父-子　系文

父-子　系文　　在　　上　似于　nested　model　　：允　将一个　象　体和　外一个　象　体　起来。而　　型的主要区　是：在 nested objects 文　中，所有　象都是在同一个文　中，而在父-子　系文　中，父　象和子　象都是完全独立的文　。

父-子　系的主要作用是允　把一个　type　的文　和　外一个　type　的文　　起来，　成一　多的　系：一个父文　可以　　多个子文　。与 nested objects 相比，父-子　系的主要　　有：

- 更新父文　　，不会重新索引子文　。
- 　建，修改或　除子文　　，不会影　父文　或其他子文　。　一点在　　景下尤其有用：子文　数　量　多，并且子文　　建和修改的　率高　。
- 子文　可以作　搜索　果独立返回。

Elasticsearch　　了一个父文　和子文　的映射　系，得益于　个映射，父-子文　　操作非常快。但是　个映射也　父-子文　　系有个限制条件：父文　和其所有子文　，都必　要存　在同一个分片中。

父-子文　ID映射存　在　[docvalues]　中。当映射完全在内存中　，　[docvalues]　提供　映射的快速　理能力，　一方面当映射非常大　，可以通　溢出到磁　提供足　的　展能力

父-子　系文　映射

建立父-子文　映射　系　只需要指定某一个文　　type 是　一个文　　type 的父　。　　系可以在如下　个　　点　置：1）　建索引　；2）在子文　type　建之前更新父文　的 mapping。

　例　明，有一个公司在多个城市有分公司，并且　一个分公司下面都有很多　工。有　　的需求：按照分　公司、　工的　度去搜索，并且把　工和他　工作的分公司　系起来。　　需求，用嵌套模型是无法　的。当然，如果使用　application-side-joins　或者　data　denormalization　也是可以　　的，但是　了演示的目的，在　里我　使用父-子文　。

我　需要告　Elasticsearch，在　建　工 employee 文　type　，指定分公司 branch 的文　type　其父　。

```
PUT /company
{
  "mappings": {
    "branch": {},
    "employee": {
      "_parent": {
        "type": "branch" ①
      }
    }
  }
}
```

① employee 文　是 branch 文　的子文　。

建父-子文　索引

父文　　建索引与　普通文　　建索引没有区　。父文　并不需要知道它有　些子文　。

```
POST /company/branch/_bulk
{ "index": { "_id": "london" }}
{ "name": "London Westminster", "city": "London", "country": "UK" }
{ "index": { "_id": "liverpool" }}
{ "name": "Liverpool Central", "city": "Liverpool", "country": "UK" }
{ "index": { "_id": "paris" }}
{ "name": "Champs Élysées", "city": "Paris", "country": "France" }
```

建子文　　，用　必　要通　parent 参数来指定　子文　的父文　ID：

```
PUT /company/employee/1?parent=london ①
{
  "name":  "Alice Smith",
  "dob":   "1970-10-24",
  "hobby": "hiking"
}
```

① 当前 employee 文　的父文　ID 是 london 。

父文　　ID　　有　个作用：　建了父文　和子文　之　的　系，并且保　了父文　和子文　都在同一个分片上。

在　[routing-value]　中，我　解　了　Elasticsearch　如何通　路由　来决定　文　属于一个分片，路由　　文　的 _id 。分片路由的　算公式如下：

```
shard = hash(routing) % number_of_primary_shards
```

如果指定了父文　的　ID，那　就会使用父文　的　ID　行路由，而不会使用当前文　_id　。也就是　，如果父文　和子文　都使用相同的　行路由，那　父文　和子文　都会　定分布在同一个分片上。

在　行　文　的　求　需要指定父文　的　ID，　文　求包括：通　GET　求　取一个子文　；　建、更新或　除一个子文　。而　行搜索　求　是不需要指定父文　的ID，　是因　搜索　求是向一个索引中的所有分片　起　求，而　文　的操作是只会向存　文　的分片　送　求。因此，如果操作　个子文　不指定父文　的 ID，那　很有可能会把　求　送到　的分片上。

父文　的 ID　在 bulk API 中指定

```
POST /company/employee/_bulk
{ "index": { "_id": 2, "parent": "london" }}
{ "name": "Mark Thomas", "dob": "1982-05-16", "hobby": "diving" }
{ "index": { "_id": 3, "parent": "liverpool" }}
{ "name": "Barry Smith", "dob": "1979-04-01", "hobby": "hiking" }
{ "index": { "_id": 4, "parent": "paris" }}
{ "name": "Adrien Grand", "dob": "1987-05-11", "hobby": "horses" }
```

