



下载APP



11 | 如何针对特定业务场景设计数据结构和高性能算法？

2021-06-10 尉刚强

《性能优化高手课》

课程介绍 >



讲述：尉刚强

时长 19:14 大小 17.62M



你好，我是尉刚强。今天这节课，我们来聊聊数据结构与算法。

可能在看到这节课的标题后，你会觉得有点儿奇怪：好像在平时的编码过程中，已经不太需要单独去关注数据结构和算法了，为什么还需要再根据场景设计数据结构和算法呢？

有这样的想法也无可厚非，因为我们确实会发现，在实际的业务领域内，需要我们开发人员直接设计数据结构与算法的机会越来越少。比如说：

在互联网服务场景中，性能开销主要集中在数据库 CRUD 操作上，所以很少会关注
内数据结构与算法设计的性能；



随着更多的核心业务算法内置到了芯片当中，对于从事嵌入式研发的工程师来说，主要工作就聚焦在了管理配置各种硬件资源上，因而并不会经常设计和使用数据结构与算法；

很多语言与标准库中已经内置了丰富的数据结构与算法，并不太需要开发人员手动去设计和开发；

.....

但事实上，我们以往所采用的性能优化手段（如热点代码分析优化、编译器优化等），对于系统性能的提升其实是按照百分制计算的，**这是一种线性粒度的性能提升**。我举个简单的例子，如果你在代码 Profiling 分析后，识别出了一个频繁调用的热点函数，将它内联或者优化后性能提升能够达到 3%~5%，就已经属于非常明显的优化提升了。

而通过数据结构与算法的设计来改进的系统性能，其获得的**性能收益很有可能是非线性**的，甚至可能是**指数级**的。就拿典型的查找问题来说，使用链表的遍历查找算法和数组向量的二分查找算法，在查找速度上性能可能会相差好多倍呢！


所以，合理设计数据结构与算法，对于软件系统的性能提升来说至关重要。

当然，可能你已经系统学习过了数据结构与算法，对相关知识原理都有比较深入的理解，也可能你对现有数据结构与算法的了解比较有限，但这都不会影响到你学习这节课的内容。另外，数据结构与算法包含的内容非常多，我不可能在一节课里介绍完整，市面上也已经有不少相关的课程书籍，我也没有必要再重复讲解。

这节课，我只聚焦于一个视角，那就是**根据业务开发中数据特征和计算逻辑的典型差异，从性能维度出发**，系统性地选择和设计数据结构与算法，以此帮助你在软件编码的过程中，更容易开发出高性能的软件。

那么接下来，我就从分析计算机软件执行原理开始，带你去了解选择不同的数据结构与算法，都会给系统性能带来什么影响。

数据结构与算法选择对性能的影响

谈起数据结构与算法的开销，可能第一时间你会想到  **大 O 标记表示法**。这的确是一个非常重要的算法复杂度表示方法，但这并不是衡量数据结构与算法性能的全部。

事实上，衡量数据结构与算法的实现复杂度，有几类比较常用的指标，包括最优时间开销、最差时间开销、平均时间开销、空间使用开销、摊销时间开销。

这里你可能要问了：平均时间开销是决定系统负载的一个关键指标，所以**是不是只要重点关注平均时间开销就可以了？**

实际上并不是，不同业务场景关注的指标都是不同的。我给你举几个例子，你就明白了：

针对内存资源极度受限的业务场景，对空间使用开销的关注度更高；

针对实时性要求非常高的场景，通常重点关注的是最差时间开销，而平均时间开销的意义并不大；

针对关注最大吞吐量的业务系统，这时的平均时间开销就变成最重要的指标了。

所以说，我们不要只关注平均时间开销，而是要关注对业务更有价值的指标。


那么接下来，你或许还会产生这样的疑问：**是不是只根据算法复杂度去选择算法就可以了？**

答案是不可以，相同的算法复杂度并不代表相同的性能。你要知道，性能还会受到软件编码实现方式、数据结构存储特性等多方面的影响。比如对于二分查找算法而言，基于循环遍历的实现与基于递归调用的实现，二者在性能上就会存在很大差异。

