前端老司机讲故事：算法相关的四个故事

之前chat的介绍和公众号铺垫也比较多了，本文不再讨论前端是否应该学习算法，因为能够看到这篇文章的朋友肯定是对算法感兴趣的朋友，我会讲解四个我经历过的跟算法相关的故事，然后总结下通过这四个故事对自己有什么感触：

1. 第一份工作是做多级联动选择器
2. 终于写了个纯真 IP 库却被鄙视
3. 我常来面试别人的题目：EventBus 实现
4. 面挂经验：LRU Cache 实现

第一份工作是做多级联动选择器

第一家公司一开始是一个生活信息分类网站，干的事情就像58赶集一样，刚刚去公司是做一个四级联动选择器，包括省市县和商圈，数据都是自己抓取。对于我来说，需要面临的挑战是：

1. 我不会抓取，当时没有node可以用js来做
2. 联动选择器应该怎么组织省市县和商圈数据，没有经验
3. 公司刚刚开始做分类信息业务，数据抓取整理后，要给后台、数据库等都需要用一套

领到任务之后，任务分解了下，主要包括两点：
1. 抓取，这块需要找老司机带带路
2. 整理数据，这块主要是为了级联选择器好用

> 每次有什么新技术想练手，我就拿建站这块来做，移动webapp火了出来就做个移动站点，node出来就用node重写抓取，垃圾站server换成node。所以很多新人一般学习完新技术之后，说项目中不用很快就忘记了，可以在自己的「小项目」中多练手，遇见问题解决问题才能巩固住。

这算是我自己学习的一个方法吧。

数据整理

级联选择器需要设计数据格式，说到底其实好的数据格式自然计算量和查找量就少了，所以我开始用了下面的树形菜单结构：

```
{
    '山东': {
        id: '233',
        citys: {
            '青岛': {
                id: '233'
                counties: {
                    '开发区': {
                        id: 233,
                        zones: {
                            '开发区': {
                                '香江路',
                                '长江路'
                            },
                            '市南区': {

                            }
                        }
                    }
                }

            }
```

```
北京：101010100
– 昌平：101010700
– 海淀：101010200
--- 八里庄街道：101010200001
--- 甘家口街道：101010200005
```

观察会发现：

> id中下一级带着上一级的公共部分，可以通过简单预算求出上一级的id

后来发现国家统计局发布的省市县划分代码更加完善和权威，看下：http://www.stats.gov.cn/tjsj/tjbz/xzqhdm/201703/t20170310_1471429.html
这个链接的数据。当时一下子就茅塞顿开。

于是我将id对应name存了一份数据：

```
var idmap = {
    110000: '北京市',
    110101: '东城区',
    110102: '西城区',
    110105: '朝阳区',
    110106: '丰台区'
    .....
}
```


类似级联选择器这种需求的衍生数据，会直接用id来存储关系，这样可以提升查找速度：

```
// 比如：
110000:[01,02,04,05…]
```

终于写了个纯真 IP 库却被鄙视

现在这个IP库支撑我厂一个Super APP的基本IP定位和运营商判断功能，比如：针对某些运营商下发不同的运营位或者针对没有定位信息的用户使用IP定位省市县，都有用到这个IP库。

事情开始的时候，公司内部有文本的IP库，也有IP查询的API接口，但是经过调研发现，API调用不符合我们需求：要求高效、低延迟的查询调用，因为无法忍受一个app开启的时候update接口下发数据太慢，会影响开屏时间。

经过一番调研之后，想到玩QQ时候的纯真IP库。零几年的时候，QQ还可以加外挂显示IP，其中比较有名的是纯真IP库。这类IP库是将文本格式的IP信息库，转成 `dat` 这类二进制IP库，而且一搜索也有对应的很多版本的解析库。

所以要做的是：
* 设计一种二进制IP库的数据结构，能够将信息存入其中，保证体积尽可能小
* 写一个解析程序，输入用户IP，能够从IP库中返回IP定位和运营商数据

想通过查找纯真IP库的设计，来做一套类似纯真IP库的设计，但是发现网上没有找到IP库详细的设计文档，通过解析代码来看纯真IP库的查询并不快。于是就有了开始自己动手搞一搞的冲动。

