

ES使用中文文档

1823218990@qq.com

**LOGO**

ES文档

2022-05-04

# 一、安装配置

### 配置文件所在位置

Elasticsearch 有三个配置文件：

elasticsearch.yml 用于配置 Elasticsearch

jvm.options 用于配置 Elasticsearch JVM 的设置

log4j2.properties 用户配置 Elasticsearch 日志

配置文件存放在 config 目录下，默认在 $ES\_HOME/config

ES\_PATH\_CONF=/path/to/my/config ./bin/elasticsearch

或者你可以通过命令行或者 shell 脚本使用 export 方法导出 ES\_PATH\_CONF 环境变量。

如果是软件包安装的系统，默认的配置文件是在 /etc/elasticsearch。

### 配置文件格式

配置文件是以 YAML 的格式。 下面是一个更改数据和日志目录路径的例子：

path:

data: /var/lib/elasticsearch logs: /var/log/elasticsearch

你也可以采用下面 的方式设置：

path.data: /var/lib/elasticsearch

path.logs: /var/log/elasticsearch

环境变量替换

配置文件中可以使用 ${...} 符号引用环境变量，例如：

node.name: ${HOSTNAME}

network.host: ${ES\_NETWORK\_HOST}

# 二、JVM 参数设置

默认文件在 config/jvm.options或者 /etc/elasticsearch/jvm.options

以#开头的行被视为注释而被忽略

以 - 开头的行被视为 JVM 选项

-Xmx2g

以数字开头且后跟:- 的行被视为 JVM 选项，仅当 JVM 的版本与数字匹配。

8:-Xmx2g

以数字开头的行，后跟 -:- 被视为 JVM 选项，仅当 JVM 的版本大于或等于数字时才适用。

8-:-Xmx2g

以数字开头，后跟 - 在跟数字和:- 行被视为 JVM 选项，仅当 JVM 的版本在这两个数字的范围内时才应用该选项

8-9:-Xmx2g

设置 JAVA 虚拟机参数是可以通过修改 ES\_JAVA\_OPTS 环境变量。例如：

export ES\_JAVA\_OPTS="$ES\_JAVA\_OPTS -Djava.io.tmpdir=/path/to/temp/dir"

./bin/elasticsearch

# 三、安全设置

某些参数设置是敏感的，如果仅靠文件系统的权限去保护是远远不够的。对于这些参数设置，Elasticsearch 提供了一个秘钥库和 elasticsearch-keystore 工具来管理秘钥库中的设置。

注意: 这里所有的命令都以 Elasticsearch 的用户身份运行。

重要: 只有一些参数需要在秘钥库里读取，但是秘钥库没有验证是否支持这些参数，如果这些参数不被认证将会导致 Elasticsearch 无法启动。请参阅文档，秘钥库是否支持每个参数。

注意: 对秘钥库的任意修改都只有重启 Elasticsearch 才会生效。

注意: Elasticsearch 秘钥库目前只提供混淆操作，之后将会添加密码保护功能。

这些设置就修改 elasticsearch.yml 配置文件中的常规设置一样，需要在集群中的每个节点上指定。目前所有安全参数都需要指定节点，且每个节点的参数都需要相同。

### Creating the keystore

创建 elasticsearch.keystore, 使用 create 命令:

bin/elasticsearch-keystore create

文件 elasticsearch.keystore 将会跟 elasticsearch.yml 一起创建。

### Listing settings in the keystore

秘钥库中的参数列表可通过 list 命令获得:

bin/elasticsearch-keystore list

### Adding string settings

可使用 add 命令添加敏感的字符串，例如云插件的身份验证的凭证：

bin/elasticsearch-keystore add the.setting.name.to.set

工具会提示你输入的值，通过 --stdin 方法传递数据：

cat /file/containing/setting/value | bin/elasticsearch-keystore add --stdin the.setting.name.to.set

### Adding file settings

你可通过 add-file 命令添加敏感文件，例如云插件身份验证秘钥文件。请确保文件路径作为参数在参数名之后：

bin/elasticsearch-keystore add-file the.setting.name.to.set /path/example-file.json

