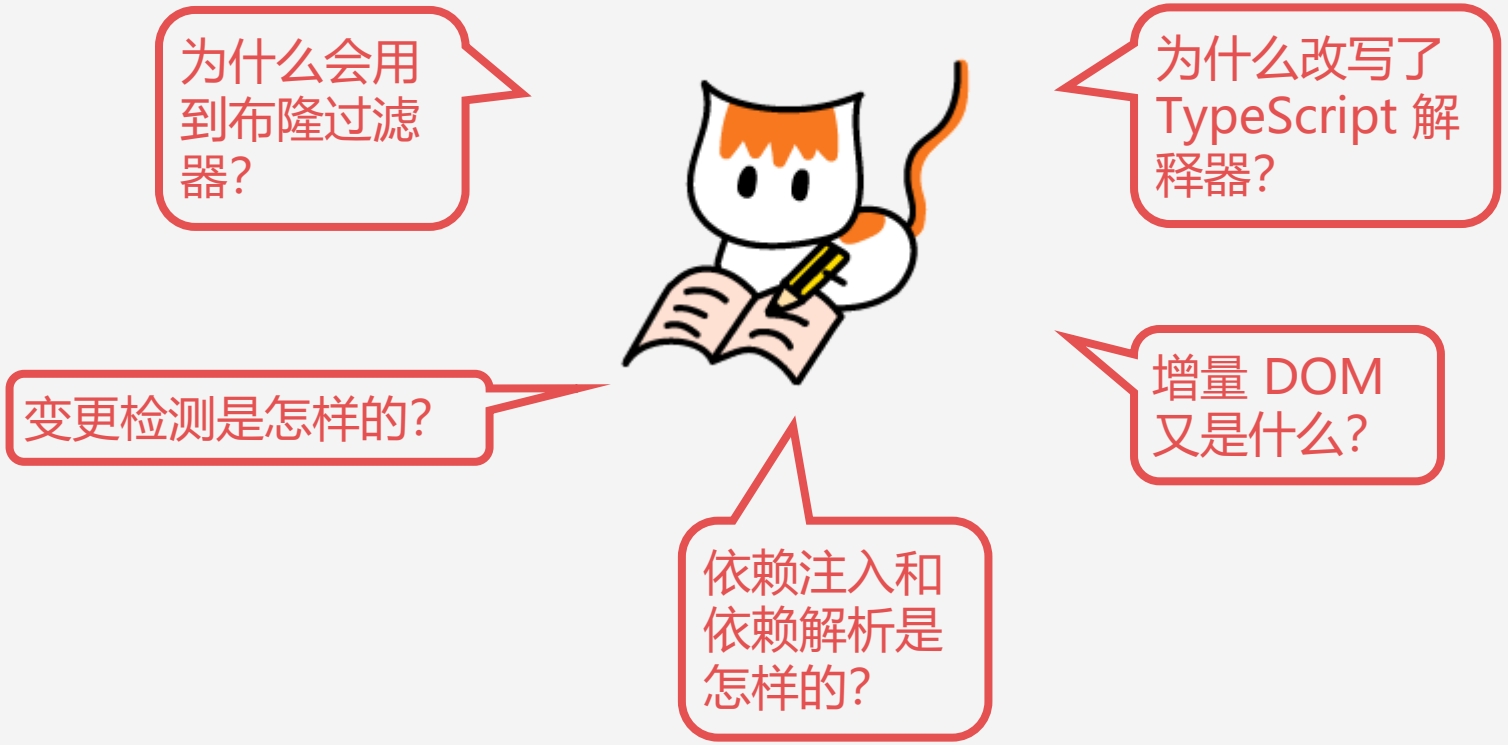
# Angular 冷知识
# 之
# 布隆过滤器

@被删

# Angular 知多少


为什么会用到布隆过滤器?
为什么改写了 TypeScript 解释器?
变更检测是怎样的?
增量 DOM 又是什么?
依赖注入和依赖解析是怎样的?

