前端领域中的算法（上）

@被删

面试造火箭
入职拧螺丝

1

前端常见的算法问题

排序算法

- 快速排序
- 冒泡排序
- 插入排序
- 归并排序
- 堆排序
- 希尔排序
- 选择排序
- 计数排序
- 桶排序
- 基数排序
- ...

JavaScript 中的 Array.prototype.sort() 用的哪种排序算法?

Array.prototype.sort()

数组长度小于 10 用插入排序，否则用快速排序？？

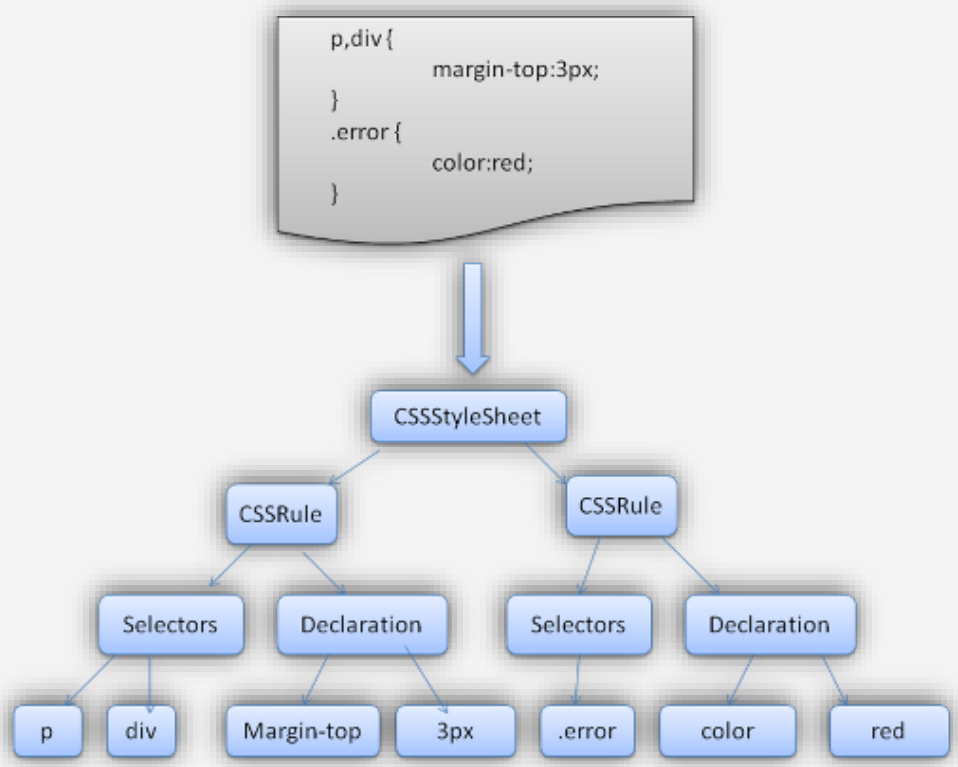
- V8 原本是使用快速排序和插入排序来处理，目前改用 Timsort
- Timsort 是一种混合、稳定的排序算法，从归并排序和插入排序派生而来的
- Mergesort 通常以递归方式工作，而 Timsort 则迭代地工作

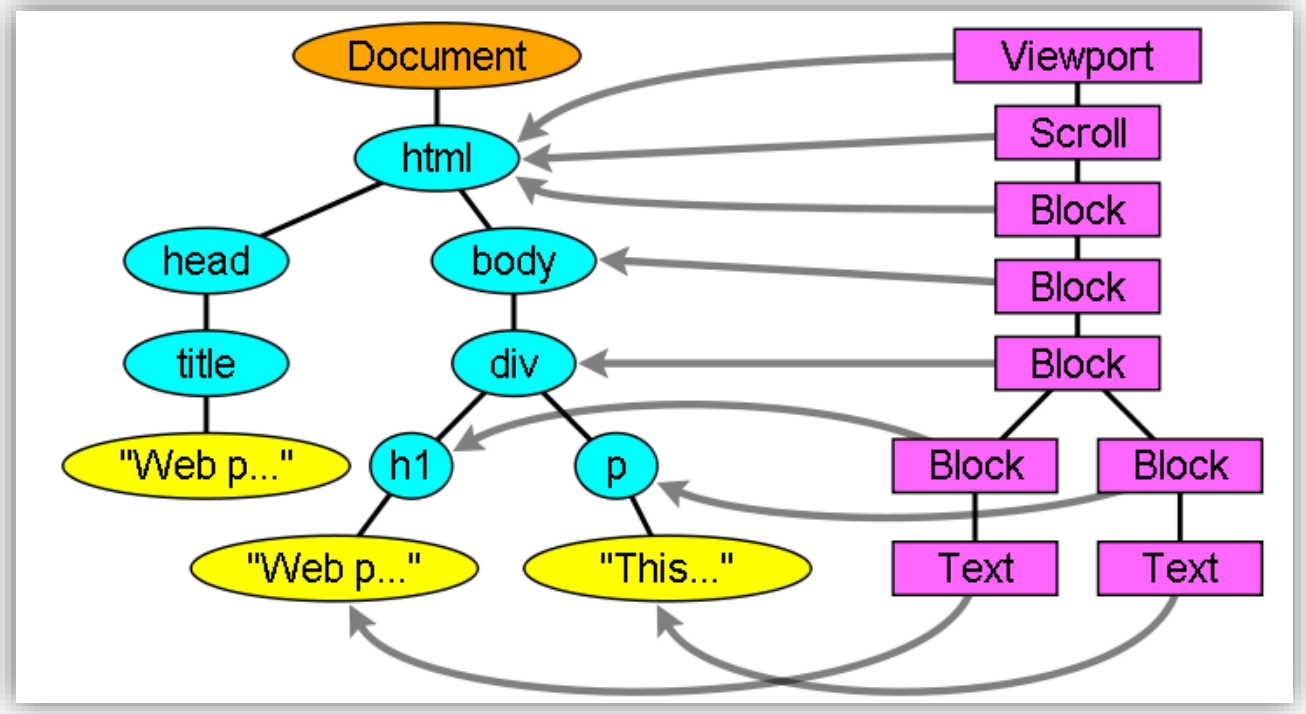
从 2.3 版本起，Timsort 一直是 Python 的标准排序算法。它还被 Java SE7[4], Android platform[5], GNU Octave,[6] 谷歌浏览器,[7] 和 Swift[8] 用于对非原始类型的数组排序。

