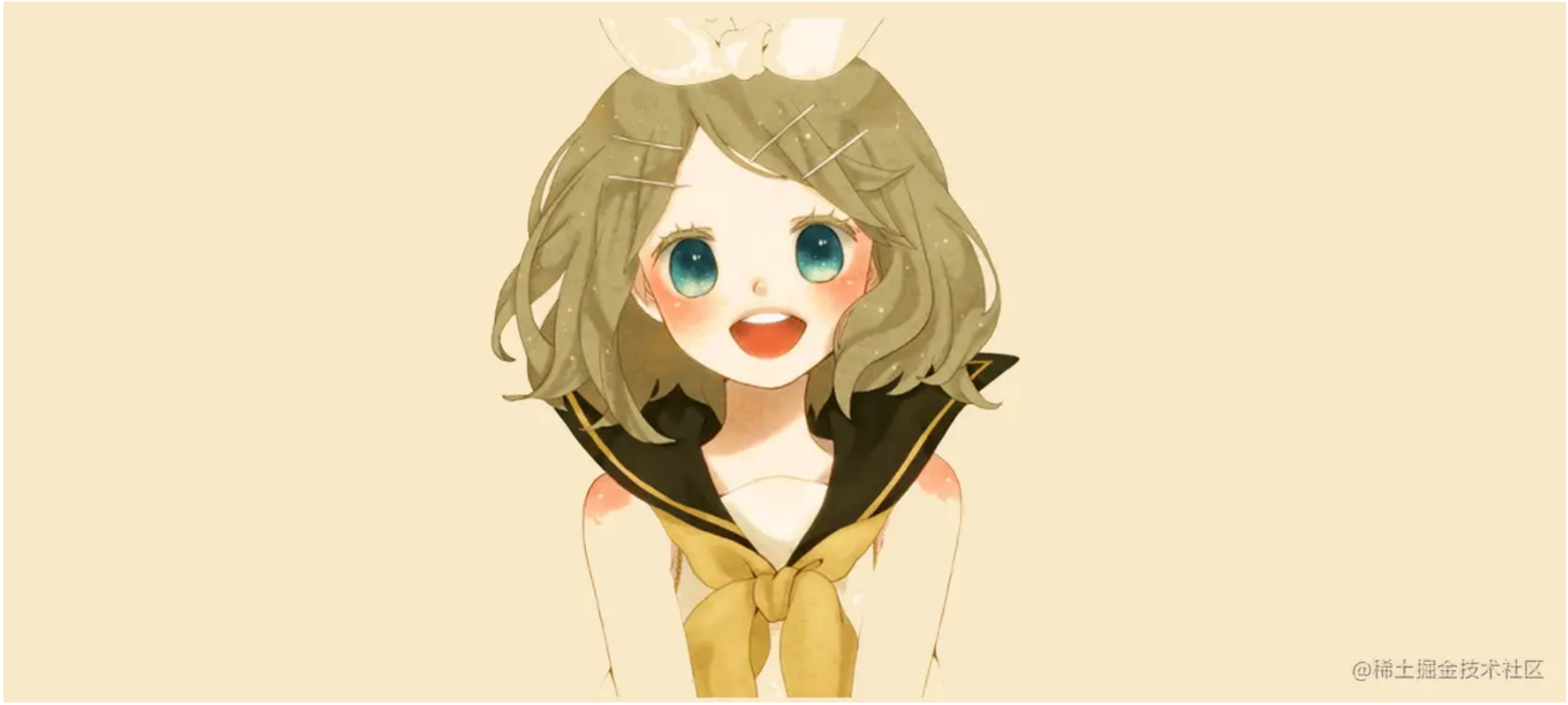
前端进阶算法3：从浏览器缓存淘汰策略和Vue的keep-alive学习LRU算法

前端瓶子君 Lv5

2020年04月07日 00:54 · 阅读 6718



引言

这个标题已经很明显的告诉我们：前端需要了解 LRU 算法！

这也是前端技能的亮点，当面试官在问到你前端开发中遇到过哪些算法，你也可以把这部分丢过去！

本节按以下步骤切入：

- 由浏览器缓存策略引出 LRU 算法原理
- 然后走进 `vue` 中 `keep-alive` 的应用
- 接着，透过 `vue` 中 `keep-alive` 源码看 `LRU` 算法的实现
- 最后，来一道 leetcode 题目，我们来实现一个 LRU 算法

