【前端面试】如何使用JS完成一个LRU缓存?

前言

LRU 缓存算法是一个非常经典的算法，在很多面试中经常问道，不仅仅包括前端面试。小伙伴们如果刷过 Leetcode 算法题，相信你一定遇到过 LRU 算法的题，那么 LRU 算法到底是一个怎样的算法呢？今天我们就给大家好好讲讲，顺便使用 JS 把它实现出来！

1.什么是 LRU?

LRU 英文全称是 Least Recently Used，英译过来就是”**最近最少使用“的意思。**它是页面置换算法中的一种，我们先来看一段百度百科的解释。

百度百科：

> LRU 是一种常用的页面置换算法，选择最近最久未使用的页面予以淘汰。该算法赋予每个页面一个访问字段，用来记录一个页面自上次被访问以来所经历的时间 t，当须淘汰一个页面时，选择现有页面中其 t 值最大的，即最近最少使用的页面予以淘汰。

百度百科解释的比较窄，它这里只使用了页面来举例，我们通俗点来说就是：假如我们最近访问了很多个页面，内存把我们最近访问的页面都缓存了起来，但是随着时间推移，我们还在不停的访问新页面，这个时候为了减少内存占用，我们有必要删除一些页面，而删除哪些页面呢？我们可以通过访问页面的时间来决定，或者说是一个标准：在最近时间内，最久未访问的页面把它删掉。

百度百科的解释只是单纯的解释算法，而我们这里可以结合我们的前端和实际应用场景来给大家解释一下。

通俗的解释：

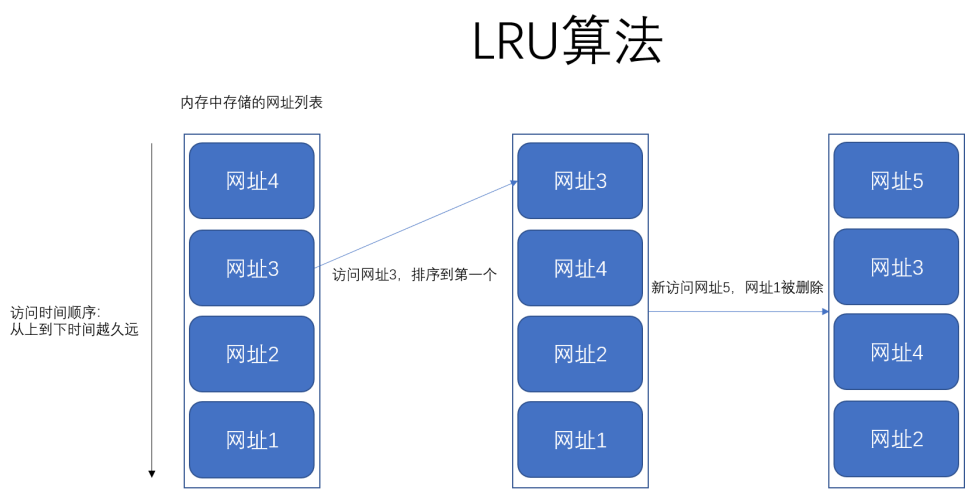
> 假如我们有一块内存，专门用来缓存我们最近发访问的网页，访问一个新网页，我们就会往内存中添加一个网页地址，随着网页的不断增加，内存存满了，这个时候我们就需要考虑删除一些网页了。这个时候我们找到内存中最早访问的那个网页地址，然后把它删掉。
>
> 这一整个过程就可以称之为 LRU 算法。

虽然上面的解释比较好懂了，但是我们还有很多地方没有考虑到，比如如下几点：

- 当我们访问内存中已经存在了的网址，那么该网址是否需要更新在内存中的存储顺序。

- 当我们内存中还没有数据的时候，是否需要执行删除操作。

最后我们在上一张图，大家应该就更容易理解了，如下图：



上图就很好的解释了 LRU 算法在干嘛了，其实非常简单，无非就是我们往内存里面添加或者删除元素的时候，遵循**最近最少使用原则**。

2.使用场景

LRU 算法使用的场景非常多，这里简单举几个例子即可：

1. 我们操作系统底层的内存管理，其中就包括有 LRU 算法
2. 我们常见的缓存服务，比如 redis 等等
3. 比如浏览器的最近浏览记录存储，如下图：

总之 LRU 算法的运用场景还是蛮多的，所以我们很有必要掌握它。

3.梳理实现 LRU 思路

我们学习了 LRU 算法的基本概念和使用场景之后，那么我们就应该考虑如何实现它了。要想实现一个算法，我们很有必要梳理一下思路，这样才能让我们更好更快的编写出代码。

首先我们来梳理一下 LRU 算法的特点。

特点分析：

- 我们需要一块有限的存储空间，因为无限的化就没必要使用 LRU 算发删除数据了。
- 我们这块存储空间里面存储的数据需要是有序的，因为我们必须要顺序来删除数据，所以可以考虑使用 Array、Map 数据结构来存储，不能使用 Object，因为它是无序的。
- 我们能够删除或者添加以及获取到这块存储空间中的指定数据。
- 存储空间存满之后，在添加数据时，会自动删除时间最久远的那条数据。