https://v8.dev/blog/array-sort#timsort

有没有一些实用点的算法和数据结构

浏览器渲染机制

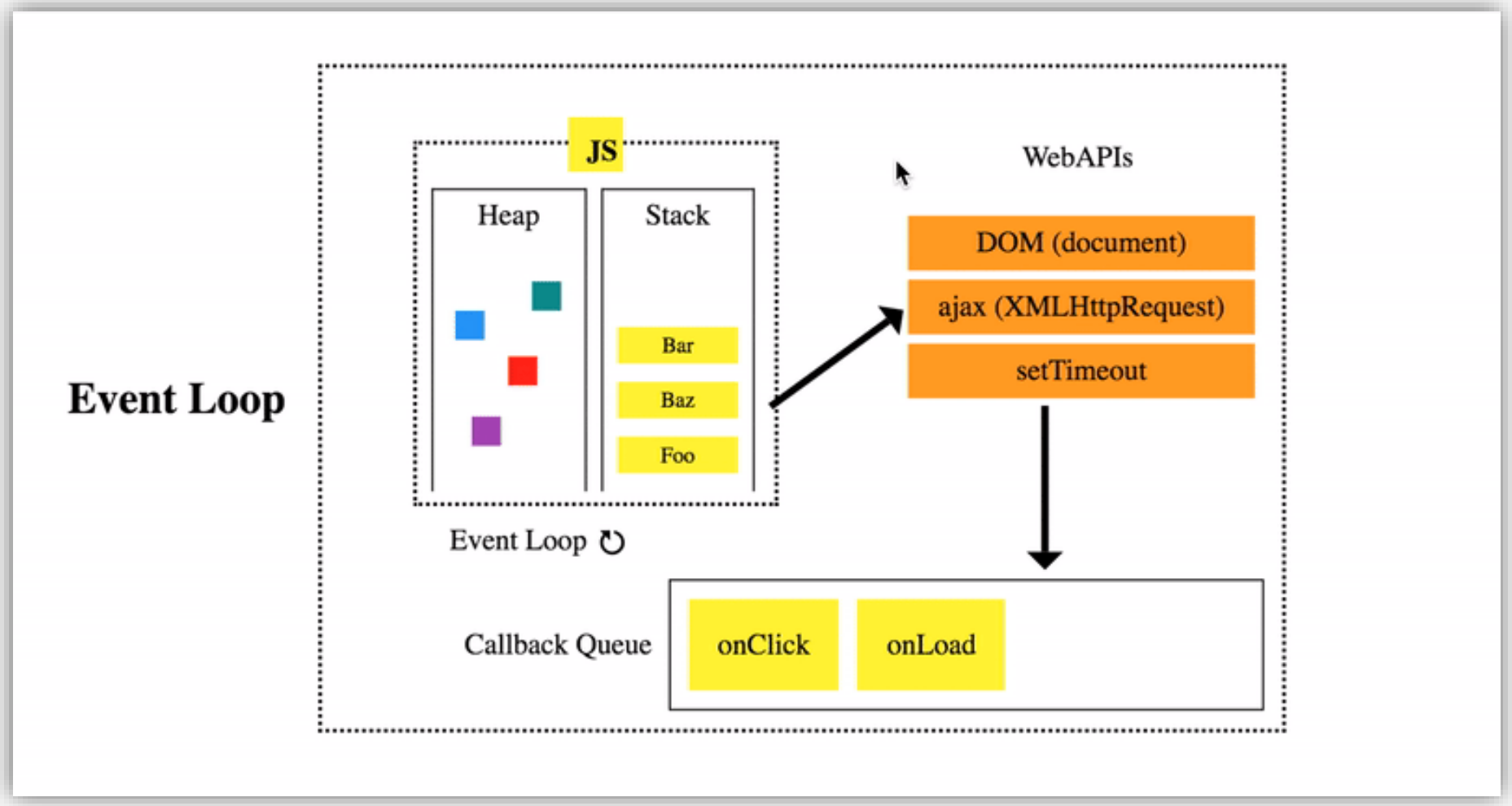




《How Browsers Work: Behind the scenes of modern web browsers》

《Inside look at modern web browser》

浏览器运行机制


Event Loop
JS
Heap
Stack
Bar
Baz
Foo
Event Loop
WebAPIs
DOM (document)
ajax (XMLHttpRequest)
setTimeout
Callback Queue
onClick
onLoad