这里我给你举一个具体的例子。

注：虽然该示例中使用的是 C++ 语言和 STL 库，但解释的原理与具体语言无关。

首先我们来看一个类定义，其中包含了一个构造函数和比较运算符，代码如下：

 复制代码

```
1 struct Kv
2 {
3     char const *key;
4     unsigned int value;
5     Kv(const char *key, unsigned int value) : //构造函数
6         key(key), value(value)
7     {
8     }
9     bool operator==(Kv const &rht) // 比较运算符，当两个对象实例比较时使用
```

```
10     {  
11         return (strcmp(key, rht.key) == 0) && (value == rht.value);  
12     }  
13 };
```

那么针对这个类，我选择了两种数据结构进行记录，然后使用相同的查询算法来对比性能。

第一种数据结构类型为**数组**：

```
1 Kv arrayKvs[] = {...}
```

[复制代码](#)

然后，使用 STD 标准库中的线性查找算法，算法复杂度为 $O(n)$ ，如下所示：

```
1 Kv *result = std::find(std::begin(arrayKvs), std::end(arrayKvs), Kv("bbb", 2))
```

[复制代码](#)

第二种数据结构类型为**链表**：

```
1 std::list<Kv> listKvs;
```

[复制代码](#)

然后，这里我使用的也是标准库中的线性查找算法，算法复杂度为 $O(n)$ ，如下所示：

```
1 std::list<Kv>::iterator result = std::find(listKvs.begin(), listKvs.end(), Kv("
```

[复制代码](#)

到这里，你可以先思考一下，以上两种实现选择了相同的算法，实现复杂度一样，那么其性能表现是一致的吗？


显然是不一致的。当使用数组时，顺序访问数据的局部性高（数据内存地址是连续的）；而使用链表时，由于链表中的元素位置不相邻，而且数据不连续，就潜在导致了内存

Cache Miss (缓存未命中) 的概率显著增大，从而造成性能开销变大。

所以说，单纯的算法复杂度实际并不能准确地反映性能，数据结构对性能的影响也很大，而这部分并没有很好地在算法复杂度上体现出来。

OK，最后我们再来思考一个问题：**选择数据结构与算法之后，软件性能就决定了吗？**

答案也是否定的，因为数据结构和算法转换成的二级制代码执行是否高效，会受到很多因素的影响，比如编码实现、编译优化等。这里咱们再来分析一下上述业务场景中的比较逻辑：

 复制代码

```
1 bool Kv::operator==(Kv const &rht)
2 {
3     return (strcmp(key, rht.key) == 0) && (value == rht.value);
4     /*先比较字符串key，再比较数字value */
5 }
```

如果这个类的所有节点数据中，几乎所有的 value 值都不相同，而且 key 长度比较大，那么我们可以调整下代码中的比较顺序，因为整数比较的效率更高，还可以进一步提升性能。

总而言之，数据结构与算法的不同编码实现过程和方法，对软件的性能来说很重要，你在软件实现过程中，不仅要关注数据结构和算法的选择，还需要关注它的具体编码实现过程，这样才能真正开发出高性能的软件。

好了，在理解了数据结构和算法如何影响软件性能之后，下面咱们就进一步来探讨，如何根据领域数据的特征来选择对应的算法。

根据领域数据特征去选择算法

可能你之前已经发现了，我们从教科书上学习的数据结构与算法，通常都是标准的，但是在解决具体的业务问题时，我们需要处理的数据与算法却经常不是标准的。

怎么个不标准法儿呢？我认为主要体现在以下两个方面。

一方面，很多场景的领域数据是不标准的。

在大 O 标记法中，有一个假设是任意数据集上，通过软件所实现算法的运行时间基本相同的，但其实不少算法对数据的特性是非常敏感的。比如针对排序算法，如果待排序的数据集已经很接近有序状态，那么相比快速排序，选择直接插入排序算法的优势会更大。

我们来看一个例子。假设有一个数据集，它的特点如下：

1. 数据集规模为 10 万条；
2. 数据集完全乱序；
3. 这 10 万条数据中，有 1/3 数值小于 1000，另外 1/3 数值在 1000 到 2000 之间，还有 1/3 的数值是大于 2000 的。