# 布隆过滤器
# 到底是什么来的?

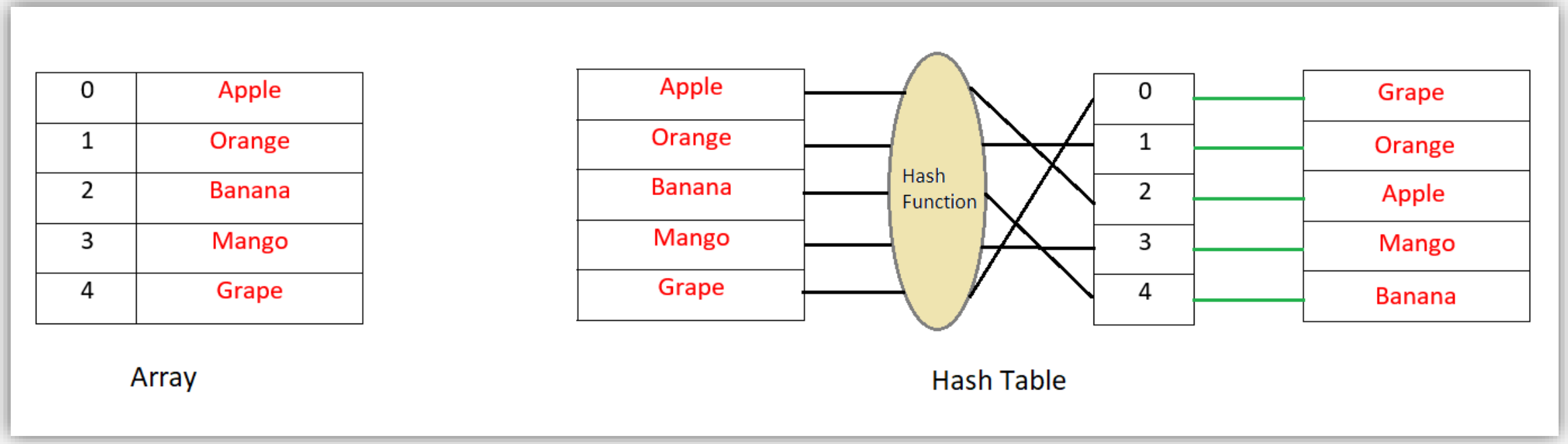
1

# Angular 中的布隆过滤器

# 哈希表的工作原理

在简单数组或列表中插入新数据时，插入此数据的索引不是由要插入的值确定的。这意味着“键（索引）”和“值（数据）”之间没有直接关系。

因此，如果需要在数组中搜索某个值，则必须在所有索引中进行搜索。

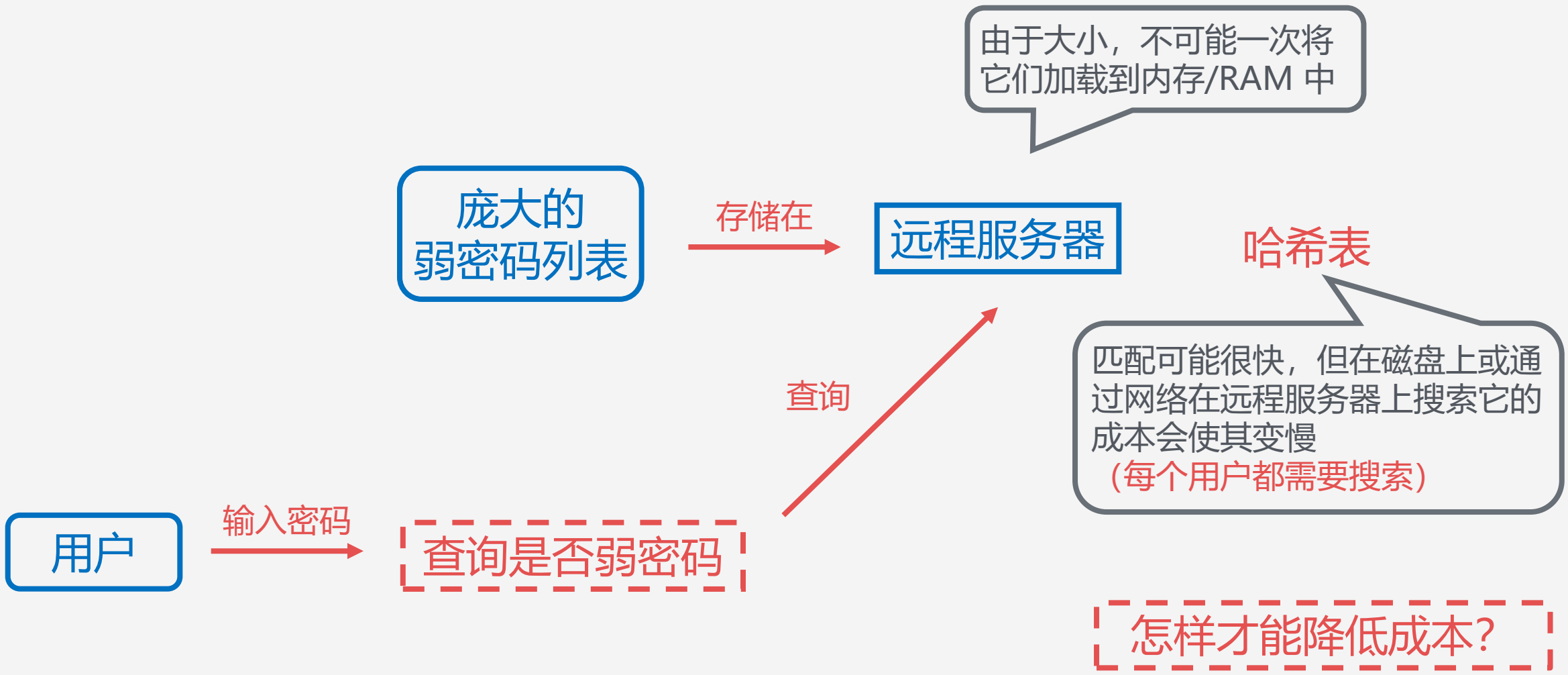



在哈希表中，可以通过散列“值”来确定“键”或“索引”。

每次需要检查列表中是否存在该值时，只需对值进行散列并搜索该键。

O(1)搜索时间

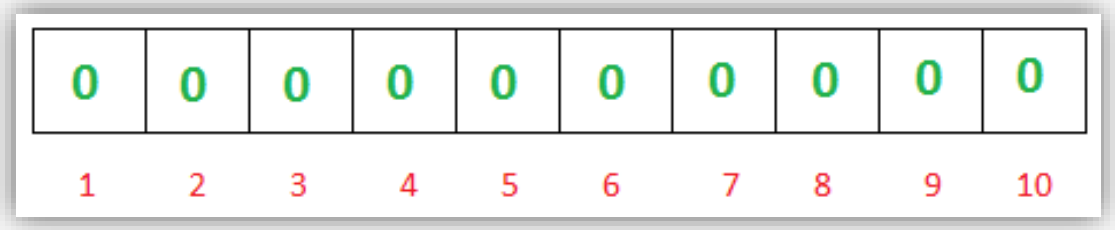
# 哈希表的搜索成本


由于大小，不可能一次将它们加载到内存/RAM 中
