如何衡量程序运行的效率?

咱们这个目标是想教会你利用数据结构的知识，建立算法思维，并完成代码效率的优化。为了达到这个目标，在第一节课，我们先来讲一讲如何衡量程序运行的效率。

当你在大数据环境中开发代码时，你一定遇到过程序执行好几个小时、甚至好几天的情况，或者是执行过程中电脑几乎死机的情况：

> 如果这个效率低下的系统是离线的，那么它会让我们的开发周期、测试周期变得很长。
>
> 如果这个效率低下的系统是在线的，那么它随时具有时间爆炸或者内存爆炸的可能性。

因此，衡量代码的运行效率对于一个工程师而言，是一项非常重要的基本功。本课时我们就来学习程序运行效率相关的度量方法。

复杂度是什么

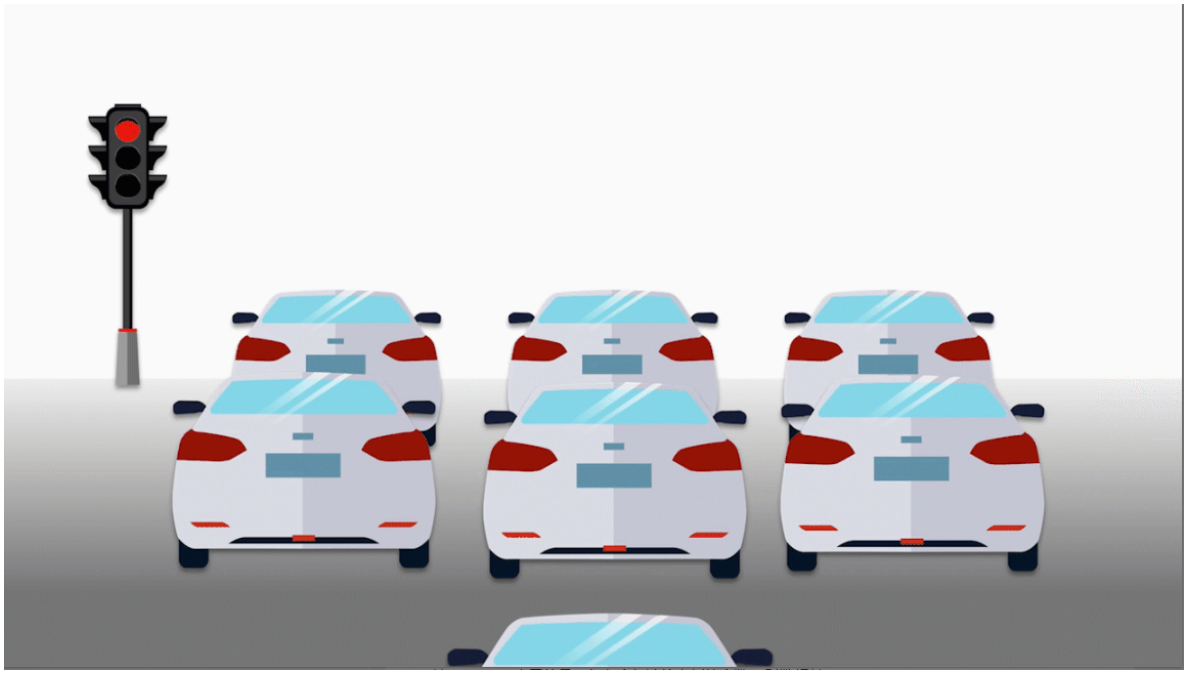
复杂度是衡量代码运行效率的重要的度量因素。在介绍复杂度之前，有必要先看一下复杂度和计算机实际任务处理效率的关系，从而了解降低复杂度的必要性。

计算机通过一个个程序去执行计算任务，也就是对输入数据进行加工处理，并最终得到结果的过程。每个程序都是由代码构成的。可见，编写代码的核心就是要完成计算。但对于同一个计算任务，不同计算方法得到结果的过程复杂程度是不一样的，这对你实际的任务处理效率就有了非常大的影响。