所以这里主要讲两点：

1. 数据库的数据结构设计
2. 实现快速查找

文本IP库格式

先说下一开始拿到的文本版本的IP信息库格式：

这种文本的IP库，国内数据大概 48M 大小，大概40多万条数据。

二进制数据库设计

上面的文本文件格式很简单，一行一个IP端，但是在实际查找一个IP对应的运营商和地理信息的时候很不方便，总不能把text文件逐行遍历查找吧，那样性能极低，而且文本文件格式也较大，所以需要转成类似纯真IP库的二进制格式，然后加上索引区，通过多级索引区设计，可以快速查找到IP所在的数据区的起止位置，然后利用遍历找到对应的IP段，这样也可以缩小IP文件大小。

简单说下数据库设计原理，这里就不涉及太多细节：

1. 文件分为索引区和数据区两部分
2. 首4个字节是索引区最后的offset，即数据区开始的offset
3. 索引区分超级索引和普通索引，超级索引是：前1024个字符，每四个代表一个ip大段 [0~255]
4. 普通索引区每8个字节代表一个索引，前四个是ip段，5~7是对应的ip数据所在数据区的offset，最后是数据长度
5. 数据区域就是存的数据，为了使用方便，我又将数据分为包含省市中文名字、不包含中文名包含城市ID和全球IP数据，做出了3个IP库

这里需要介绍下： `ip2long` 这个函数。

ip2long

写过php的应该知道ip值可以通过ip2long方法转换成int值，node当中可以直接用buffer模块来实现：

```
function ip2long(ip) {
    return new Buffer(ip.split('.')).readInt32BE(0)
}
```

这样一个ip就可以转成整数存储，而二级索引内的实际是按大小排好顺序存储单元（8个字节为一个单元，0~3是存储的ip end信息）

查找IP

假如我们查找IP为 `1.193.92.10` ，对应数据为：

```
1.193.92.0|1.193.93.255|CN|CHINANET|河南|郑州|二七|100|85|90|90|50
```


整个的查找过程是这样的：

1. 把 `ip.split('.')` ，得到 `ipArr = [1,193,92,0]`
2. 根据ipArr[0]，可以在一级索引区域找到对应的值： `ipArr[0]*4`
3. 将这个值在二级索引查找开始，然后往后查找比较二级索引一条数据单位，直到找到这个结束位置ip-end
4. 取出二级索引这个ip-end位置的数据区offset和长度，解二进制就可以得到数据

这里主要是二级（普通）索引的查找，需要遍历，比如： `1.193.92.0|1.193.93.255` 这个区间，就需要从 `1` 开始查找，直到找到 `1.193.93.255` 相比小，找到为止。

其实这个就是个顺序查找算法，以我当时的能力自然想到的就是「遍历」！

```
for (;
    (end - start) / 2 >= 1;) {
    var tmp = Math.floor((end - start) / 2 + start);
    m = tmp * 8 + 1024;
    if (indexBuffer.slice(m, m + 4).readInt32BE(0) >= ipInt) {
        end = tmp;
    } else {
        start = tmp;
    }
}
find = end * 8 + 1024;
```

而自己又查找了好多资料，原来这就是二分查找。。

这个故事讲完了，本来我以为设计的数据结构已经够完美了，值得炫耀一番，没想到查找算法上面反而被嘲笑了。

我常来面试别人的题目：EventBus 实现

这个题目本身是这样的：首先类似微博这种复杂交互的页面，分模块开发，需要多模块之间实现通信配合，如果候选人提到通过自定义事件之类方式来实现模块间通信，那么就会让他写个EventBus，包括：事件的订阅、发布基本功能。如果可以实现，则进一步问，如何实现 `once` 和 `context`（支持事件回调函数 `this` 绑定）功能。

这里主要考查点是：

1. 是否了解页面组件化开发通信机制
2. 能否实现自定义事件机制，以及实现的代码灵活度

其实对于EventBus的代码，主要设计在存储事件的数据结构上，假设这个存储事件的数据为 `listeners` ：

但是这种结构并不满足支持 once （触发一次就删除）的要求，所以 listeners 的事件类型数组（队列）还需要详细设计下：

```
const listeners = {
    type1: [{listener: fn1, once: true}, {listener: fn2,
once:false}…],
    type2: [{listener: foo1, once: true}, {listener: foo2, once:
fasle}…]
}
```

这样，数组的单个值变成了对象，这个对象内包括了函数的 once 信息，标识是只执行一次的，这样执行完了事件回调函数之后，就可以按照这个once来移除 listener 。

我用来面试别人的题目讲完了，其实就是一道由浅入深的面试题。我用它面过很多人，一部分人只知道DOM事件；一部分人虽然用过但是不会写；也有虽然没用过，但是可以在引导下一步步实现的。