庞大的弱密码列表
存储在
远程服务器
哈希表
匹配可能很快，但在磁盘上或通过网络在远程服务器上搜索它的成本会使其变慢
（每个用户都需要搜索）
查询
用户
输入密码
查询是否弱密码
怎样才能降低成本?

# 布隆过滤器

# 布隆过滤器

布隆过滤器可以检查值是“可能在集合中”，还是“绝对不在集合中”



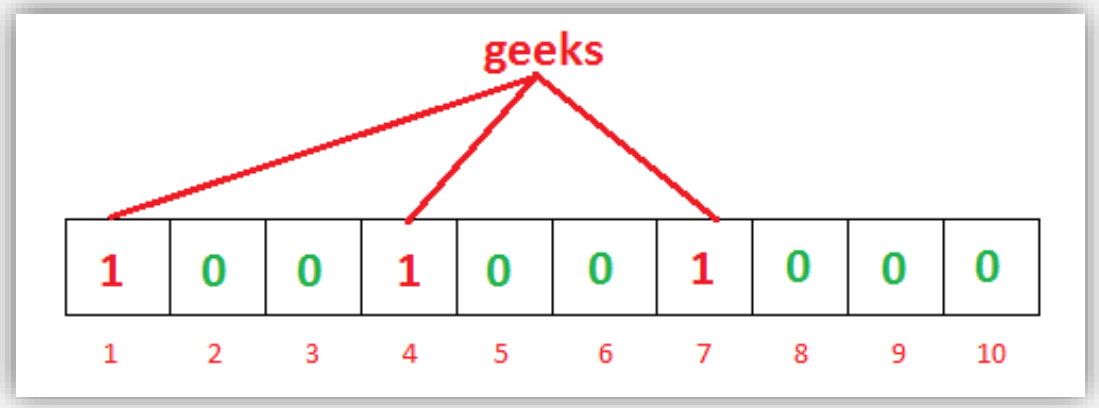

由长度为 m 的位向量或位列表（仅包含0或1位值的列表）组成

在哈希表中，使用单个哈希函数，因此仅获得单个索引作为输出。

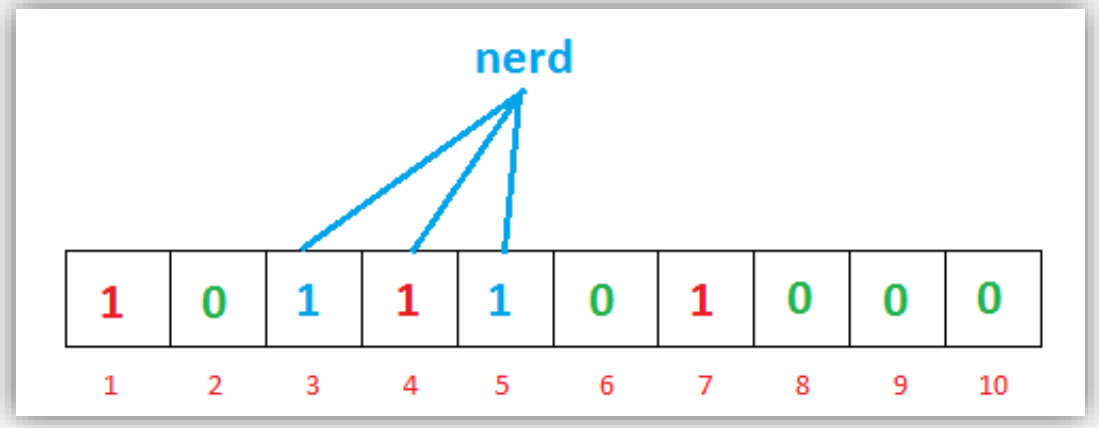
但是在布隆过滤器的情况下，使用多个哈希函数，将提供多个索引。

# 布隆过滤器

布隆过滤器可以检查值是“可能在集合中”，还是“绝对不在集合中”