### Removing settings

如从秘钥库删除参数，请使用 remove 命令：

bin/elasticsearch-keystore remove the.setting.name.to.remove

### Upgrading the keystore

有时候，秘钥库里面格式会改变，从包管理安装 Elasticsearch 时，将在升级期间将磁盘上的秘钥库升级到最新版本。在其他情况下， Elasticsearch 将在节点重启期间重启期间执行升级命令。这就要求 Elasticsearch 对包含秘钥库的目录具有写权限。或者你可以使用 upgrade 命令手动更新：

bin/elasticsearch-keystore upgrade

### Reloadable secure settings

就像 elasticsearch.yml 里设置的一样，秘钥库参数的更改不会自动更改正在运行的 Elasticsearch 节点上，重新读取参数需要重启这些节点。但是某些设置被标记成 “可自动重载”，此类参数更改之后，将会自动更新到正在运行的节点上。

所有安全设置（无论是否会自动加载）的值在所有集群节点上必须相同，进行设置更改后，使用 bin/elasticsearch-keystore add 命令之后再调用：

POST \_nodes/reload\_secure\_settings

这个 API 将解密并且重新读取每个集群节点上的秘钥库，但仅适用 reloadable

类型安全设置。对其他设置的更改在下次重启之前不会生效。重新加载成功，意味着依赖这些设置的所有内部数据结构都已更改，所有设置都被重置了一遍。

# 日志配置

Elasticsearch 适用 Log4j 2 作为日志驱动。可以通过 log4j2.properties 文件配置 Log4j 2 。

Elasticsearch 对外有三个属性： ${sys:es.logs.base\_path}, ${sys:es.logs.cluster\_name} 和 ${sys:es.logs.node\_name} , 这些参数在配置文件中引用，以确定日志文件的位置。

${sys:es.logs.base\_path} :文件的位置。

${sys:es.logs.cluster\_name} : 解析为集群名称 （集群名称是日志文件的默认前缀） ${sys:es.logs.node\_name} : 解析为节点名称（如果节点名称已定义）。

例如：如果你的日志文件（path.logs）在 /var/log/elasticsearch 目录下，

集群名被命名成 production

${sys:es.logs.base\_path} 将被解析为

/var/log/elasticsearc

${sys:es.logs.base\_path} ${sys:file.separator} ${sys:es.logs.cluster\_name}.log

被解析成 /var/log/elasticsearch/production.log。

### Rolling参数说明

######## Server JSON ############################

01 appender.rolling.type = RollingFile

appender.rolling.name = rolling

# 配置 RollingFile appender

02 appender.rolling.fileName = ${sys:es.logs.base\_path}${sys:file.separator}${sys:es.logs.cluster\_name}\_server.json

# 日志位置： /var/log/elasticsearch/production.json

03 appender.rolling.layout.type = ESJsonLayout

# 使用 JSON 布局

04 appender.rolling.layout.type\_name = server

# type\_name 是一个标识， 用于填充 ESJsonLayout 中的 type 字段。解析日志时，可以更加轻松地区别不同类型的日志。

05 appender.rolling.filePattern = ${sys:es.logs.base\_path}${sys:file.separator}${sys:es.logs.cluster\_name}-%d{yyyy-MM-dd}-%i.json.gz

appender.rolling.policies.type = Policies

# 日志回滚到：/var/log/elasticsearch/production-yyyy-MM-dd-i.json; 每次回滚日志都将会被压缩，日志名称 i 参数每次递增。

06 appender.rolling.policies.time.type = TimeBasedTriggeringPolicy

# 使用基于时间回滚策略

07 appender.rolling.policies.time.interval = 1

# 每天回滚日志

08 appender.rolling.policies.time.modulate = true

# 在日边界对齐（凌晨）回滚 (而不是每 24 小时回滚一次)

09 appender.rolling.policies.size.type = SizeBasedTriggeringPolicy

# 使用基于大小的回滚策略

10 appender.rolling.policies.size.size = 256MB

appender.rolling.strategy.type = DefaultRolloverStrategy

appender.rolling.strategy.fileIndex = nomax

# 日志大于 256M 进行回滚

11 appender.rolling.strategy.action.type = Delete

appender.rolling.strategy.action.basepath = ${sys:es.logs.base\_path}

# 使用删除操作回滚日志

12 appender.rolling.strategy.action.condition.type = IfFileName

# 仅删除匹配到的日志

13 appender.rolling.strategy.action.condition.glob = ${sys:es.logs.cluster\_name}-\*