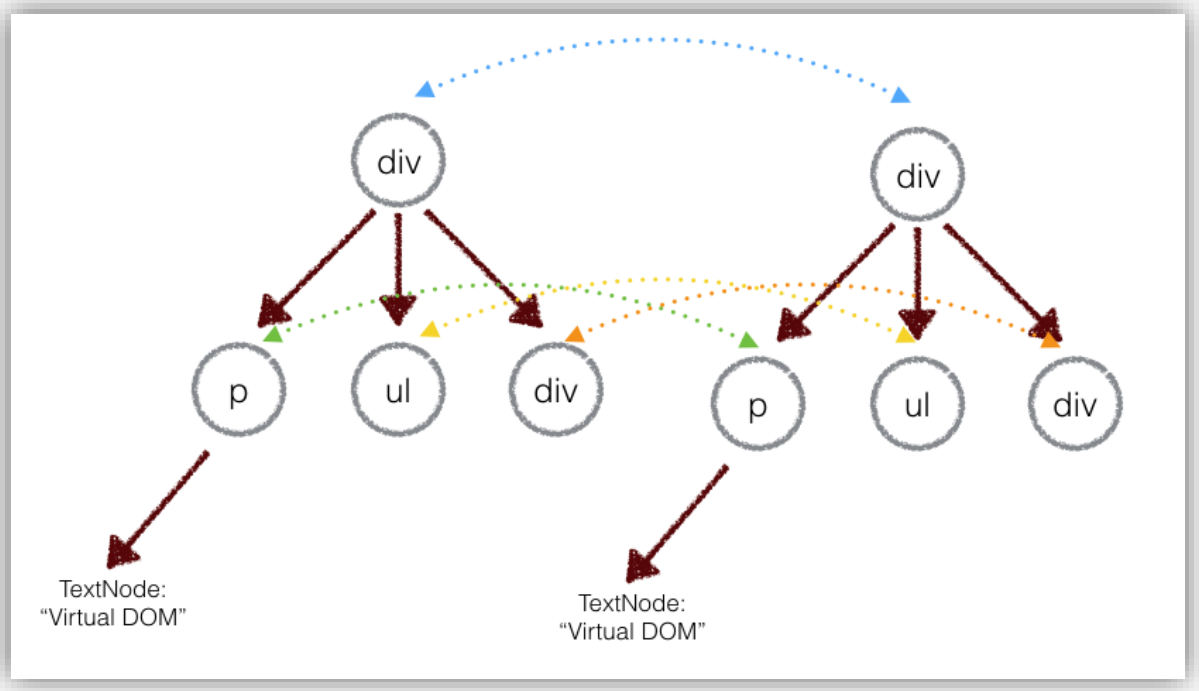
2

前端开发中的算法和数据结构

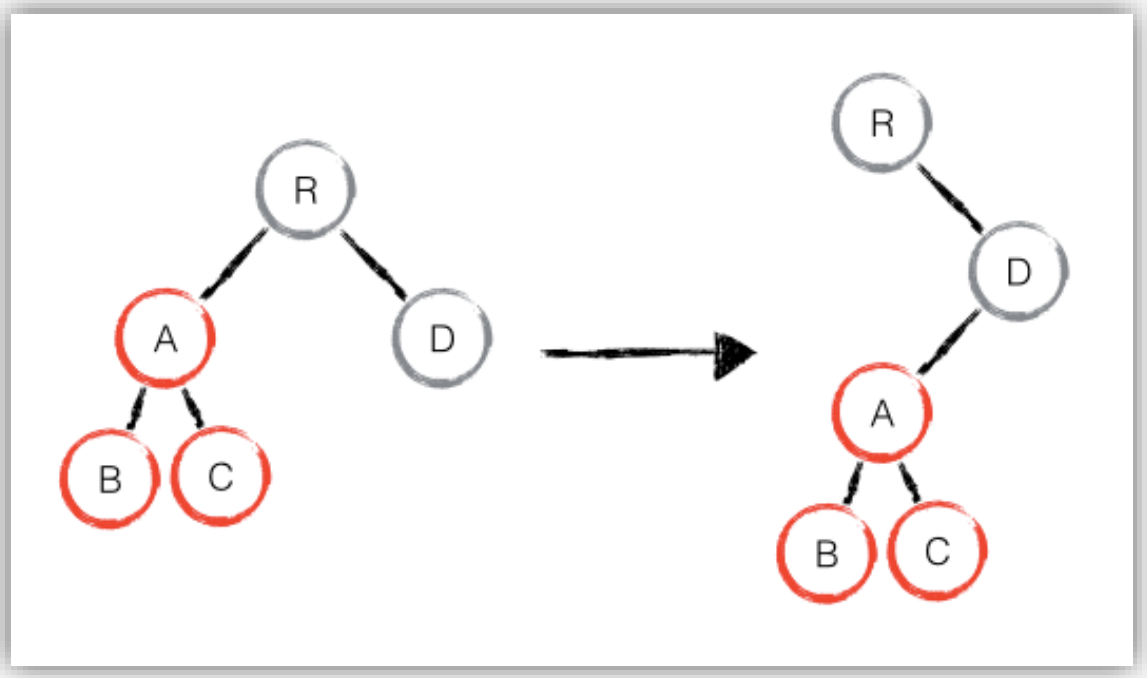
虚拟 DOM

1. 用 JavaScript 对象描述 DOM 树结构，然后用它来构建真正的 DOM 树插入文档
2. 当状态发生改变之后，重新构造新的 JavaScript 对象结构，新的和旧的作对比，得出差异
3. 针对差异之处，进行视图更新

