### 内存查看

很多人都想着，都说WeakMap可以释放内存，真的假的啊？我怎么去验证啊？

其实你大可不必去验证，你可以放心的相信官方。不过眼见为实也是好的，所以我们用几个例子来验证一下，首先确定几个场景：

前提： 赋值一个对象给某一个变量当作KEY

* 将这个变量做key，一个长度为10,000,000的数组作为value，然后map.set(key, value)
* 将这个变量做key，一个长度为10,000,000的数组作为value，然后map.set(key, value)。然后给key重新赋值，改变地址。
* 将这个变量做key，一个长度为10,000,000的数组作为value，然后weekmap.set(key, value)
* 将这个变量做key，一个长度为10,000,000的数组作为value，然后weekmap.set(key, value)。然后给key重新赋值，改变地址。

我们通过浏览器的内存工具，挨个查看结果。我将代码和结果贴出来

#### Map -> 不修改key的数据

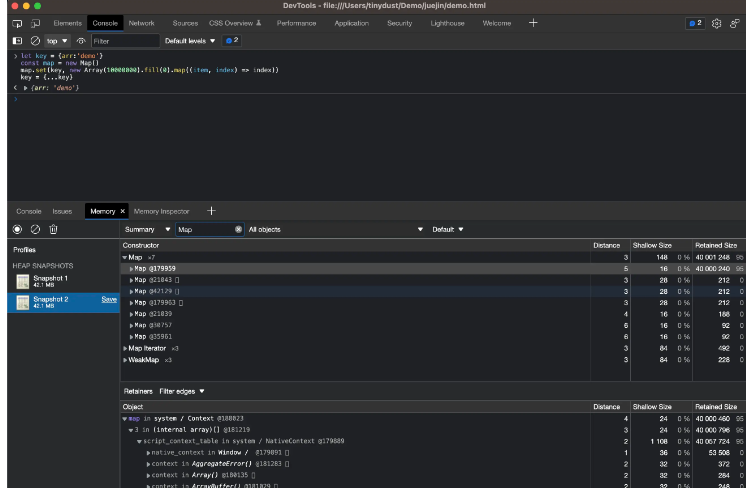
const key = {arr:'demo'}

const map = new Map()

map.set(key, new Array(10000000).fill(0).map((item, index) => index))

复制代码

内存结果如下，总内存42.1MB



#### Map -> 修改key的数据

let key = {arr:'demo'}

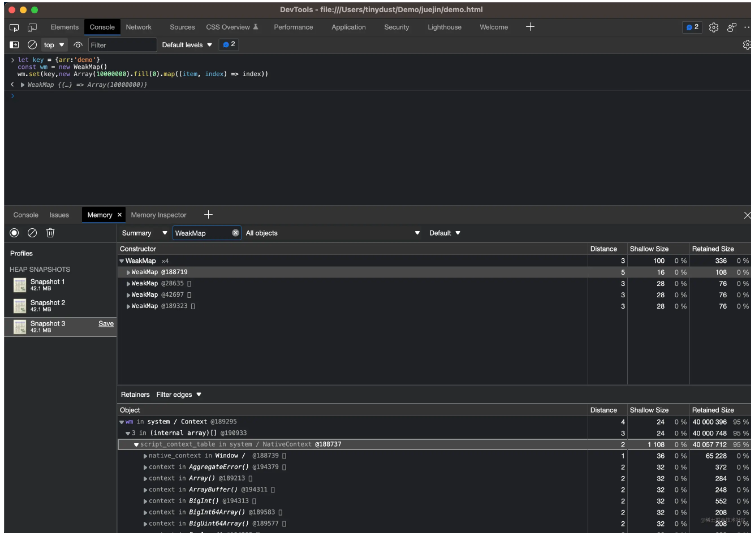
const map = new Map()

map.set(key, new Array(10000000).fill(0).map((item, index) => index))

key = {...key}

复制代码

内存结果如下，总内存还是42.1MB



#### WeakMap -> 不修改key的数据

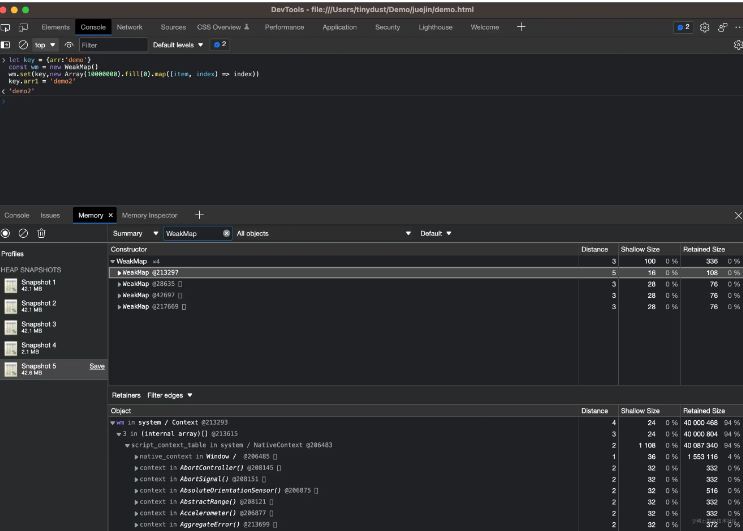
let key = {arr:'demo'}

const wm = new WeakMap()