举个例子，你要在一个在线系统中实时处理数据。假设这个系统平均每分钟会新增 300M 的数据量。如果你的代码不能在 1 分钟内完成对这 300M 数据的处理，那么这个系统就会发生时间爆炸和空间爆炸。表现就是，电脑执行越来越慢，直到死机。因此，我们需要讲究合理的计算方法，去通过尽可能低复杂程度的代码完成计算任务。

那提到降低复杂度，我们首先需要知道怎么衡量复杂度。而在实际衡量时，我们通常会围绕以下2 个维度进行。首先，这段代码消耗的资源是什么。一般而言，代码执行过程中会消耗计算时间和计算空间，那需要衡量的就是时间复杂度和空间复杂度。

我举一个实际生活中的例子。某个十字路口没有建立立交桥时，所有车辆通过红绿灯分批次行驶通过。当大量汽车同时过路口的时候，就会分别消耗大家的时间。但建了立交桥之后，所有车辆都可以同时通过了，因为立交桥的存在，等于是消耗了空间资源，来换取了时间资源。



其次，这段代码对于资源的消耗是多少。我们不会关注这段代码对于资源消耗的绝对量，因为不管是时间还是空间，它们的消耗程度都与输入的数据量高度相关，输入数据少时消耗自然就少。为了更客观地衡量消耗程度，我们通常会关注时间或者空间消耗量与输入数据量之间的关系。

好，现在我们已经了解了衡量复杂度的两个纬度，那应该如何去计算复杂度呢?

复杂度是一个关于输入数据量 n 的函数。假设你的代码复杂度是 f(n)，那么就用个大写字母 O 和括号，把 f(n) 括起来就可以了，即 O(f(n))。例如，O(n) 表示的是，复杂度与计算实例的个数 n 线性相关；O(logn) 表示的是，复杂度与计算实例的个数 n 对数相关。

通常，复杂度的计算方法遵循以下几个原则：

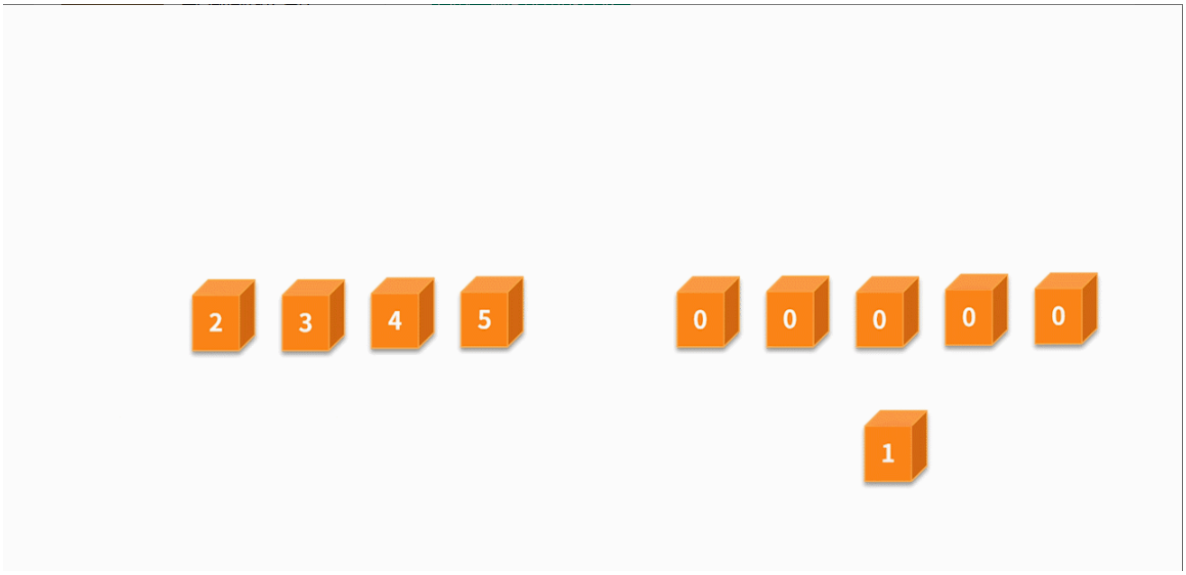
> 首先，复杂度与具体的常系数无关，例如 `O(n)` 和 `O(2n)` 表示的是同样的复杂度。我们详细分析下，`O(2n)` 等于 `O(n+n)`，也等于 `O(n) + O(n)`。也就是说，一段 `O(n)` 复杂度的代码只是先后执行两遍 `O(n)`，其复杂度是一致的。
>
> 其次，多项式级的复杂度相加的时候，选择高者作为结果，例如 `O(n²)+O(n)` 和 `O(n²)` 表示的是同样的复杂度。具体分析一下就是，`O(n²)+O(n) = O(n²+n)`。随着 `n` 越来越大，二阶多项式的变化率是要比一阶多项式更大的。因此，只需要通过更大变化率的二阶多项式来表征复杂度就可以了。

