简介

ES 以其高性能的近实时的全文检索闻名，本文试图从其设计的底层原理和数据结构来分析 ES 到底是如何实现数据搜索的。本文的主要内容如下：

1. 索引原理
2. 磁盘IO与预读
3. 倒排索引
4. FST

索引原理

索引是加速数据查询的重要手段，其核心原理是通过不断的缩小想要获取数据的范围来筛选出最终想要的结果，同时把随机的事件变成顺序的事件。也就是说，有了这种索引机制，我们可以总是用同一种查找方式来锁定数据。

磁盘IO与预读

磁盘IO程序设计中非常高昂的操作，也是影响程序性能的重要因素，因此应当尽量避免过多的磁盘IO，有效的利用内存可以大大的提升程序的性能。在操作系统层面，发生一次IO时，不光把当前磁盘地址的数据，而是把相邻的数据也都读取到内存缓冲区内，局部预读性原理告诉我们，当计算机访问一个地址的数据的时候，与其相邻的数据也会很快被访问到。每一次IO读取的数据我们称之为一页(page)。具体一页有多大数据跟操作系统有关，一般为4k或8k，也就是我们读取一页内的数据时候，实际上才发生了一次IO，这个理论对于索引的数据结构设计非常有帮助。

倒排索引

当数据写入 ES 时，数据将会通过 **分词** 被切分为不同的 **term**，ES 将 term 与其对应的文档列表建立一种映射关系，这种结构就是 **倒排索引**。如下图所示：

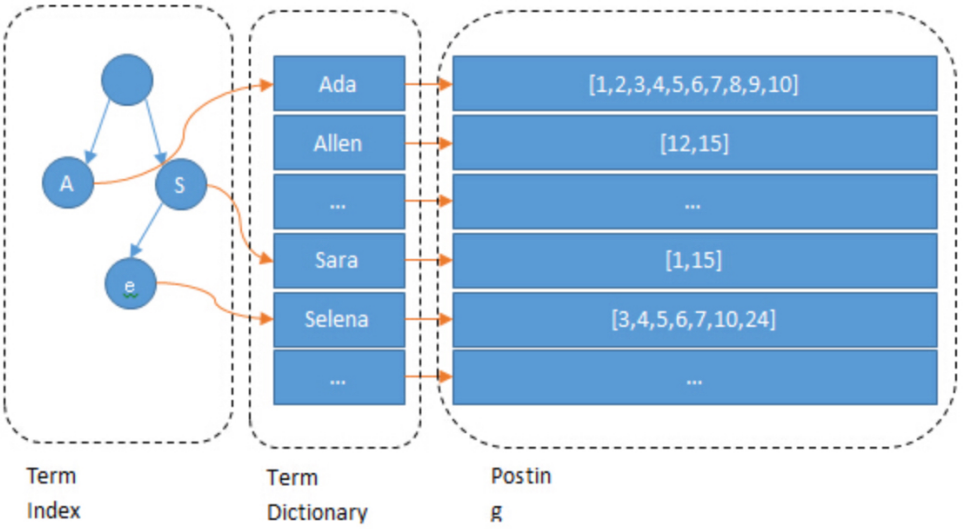
以英文为例，下面是要被索引的文本：

- $T_0 =$ `"it is what it is"`
- $T_1 =$ `"what is it"`
- $T_2 =$ `"it is a banana"`

我们就能得到下面的反向文件索引：

```
"a":      {2}
"banana": {2}
"is":     {0, 1, 2}
"it":     {0, 1, 2}
"what":   {0, 1}
```

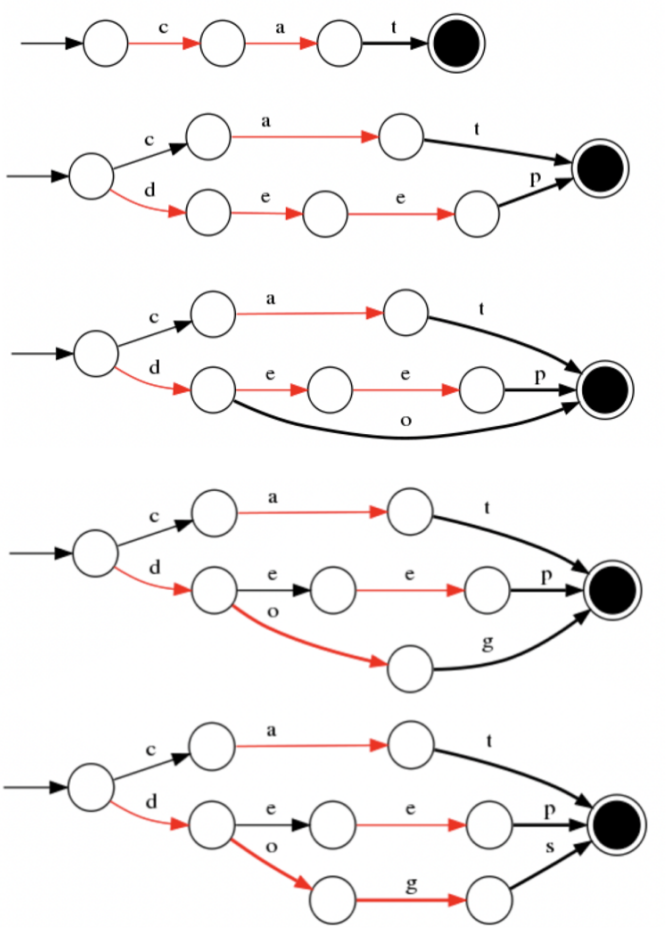
为了进一步提升索引的效率，ES 在 term 的基础上利用 term 的前缀或者后缀构建了 term index, 用于对 term 本身进行索引，ES 实际的索引结构如下图所示：



这样当我们去搜索某个关键词时，ES 首先根据它的前缀或者后缀迅速缩小关键词的在 term dictionary 中的范围，大大减少了磁盘IO的次数。

FST

当数据量不断增长，内存中无法存储完整的 Term index，此时必须将数据存储再磁盘上，这样就增加了对磁盘的IO操作。为了尽量多的将 Term index 加载到内存中，ES 采用了 FST 数据结构对 Term index 进行压缩。FST 数据结构有两个优点：1）空间占用小。通过对词典中单词前缀和后缀的重复利用，压缩了存储空间；2）查询速度快，O(len(str))字符长度的查询时间复杂度。当使用FST插入cat、deep、do、dog、dogs等词时过程如下：


c
a
t
c
a
t
d
e
e
p
c
a
t
d
e
e
p
o
c
a
t
d
e
e
p
o
g
c
a
t
d
e
e
p
o
s
g