wm.set(key,new Array(10000000).fill(0).map((item, index) => index))

复制代码

内存结果如下，总内存依旧是42.1MB



#### WeakMap -> 修改key的属性值，不改变地址

const key = {arr:'demo'}

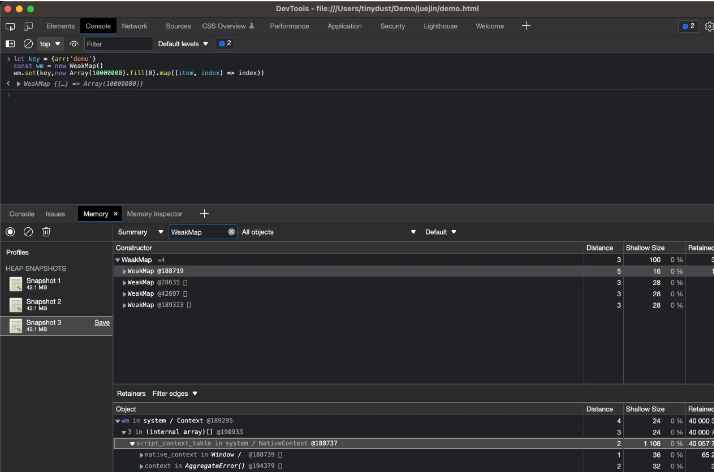
const wm = new WeakMap()

wm.set(key,new Array(10000000).fill(0).map((item, index) => index))

key.arr1 = 'demo2'

复制代码

内存结果如下，总内存变成了42.6MB



#### WeakMap -> 修改key的地址引用

let key = {arr:'demo'}

const wm = new WeakMap()

wm.set(key,new Array(10000000).fill(0).map((item, index) => index))

key = {...key}

复制代码

内存结果如下，总内存变成了2.1MB !!!

有了这个神器，我还用什么Map、Set！直接WeakMap一把梭了啊

非也非也~ 客官你继续看

### 使用的场景

我之前看了很多的文章，说实话。他们说得使用场景，我还是没理解。Set去重是唯一记住的😕。

我认为，我们要知道东西的使用场景，首先得清楚这个工具有什么特点。比如我知道锤子不能拧螺丝（你非要想把螺丝拧进去，可能也不是不行，就是肯定没有专业工具更方便）。

**Map的特点/特性**

首先来个MDN[Map - JavaScript | MDN (mozilla.org)](https://link.juejin.cn?target=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FJavaScript%2FReference%2FGlobal_Objects%2FMap)的文档打个底。我们将特性给记下来，至于它的属性方法，自己去看，都是直接看名字就知道意思的，没必要去讲怎么去set，去get等等。

* 能够记住键的原始插入顺序，即你在0位置插入，后续set同一个key时，它依旧是在0的位置
* 任何值都可以做key和val，强调一下，是 **任何值！** 也就是说什么正则、Error、Function、Symbol都是可以的哦
* key是独一无二的
* 需要维护哈希表和链表等，内存占用相对会大，同时由于基于哈希，所以查询的时间复杂度是O(1)，看不懂没关系，就是你只要知道它不会因为数据规模的增长而增长就好了
* 频繁的增删键值对性能更好，插入千万个number数据，Map耗时仅仅10s左右，但是通过Object[key] = val方式，运行了400s还没有执行结束
* 在大量数据的查询下性能更好，在百万基础数据量，查询12000条数据时，时间接近。但是查询500000条数据时，Map碾压式领先

OK~OK~ 知道了这些，我们是不是立刻就想到了可以在哪些场景下使用了啊。。。。。个锤子啊！完全想不起来啊！



冷静一下想想，诶~好像真有那么一些场景哦

* 如果将网络请求的路径和参数解析成一个唯一键，值是过期的时间，那对于请求的重复拦截和指定时间内的重复拦截，以及过期后的时间重置。好像是比Object要方便的多，因为频繁的读写肯定是Map性能更好。
* 如果有一个平铺开的省市区列表，需要整理成Tree，亦或者把相同属性的对象，其他数据合并起来。例如[{ id:1, val: [1, 2]}, { id:2, val: [2] }]
* 还有就是如果你发布文章，再或者购物，我只需要Map(User, Product)，就不需要单独在两边数据加上属性关联了
* 如果我要统计一个数据的出现个数，数据不仅仅还是基础类型。好像也是很easy