按这个步骤来，完全掌握 LRU 算法，点亮前端技能，下面就开始吧👇

一、LRU 缓存淘汰策略

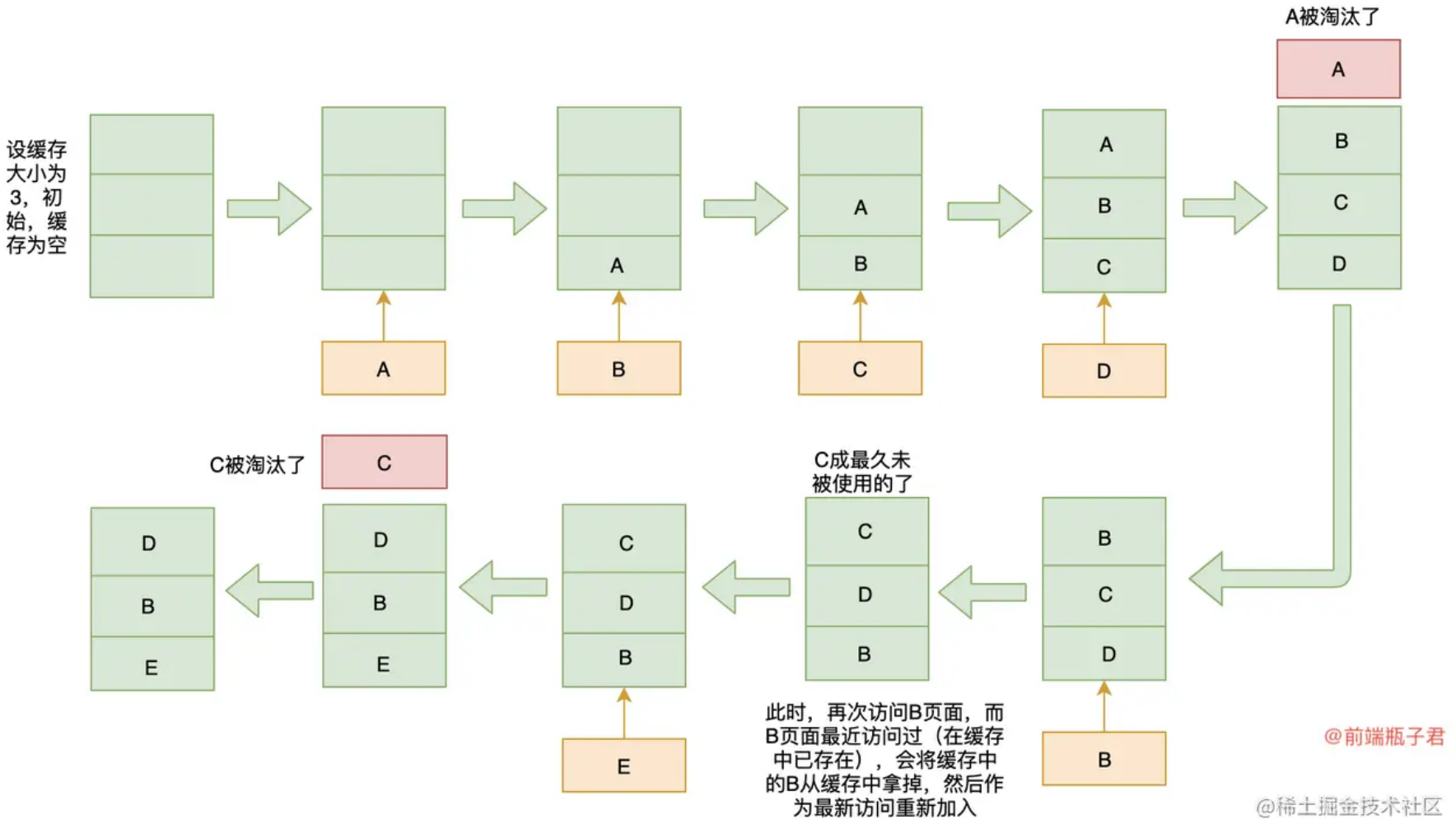
缓存在计算机网络上随处可见，例如：当我们首次访问一个网页时，打开很慢，但当我们再次打开这个网页时，打开就很快。