值得一提的是，O(1) 也是表示一个特殊复杂度，含义为某个任务通过有限可数的资源即可完成。此处有限可数的具体意义是，与输入数据量 n 无关。

例如，你的代码处理 10 条数据需要消耗 5 个单位的时间资源，3 个单位的空间资源。处理 1000 条数据，还是只需要消耗 5 个单位的时间资源，3 个单位的空间资源。那么就能发现资源消耗与输入数据量无关，就是 O(1) 的复杂度。

为了方便你理解不同计算方法对复杂度的影响，我们来看一个代码任务：对于输入的数组，输出与之逆序的数组。例如，输入 a=[1,2,3,4,5]，输出 [5,4,3,2,1]。

先看方法一，建立并初始化数组 b，得到一个与输入数组等长的全零数组。通过一个 for 循环，从左到右将 a 数组的元素，从右到左地赋值到 b 数组中，最后输出数组 b 得到结果。



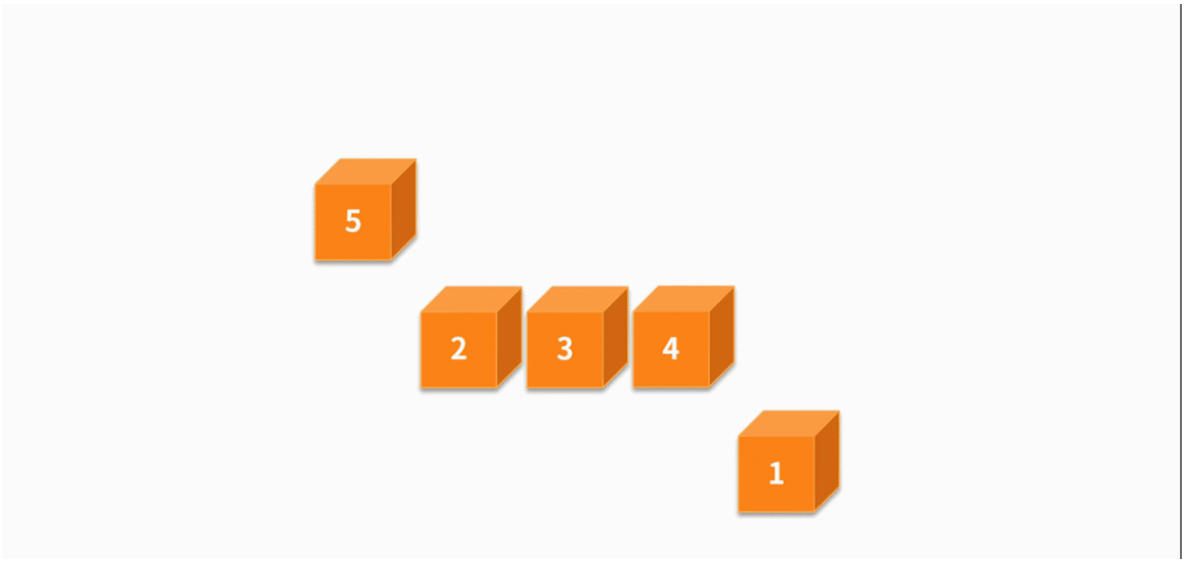
代码如下：
复制

```
public void s1_1() {
    int a[] = { 1, 2, 3, 4, 5 };
    int b[] = new int[5];
    for (int i = 0; i < a.length; i++) {
        b[i] = a[i];
    }
    for (int i = 0; i < a.length; i++) {
        b[a.length - i - 1] = a[i];
    }
    System.out.println(Arrays.toString(b));
}
```