总结来说就是频繁的读写和查询，键值复杂这类的场景肯定是Map。但是注意哦，它会无限期的维护着它的键值的引用，所以注意内存的问题。及时清理无用的数据。

#### WeakMap的特点/特性

* 键是弱引用 这是最为重要的特性
* 键只能是对象
* 因为是弱引用，所以它的key是无法枚举的，看他的方法就知道，只有has、set、get、delete

这个样子的话，那它的适用场景，好像有点清晰了呢。。。 例如：

* 我需要对一些DOM的节点进行存储，并且保存这个节点和其他数据之间的关系，例如对这个节点的操作、事件监听之类的。
* 如果我需要一个单例，那我就可以使用WeakMap维护对应的函数/Class
* 再有就是像开头提到的Vue源码中，WeakMap被用到的场景更多
  + 比如实现组件的一些私有属性不被外部访问
  + 组件的一些事件监听的存储，这些需要在组件销毁的时候自动被释放
  + 还有就是模板字符串的编译缓存也是利用了WeakMap

总结的话，那就是你不需要大量的数据操作、你也不需要对数据进行迭代，你只想实现缓存、一些引用数据和它依赖之间的关系处理，并且你不想手动去处理这些数据的回收问题，那就用WeakMap!

### Set WeakSet

这俩实际上跟数组类似，不过Set是不能有重复数据的，具体表现在于，如果是基础数据类型 那就直接去重。对于引用数据类型，只会比较他们的地址是否相同。另外还有一些特殊的情况，我直接摘抄来自MDN中对值的相等的描述：

### [值的相等](https://link.juejin.cn?target=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FJavaScript%2FReference%2FGlobal_Objects%2FSet%23%25E5%2580%25BC%25E7%259A%2584%25E7%259B%25B8%25E7%25AD%2589)

因为 Set 中的值总是唯一的，所以需要判断两个值是否相等。在 ECMAScript 规范的早期版本中，这不是基于和===操作符中使用的算法相同的算法。具体来说，对于 Set，+0（+0 严格相等于 -0）和 -0 是不同的值。然而，在 ECMAScript 2015 规范中这点已被更改。有关详细信息，请参阅[浏览器兼容性](https://link.juejin.cn?target=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FJavaScript%2FReference%2FGlobal_Objects%2FSet%23%25E6%25B5%258F%25E8%25A7%2588%25E5%2599%25A8%25E5%2585%25BC%25E5%25AE%25B9%25E6%2580%25A7)表中的“Key equality for -0 and 0”。

另外，NaN 和 undefined 都可以被存储在 Set 中，NaN 之间被视为相同的值（NaN 被认为是相同的，尽管 NaN !== NaN）。

另外，在最近的TC39（第39号技术委员会，负责迭代和发展 ECMAScript 语言规范的委员会）提案中，有提到对Set的方法进行补充，提案里提到的补充有以下这些：

* Set.prototype.intersection(other) 交集
* Set.prototype.union(other) 并集
* Set.prototype.difference(other) 差分
* Set.prototype.symmetricDifference(other) 对称差分
* Set.prototype.isSubsetOf(other) 子集
* Set.prototype.isSupersetOf(other) 超集
* Set.prototype.isDisjointFrom(other) 不相交

当前的这个提案处于第三阶段（Candidate 候选方案：进一步完善并接受一些用户的反馈），如果上面的方法在未来发布到了新的ES标准里，那对于我们一些业务的数据计算提供极大的便捷~ 一起期待吧

### 总结

日常我们的开发中，实际上对Map、WeakMap应用的不多，大部分的业务都是copy，然后就是Object、Array一把梭。我之前出现的问题就是：

我知道Map、WeakMap的概念，也知道它们方法，甚至不需要知道他们的方法，IDE会给我对应的提示，但是我不知道什么时候去使用，好像每一个上文中的每个场景，我都可以用Object、Array去完成，那问题在哪里呢？

问题就在于我不懂如何让它们有更好的可读性、便捷性以及平时也不在乎哪些内存的占用问题。但是当我空闲下来了，突然想起来之前有个地方好像有点复杂的时候，就可以回头来看看，是否可以对它们升级一下。所以我想，当你开发时，遇到大量的数组处理、对象关系处理、频繁操作dom等情况时，不妨先加一个 // TODO: 待优化。

结尾加一句：如果发现文章有不对的地方，还请指正。在错误中成长的更快🍻🍻🍻