这就涉及缓存在浏览器上的应用：**浏览器缓存**。当我们打开一个网页时，例如 `https://github.com/sisterAn/JavaScript-Algorithms`，它会在发起真正的网络请求前，查询浏览器缓存，看是否有要请求的文件，如果有，浏览器将会拦截请求，返回缓存文件，并直接结束请求，不会再去服务器上下载。如果不存在，才会去服务器请求。

其实，浏览器中的缓存是一种在本地保存资源副本，它的大小是有限的，当我们请求数过多时，缓存空间会被用满，此时，继续进行网络请求就需要确定缓存中哪些数据被保留，哪些数据被移除，这就是**浏览器缓存淘汰策略**，最常见的淘汰策略有 FIFO（先进先出）、LFU（最少使用）、LRU（最近最少使用）。

LRU （ `Least Recently Used` ：最近最少使用 ）缓存淘汰策略，故名思义，就是根据数据的历史访问记录来进行淘汰数据，其核心思想是**如果数据最近被访问过，那么将来被访问的几率也更高**，优先淘汰最近没有被访问到的数据。

画个图帮助我们理解：



二、LRU 在 keep-alive (Vue) 上的实现

keep-alive 在 vue 中用于实现组件的缓存，当组件切换时不会对当前组件进行卸载。

复制代码

```
<!-- 基本 -->
<keep-alive>
  <component :is="view"></component>
</keep-alive>
```

最常用的两个属性：`include` 、 `exculde` ，用于组件进行有条件的缓存，可以用逗号分隔字符串、正则表达式或一个数组来表示。

在 2.5.0 版本中，`keep-alive` 新增了 `max` 属性，用于最多可以缓存多少组件实例，一旦这个数字达到了，在新实例被创建之前，已缓存组件中最久没有被访问的实例会被销毁掉，**看，这里就应用了 LRU 算法**。即在 `keep-alive` 中缓存达到 `max` ，新增缓存实例会优先淘汰最近没有被访问到的实例🎉🎉🎉

下面我们透过 vue 源码看一下具体的实现👇

2. 从 vue 源码看 keep-alive 的实现

复制代码

```
export default {
  name: "keep-alive",
  // 抽象组件属性 ,它在组件实例建立父子关系的时候会被忽略,发生在 initLifecycle 的过程中
  abstract: true,
  props: {
    // 被缓存组件
    include: patternTypes,
    // 不被缓存组件
```

```
    // 指定缓存大小
    max: [String, Number]
  },
  created() {
    // 初始化用于存储缓存的 cache 对象
    this.cache = Object.create(null);
    // 初始化用于存储VNode key值的 keys 数组
    this.keys = [];
  },
  destroyed() {
    for (const key in this.cache) {
      // 删除所有缓存
      pruneCacheEntry(this.cache, key, this.keys);
    }
  },
  mounted() {
    // 监听缓存（include）/不缓存（exclude）组件的变化
    // 在变化时，重新调整 cache
    // pruneCache：遍历 cache，如果缓存的节点名称与传入的规则没有匹配上的话，就把这个节点从缓存中移除
    this.$watch("include", val => {
      pruneCache(this, name => matches(val, name));
    });
    this.$watch("exclude", val => {
      pruneCache(this, name => !matches(val, name));
    });
  },
  render() {
    // 获取第一个子元素的 vnode
    const slot = this.$slots.default;
    const vnode: VNode = getFirstComponentChild(slot);
```

```
    if (componentOptions) {
      // name 不在 inlcude 中或者在 exlude 中则直接返回 vnode，否则继续进行下一步
      // check pattern
      const name: ?string = getComponentName(componentOptions);
      const { include, exclude } = this;
      if (
        // not included
        (include && (!name || !matches(include, name))) ||
        // excluded
        (exclude && name && matches(exclude, name))
      ) {
        return vnode;
      }

      const { cache, keys } = this;
      // 获取键，优先获取组件的 name 字段，否则是组件的 tag
      const key: ?string =
        vnode.key == null
          ? // same constructor may get registered as different local components
            // so cid alone is not enough (#3269)
            componentOptions.Ctor.cid +
            (componentOptions.tag ? `::${componentOptions.tag}` : "")
          : vnode.key;

      // ------------------------------------------------
      // 下面就是 LRU 算法了，
      // 如果在缓存里有则调整，
      // 没有则放入（长度超过 max，则淘汰最近没有访问的）
      // ------------------------------------------------
      // 如果命中缓存，则从缓存中获取 vnode 的组件实例，并且调整 key 的顺序放入 keys 数组的末尾
```

```
        // make current key freshest
        remove(keys, key);
        keys.push(key);
      }
      // 如果没有命中缓存,就把 vnode 放进缓存
      else {
        cache[key] = vnode;
        keys.push(key);
        // prune oldest entry
        // 如果配置了 max 并且缓存的长度超过了 this.max，还要从缓存中删除第一个
        if (this.max && keys.length > parseInt(this.max)) {
          pruneCacheEntry(cache, keys[0], keys, this._vnode);
        }
      }

      // keepAlive标记位
      vnode.data.keepAlive = true;
    }
    return vnode || (slot && slot[0]);
  }
};

// 移除 key 缓存
function pruneCacheEntry (
  cache: VNodeCache,
  key: string,
  keys: Array<string>,
  current?: VNode
) {
  const cached = cache[key]
```

```
  }
  cache[key] = null
  remove(keys, key)
}

// remove 方法（shared/util.js）
/**
 * Remove an item from an array.
 */
export function remove (arr: Array<any>, item: any): Array<any> | void {
  if (arr.length) {
    const index = arr.indexOf(item)
    if (index > -1) {
      return arr.splice(index, 1)
    }
  }
}
```

