1. **Mapping：**

(1) 概念：mapping就是ES数据字段field的type元数据，ES在创建索引的时候，dynamic mapping会自动为不同的数据指定相应mapping，mapping中包含了字段的类型、搜索方式（exact value或者full text）、分词器等。

(2) 查看mapping

GET /product/_mappings

(3) Dynamic mapping

1. “Elasticsearch”：text/keyword
2. 123456 => long ？为什么不是integer
3. 123.123 => double
4. true false => boolean
5. 2020-05-20 => date

为啥price是long类型而不是integer？因为es的mapping_type是由JSON分析器检测数据类型，而Json没有隐式类型转换（integer=>long or float=> double）,所以dynamic mapping会选择一个比较宽的数据类型。

(4) 搜索方式：

1. exact value 精确匹配：在倒排索引过程中，分词器会将field作为一个整体创建到索引中，
2. full text全文检索：分词、近义词同义词、混淆词、大小写、词性、过滤、时态转换等（normaliztion）

(5) ES数据类型：

1. 核心类型

1) **数字类型**：
   a. long, integer, short, byte, double, float, half_float, scaled_float
   b. 在满足需求的情况下，尽可能选择范围小的数据类型。
2) **字符串：string**：
   a. **keyword**：适用于索引结构化的字段，可以用于过滤、排序、聚合。keyword类型的字段只能通过精确值（exact value）搜索到。Id应该用keyword
   text：当一个字段是要被全文搜索的，比如Email内容、产品描述，这些字段应该使用text类型。设置text类型以后，字段内容会被分析，在生成倒排索引以前，字符串会被分析器分成一个一个**词项。text类型的字段不用于排序，很少用于聚合。**（解释一下为啥不会为text创建索引：**字段数据会占用大量堆空间，尤其是在加载高基数text字段时。字段数据一旦加载到堆中，就在该段的生命周期内保持在那里。同样，加载字段数据是一个昂贵的过程，可能导致用户遇到延迟问题。这就是默认情况下禁用字段数据的原因**）
   b. 有时，在同一字段中同时具有全文本（text）和关键字（keyword）版本会很有用：一个用于全文本搜索，另一个用于聚合和排序。
3) **date**（时间类型）：exact value
4) 布尔类型：boolean
5) binary（二进制）：binary
6) range（区间类型）：integer_range、float_range、long_range、double_range、date_range

2. 复杂类型：
   1) Object：用于单个JSON对象
   2) Nested：用于JSON对象数组
3. 地理位置：
   1) Geo-point：纬度/经度积分
   2) Geo-shape：用于多边形等复杂形状
4. 特有类型：
   1) IP地址：ip 用于IPv4和IPv6地址
   2) Completion：提供自动完成建议
   3) Tocken_count：计算字符串中令牌的数量
   4) Murmur3：在索引时计算值的哈希并将其存储在索引中
   5) Annotated-text：索引包含特殊标记的文本（通常用于标识命名实体）

6) Percolator：接受来自query-dsl的查询
7) Join：为同一索引内的文档定义父/子关系
8) Rank features：记录数字功能以提高查询时的点击率。
9) Dense vector：记录浮点值的密集向量。
10) Sparse vector：记录浮点值的稀疏向量。
11) Search-as-you-type：针对查询优化的文本字段，以实现按需输入的完成
12) Alias：为现有字段定义别名。
13) Flattened：允许将整个JSON对象索引为单个字段。
14) Shape：shape 对于任意笛卡尔几何。
15) Histogram：histogram 用于百分位数聚合的预聚合数值。
16) Constant keyword：keyword当所有文档都具有相同值时的情况的 专业化。

5. Array（数组）：在Elasticsearch中，数组不需要专用的字段数据类型。默认情况下，任何字段都可以包含零个或多个值，但是，数组中的所有值都必须具有相同的数据类型。
6. ES 7新增：
   1) Date_nanos：date plus 纳秒
   2) Features：
   3) Vector：as

(6) 手工创建mapping

```
PUT /product
{
  "mappings": {
    "properties": {
      "field": {
        "mapping_parameter": "parameter_value"
      }
    }
  }
}
```

(7) Mapping parameters

1. **index**：是否对创建对当前字段创建索引，默认true，如果不创建索引，该字段不会通过索引被搜索到,但是仍然会在source元数据中展示
2. `analyzer:`指定分析器（character filter、tokenizer、Token filters）。
3. boost：对当前字段相关度的评分权重，默认1
4. **coerce**：是否允许强制类型转换 true “1”=> 1 false “1”=< 1
5. copy_to：

```
"field": {
    "type": "text",
    "copy_to": "other_field_name"
},
```