这段代码的输入数据是 a，数据量就等于数组 a 的长度。代码中有两个 for 循环，作用分别是给b 数组初始化和赋值，其执行次数都与输入数据量相等。因此，代码的时间复杂度就是 O(n)+O(n)，也就是 O(n)。

空间方面主要体现在计算过程中，对于存储资源的消耗情况。上面这段代码中，我们定义了一个新的数组 b，它与输入数组 a 的长度相等。因此，空间复杂度就是 O(n)。

接着我们看一下第二种编码方法，它定义了缓存变量 tmp，接着通过一个 for 循环，从 0 遍历到a 数组长度的一半（即 len(a)/2）。每次遍历执行的是什么内容？就是交换首尾对应的元素。最后打印数组 a，得到结果。



代码如下：
复制

```
public void s1_2() {
   int a[] = { 1, 2, 3, 4, 5 };
   int tmp = 0;
   for (int i = 0; i < (a.length / 2); i++) {
      tmp = a[i];
      a[i] = a[a.length - i - 1];
      a[a.length - i - 1] = tmp;
   }
      System.out.println(Arrays.toString(a));
}
```

这段代码包含了一个 for 循环，执行的次数是数组长度的一半，时间复杂度变成了 O(n/2)。根据复杂度与具体的常系数无关的性质，这段代码的时间复杂度也就是 O(n)。

空间方面，我们定义了一个 tmp 变量，它与数组长度无关。也就是说，输入是 5 个元素的数组，需要一个 tmp 变量；输入是 50 个元素的数组，依然只需要一个 tmp 变量。因此，空间复杂度与输入数组长度无关，即 O(1)。

可见，对于同一个问题，采用不同的编码方法，对时间和空间的消耗是有可能不一样的。因此，工程师在写代码的时候，一方面要完成任务目标；另一方面，也需要考虑时间复杂度和空间复杂度，以求用尽可能少的时间损耗和尽可能少的空间损耗去完成任务。
时间复杂度与代码结构的关系

好了，通过前面的内容，相信你已经对时间复杂度和空间复杂度有了很好的理解。从本质来看，时间复杂度与代码的结构有着非常紧密的关系；而空间复杂度与数据结构的设计有关，关于这一点我们会在下一讲进行详细阐述。接下来我先来系统地讲一下时间复杂度和代码结构的关系。

代码的时间复杂度，与代码的结构有非常强的关系，我们一起来看一些具体的例子。

例 1，定义了一个数组 a = [1, 4, 3]，查找数组 a 中的最大值，代码如下：
复制

```
public void s1_3() {
   int a[] = { 1, 4, 3 };
   int max_val = -1;
   for (int i = 0; i < a.length; i++) {
      if (a[i] > max_val) {
         max_val = a[i];
      }
   }
   System.out.println(max_val);
}
```

这个例子比较简单，实现方法就是，暂存当前最大值并把所有元素遍历一遍即可。因为代码的结构上需要使用一个 for 循环，对数组所有元素处理一遍，所以时间复杂度为 O(n)。

例2，下面的代码定义了一个数组 a = [1, 3, 4, 3, 4, 1, 3]，并会在这个数组中查找出现次数最多的那个数字：
复制

```
public void s1_4() {
   int a[] = { 1, 3, 4, 3, 4, 1, 3 };
   int val_max = -1;
   int time_max = 0;
   int time_tmp = 0;
   for (int i = 0; i < a.length; i++) {
      time_tmp = 0;
      for (int j = 0; j < a.length; j++) {
         if (a[i] == a[j]) {
         time_tmp += 1;
      }
      if (time_tmp > time_max) {
         time_max = time_tmp;
         val_max = a[i];
      }
      }
   }
   System.out.println(val_max);
}
```