keep-alive源码路径

在 `keep-alive` 缓存超过 `max` 时，使用的缓存淘汰算法就是 LRU 算法，它在实现的过程中用到了 `cache` 对象用于保存缓存的组件实例及 `key` 值， `keys` 数组用于保存缓存组件的 `key` ，当 `keep-alive` 中渲染一个需要缓存的实例时：

- 判断缓存中是否已缓存了该实例，缓存了则直接获取，并调整 `key` 在 `keys` 中的位置（移除 `keys` 中 `key` ，并放入 `keys` 数组的最后一位）
- 如果没有缓存，则缓存该实例，若 `keys` 的长度大于 `max` （缓存长度超过上限），则移除 `keys[0]` 缓存

三、leetcode：LRU 缓存机制

运用你所掌握的数据结构，设计和实现一个 LRU (最近最少使用) 缓存机制。它应该支持以下操作： 获取数据 `get` 和写入数据 `put` 。

获取数据 `get(key)` - 如果密钥 (`key`) 存在于缓存中，则获取密钥的值（总是正数），否则返回 `-1` 。 写入数据 `put(key, value)` - 如果密钥不存在，则写入数据。当缓存容量达到上限时，它应该在写入新数据之前删除最久未使用的数据，从而为新数据留出空间。

进阶:

你是否可以在 **O(1)** 时间复杂度内完成这两种操作?

示例:

复制代码

```
LRUCache cache = new LRUCache( 2 /* 缓存容量 */ );

cache.put(1, 1);
cache.put(2, 2);
cache.get(1);       // 返回  1
cache.put(3, 3);    // 该操作会使得密钥 2 作废
cache.get(2);       // 返回 -1 (未找到)
cache.put(4, 4);    // 该操作会使得密钥 1 作废
cache.get(1);       // 返回 -1 (未找到)
cache.get(3);       // 返回  3
cache.get(4);       // 返回  4
```