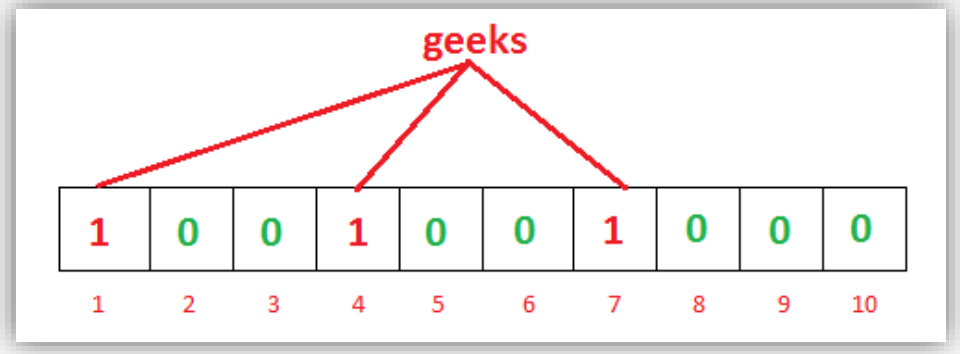


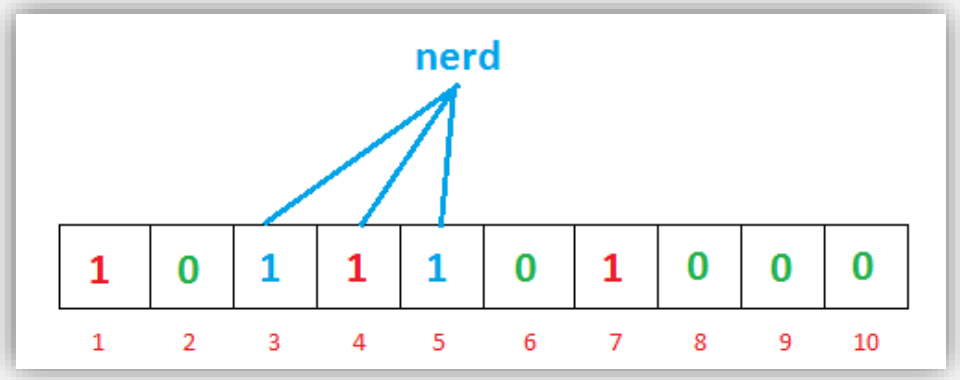

对于给定的输入“geeks”，
我们的 3 个哈希函数将给出 3
个不同的输出：1、4 和 7

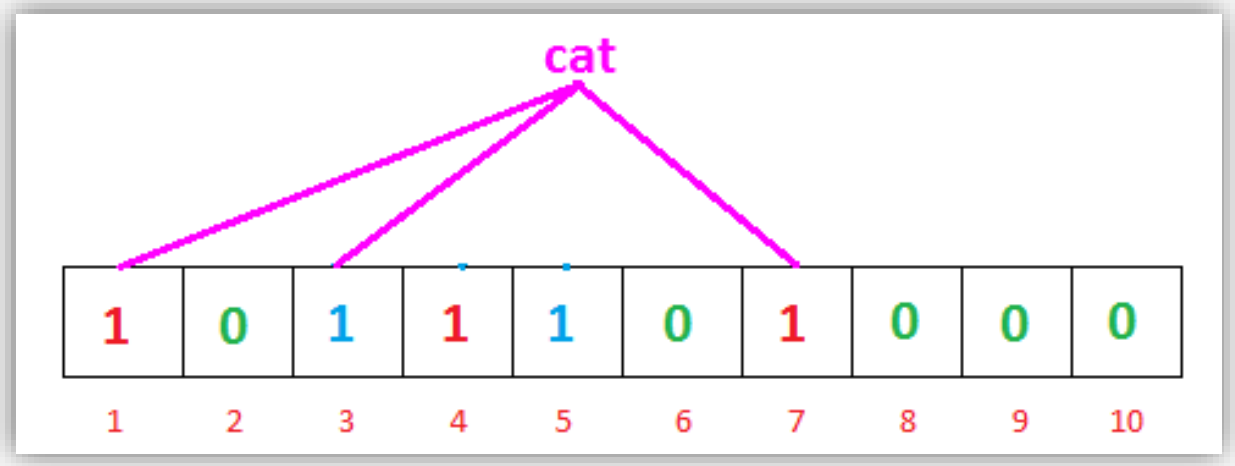



对于另一个输入 “nerd”，
哈希函数为我们提供 3、4 和 5

