算法的衡量

如何衡量一个算法的好坏呢？

课程目录

- 课前思考
- 预习
- 复杂度的表示方法
  - 复杂度的描述（O, Θ和Ω ）
  - 复杂度分析举例
    - 插入排序的复杂度
    - 归并排序的复杂度
    - 快速排序的复杂度
  - 递归结构的分析方法
- 实战问题精选
  - 100W数据随机打乱和排序算法
  - 大型系统的路由匹配算法

前言：前端提高算法性能的意义

我们常常听说，前端只要实现功能就可以，不用在意程序的执行时间，真的是这样吗？ 其实不然，相比服务端，前端在很大程度上对性能有更高的要求，主要是基于以下两点考虑：

- 流畅问题——为了保证动画的流畅进行，每一帧动画我们只有(1000/60)ms. = 16ms 时间来执行其他计算过程。播放动画的同时我们可能还在请求数据、改变DOM结构、响应事件。(从某种意义上说服务端轻松得多，因为他们的接口程序通常只需要在100ms内完成执行）

- 老旧机型——前端还有大量机型需要适配，这些机型可能性能没有那么好（比如华为荣耀4a, Android 5.1），在这台机器上亲测一个未经优化设计50多张页面的路由算法执行时间为500ms。 如果算法设计失误，在一台机器上执行的算法可能会在另一台机器上产生**雪崩效果**。

预习：算法依赖的模型

我们假定CPU会顺序的执行所有的指令，而内存随机访问的代价是相同的，例如：

而与内存最相近的数据结构就是数组，那么我们认为，对于任何一个我们定义的数组：

```
const a = [1,2,3,4,5]
```

它的索引操作，占用1单位时间（也就是消耗1的CPU指令）：

```
a[2]
```

预习：算法依赖的模型

整数变量赋值，也消耗1的CPU时间

```
const b = 1 // 1
const c = 2 // 1
const m = -2 // 1
```

而字符串赋值，我们不能简单的认为是1的时间：

```
const str = 'hello world!'
```

字符串赋值更类似于数组赋值，相当于一个一个字符赋值，
也就是hello world!一共12个字符，我们可以简单认为需要12的时间

预习：算法依赖的模型

对象的赋值，

```
const obj = {} // 创建对象的成本
obj['x'] = 1 // 1次索引，1次赋值
```

我们同样不能简单认为是1的时间，而是要具体问题具体分析

```
const d = new Date()
```

但我们认为上述操作都是耗时很少的操作，可以在常数时间内执行完成

预习：算法依赖的模型

因为CPU通常提供一些指令，比如加减乘除，所以我们可以认为下列操作可以在1的时间内完成：

```
1 + 2
100 + 200
499*21
500/3
10%5
```

但对一些更加复杂的操作，就不能这样认为了：

```
"123" + "456"
Math.pow(10, 10)
Math.sqrt(100)
```

但是我们通常可以认为这些操作也是相对较快的常熟级别操作

预习：算法依赖的模型

我们认为一部分逻辑运算符也可以在1的单位时间计算完成：

```
1 > 2
a > b
x <= 3
```

预习：线性时间的算法

从一个有序数组中搜索一个值，最暴力的方法就是遍历了。当然也是最慢的做法。 我们来分析下这个算法的复杂度，我们下面来分析一下它的运行时间。

```
function find(arr, value) {
  for(let i = 0; i < arr.length; i++){
    if( arr[i] === value ) {
      return value
    }
  }
  return null
}
```

预习：线性时间的算法

在最坏的情况下，没有找到值：

- 第2行：i=0执行了1次；i<arr.length执行了 N+1次; i++执行了N次。所以总共执行了 2N + 2次。
- 第3行：比较操作执行了N次
- 第4行：执行0次
- 第7行：执行1次

所以算法最坏的情况下，用时： T = 2N+2+N+1 = 3N + 3

这种最坏情况下，复杂度和数据规模N相关的算法很常见，我们成为线性时间复杂度。

珠峰培训
最专业的前端培训

课前思考题目：100W整数数据的排序

- 先生成1-100W的整数
- 写一个算法将他们随机打乱
- 再写一个算法对他们进行排序
- 最后输出一下自己程序的总执行时间

前置知识：大数定理

- 在随机事件的大量重复出现中，往往呈现几乎**必然**的规律，这个规律就是大数定律
- 比如抛硬币，次数多了之后（比如1万次），正面朝上和反面朝上的数量会趋同

前置知识：需要用到的两个希腊字母

算法复杂度的衡量用到3个字母，分别是O，。 Θ，Ω是两个希腊字母，发音如下：

小写	大写	发音
θ	Θ	theta
ω	Ω	omega

珠峰培训
最专业的前端培训

前置知识：对数函数

在衡量算法复杂度的过程当中常常用到对数函数，对数是指数的逆运算：

- $2^{10} = 1024 \Leftrightarrow \log_2 1024 = 10$

比如需要从100万数据中查找结果，算法可以在$\log_{10} N$的时间内完成，那么100W数据查找结果，需要处理 $\log_{10} 1000000 = 6$次计算。 那么这样的查找操作，速度是相对比较快的。

前置知识：一些对数函数

- $\lg n = \log_2 n$
- $\ln n = \log_e n$
- $\lg^k n = (\lg n)^k$
- $\lg \lg n = \lg(\lg n)$

e是一个神奇的数字， $e = 2.718281828459...$,比如斐波那契数列

1 1 2 3 5 8 13 ⋯ 下一项等于前两项的和，其实下一项还等于上一项乘以e然后去掉小数部分取整。

前置知识：等差数列求和公式

- 等差数列就是形如1,2,3,...n的数列，两个相临项之和相等。 再比如: 2,4,6,8,...,n， 是第一项是2，相邻项差为2的等差数列。
- 在我们分析程序的执行效率的时候，经常需要求等差数列的和。

比如： 1+2+3+...+n = (1+n) + (2 + n-1) + (3 + n-2) + ...
等差数列求和就相当于，第一项和最后一项相加， 第二项和倒数第二项相加， 依此类推。但是每项和都是相同的。
最后：

$$(1+2+\cdots+n) = (1+n) + (2+n-1) + \ \ldots\ = (n+1)\left(\frac{n}{2}\right)$$

上述我们表示为：

- $\sum_{k=1}^{n} k = \frac{1}{2}\, n\,(n+1)$