实现需求：

- 实现一个 LRUCache 类型，用来充当存储空间
- 采用 Map 数据结构存储数据，因为它的存取时间复杂度为 O(1)，数组为 O(n)
- 实现 get 和 set 方法，用来获取和添加数据
- 我们的存储空间有长度限制，所以无需提供删除方法，存储满之后，自动删除最久远的那条数据

- 当使用 get 获取数据后，该条数据需要更新到最前面

现在我们已经把 LRU 算法的特点以及实现思路列了出来，那么接下来就然我们一起去实现它吧！

4.具体实现

首先我们定义一个 LRUCache 类，封装所有的方法和变量。

代码如下：

```
<script>
  class LRUCache {
    constructor(lenght) {
      this.length = lenght; // 存储长度
      this.data = new Map(); // 存储数据
    }
    // 存储数据，通过键值对的方式
    set(key, value) { }
    // 获取数据
    get(key) { }
  }
  const lruCache = new LRUCache(5);
</script>
```

上段代码只是我们最简单的一个架子，我们需要去实现具体的 get 和 set 方法。

代码如下：

```
<script>
  class LRUCache {
    constructor(lenght) {
      this.length = lenght; // 存储长度
      this.data = new Map(); // 存储数据
    }
    // 存储数据，通过键值对的方式
    set(key, value) {
      const data = this.data;
      if (data.has(key)) {
        data.delete(key)
      }
      data.set(key, value);

      // 如果超出了容量，则需要删除最久的数据
      if (data.size > this.length) {
        const delKey = data.keys().next().value;
        data.delete(delKey);
      }
    }
    // 获取数据
    get(key) {
      const data = this.data;
```

```
      // 未找到
      if (!data.has(key)) {
        return null;
      }
      const value = data.get(key); // 获取元素
      data.delete(key); // 删除元素
      data.set(key, value); // 重新插入元素
    }
  }
  const lruCache = new LRUCache(5);
</script>
```

上段代码中实现实现了 get 和 set 方法，下面说一下这两个方法的实现思路：

- set 方法：往 map 里面添加新数据，如果添加的数据存在了，则先删除该条数据，然后再添加。如果添加数据后超长了，则需要删除最久远的一条数据。data.keys().next().value 便是获取最后一条数据的意思。
- get 方法：首先从 map 对象中拿出该条数据，然后删除该条数据，最后再重新插入该条数据，确保将该条数据移动到最前面。

接下来我们使用一些测试用例来试试行不行。

存储数据 set：

```
lruCache.set('name', '小猪课堂');
lruCache.set('age', 22);
lruCache.set('sex', '男');
lruCache.set('height', 176);
lruCache.set('weight', '100');
console.log(lruCache);
```

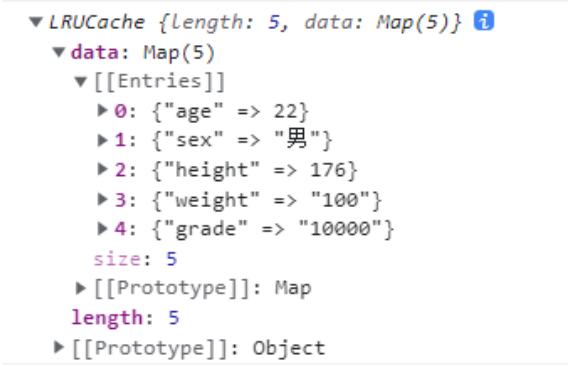
输出结果：

```
▼LRUCache {length: 5, data: Map(5)}
  ▼data: Map(5)
    ▼[[Entries]]
      ▶0: {"name" => "小猪课堂"}
      ▶1: {"age" => 22}
      ▶2: {"sex" => "男"}
      ▶3: {"height" => 176}
      ▶4: {"weight" => "100"}
      size: 5
    ▶[[Prototype]]: Map
    length: 5
  ▶[[Prototype]]: Object
```

继续插入数据，此时会超长，**代码如下：**

```
lruCache.set('grade', '10000');
console.log(lruCache);
```

输出结果：

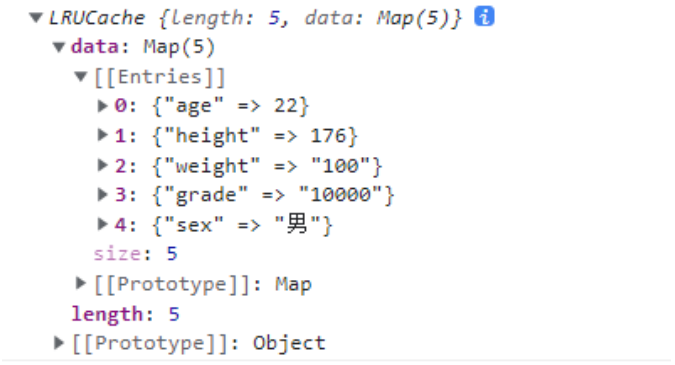


此时我们发现存储时间最久的 name 已经被移除了，新插入的数据变为了最前面的一个。

我们使用 get 获取数据，**代码如下：**

```
lruCache.get('sex');
console.log(lruCache);
```

输出结果：



我们发现此时 sex 字段已经跑到最前面去了。

总结

LRU 算法其实逻辑非常的简单，明白了原理之后实现起来非常的简单。最主要的是我们需要使用什么数据结构来存储数据，因为 map 的存取非常快，所以我们采用了它，当然数组其实也可以实现的。还有一些小伙伴使用链表来实现 LRU，这当然也是可以的。