# 布隆过滤器



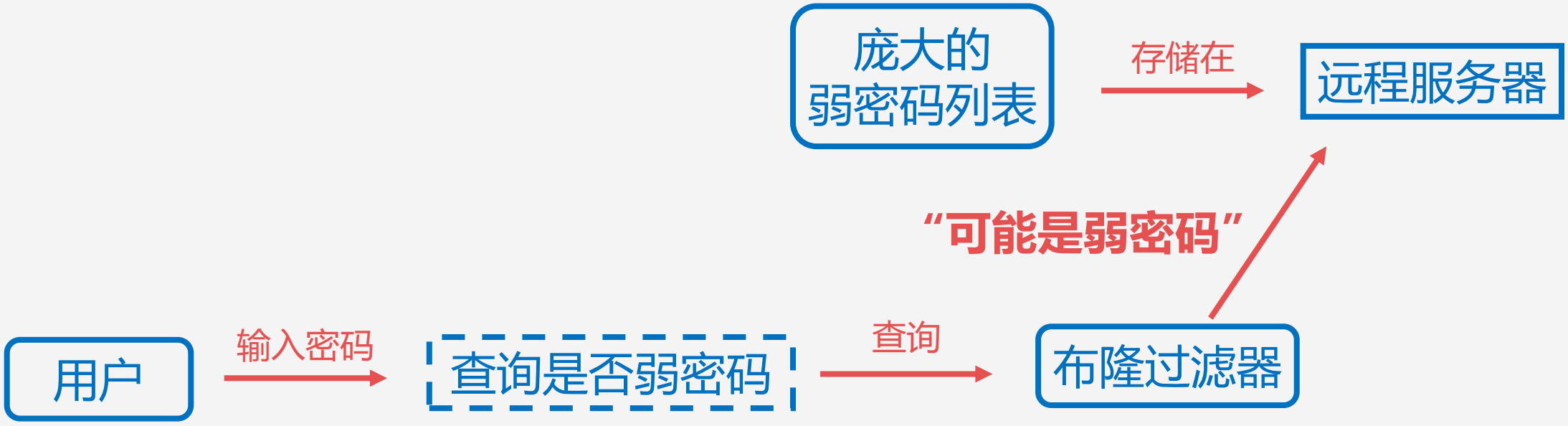




搜索“cat”，哈希函数这次给了 1、3 和 7

可以检查值是“可能在集合中”
还是“绝对不在集合中”

# 弱密码的搜索

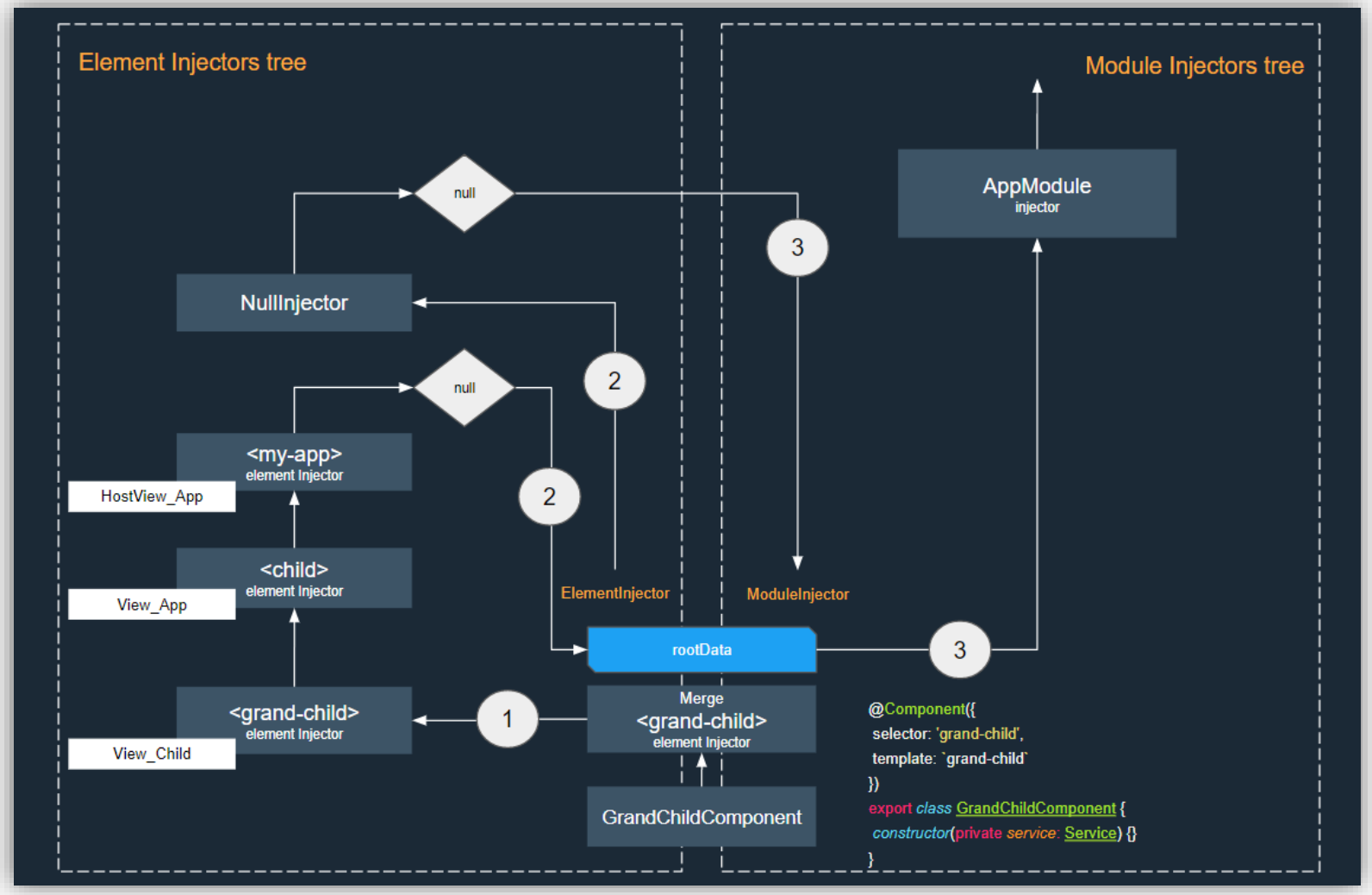
由于布隆过滤器的大小不会很大并且是固定大小，因此可以很容易地将其存储在内存中，必要时也可以存储在客户端。