前面已经介绍过了 `keep-alive` 中LRU实现源码，现在来看这道题是不是很简单😊😊😊，可以尝试自己解答一下 ，然后思考一下有没有什么

欢迎将答案提交到 github.com/sisterAn/Ja... ，让更多人看到，瓶子君也会在明日放上自己的解答。

四、往期系列

瓶子君前端进阶算法营首周总结

前端进阶算法2：从Chrome V8源码看JavaScript数组（附赠腾讯面试题）

前端进阶算法1：如何分析、统计算法的执行效率和资源消耗？

五、认识更多的前端道友，一起进阶前端开发

前端算法集训营第一期免费开营啦🎉🎉🎉，免费哟！

在这里，你可以和志同道合的前端朋友们（600+）一起进阶前端算法，从0到1构建完整的数据结构与算法体系。

在这里，瓶子君不仅介绍算法，还将算法与前端各个领域进行结合，包括浏览器、HTTP、V8、React、Vue源码等。

在这里，你可以每天学习一道大厂算法题（阿里、腾讯、百度、字节等等）或 leetcode，瓶子君都会在第二天解答哟！

更多福利等你解锁🔓🔓🔓！

扫码加入【前端算法交流群交流群】,**若二维码人数已经达到上限，可扫底部二维码，在公众号「前端瓶子君」内回复「算法」自动拉你进群学习**

瓶子君前端算法交流群2

分类： 前端 标签： 前端 算法

评论

输入评论（Enter换行，Ctrl + Enter发送）

128 18 收藏

热门评论

Fiction 前端 1年前

有个地方不太理解，当缓存中存在BCD时，再次访问B页面，此时的流程是先把B淘汰，再把B重新推向第一位吗

点赞 8

前端瓶子君 Lv5 （作者） 1年前

嗯嗯

点赞 回复

Fiction 1年前

😂 为啥不直接把B移动到第一个位置呢，这样好像耗时更少吧，不然还有一个出队，入队的操作

点赞 回复

查看更多回复

全部评论 18

最新 最热

Fiction 前端 1年前

有个地方不太理解，当缓存中存在BCD时，再次访问B页面，此时的流程是先把B淘汰，再把B重新推向第一位吗

点赞 8

128 18 收藏

嗯嗯

点赞 回复

Fiction 1年前

😂 为啥不直接把B移动到第一个位置呢，这样好像耗时更少吧，不然还有一个出队，入队的操作

点赞 回复

查看更多回复

噢噢噢噢我知道啦 吐槽师 @ 吐槽部 1年前

群满了?

点赞 1

wyss 1年前

这群人这么快就满了。。。

点赞 回复

Milk就是我 1年前

这个群已经不让自己进了

1 回复

lohiccan 创始人 @ isrun 1年前

128 18 收藏

点赞 回复

前端瓶子君 Lv5 （作者） 高级前端 1年前

每天一道算法题，扫码进群学习 😄

3 2

相关推荐

前端胖头鱼 | 1月前 | 前端 · JavaScript · 算法

面试官：你都工作3年了，这个算法题都不会?

5.0w 306 135

Gaby | 5月前 | JavaScript · 面试

🔥 连八股文都不懂还指望在前端混下去么

18.0w 4958 241

渣渣xiong | 7月前 | 算法 · 前端

DIff算法看不懂就一起来砍我(带图)

2.4w 774 140

vortesnail | 1月前 | 前端 · 面试

128 18 收藏

13.3w 3411 199

Chris威 | 4年前 | Node.js · JavaScript · 前端

JS中的算法与数据结构——链表(Linked-list)

2.0w 182 23

杰出D | 8月前 | 前端 · 算法

面试了十几个高级前端，竟然连（扁平数据结构转Tree）都写不出来

14.9w 2634 1425

人类观察所主任 | 1年前 | 算法

图解 LRU LFU ARC FIFO 缓存淘汰算法

2651 4 评论

ConardLi | 2年前 | JavaScript

一名【合格】前端工程师的自检清单

24.9w 6441 584

Try to do | 7月前 | 前端 · JavaScript

keep-alive 如何清除缓存

4066 14 4

超级索尼 | 3年前 | Vue.js · 算法 · 前端

128 18 收藏

1.2w 200 18

wilton | 2年前 | Vue.js

彻底揭秘keep-alive原理

2.8w 415 28

ssh_晨曦时梦见兮 | 2年前 | JavaScript · 面试

写给女朋友的中级前端面试秘籍（含详细答案，15k级别）

18.2w 2880 195

程序员依扬 | 2年前 | 面试 · 前端

【1 月最新】前端 100 问：能搞懂 80% 的请把简历给我

52.7w 9275 308

张熠 | 3年前 | Vue.js · TypeScript

值得一看，Vue 作者尤雨溪的回答【TypeScript 不适合在 vue 业务开发中使用吗？】

9837 63 13

yeyan1996 | 2年前 | JavaScript

字节跳动面试官：请你实现一个大文件上传和断点续传

21.5w 5039 533

程序员鱼皮 | 8月前 | 前端

128 18 收藏

13.6w 900 223

Sunshine_Lin | 6月前 | 前端 · JavaScript · 算法

太震撼了！我把七大JS排序算法做成了可视化！！！太好玩了！

2.6w 646 105

非优秀程序员 | 4月前 | 前端 · JavaScript

如何用 CSS 中写出超级美丽的阴影效果

30.5w 231 12

荒山 | 2年前 | 前端 · 团队管理

if 我是前端团队 Leader，怎么制定前端协作规范?

15.7w 4222 227

帅地 | 2年前 | 算法

面试挂在了 LRU 缓存算法设计上

8135 80 17

128 18 收藏