# 只删除主日志

14 appender.rolling.strategy.action.condition.nested\_condition.type = IfAccumulatedFileSize

# 仅日志大于多少才删除

15 appender.rolling.strategy.action.condition.nested\_condition.exceeds = 2GB

# 当日志大于 2G 时压缩

################################################

### Old style pattern

######## Server - old style pattern ###########

appender.rolling\_old.type = RollingFile

appender.rolling\_old.name = rolling\_old

appender.rolling\_old.fileName = ${sys:es.logs.base\_path}${sys:file.separator}${sys:es.logs.cluster\_name}\_server.log

appender.rolling\_old.layout.type = PatternLayout

appender.rolling\_old.layout.pattern = [%d{ISO8601}][%-5p][%-25c{1.}] [%node\_name]%marker %m%n

appender.rolling\_old.filePattern = ${sys:es.logs.base\_path}${sys:file.separator}${sys:es.logs.cluster\_name}-%d{yyyy-MM-dd}-%i.old\_log.gz

############################################

old style 配置参数为追加模式，这些日志将被保存在 \*.log 文件中，如果日志被归档了，则将保存在 \* .log.gz 文件中。请注意，如果配置被弃用，之后需删除。

你可以用 appender.rolling.filePattern 中的 .zip 替代.gz 以压缩回滚日志。如果删除 .gz 扩展名，则日志在回滚时将不会被压缩。

如果你想在指定时间段内保存日志文件，则可以将过渡策略和删除操作一起使用。

01 appender.rolling.strategy.type = DefaultRolloverStrategy

# 配置 DefaultRolloverStrategy

02 appender.rolling.strategy.action.type = Delete

# 配置回滚过程中类型为删除

03 appender.rolling.strategy.action.basepath = ${sys:es.logs.base\_path}

# Elasticsearch 日志路径地址

04 appender.rolling.strategy.action.condition.type = IfFileName

# 回滚时的条件

05 appender.rolling.strategy.action.condition.glob = ${sys:es.logs.cluster\_name}-\*

# 从基本路径中删除与全局 $ {sys：es.logs.cluster\_name}-\* 相匹配的文件；这是日志文件滚动到的位置；仅删除滚动的 Elasticsearch 日志，而不删除过时和慢速日志，这是必需的

06 appender.rolling.strategy.action.condition.nested\_condition.type = IfLastModified

# 嵌套条件适用于与 glob 匹配的文件

07 appender.rolling.strategy.action.condition.nested\_condition.age = 7D

# 保留日志 7 天

可以加载多个配置文件 (在这种情况下，这些配置文件将被合并)，只要它们的名为 log4j2.properties 并将 Elasticsearch 配置目录作为父类；这对于公开其他记录器的插件很有用。日志驱动部分包含 java 包及其相应的日志级别，有关如何自定义日志记录和所有受支持的附加器的详细信息，请参阅 Log4j 文档.

(<http://logging.apache.org/log4j/2.x/manual/configuration.html>)

### 配置日志等级

有四种方式日志记录等级的方法，每种方式都有对应的适用场景。

01 修改命令行

-E <name of logging hierarchy>=<level>

(e.g., -E logger.org.elasticsearch.transport=trace)。

当你在单节点上临时调试问题（例如，问题重现或者开发时），这是最合适的方式。

02 修改 elasticsearch.yml 文件

<name of logging hierarchy>: <level>

(e.g., logger.org.elasticsearch.transport: trace)

当你临时调试的时候，但没有通过命令行（或者通过服务）去启动 Elasticsearch ，或者你想永久的改变日志记录级别，这是最合适的方法。

03 集群设置修改

PUT /\_cluster/settings

{

"transient": {

"<name of logging hierarchy>": "<level>"

}

}

例如

PUT /\_cluster/settings

{

"transient": {

"logger.org.elasticsearch.transport": "trace"

