对键的删除操作
- 优点：节约内存，到时就删除，快速释放掉不必要的内存占用
- 缺点：CPU压力很大，无论CPU此时负载量多高，均占用CPU，会影响redis服务器响应时间和指令吞吐量
- 总结：用处理器性能换取存储空间（拿时间换空间）

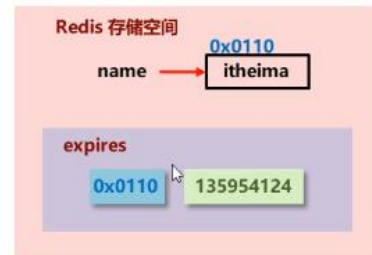


## 数据删除策略



### 惰性删除

- 数据到达过期时间，不做处理。等下次访问该数据时
  - 如果未过期，返回数据
  - 发现已过期，删除，返回不存在

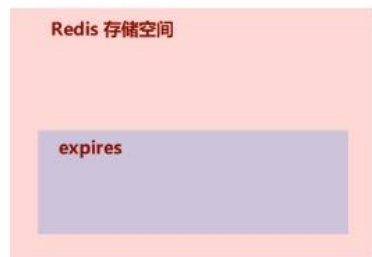


## 数据删除策略



### 惰性删除

- 数据到达过期时间，不做处理。等下次访问该数据时
  - 如果未过期，返回数据
  - 发现已过期，删除，返回不存在
- 优点：节约CPU性能，发现必须删除的时候才删除
- 缺点：内存压力很大，出现长期占用内存的数据
- 总结：用存储空间换取处理器性能（拿时间换空间）



## 数据删除策略



### 定期删除



两种方案都走极端，有没有折中方案？

- Redis启动服务器初始化时，读取配置server.hz的值，默认为10
- 每秒钟执行server.hz次 `serverCron()`
  - ↳ `databasesCron()`
  - ↳ `activeExpireCycle()`
- `activeExpireCycle()`对每个 `expires[*]` 逐一进行检测，每次执行 `250ms/server.hz`
- 对某个 `expires[*]` 检测时，随机挑选 `W` 个key检测
  - 如果key超时，删除key
  - 如果一轮中删除的key的数量  $> W * 25\%$ ，循环该过程
  - 如果一轮中删除的key的数量  $\leq W * 25\%$ ，检查下一个 `expires[*]`，0-15循环
  - `W`取值=`ACTIVE_EXPIRE_CYCLE_LOOKUPS_PER_LOOP`属性值
- 参数 `current_db` 用于记录 `activeExpireCycle()` 进入哪个 `expires[*]` 执行
- 如果 `activeExpireCycle()` 执行时间到期，下次从 `current_db` 继续向下执行



可以通过 `info server` 来查询这个值，在里面他叫 `hz`

## ■ 数据删除策略

### 定期删除

- 周期性轮询redis库中的时效性数据，采用随机抽取的策略，利用过期数据占比的方式控制删除频度
- 特点1：CPU性能占用设置有峰值，检测频度可自定义设置
- 特点2：内存压力不是很大，长期占用内存的冷数据会被持续清理
- 总结：周期性抽查存储空间（随机抽查，重点抽查）

## ■ 数据删除策略



### 删除策略比对

- |         |            |                    |             |
|---------|------------|--------------------|-------------|
| 1. 定时删除 | ● 节约内存，无占用 | ● 不分时段占用CPU资源，频度高  | ● 拿时间换空间    |
| 2. 惰性删除 | ● 内存占用严重   | ● 延时执行，CPU利用率高     | ● 拿空间换时间    |
| 3. 定期删除 | ● 内存定期随机清理 | ● 每秒花费固定的CPU资源维护内存 | ● 随机抽查，重点抽查 |

## ■ 逐出算法

### 新数据进入检测



#### 当新数据进入redis时，如果内存不足怎么办？

- Redis使用内存存储数据，在执行每一个命令前，会调用`freeMemoryIfNeeded()`检测内存是否充足。如果内存不满足新加入数据的最低存储要求，redis要临时删除一些数据为当前指令清理存储空间。清理数据的策略称为逐出算法。
- 注意：逐出数据的过程不是100%能够清理出足够的可使用的内存空间，如果不成功则反复执行。当对所有数据尝试完毕后，如果不能达到内存清理的要求，将出现错误信息。

```
(error) OOM command not allowed when used memory > 'maxmemory'
```

### 影响数据逐出的相关配置

- 最大可使用内存

```
maxmemory
```

占用物理内存的比例，默认值为0，表示不限制。生产环境中根据需求设定，通常设置在50%以上。

- 每次选取待删除数据的个数

```
maxmemory-samples
```

选取数据时并不会全库扫描，导致严重的性能消耗，降低读写性能。因此采用随机获取数据的方式作为待检测删除数据

- 删除策略

```
maxmemory-policy
```

达到最大内存后的，对被挑选出来的数据进行删除的策略

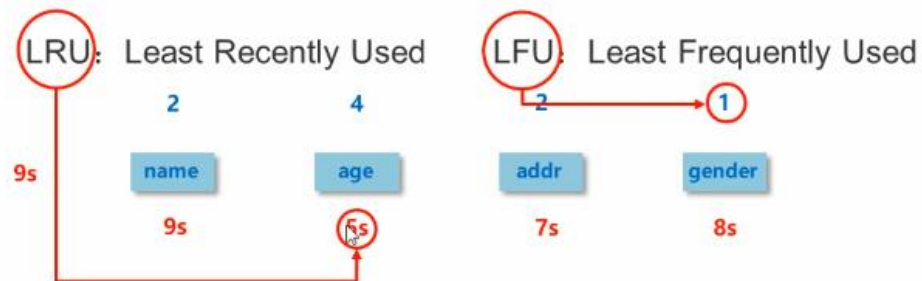
## ■ 逐出算法



### 影响数据逐出的相关配置

- 检测易失数据（可能会过期的数据集server.db[i].expires）

- ① volatile-lru：挑选最近最少使用的数据淘汰
- ② volatile-lfu：挑选最近使用次数最少的数据淘汰
- ③ volatile-ttl：挑选将要过期的数据淘汰
- ④ volatile-random：任意选择数据淘汰



- 检测全库数据（所有数据集server.db[i].dict）

- ⑤ allkeys-lru：挑选最近最少使用的数据淘汰
- ⑥ allkeys-lfu：挑选最近使用次数最少的数据淘汰
- ⑦ allkeys-random：任意选择数据淘汰

- 放弃数据驱逐

- ⑧ no-eviction（驱逐）：禁止驱逐数据（redis4.0中默认策略），会引发错误OOM（Out Of Memory）

```
maxmemory-policy volatile-lru
```



## 数据逐出策略配置依据

- 使用INFO命令输出监控信息，查询缓存 hit 和 miss 的次数，根据业务需求调优Redis配置

## 总结



### Redis删除策略

- ◆ 数据删除策略
  - ◆ 定时删除
  - ◆ 惰性删除
  - ◆ 定期删除
- ◆ 数据逐出策略
  - ◆ 8种策略