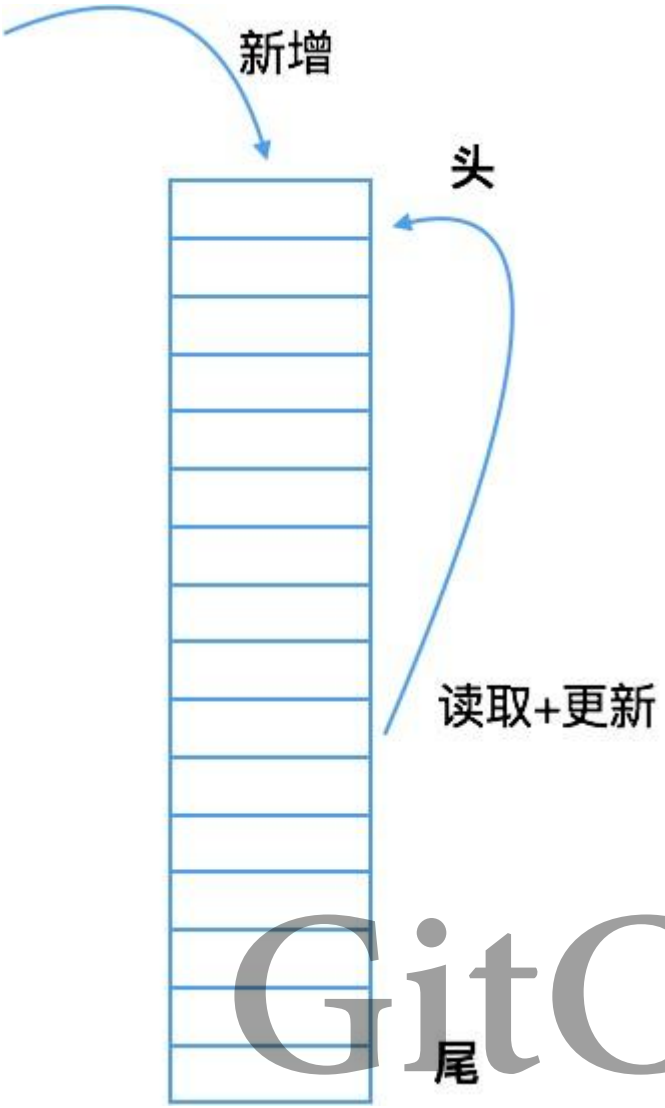
相对于项目经验丰富的，用过也能实现的候选人，我更喜欢没用过但是却能够理解并且实现的，这种候选人可能是因为项目经验不足，但他理解能力、学习能力强，可塑性高。题目本身很简单，算是数据结构的基本知识，但是对于我来说面试这么多人，感触还是挺多的。

面挂经验：LRU Cache 实现

这是某一次面试经历，当时二面官是个安卓的研发，所以没啥共同语言可以聊，刚开始聊的多数是项目和人生之类的，突然最后问了一个技术题目：你知道LRU cache吧，给写一下吧！