这段代码中，我们采用了双层循环的方式计算：第一层循环，我们对数组中的每个元素进行遍历；第二层循环，对于每个元素计算出现的次数，并且通过当前元素次数 time_tmp 和全局最大次数变量 time_max 的大小关系，持续保存出现次数最多的那个元素及其出现次数。由于是双层循环，这段代码在时间方面的消耗就是 n*n 的复杂度，也就是 $O(n^2)$。

在这里，我们给出一些经验性的结论：

> 一个顺序结构的代码，时间复杂度是 `O(1)`。
>
> 二分查找，或者更通用地说是采用分而治之的二分策略，时间复杂度都是 `O(logn)`。这个我们会在后续课程讲到。
>
> 一个简单的 `for` 循环，时间复杂度是 `O(n)`。
>
> 两个顺序执行的 `for` 循环，时间复杂度是 `O(n)+O(n)=O(2n)`，其实也是 `O(n)`。
>
> 两个嵌套的 `for` 循环，时间复杂度是 $O(n^2)$。

有了这些基本的结论，再去分析代码的时间复杂度将会轻而易举。

降低时间复杂度的必要性

很多新手的工程师，对降低时间复杂度并没有那么强的意识。这主要是在学校或者实验室中，参加的课程作业或者科研项目，普遍都不是实时的、在线的工程环境。

实际的在线环境中，用户的访问请求可以看作一个流式数据。假设这个数据流中，每个访问的平均时间间隔是 t。如果你的代码无法在 t 时间内处理完单次的访问请求，那么这个系统就会一波未平一波又起，最终被大量积压的任务给压垮。这就要求工程师必须通过优化代码、优化数据结构，来降低时间复杂度。

为了更好理解，我们来看一些数据。假设某个计算任务需要处理 10万 条数据。你编写的代码：

> 如果是 O(n²) 的时间复杂度，那么计算的次数就大概是 100 亿次左右。
>
> 如果是 O(n)，那么计算的次数就是 10万 次左右。
>
> 如果这个工程师再厉害一些，能在 O(log n) 的复杂度下完成任务，那么计算的次数就是 17 次左右（log 100000 = 16.61，计算机通常是二分法，这里的对数可以以 2 为底去估计）。

数字是不是一下子变得很悬殊？通常在小数据集上，时间复杂度的降低在绝对处理时间上没有太多体现。但在当今的大数据环境下，时间复杂度的优化将会带来巨大的系统收益。而这是优秀工程师必须具备的工程开发基本意识。

总结

OK，今天的内容到这儿就结束了。相信你对复杂度的概念有了进一步的认识。

复杂度通常包括时间复杂度和空间复杂度。在具体计算复杂度时需要注意以下几点。

> 它与具体的常系数无关，O(n) 和 O(2n) 表示的是同样的复杂度。
>
> 复杂度相加的时候，选择高者作为结果，也就是说 O(n²)+O(n) 和 O(n²) 表示的是同样的复杂度。
>
> O(1) 也是表示一个特殊复杂度，即任务与算例个数 n 无关。

复杂度细分为时间复杂度和空间复杂度，其中时间复杂度与代码的结构设计高度相关；空间复杂度与代码中数据结构的选择高度相关。会计算一段代码的时间复杂度和空间复杂度，是工程师的基本功。这项技能你在实际工作中一定会用到，甚至在参加互联网公司面试的时候，也是面试中的必考内容。

练习题

下面的练习题，请你独立思考。评估一下，如下的代码片段，时间复杂度是多少？

复制

```
for (i = 0; i < n; i++) {
  for (j = 0; j < n; j++) {
    for (k = 0; k < n; k++) {

    }
    for (m = 0; m < n; m++) {

    }
}

}
```