}

}

当你需要动态调整集群的日志记录级别，这是最合适的。

04 通过修改 log4j2.properties 配置文件

logger.<unique\_identifier>.name = <name of logging hierarchy>

logger.<unique\_identifier>.level = <level>

例如

logger.transport.name = org.elasticsearch.transport

logger.transport.level = trace

当你需要对日志进行细粒度控制时（例如需要将日志写到到其他文件中去，或者以不同的方式管理日志，这是一种罕见的用法），这是最适合的方法。

### 弃用日志

除了常规日志记录外，Elasticsearch 允许你启动弃用日志操作。例如，这让你提早下定决心确认是否迁移某些功能。默认情况下，在 WARN 级别启动弃用日志，该级别上所有日志都将被弃用。

logger.deprecation.level = warn

这将在日志目录中创建每日回滚弃用日志文件，定期检查此文件，特别是当你打算升级到某一个新主版本时。

默认的日志记录配置，将弃用日志的回滚策略设为：1G 之后回滚压缩，并最多保留五个日志文件（四个回滚日志和 1 个活动日志）

你也可通过在 config/log4j2.properties 文件中禁用它，弃用日志配置设置为 error，如图所示：

logger.deprecation.name = org.elasticsearch.deprecation

logger.deprecation.level = error

### JSON 日志格式

为了简化对 Elasticsearch 日志的解析，日志以 JSON 格式打印。这是通过

Log4J 的布局属性 appender.rolling.layout.type = ESJsonLayout 进行配置。需要设置 type\_name，该属性用于解析时区分日志流。

appender.rolling.layout.type = ESJsonLayout

appender.rolling.layout.type\_name = server

每行包含一个 JSON 文档，其属性配置在 ESJsonLayout 中。有关更多详细信息，请参见此类 javadoc。但是如果 JSON 文档异常，它将被打印成多行。第一行将包含常规属性，随后的行将包含格式为 JSON 数组的 stacktrace。

NOTE: 你仍然可以使用自己的自定义布局。为此，将 appender.rolling.layout.type 替换为其他布局。请参考以下示例:

appender.rolling.type = RollingFile

appender.rolling.name = rolling

appender.rolling.fileName = ${sys:es.logs.base\_path}${sys:file.separator}${sys:es.logs.cluster\_name}\_server.log

appender.rolling.layout.type = PatternLayout

appender.rolling.layout.pattern = [%d{ISO8601}][%-5p][%-25c{1.}] [%node\_name]%marker %.-10000m%n

appender.rolling.filePattern = ${sys:es.logs.base\_path}${sys:file.separator}${sys:es.logs.cluster\_name}-%d{yyyy-MM-dd}-%i.log.gz

# 五、重要elasticsearch配置

### 数据、日志路径设置

如果您正在使用 .zip 或 .tar.gz 归档， data 和 logs 目录在都在 $ES\_HOME 下。如果这些比较重要的文件夹保留在默认位置，则 Elasticsearch 升级到新版本时，很有可能被删除。

在生产环境中，你可以通过下面这种方式修改数据和日志文件所在的位置：

path:

logs: /var/log/elasticsearch

data: /var/data/elasticsearch

path.data 可以设置为多个路径，在这种情况下，所有路径将用于存储数据 （属于单个分片的文件将全部存储在同一数据路径上）：

path:

data:

- /mnt/elasticsearch\_1

- /mnt/elasticsearch\_2

- /mnt/elasticsearch\_3

### 集群名称

当节点与集群中的所有其他节点共享 cluster.name 时，该节点就能加入集群。

默认名称是 elasticsearch，但是你应该将其更改为描述集群用途的适当名称。

cluster.name: logging-prod

确保不要在不同的环境中重复使用相同的集群名称，否则最终可能会导致节点加入错误的集群

### 节点名称

Elasticsearch 使用 node.name 作为 Elasticsearch 特定实例的可读标识符，因此它被包含在许多 API 的响应中。它默认机器在 Elasticsearch 启动时的主机名，但可以在 elasticsearch.yml 中更改配置，如下所示：

node.name: prod-data-2

### 主机网络

默认情况下， Elasticsearch 仅绑定回送地址，例如 127.0.0.1 和 [::1]，这样足以在服务器上运行单节点集群。

TIP: 实际上，可以从单节点上相同的 $ ES\_HOME 位置启动多个节点。这对于测试 Elasticsearch 形成集群的能力很有用，但不建议在生产中使用此配置。

为了与其他服务器上的节点形成集群，您的节点将需要绑定到非回送地址。尽管有许多网络设置方式，但通常您需要配置的只是 network.host：

network.host: 192.168.1.10

network.host 还可以设置一些特殊值，例如\_local\_,\_site\_,\_global\_ 以及诸如 :ip4 和 :ip6, 详细信息可以在 network.host 的特殊值.