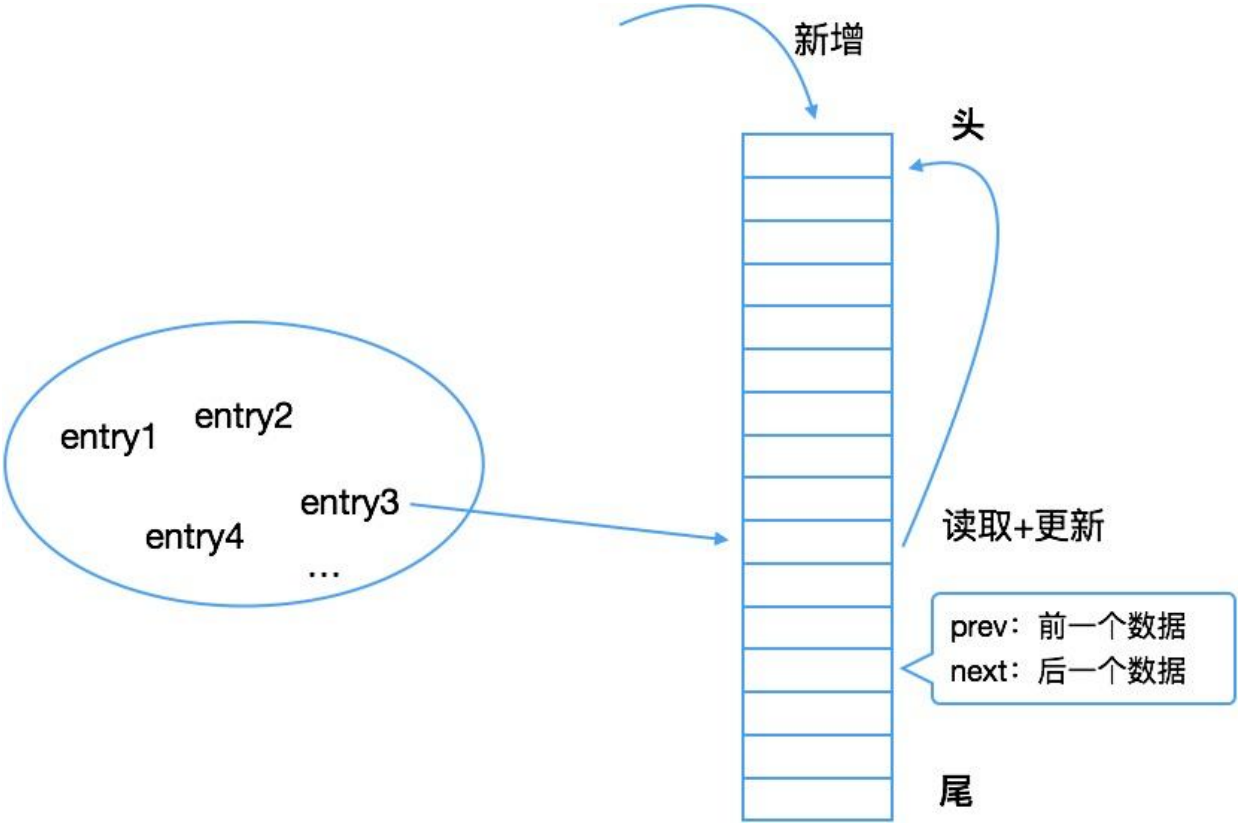
突然画风转的有点摸不着头脑，还好我当时知道LRU大概是什么，根据LRU的特点估计可以写出来。先说下具体题目，再说面试的故事。

先说一下最简单通过一个数组的实现。



1. 创建一个数组（array）
2. 新数据加上访问时间，插入（unshift）头部
3. 命中的数据，从数组中拿出（splice），更新访问时间，再插到（unshift）数组头部
4. 当数组长度超过设定长度（ `array.length=maxSize` ），则将尾部数据丢掉

缺点：每次都需要遍历查找数据，然后将数据重新推到表头。典型的时间换空间的思路，也是最常想到的最简单实现。



1. 读取的时候，按key从字典中读取出entry，返回 `entry.value`
2. 更新，将链表 `head` 改成entry， `entry.next` 改成之前的head
3. 修复链表链接： `entry.prev` 和 `entry.next` 连接起来

所有的操作都是常数级别的，不需要遍历！

最后再说面试故事

继续故事，知道LRU是什么东西，我就用对象来快速查找返回缓存的数据，利用数组遍历的方法来完成链表的维护，但是似乎还是怪怪的感觉，所以用js简单写了下数组和对象，后来越写越觉得太多此一举了，完全一个数组就搞定了，然后想跟面试官沟通，跟他解释了半天用对象和对象来做，他愣是没明白，我也知道估计是跨语言，他不懂，然后就随便说字典和队列，然后面试官说：我知道了！

然后就没有然后了

故事讲完了，关于前端工程师应不应该掌握算法，我的答案是：肯定需要的。不得不说，现在前端工程师门槛越来越低，前端分工也越来越细，很多前端工程师从事的工作越来越「螺丝钉」，但是对于本Chat参与者，希望通过上面的讲述，能够得到下面的启发：

1. 知识面广度和深度都拓展，做好知识储备，迎接新机遇很重要（级联选择器）
2. 灵活的脑子，也需要系统的知识学习（IP库）
3. 算法和数据结构至少做到下面几点：
    1. 了解基础数据结构和应用场景，（LRU）
    2. 了解常见的排序算法和查找算法：比如冒泡、快排、二分法
    3. 了解常见算法思想：分治、动态规划、广度和深度搜索等
    4. 抽象能力和算法思维是工程师必备的技能！
4. 多联系工作和生活场景，活学活用

对于这个chat，你有什么启发或疑惑，欢迎继续后面通过微信群来提问沟通~