6. **doc_values**：为了提升排序和聚合效率，默认true，如果确定不需要对字段进行排序或聚合，也不需要通过脚本访问字段值，则可以禁用doc值以节省磁盘空间（不支持text和annotated_text）
7. dynamic：控制是否可以动态添加新字段
   1) true 新检测到的字段将添加到映射中。（默认）
   2) false 新检测到的字段将被忽略。这些字段将不会被索引，因此将无法搜索，但仍会出现在_source返回的匹配项中。这些字段不会添加到映射中，必须显式添加新字段。
   3) strict 如果检测到新字段，则会引发异常并拒绝文档。必须将新字段显式添加到映射中
8. **eager_global_ordinals：用于聚合的字段上，优化聚合性能。**
   1) Frozen indices（冻结索引）：有些索引使用率很高，会被保存在内存中，有些使用率特别低，宁愿在使用的时候重新创建，在使用完毕后丢弃数据，Frozen indices的数据命中频率小，不适用于高搜索负载，数据不会被保存在内存中，堆空间占用比普通索引少得多，Frozen indices是只读的，请求可能是秒级或者分钟级。

eager_global_ordinals不适用于Frozen indices

9. enable：是否创建倒排索引，可以对字段操作，也可以对索引操作，如果不创建索引，让然可以检索并在_source元数据中展示，谨慎使用，该状态无法修改。

```
PUT my_index{
  "mappings": {
    "enabled": false
  }}

PUT my_index{
  "mappings": {
    "properties": {
      "session_data": {
        "type": "object",
        "enabled": false
      }
    }
  }}
```

10. **fielddata**：查询时**内存**数据结构，在首次用当前字段聚合、排序或者在脚本中使用时，需要字段为fielddata数据结构，并且创建倒排索引保存到堆中
11. **fields**：给field创建多字段，用于不同目的（全文检索或者聚合分析排序）
12. format：格式化

```
"date": {
      "type":  "date",
      "format": "yyyy-MM-dd"
 }
```

13. `ignore_above: 超过长度将被忽略`
14. ignore_malformed：忽略类型错误

```
PUT my_index{
  "mappings": {
    "properties": {
      "number_one": {
        "type": "integer",
        "ignore_malformed": true
      },
      "number_two": {
        "type": "integer"
      }
    }
  }}
PUT my_index/_doc/1{
  "text":       "Some text value",
  "number_one": "foo" }  //虽然有异常 但是不抛出
PUT my_index/_doc/2{
  "text":       "Some text value",
  "number_two": "foo" }  //数据格式不对
```

15. index_options：控制将哪些信息添加到反向索引中以进行搜索和突出显示。仅用于text字段
16. Index_phrases：提升exact_value查询速度，但是要消耗更多磁盘空间
17. Index_prefixes：前缀搜索
    1) min_chars：前缀最小长度，>0，默认2（包含）
    2) max_chars：前缀最大长度，<20，默认5（包含）

```
"index_prefixes": {
        "min_chars" : 1,
        "max_chars" : 10
       }
```

18. meta：附加元数据
19. normalizer：
20. norms：是否禁用评分（在filter和聚合字段上应该禁用）。
21. null_value：为null值设置默认值

"null_value": "NULL"

22. position_increment_gap:
23. proterties：除了mapping还可用于object的属性设置
24. **search_analyzer**：设置单独的查询时分析器：

```
PUT my_index{
  "settings": {
    "analysis": {
      "filter": {
        "autocomplete_filter": {
          "type": "edge_ngram",
          "min_gram": 1,
          "max_gram": 20
        }
      },
      "analyzer": {
        "autocomplete": {
          "type": "custom",
          "tokenizer": "standard",
          "filter": [
            "lowercase",
            "autocomplete_filter"
          ]
        }
      }
    }
  },
  "mappings": {
    "properties": {
      "text": {
        "type": "text",
        "analyzer": "autocomplete",
        "search_analyzer": "standard"
      }
    }
  }}
PUT my_index/_doc/1{
  "text": "Quick Brown Fox" }
GET my_index/_search{
  "query": {
    "match": {
      "text": {
        "query": "Quick Br",
        "operator": "and"
      }
    }
  }}
```

25. similarity：为字段设置相关度算法，支持BM25、claassic（TF-IDF）、boolean
26. store：设置字段是否仅查询
27. term_vector：

聚合查询：

(1) bucket和metirc：

(2) 语法：

```
aggs:{
    code...
}
```

(3) “goup by”：

1. 以tag维度每个产品的数量，即每个标签
2. 在的基础上增加筛选条件：统计价格大于1999的数据

(4) “avg”：

1. 价格大于1999的每个tag产品的平均价格

(5) 分组聚合

(6) 按照千元机：1000以下 中端机：2000-3000 高端机：3000以上分组聚合，分别计算数量