纯干货 | ES性能调优策略

原创 2017-09-04 胡晓东 中兴大数据

文 | 胡晓东

性能是大数据人永恒的追求，本文总结了ES性能调优的一些实践经验，一起来看下吧！

索引性能调优

1. 使用批量请求，批量索引的效率肯定比单条索引的效率要高。
2. 调整批量请求的条数和每条文档的大小。现场上线之前应该仔细按照自身业务场景进行压力测试，慢慢尝试获得合适的批量请求的条数和大小。
3. 调整内存大小。当频繁出现full gc后考虑增加内存大小，但是堆内存和堆外内存不要超过32G。
4. 调整写入的线程数和队列大小。不过线程数最大不能超过33个（es控制死）。
5. 多盘的话，尽量将所有盘都用上。否则磁盘可能是瓶颈。
6. 如果你的搜索结果不需要近实时的准确度，考虑把每个索引的 index.refresh_interval 改到 30s或者更大。如果你是在做大批量导入，导入期间你可以通过设置这个值为 -1 关掉刷新。别忘记在完工的时候重新开启它。

7. 客户端不生成ID或者使用对Lucene友好的ID。包括零填充序列 ID、UUID-1 和纳秒；这些 ID 都是有一致的，压缩良好的序列模式。相反的，像 UUID-4 这样的 ID，本质上是随机的，压缩比很低，会明显拖慢 Lucene。

8. 将写translog从同步改成异步能提高索引效率。但是存在丢失数据的风险。

9. 提高translog flush的大小，以减少merge的次数。

10. 如果你在做大批量导入，考虑通过设置 index.number_of_replicas:0关闭副本。文档在复制的时候，整个文档内容都被发往副本节点，然后逐字的把索引过程重复一遍。这意味着每个副本也会执行分析、索引以及可能的合并过程。相反，如果你的索引是零副本，然后在写入完成后再开启副本，恢复过程本质上只是一个字节到字节的网络传输。相比重复索引过程，这个算是相当高效的了。

11. indices.memory.index_buffer_size调整内存缓存的文档大小，最多给512M。大于这个值会触发refresh。默认值是JVM的内存10%。但是是所有切片共享。

12. 使用更好的硬件，比如磁盘使用SSD或者做raid0。

13. 测试原则：先单线程增加批量文档大小，直到性能无法增加，再增加线程数。当出现Reject之后代表es资源饱和，需要定位并找到瓶颈点。

搜索性能调优

- **扩大堆外内存**

ES非常依赖文件系统缓存，以便快速搜索。一般来说，应该至少确保物理上有一半的可用内存分配到文件系统缓存。

- **使用更好的硬件，比如使用SSD或者做raid0、使用更好的CPU**

- **优化文档模型**

应对文档进行建模，合理的文档模型对搜索至关重要。尤其是要避免关联查询。嵌套查询可以使查询慢好几倍，父子关系查询甚至可以使查询慢好几百倍。所以文档模型能够避免关联的话，能够极大的提高搜索性能。

- **预索引数据**

例如，如果所有文档都有价格字段，大多数查询都会在固定的范围中运行 range aggregations，则可以通过将范围预索引到索引中并使用term aggregations来更快地进行此聚合。

例如，如果文档如下：

```
PUT index/type/1
{
    "designation": "spoon",
    "price": 13
}
```

搜索如下：

```
GET index/_search
{
    "aggs": {
        "price_ranges": {
            "range": {
                "field": "price",
                "ranges": [
                    { "to": 10 },
                    { "from": 10, "to": 100 },
                    { "from": 100 }
                ]
            }
        }
    }
}
```

在索引的时候，文档可以扩展一个price_range字段，字段类型是keyword：

```
PUT index
{
   "mappings": {
      "type": {
         "properties": {
            "price_range": {
               "type": "keyword"
            }
         }
      }
  }
}
PUT index/type/1
{
   "designation": "spoon",
   "price": 13,
   "price_range": "10-100"
}
```

搜索的时候就可以对新增的price_range进行汇聚，而不需要做一个 range aggregation。

```
GET index/_search
{
   "aggs": {
      "price_ranges": {
         "terms": {
            "field": "price_range"
         }
      }
   }
}
```

- **优化映射**

一些数据是数字不代表它就应该被映射integer或者 long。典型的场景，一些在数据库中作为唯一标示的数字，就应该被映射成keyword，更优于integer或者long。

- **避免使用scripts**

- **搜索舍去精度的日期**

现在使用的日期字段的查询通常不可缓存，因为正在匹配的范围始终会更改。然而，切换到舍去精度的日期在用户体验方面通常是可接受的，并且具有更好地利用查询高速缓存的好处。