那么现在，你需要对这个数据集进行完整排序，应该如何选择算法呢？

如果你没有关注到第 3 点特征，选择一个非常高效的排序算法后，其实也可以将算法复杂度降低到 $O(N \log_2 N)$ 。

但是当你意识到了第 3 点特征时，以上的排序过程就可以拆分为 3 个子数据集排序，然后再将排序结果合并到一起。而基于这种方式实现后，算法复杂度就可以降低到 $O(N/3 \log_2(N/3)) * 3 = O(N \log_2(N/3))$ ，从而就可以进一步提升性能了。

除此之外，针对上面这个业务场景，我们也很容易能想到，**采用并发模式**将数据集中的 3 个子数据集的排序过程，通过子任务并发起来，从而就能进一步降低业务的处理时延。

所以说，我们一定要认真挖掘领域数据的各种特性，只要挖掘的领域数据中的特性越多，其潜在的优化数据结构与算法的性能空间也就越大。

另一方面，业务算法通常是不标准的。

要知道，除了领域数据不标准之外，业务场景中的算法通常也不是标准的，所以我们就要根据具体的业务逻辑设计算法，才能最大化地提升性能，而不是仅仅照搬现成的标准算法实现。

我给你举个真实的例子，这是我曾经参与设计的一个资源调度子系统中的一个算法案例。不过为了方便理解，我把问题做了简化抽象，也就是如何在 1000 个用户中，根据优先级选择前 10 位用户进行资源分配。

那么碰到这个问题，你选择的算法方案会是什么呢？比如，是否会以下是两种方案：

方案 1：根据 1000 个用户的优先级进行全排序，然后选择前 10 个；

方案 2：使用冒泡排序算法，对 1000 个用户全遍历 10 次，选择前 10 个用户。

如果你选择方案 1，那么你将会浪费很多无谓的计算机资源，性能注定会非常差。而这个时候，你可能就很容易地想到了方案 2，觉得这个方案效率很高。那么方案 2 会是最佳的解决方案吗？

显然也不是，我们再来看看另外一个方案：

方案 3：首先选择前 10 个用户作为优先级最高的 10 个，然后对 1000 个用户全遍历一次，当某个用户的优先级超过这 10 个用户时，就更新至前 10 个用户中。

现在你可以来想想看，方案 3 在性能上是否会优于方案 2 呢？或者还有其他的算法实现吗？相信在认真思考了这些问题之后，你就迈出了基于业务选择和优化算法的第一步。

而实际上，对于这个案例来说，因为它的业务计算逻辑是比较特殊的，所以我们就需要针对典型计算逻辑，来单独设计算法实现逻辑。因此，最后我们选择了方案 3，使用针对前 10 位用户的资源分配，取得了比较好的性能效果。

OK，在根据业务逻辑定制化设计算法和实现之后，我们还需要综合权衡各种典型操作，才能选择出最符合业务逻辑的数据结构与算法，所以下面我们就具体来看看吧。

权衡综合各种操作选择数据结构与算法

我们知道，数据结构和算法之间通常是一对多的关系，在业务中，针对同一个数据结构可能会有排序、搜索等不同的算法业务逻辑。但是，**同一个数据结构在不同的算法上性能差异是比较大的，所以这时候，我们就需要去综合各种功能操作，再选择数据结构和算法。**

举个简单的例子，对于数据结构，很典型的方法就包括了删除、增加、查找元素等。当然数据结构还可以有很多其他方法，但是每种方法的操作频率都不一样，优先级也不同，比如说：

有些业务场景，插入和删除操作非常频繁，而查询操作很少，选择链表类数据结构保存会比较适合；

有些业务场景，插入和删除操作非常少，而查询操作很频繁，因此考虑选择数组类数据结构，系统的性能会比较好；

另外，当查询操作非常频繁时，可能还需要考虑对数据保持实时排序，从而进一步提升性能。

所以，为了更好地权衡，我们在设计数据结构与算法时，有时候甚至需要同时选择多种数据结构来记录数据。比如，把绝大部分的稳定数据保存在序列数组中，针对偶尔变更的数据记录保存在链表中，毕竟业务中并没有限定必须要使用相同的结构类型，保存相同类型的数据。