React DOM Diff



tree diff：同层级 DOM 节点对比
O(n^3) -> O(n)

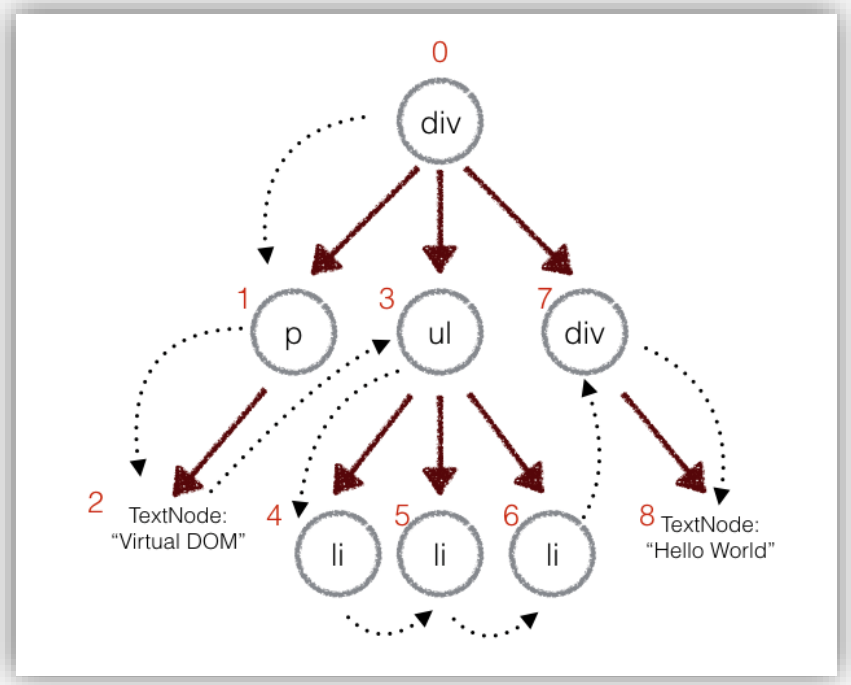


tree diff：对于不同层级的节点，只有创建和除删操作

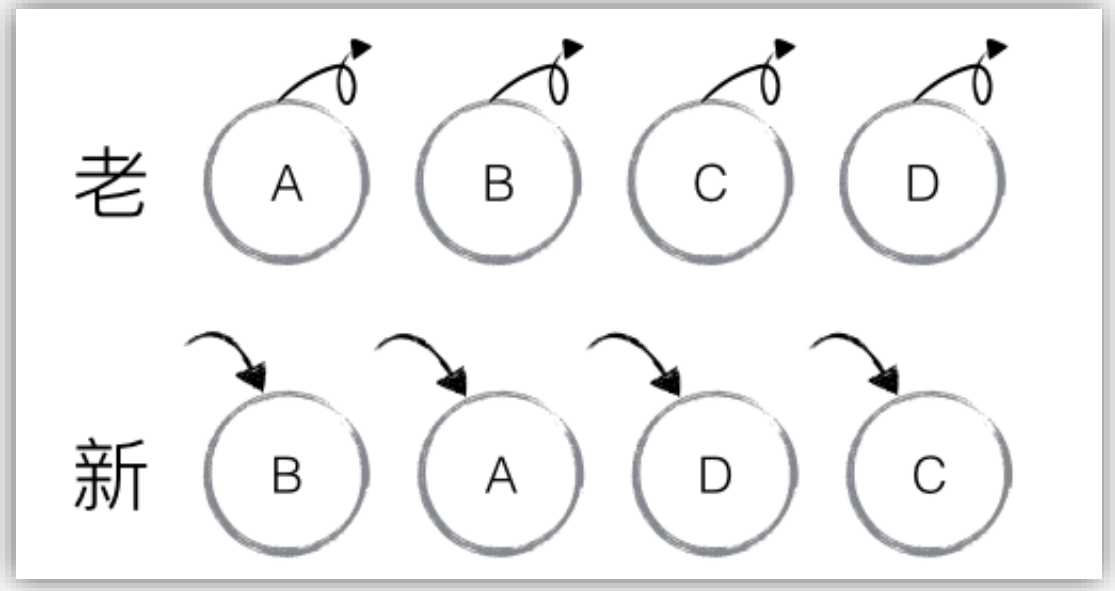
- 保持稳定的 DOM 结构会有助于性能的提升

图片来源：https://github.com/livoras/blog/issues/13

React DOM Diff



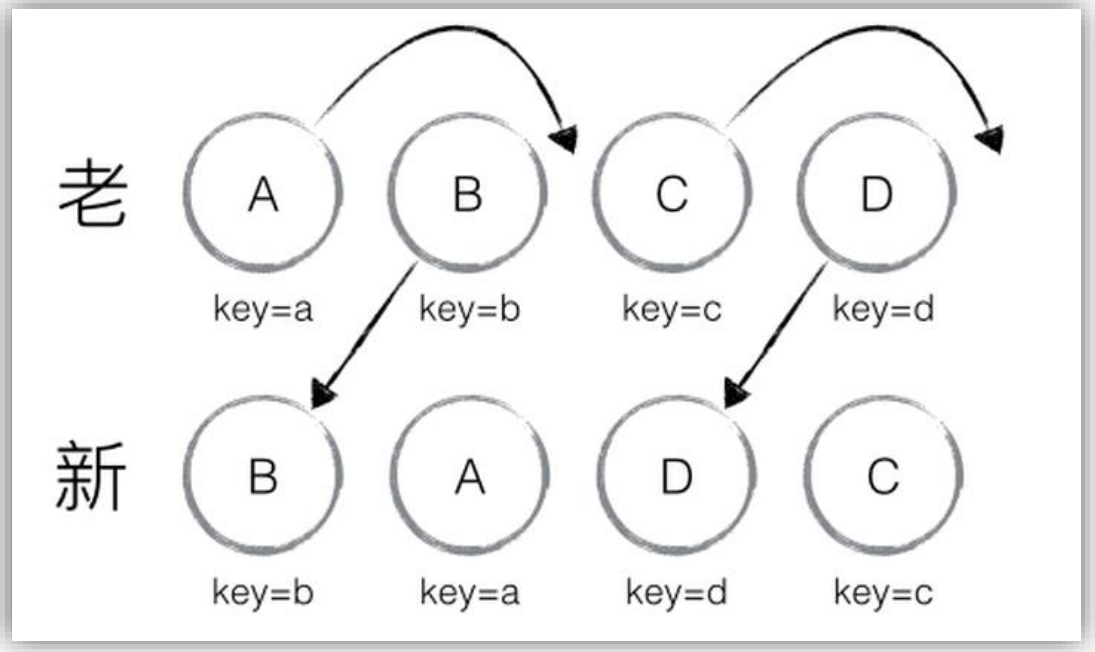
element diff：深度遍历

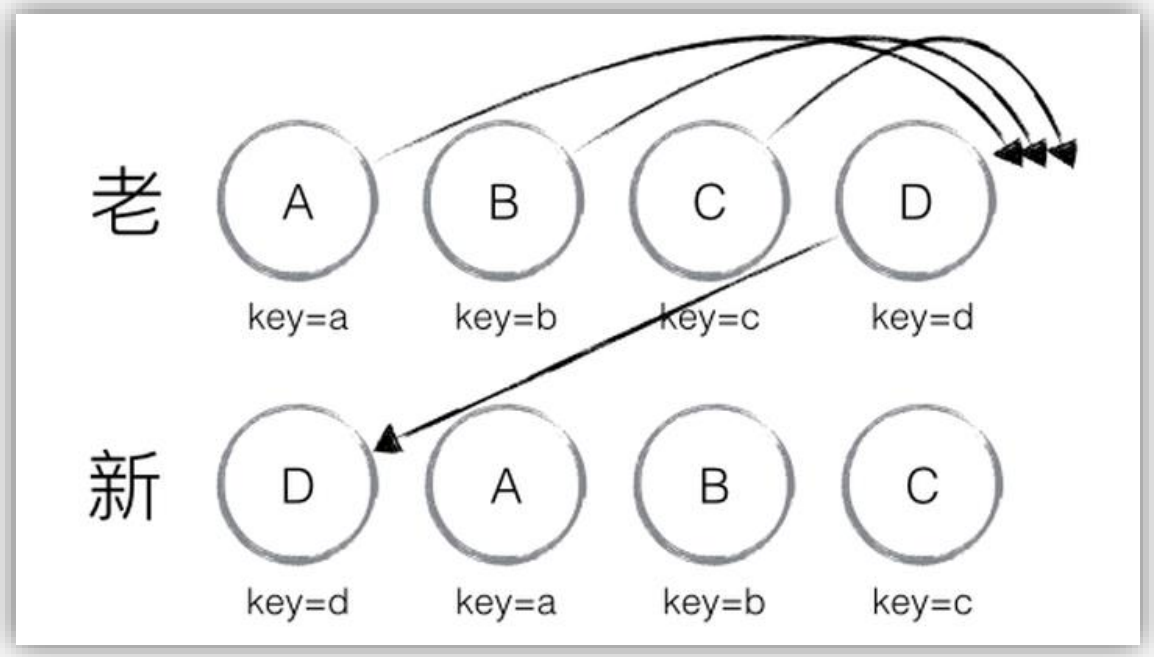