如下：

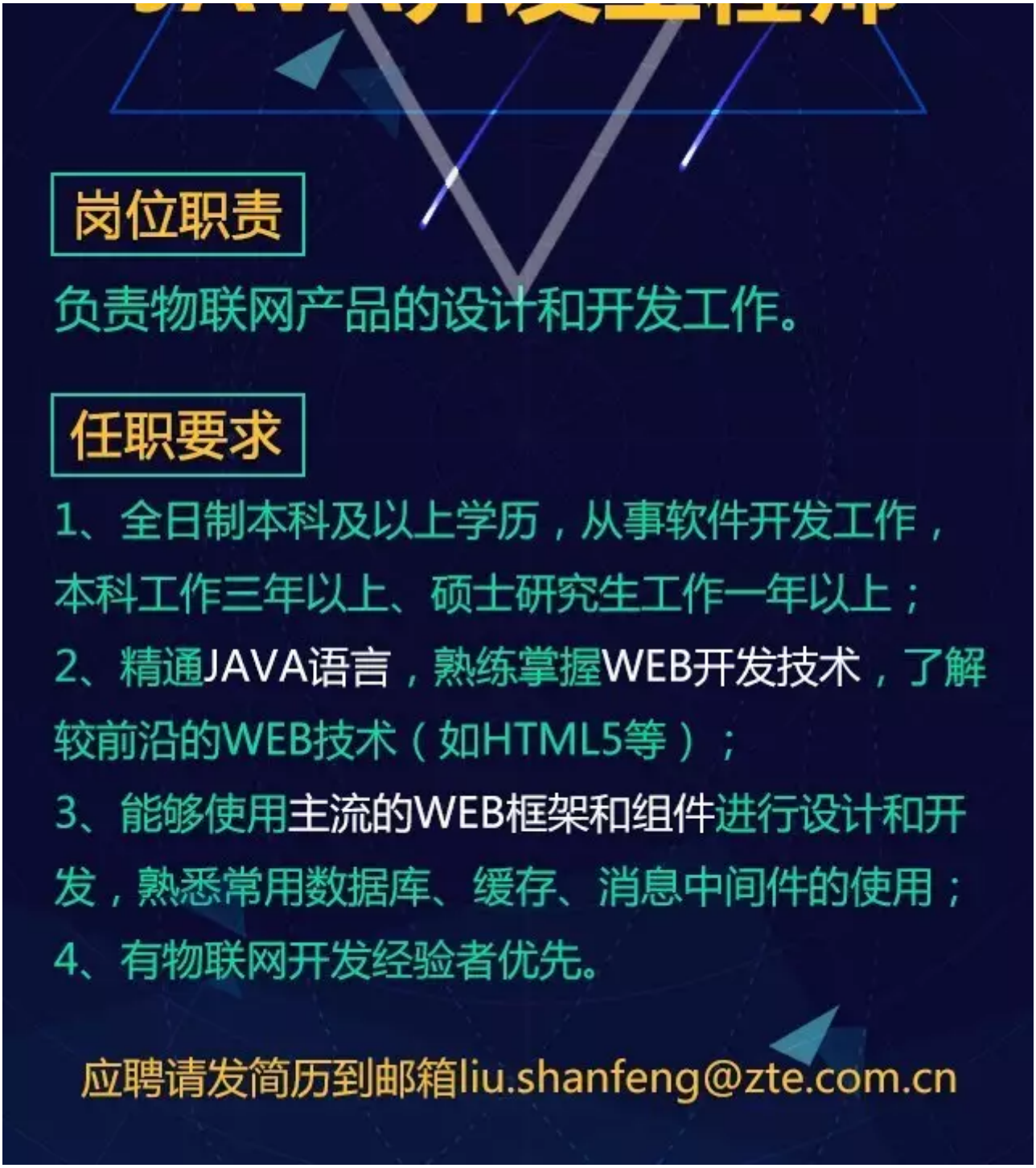
```
PUT index/type/1
{
   "my_date": "2016-05-11T16:30:55.328Z"
}

GET index/_search
{
   "query": {
      "constant_score": {
         "filter": {
            "range": {
               "my_date": {
                  "gte": "now-1h",
                  "lte": "now"
               }
            }
         }
      }
   }
}
```

上述查询可以被替换成以下：

```
GET index/_search
{
   "query": {
      "constant_score": {
         "filter": {
            "range": {
               "my_date": {
                  "gte": "now-1h/m",
                  "lte": "now/m"
               }
            }
         }
      }
   }
}
```

在这种情况下，我们四舍五入到分钟，所以如果当前时间是16:31:29，范围查询将匹配my_date字段的值在15:31:00到16:31:59之间的所有内容。如果有几个用户在同一分钟内运行包含此范围的查询，查询缓存可以帮助你加快速度。用于舍入的间隔越长，缓存被命中的概率就越大，但要注意的是，过多的舍入也可能会降低用户体验。


岗位职责
负责物联网产品的设计和开发工作。
任职要求
1、全日制本科及以上学历，从事软件开发工作，本科工作三年以上、硕士研究生工作一年以上；
2、精通JAVA语言，熟练掌握WEB开发技术，了解较前沿的WEB技术（如HTML5等）；
3、能够使用主流的WEB框架和组件进行设计和开发，熟悉常用数据库、缓存、消息中间件的使用；
4、有物联网开发经验者优先。
应聘请发简历到邮箱liu.shanfeng@zte.com.cn