WARNING	如果 想要改 一个子文 的 parent ， 通 更新 个子文 是不 的，因 新的父文 有可能在 外一个分片上。因此， 必 要先把子文 除，然后再重新索引 个子文 。

通 子文 父文

has_child 的 和 可以通 子文 的内容来 父文 。例如，我 根据如下 ，可 出所有80后 工所在的分公司：

```
GET /company/branch/_search
{
  "query": {
    "has_child": {
      "type": "employee",
      "query": {
        "range": {
          "dob": {
            "gte": "1980-01-01"
          }
        }
      }
    }
  }
}
```

似于 nested query ，has_child 可以匹配多个子文 ，并且 一个子文 的 分都不同。但是由于 一个子文 都 有 分， 些 分如何 成父文 的 得分取决于 score_mode 个参数。参数有多 取 策略： none ，会忽略子文 的 分，并且会 父文 分 置 1.0 ； 除此以外 可以 置成 avg 、 min 、 max 和 sum 。

下面的 将会同 返回 london 和 liverpool ，不 由于 Alice Smith 要比 Barry Smith 更加匹配 条件，因此 london 会得到一个更高的 分。

```
GET /company/branch/_search
{
  "query": {
    "has_child": {
      "type":       "employee",
      "score_mode": "max",
      "query": {
        "match": {
          "name": "Alice Smith"
        }
      }
    }
  }
}
```

TIP

score_mode 的 none ，会 著地比其模式要快， 是因 Elasticsearch不需要 算 一个子文 的 分。只有当 真正需要 心 分 果 ，才需要 source_mode ，例如 成 avg 、 min 、 max 或 sum 。

min_children 和 max_children

has_child 的 和 都可以接受 个参数：min_children 和 max_children 。 使用 个参数 ，只有当子文 数量在指定 内 ，才会返回父文 。

如下 只会返回至少有 个雇 的分公司：

```
GET /company/branch/_search
{
  "query": {
    "has_child": {
      "type":         "employee",
      "min_children": 2, ①
      "query": {
        "match_all": {}
      }
    }
  }
}
```

① 至少有 个雇 的分公司才会符合 条件。

有 min_children 和 max_children 参数的 has_child 或 ，和允 分的 has_child 的性能非常接近。

has_child Filter

has_child 和 在 行机制上 似，区 是 has_child 不支持 source_mode 参数。has_child 用于 内容—如内部的一个 filtered —和其他 行 似：包含或者排除，但没有 行 分。

has_child 的 果没有被 存，但是 has_child 内部的 方法 用于通常的 存 。

通 父文 子文

然 nested 只能返回最 的文 ，但是父文 和子文 本身是彼此独立并且可被 独 的。我 使用 has_child 句可以基于子文 来 父文 ，使用 has_parent 句可以基于父文 来 子文 。

has_parent 和 has_child 非常相似，下面的 将会返回所有在 UK 工作的雇 ：

```
GET /company/employee/_search
{
  "query": {
    "has_parent": {
      "type": "branch", ①
      "query": {
        "match": {
          "country": "UK"
        }
      }
    }
  }
}
```

① 返回父文 type 是 branch 的所有子文

has_parent 也支持 score_mode 个参数，但是 参数只支持 ： none （ ）和 score 。 个子文 都只有一个父文 ，因此 里不存在将多个 分 一个的情况， score_mode 的取 score 和 none 。

不 分的 has_parent

当 has_parent 用于非 分模式（比如 filter 句） ， score_mode 参数就不再起作用了。因 模式只是 地包含或排除文 ，没有 分，那 score_mode 参数也就没有意 了。

子文 聚合

在父-子文 中支持 子文 聚合， 一点和 [nested-aggregation] 似。但是， 于父文 的聚合 是不支持的（和 reverse_nested 似）。

我 通 下面的例子来演示按照国家 度 看最受雇 迎的 余 好：

```
GET /company/branch/_search
{
  "size" : 0,
  "aggs": {
    "country": {
      "terms": { ①
        "field": "country"
      },
      "aggs": {
        "employees": {
          "children": { ②
            "type": "employee"
          },
          "aggs": {
            "hobby": {
              "terms": { ③
                "field": "hobby"
              }
            }
          }
        }
      }
    }
  }
}
```

① country 是 branch 文 的一个字段。

② 子文 聚合 通 employee type 的子文 将其父文 聚合在一起。

③ hobby 是 employee 子文 的一个字段。

祖 与 系

父子 系可以延展到更多代 系，比如生活中 与祖 的 系 — 唯一的要求是 足 些系的文 必 在同一个分片上被索引。

我 把上一个例子中的 country 型 定 branch 型的父 ：

```
PUT /company
{
  "mappings": {
    "country": {},
    "branch": {
      "_parent": {
        "type": "country" ①
      }
    },
    "employee": {
      "_parent": {
        "type": "branch" ②
      }
    }
  }
}
```