图片来源：https://github.com/livoras/blog/issues/13

React DOM Diff

通过设置唯一 key 的策略，对 element diff 进行算法优化





- 尽量减少类似将最后一个节点移动到列表首部的操作

图片来源：https://zhuanlan.zhihu.com/p/20346379

React DOM Diff

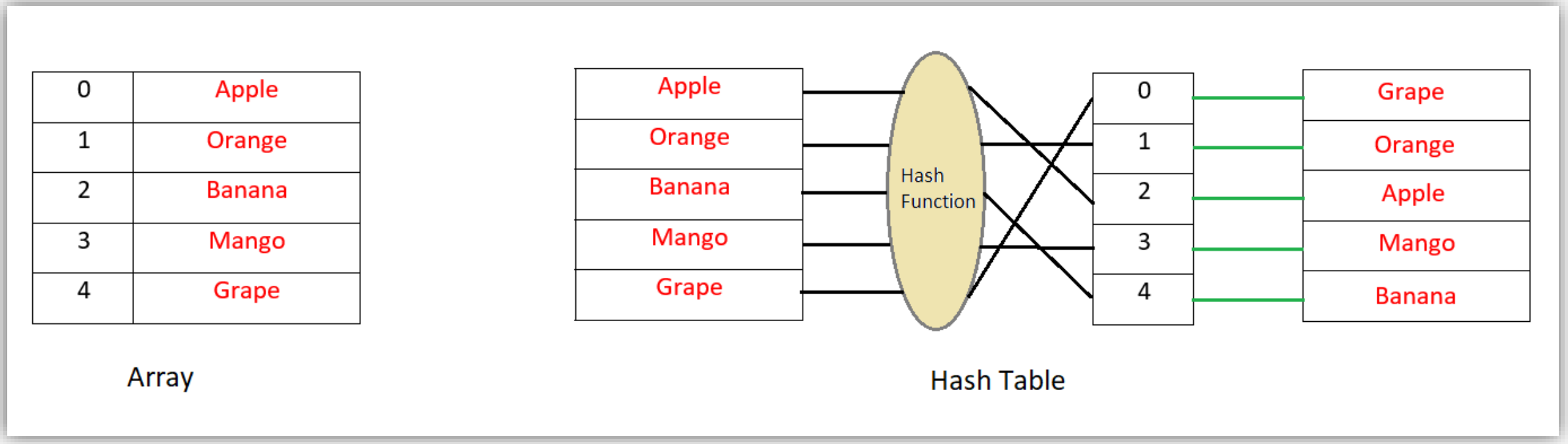
component diff

- 同一类型组件：会进行 Virtual DOM 进行比较
- 不同类型组件：会替换整个组件下的子节点
- 提供了一个 shouldComponentUpdate，决定是否更新