# 为什么 Angular 需要用到布隆过滤器?

# 2 Angular 中的视图数据

# 依赖注入查询过程




1. 首先查看子元素注入器。
2. 然后遍历所有父视图元素（1），并检查元素注入器中的提供者。
3. 如果下一个父视图元素等于null（2），则返回到 startView，检查 startView.rootData.elnjector（3）。
4. 只有在找不到令牌的情况下，才检查 startView.rootData module.injector（4）。

# Angular 中的视图数据

Ivy 渲染器将节点的注入信息存储在视图数据中，每个嵌入式视图和组件视图都有自己的 LView。

```
export interface LView extends Array<any> {
  // 插入该 LView 的节点
  [HOST]: RElement | null;

  // 此视图的静态数据
  readonly [TVIEW]: TView;

  // 父视图
  [PARENT]: LView | LContainer | null;

  // 下一个同级视图或容器
  [NEXT]: LView | LContainer | null;

  // 对此视图有效的查询-视图中的节点将报告给这些查询
  [QUERIES]: LQueries | null;

  // 存储当前 LView 插入位置的 TNode
  [T_HOST]: TNode | null;
```

LView 存储了从模板调用指令时，处理指令所需的所有信息：

- LView 中存储了足够多的信息，这样的设计使单个数组可以以紧凑的形式包含模板渲染所需的所有必要数据
- 除此之外， LView 还存储了除此之外的所有渲染模板需要的信息

LView 的设计，可以为每个视图保留单独的状态以方便视图的插入/删除，因此我们不必根据存在的视图来编译数据数组。

# Angular 中的视图数据

在 Angular Ivy 中，使用了 LView 和 TView.data 来管理和跟踪渲染模板所需要的内部数据。

LView 和 TView.data 都是数组，它们的索引指向相同的项目，它们的数据视图布局如下：

存储实例

存储令牌

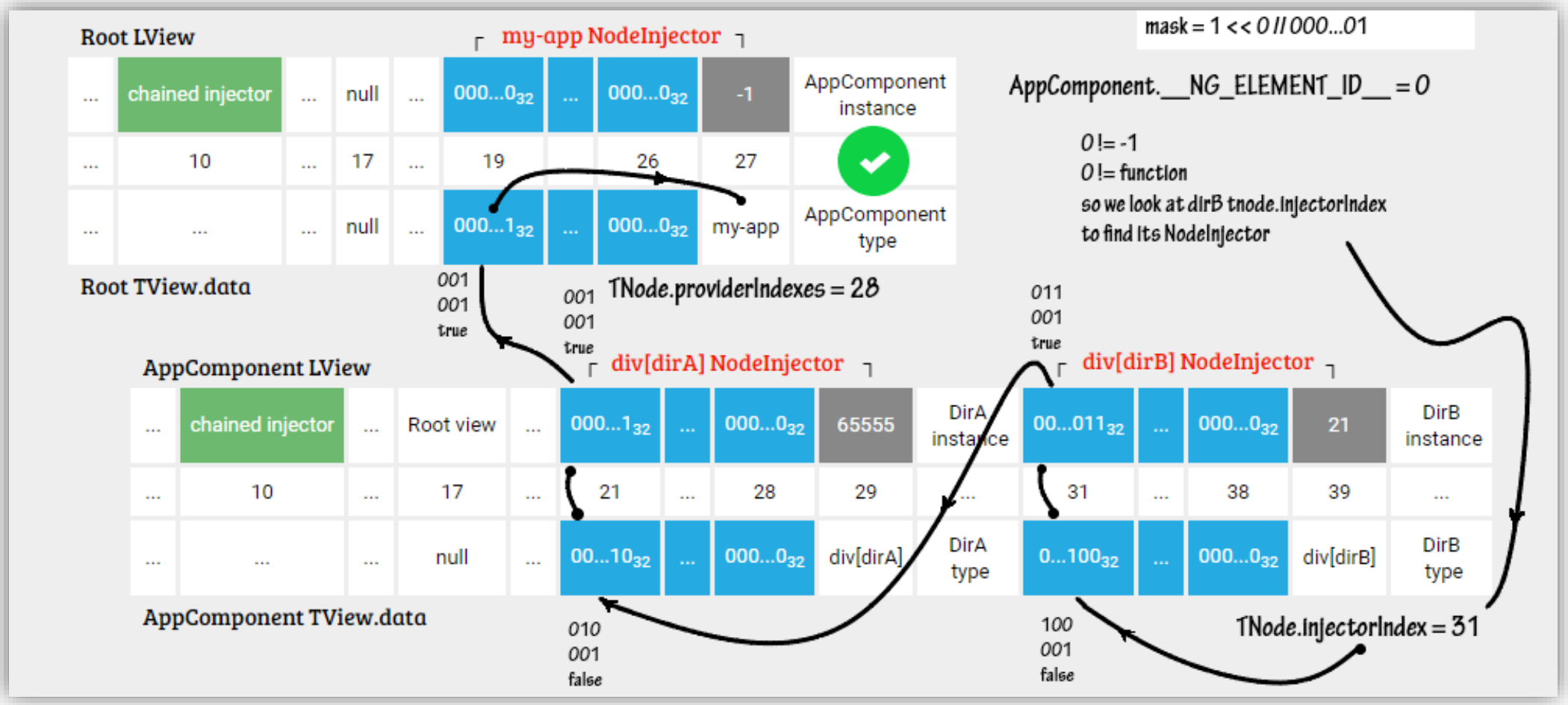
| Section | `LView` | `TView.data` |
| --- | --- | --- |
| `HEADER` | contextual data | mostly `null` |
| `DECLS` | DOM, pipe, and local ref instances | |
| `VARS` | binding values | property names |
| `EXPANDO` | host bindings; directive instances; providers; dynamic nodes | host prop names; directive tokens; provider tokens; `null` |