那么为了更直观地说明从业务操作的不同频率出发，选择数据结构与算法的意义，这里我就通过两种比较典型的数据库类型的设计原理，来给你举例说明下。

第一种是分析数据库，比如 ClickHouse。它绝大部分的操作请求都会集中在批量数据分析上，所以在设计时，就必须保证批量数据分析的性能，而这样就会造成数据的修改性能开销比较大。

第二种是文档数据库，比如 MongoDB 等。不过很多时候，我们为了追求单文档级别的 CRUD 性能，就不得已在批量数据分析计算性能上做出让步。

实际上，针对大型业务系统，我们通常需要选择多种数据存储方案进行数据冗余，从而综合满足各种业务场景下的性能需求。同样，**我们在业务内部设计数据结构与算法时，不能既要、又要，必须要作出取舍，只有在综合各种操作之后，才能权衡利弊进行选择。**

学会降低算法精确度提升性能

好了，最后我要带你掌握的知识点，就是要学会降低算法精确度，以此来进一步提升系统性能。

我们都知道，算法通常都是精确的、严格的，但在很多业务场景下，我们并不需要那么高的精确性。就拿我自己来说，我过去参与的诸多项目中，有过太多次降低算法精度与性能之间的权衡，所以接下来，我也用一个简单的例子来给你说明下原因。

假设现在有一个已经排序后的链表：

```
1 std::list<Kv> SortedKvs;
```

[复制代码](#)

然后，它在每个周期内都会有新数据输入，而在正常情况下，每插入一条数据都需要遍历寻找插入点，从而确保整个链表中的数据都是有序的。

这样通过分析业务发现，排序的正确性其实并不需要非常高，并且通过认真评估分析和验证后，我们发现其实可以把待插入数据首先放入链表的尾部，这样当积攒了 5 到 10 条待插入数据之后，再遍历一遍链表插入所有数据，通过这种实现方式，就可以将插入数据的运行开销降低数倍。

可见，**在实际的业务场景下，我们一定要根据性能要求标准来选择合适的算法精确度。**

OK，我们再来看看前面我介绍的那个资源调度例子，想一想，从 1000 个用户选择 10 个高优先级用户进行资源调度，还有没有其他降低算法精确度来提升系统性能的方案呢？

其实，我们可以将 1000 个用户拆分成 2 个组，每个组包含 500 个用户，然后交替在 2 个组内选择 10 个高优先级用户进行资源分配。这样通过在代码实现上的较少改动，就可以在性能上提升接近一倍。

但这里你要注意一点，就是你还需要**验证调整后的算法实现是否满足了业务需求**。有很多种降低算法精确度的实现方式，你需要准确分析并验证，选择背后的业务逻辑是否还能满足业务需求。

小结

在我的认知里，现成的数据结构和算法更像是一个工具库。相较于熟悉所有的数据结构与算法而言，我认为**更重要的是如何理解业务，这样在权衡利弊之下，选择并优化的数据结**

构与算法才会更加合适。

而且我在从事人工智能算法设计的工作期间，也更加深刻地印证了这一认识，因为深入理解所有人工智能算法的价值是相对有限的，也不现实。

所以在最后，我想告诉你的是，我并不是要反对你系统学习各种数据结构和算法，而是我希望你能够懂得如何理解业务，然后从业务出发，主动选择与优化数据结构和算法。

思考题

选择不同的数据结构和算法，它们在并发模式下的性能和串行模型的性能差别大吗？

欢迎给我留言，分享你的思考和看法。如果觉得有收获，也欢迎你把今天的内容分享给更多的朋友。

分享给需要的人，Ta订阅后你可得 **20** 元现金奖励

 赞 0  提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 10 | 性能模式（下）：如何解决核心的性能问题？

下一篇 12 | 我们要先实现业务功能，还是先优化代码？

更多学习推荐

Java 面试必考 300 题

最新汇总

限时免费领取 



精选留言

 写留言

由作者筛选后的优质留言将会公开显示，欢迎踊跃留言。