图片来源：https://zhuanlan.zhihu.com/p/20346379

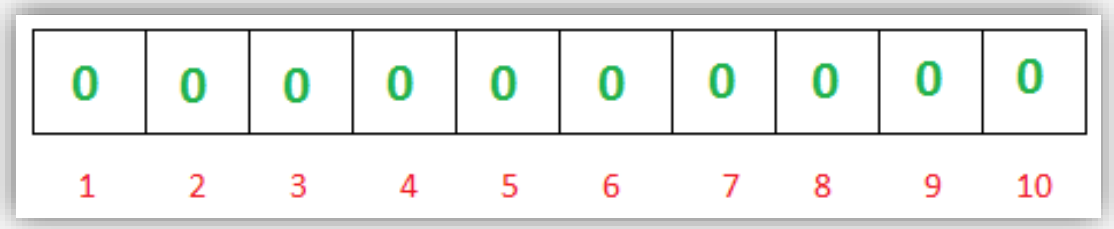
Angular 中的布隆过滤器

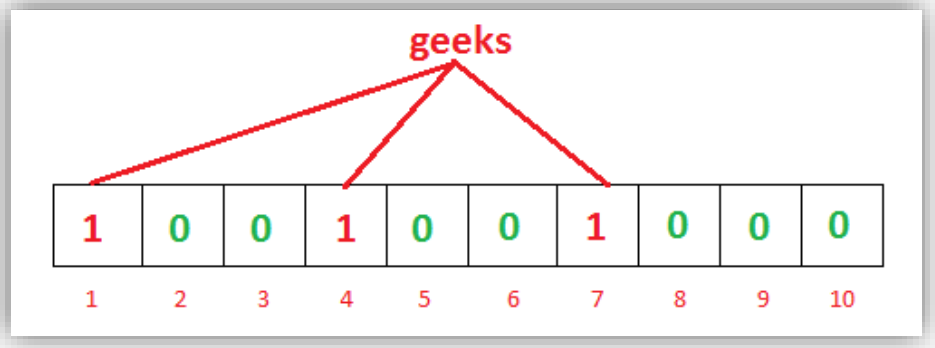
哈希表的搜索成本

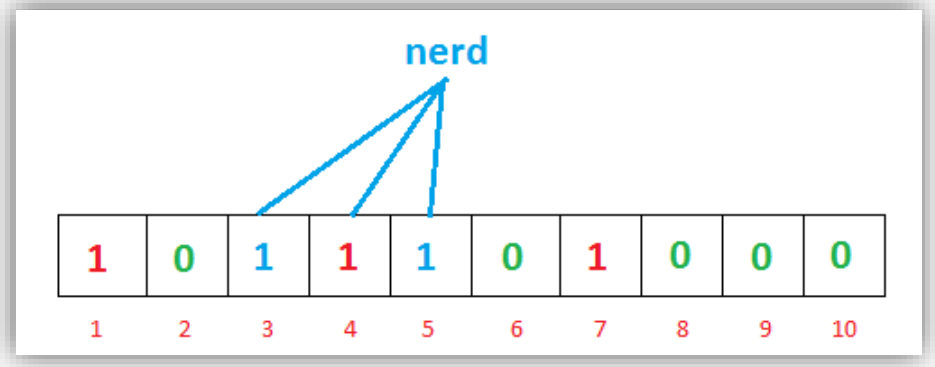


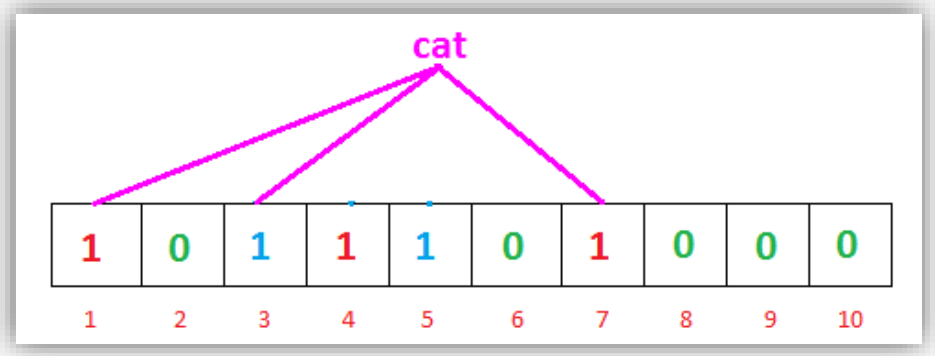
匹配可能很快，但是在磁盘上或在远程服务器上，通过网络搜索它的成本将使其变慢

布隆过滤器









- 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
- 可以检查值是“可能在集合中”，还是“绝对不在集合中”

使用布隆过滤器的经典示例：
减少对不存在的密钥的昂贵磁盘（或网络）查找，比如弱密码的查询

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

- LView数组：包含描述特定模板的数据
- TView.data：包含在模板之间共享的信息

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`

LView

TView	...	Injector	...	Declaration View	[...contst]	[...vars]	$000...0_{32}$	...	$000...0_{32}$	parent Location	[...viewprovider instances]	[...provider instances]	[...component instances]	[...directive instances]
0	...	10	...	17			n	...	n + 7	n + 8	...	...	...	...
null	...	...	...	null	[...TNode]	[...prop names]	$000...0_{32}$	...	$000...0_{32}$	TNode	[...viewprovider types]	[...provider types]	[...component types]	[...directive types]