Ivy 将所有在 TView.data 中，将存储在 LView 中，因此我们可以检索查看该视图的所有注入器。

# Angular 中的视图数据查询

Ivy 使用 LView 和 TView.data 数组来存储视图数据，其中便包括了节点的注入信息。

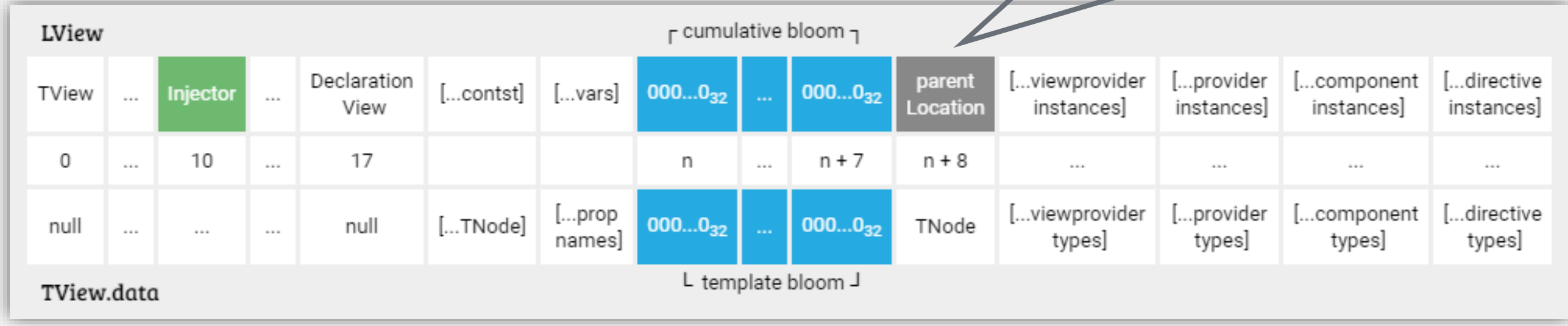
这意味着，NodeInjector 需要从 LView 和 TView.data 数组中得到具体的视图数据信息。