① branch 是 country 的子 。

② employee 是 branch 的子 。

country 和 branch 之 是一 的父子 系，所以我 的 操作 与之前保持一致：

```
POST /company/country/_bulk
{ "index": { "_id": "uk" }}
{ "name": "UK" }
{ "index": { "_id": "france" }}
{ "name": "France" }

POST /company/branch/_bulk
{ "index": { "_id": "london", "parent": "uk" }}
{ "name": "London Westmintster" }
{ "index": { "_id": "liverpool", "parent": "uk" }}
{ "name": "Liverpool Central" }
{ "index": { "_id": "paris", "parent": "france" }}
{ "name": "Champs Élysées" }
```

parent ID 使得 一个 branch 文 被路由到与其父文 country 相同的分片上 行操作。然而，当我 使用相同的方法来操作 employee 个 文 ，会 生什 ？

```
PUT /company/employee/1?parent=london
{
  "name":  "Alice Smith",
  "dob":   "1970-10-24",
  "hobby": "hiking"
}
```

employee 文 的路由依 其父文 ID — 也就是 <code>london</code> — 但是

<code>london</code> 文 的路由却依 其本身的 父文 ID — 也就是 <code>uk</code> 。此 情况下， 文 很有可能最 和父 、祖 文 不在同一分片上， 致不足祖 和 文 必 在同一个分片上被索引的要求。

解决方案是添加一个 外的 routing 参数，将其 置 祖 的文 ID ，以此来保 三代文 路由到同一个分片上。索引 求如下所示：

```
PUT /company/employee/1?parent=london&routing=uk ①
{
  "name":  "Alice Smith",
  "dob":   "1970-10-24",
  "hobby": "hiking"
}
```

① routing 的 会取代 parent 的 作 路由 。

parent 参数的 然可以 employee 文 与其父文 的 系，但是 routing 参数保 文 被存到其父 和祖 的分片上。routing 在所有的文 求中都要添加。

合多代文 行 和聚合是可行的，只需要一代代的 行 定即可。例如，我 要 到喜 足的雇者的城市，此 需要 合 country 和 branch，以及 branch 和 employee：

```
GET /company/country/_search
{
  "query": {
    "has_child": {
      "type": "branch",
      "query": {
        "has_child": {
          "type": "employee",
          "query": {
            "match": {
              "hobby": "hiking"
            }
          }
        }
      }
    }
  }
}
```

使用中的一些建

当文 索引性能 比 性能重要的 候，父子 系是非常有用的，但是它也是有巨大代 的。其 速度会比同等的嵌套 慢5到10倍!

全局序号和延

父子 系使用了全局序数 来加速文 的 合。不管父子 系映射是否使用了内存 存或基于硬 的 doc values，当索引 更 ，全局序数要重建。

一个分片中父文 越多，那 全局序数的重建就需要更多的 。父子 系更 合于父文 少、子文 多的情况。

全局序数 情况下是延 建的：在refresh后的第一个父子 会触 全局序数的 建。而 个 建会致用 使用 感受到明 的 。 可以使用全局序数 加 来将全局序数 建的 由query 段移到refresh 段， 置如下：

```
PUT /company
{
  "mappings": {
    "branch": {},
    "employee": {
      "_parent": {
        "type": "branch",
        "fielddata": {
          "loading": "eager_global_ordinals" ①
        }
      }
    }
  }
}
```

① 在一个新的段可搜索前，`_parent` 字段的全局序数会被 建。

当父文 多 ，全局序数的 建会耗 很多 。此 可以通 加 `refresh_interval` 来 少 refresh 的次数，延 全局序数的有效 ， 也很大程度上 小了全局序数 秒重建的cpu消耗。

多代使用和

多代文 的 合 （ 看 祖 与 系） 然看起来很吸引人，但必 考 如下的代 ：

- 合越多，性能越差。
- 一代的父文 都要将其字符串 型的 `_id` 字段存 在内存中， 会占用大量内存。

当 考 父子 系是否 合 有 系模型 ， 考 下面 些建 ：

- 尽量少地使用父子 系， 在子文 多于父文 使用。
- 避免在一个 中使用多个父子 合 句。
- 在 has_child 中使用 filter 上下文，或者 置 score_mode none 来避免 算文 得分。
- 保 父 IDs 尽量短，以便在 doc values 中更好地 ，被 入 占用更少的内存。

最重要的是: 先考 下我 之前 的其他方式来 到父子 系的效果。