cumulative bloom

template bloom

TView.data

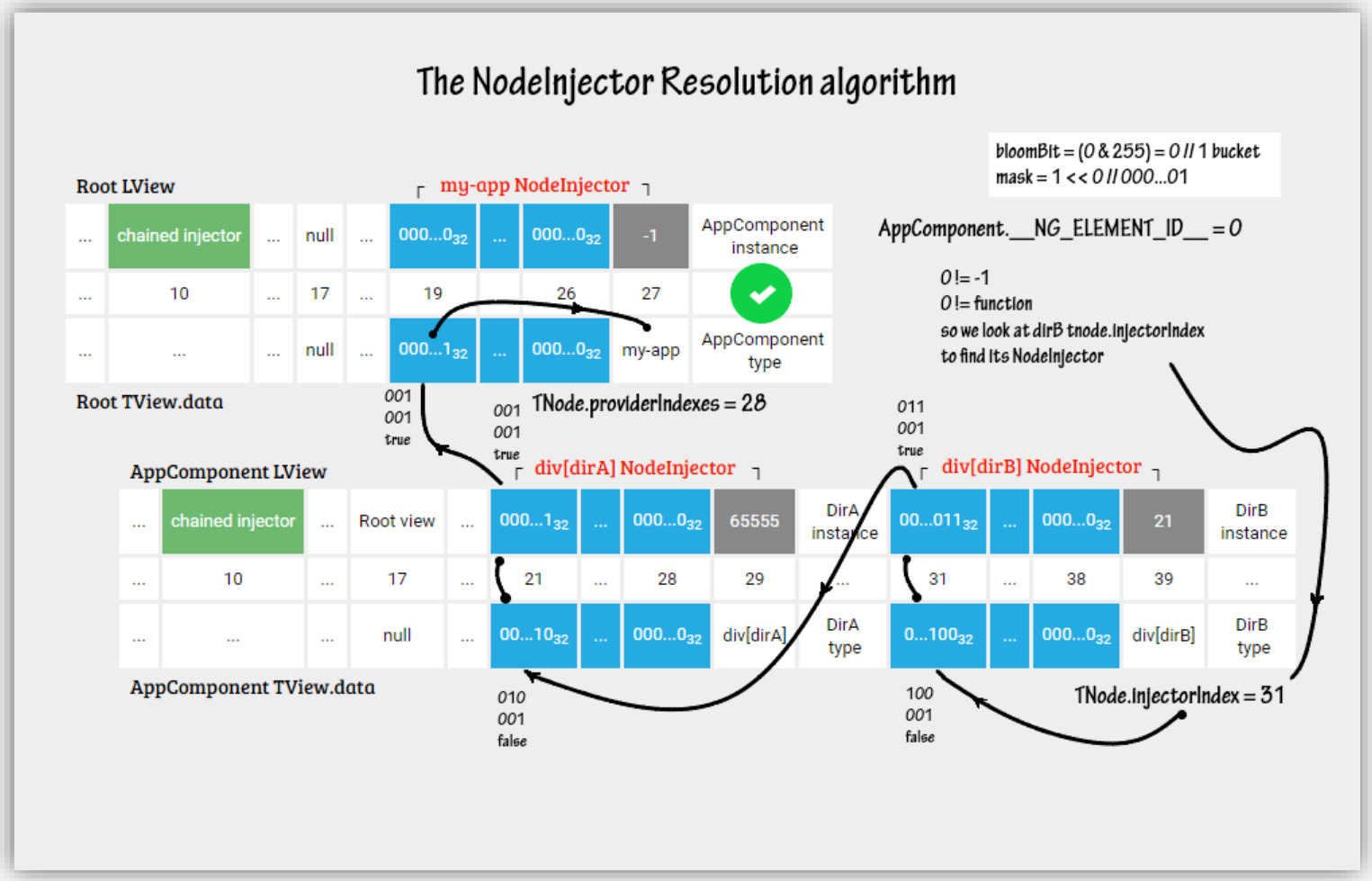
布隆过滤器，长度为8个时隙（[n，n + 7]索引）

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 中的依赖查询

Angular 通过递增整数值为令牌生成（如果尚未定义）令牌的唯一 ID，并将其置于静态__NG_ELEMENT_ID__属性中

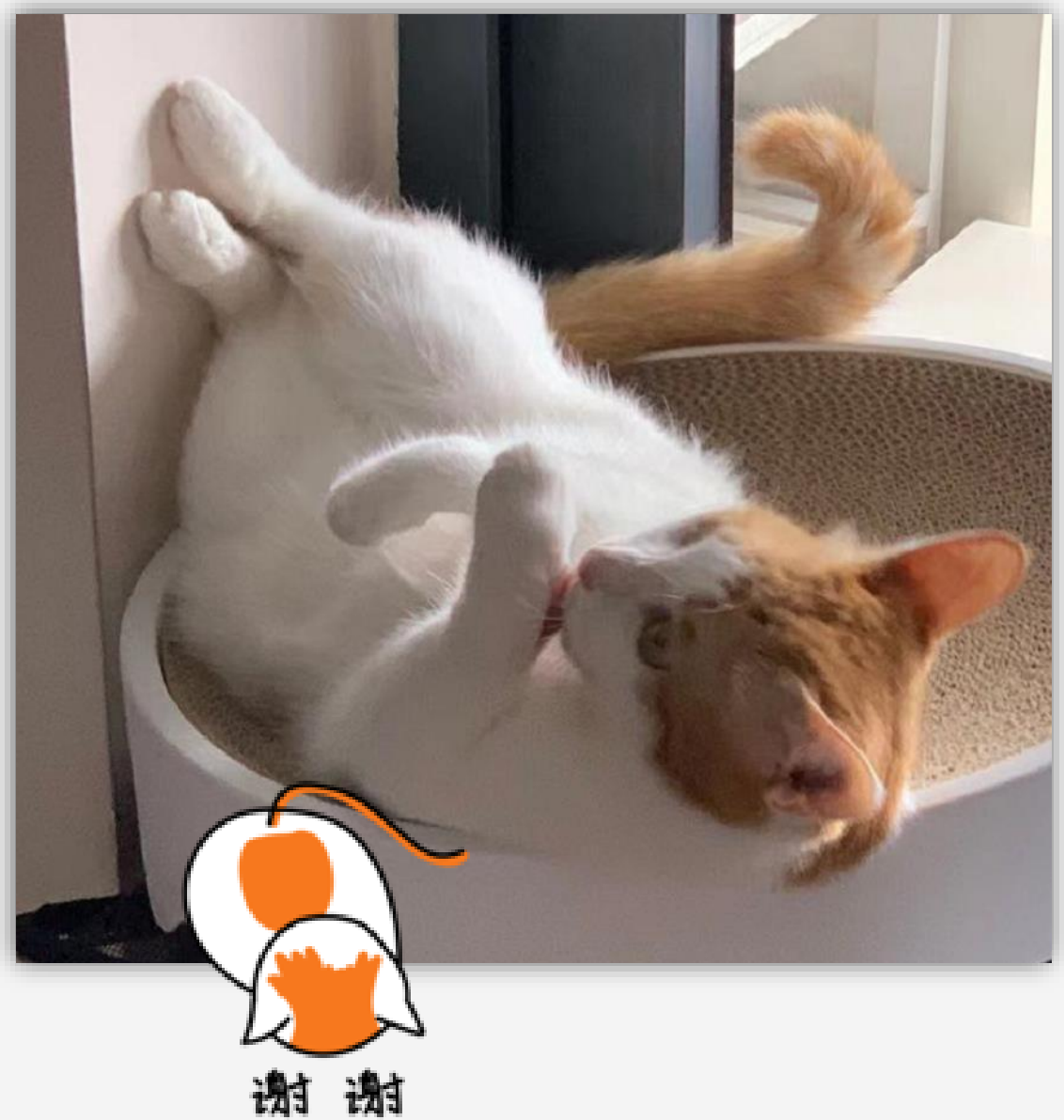
1. 在 SomeClass.__NG_ELEMENT_ID__ 静态属性中查找哈希。
2. 如果该哈希是工厂函数，则还有另一种特殊情况，即应通过调用该函数来初始化对象。
3. 如果该哈希等于-1，则是一种特殊情况，我们将获得`NodeInjector`实例。
4. 如果该哈希是一个数字，那么我们会从`TNode`获取`injectorIndex`。
5. 查看模板布隆过滤器（TView.data [injectorIndex]），如果为真，那么我们将搜索 SomeClass 令牌。
6. 如果模板布隆过滤器返回错误，那么会查看一下累积布隆过滤器。
7. 如果它为真，则继续进行遍历，否则将切换到ModuleInjector。



布隆过滤器用于检查给定的 ID 是否在集合中

除了前端框架/工具库，前端业务/应用开发中？？

敬请期待下集



Github: godbasin
@被删