# Angular 中的布隆过滤器

布隆过滤器用于检查给定的 ID 是否在集合中

每个布隆过滤器在“压缩的”parentLocation 插槽（n + 8 索引）中都有一个指向父布隆过滤器的指针



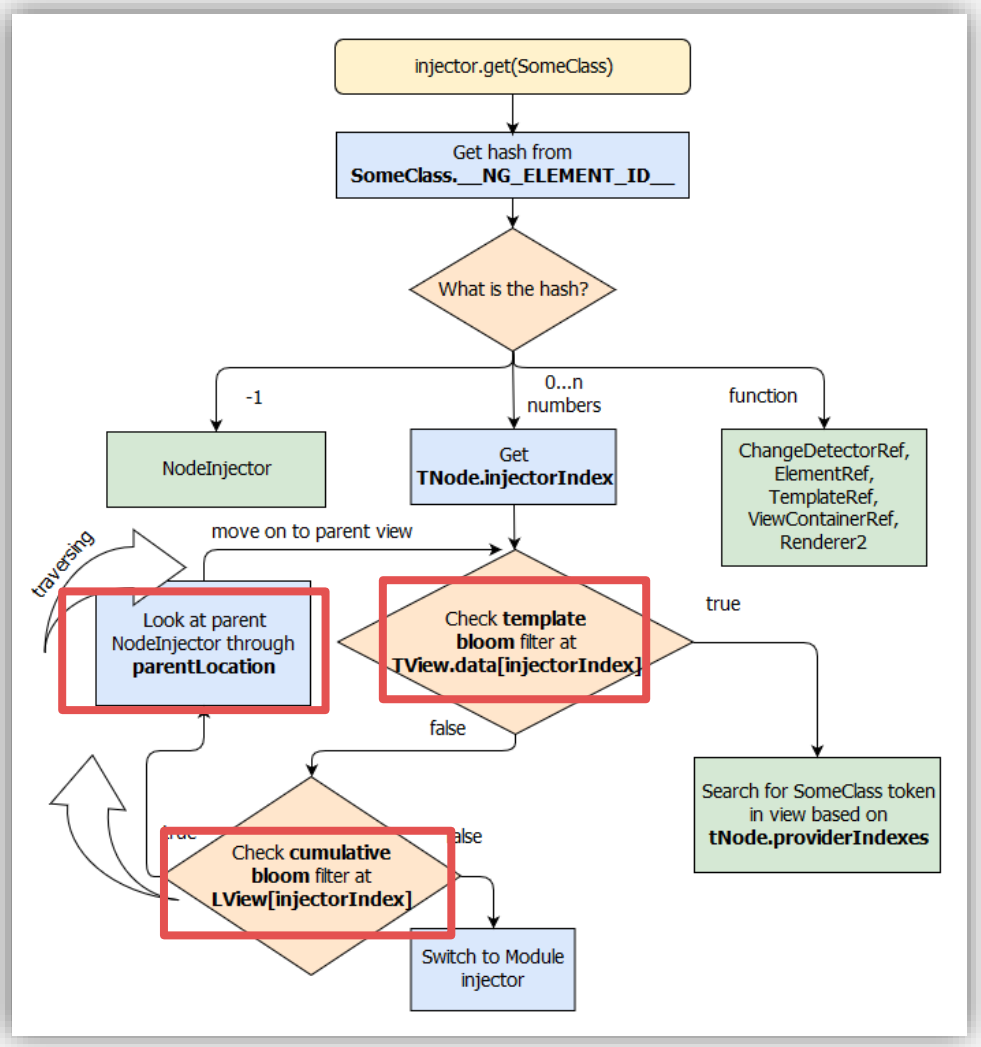

布隆过滤器存储在 LView/TView.data 布局中的 EXPANDO 部分

长度为8个时隙（[n，n + 7]索引）

1. 模板布隆过滤器：用于保留有关当前节点令牌的信息，并且可以在 TView 中共享的过滤器。
2. 累积布隆过滤器：存储有关当前节点的令牌，以及其祖先节点的令牌的信息的过滤器。

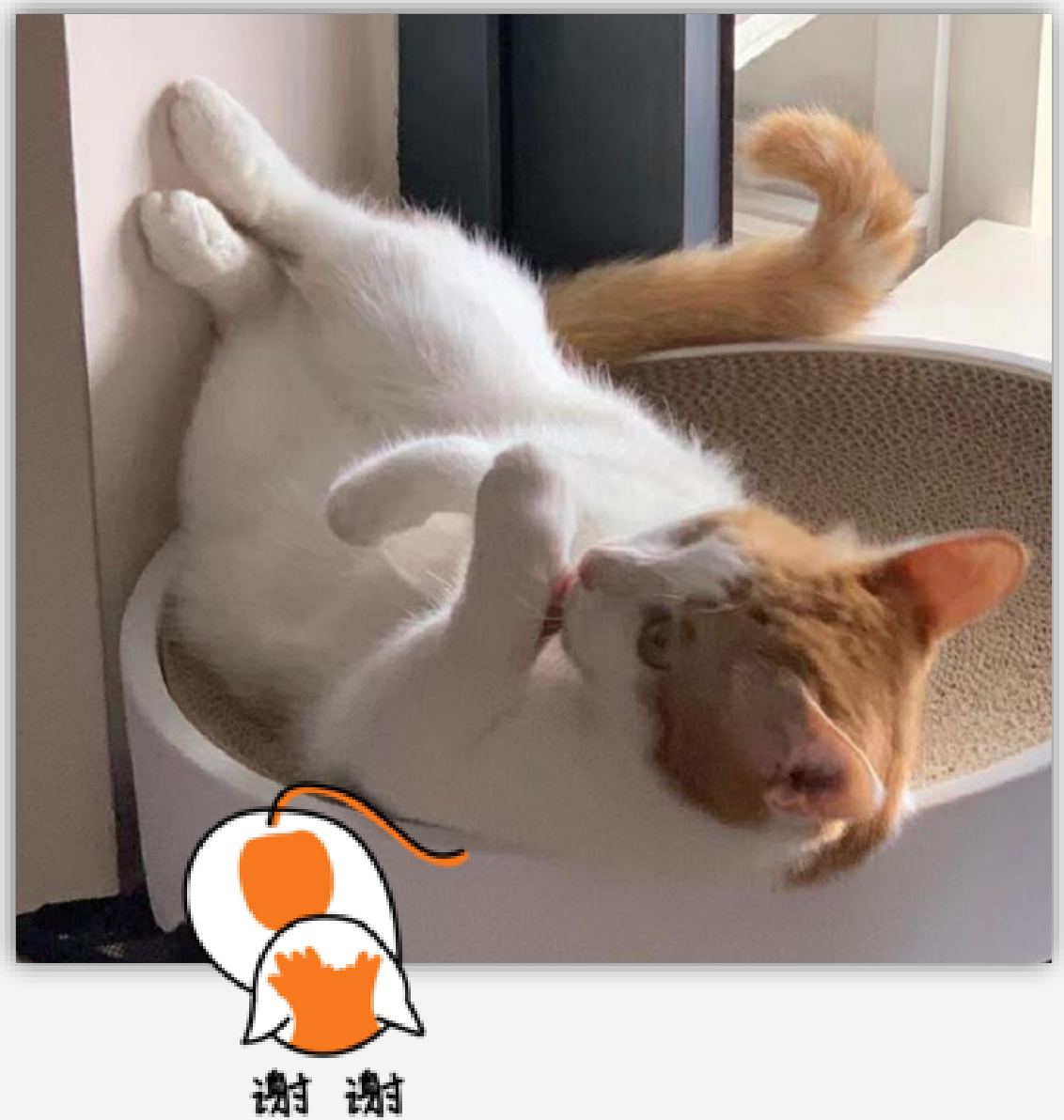
# Angular 中的视图数据查询

## 在 Ivy 中，使用 NodeInjector 来解析依赖关系的过程



# Angular 框架源码分析和解读


- https://github.com/godbasin/godbasin.github.io
- https://github.com/godbasin/front-end-playground




Github: godbasin
@被删