重要：为 network.host 提供自定义设置后，Elasticsearch 会假设你正在从开发者模式过渡到生产模式，并将许多系统启动检查从警告升级到异常。有关更多信息，请参考开发模式与生产模式

### 重要的发现和集群初始化设置

在投入生产之前，应配置两个重要的发现和集群形成设置，以便集群中的节点可以彼此发现并选举一个主节点。

discovery.seed\_hosts

开箱即用，无需任何网络配置，Elasticsearch 将绑定到可用的回送地址，并将扫描本地端口 9300 至 9305，以尝试连接到在同一服务器上运行的其他节点，这里 ES 提供了自动群集检查，而无需进行任何配置。

如果要与其他主机上的节点组成集群，则必须设置 discovery.seed\_hosts，提供集群中其他主机的列表，这些节点都具备主机资格，并且可能在其中处于活动状态并可以联系，以便为发现过程。此设置通常应包含群集中所有符合主机条件的节点的地址，是主机地址数组或逗号分隔的字符串。每个值都采用 host:port 或者 host 形式（其中 port 默认设置为 transport.profiles.default.port，如果未设置则返回 transport.port）。

注意，IPV6 主机放在方括号里。默认值是： 127.0.0.1, [::1].

cluster.initial\_master\_nodes

首次启动全新的 Elasticsearch 集群时，会出现集群选举 情况，该步骤确定了在第一次选举中符合主机资格的节点的集合。在开发模式中 , 如果没有配置了发现设置，则此步骤由节点本身自动执行。由于这种自动引导是 不安全的 , 当你开启新的集群是 生产模式 , 你必须明确列出首次选举中符合主机资格的节点，使用 cluster.initial\_master\_nodes 参数设置该列表。重新启动集群或者有新节点加入集群时，不应使用此设置。

discovery.seed\_hosts:

- 192.168.1.10:9300

- 192.168.1.11

- seeds.mydomain.com

cluster.initial\_master\_nodes:

- master-node-a

- master-node-b

- master-node-c

端口将默认为 transport.profiles.default.port，如未指定，则回退为 transport.port

如果主机名解析为多个 IP 地址，则该节点将尝试在所有解析的地址发现其他节点

初始主节点应通过其 node.name进行标识，默认为其主机名.。请确保 cluster.initial\_master\_nodes 中的值与 node.name 完全匹配。如果想您使用完全限定域名，例如 master-node-a.example.com 作为节点名称，则必须在此列表中使用完全限定名称；相反，如果 node.name 是没有任何尾随限定符的裸主机名，那么您还必须在 cluster.initial\_master\_nodes 中省略尾随限定符

### 堆大小设置

默认情况下，Elasticsearch 会告诉 JVM 使用最大和最小 1 GB 的堆。在转移至生产环境中，配置堆大小以确保 Elasticsearch 有足够的可用堆是十分重要。

Elasticsearch 将通过 [jvm.options] (3.2.1 JVM 参数设置 "设置 JVM 参数") 中指定整个堆。Xms(最小堆大小) 和 Xmx(最大堆大小) 设置，你应该让这 2 个参数相等。

这些设置的值取决于服务器上的内存大小:

01  Xmx 和 Xms 设置不应该超过宿主机内存的 50%。Elasticsearch 需要用户 JVM 堆以外的其他用途的内存，为此留出多余部分是非常重要的。例如：Elasticsearch 使用堆外缓冲区进行高效的网络通信，依赖操作系统的文件系统缓存来有效的访问文件，而 JVM 本身也需要一些内存。通常观察到 Elasticsearch 进程使用内存多于 Xmx 设置限制。

 02 Xmx 和 Xms 设置不超过 JVM 用于压缩对象指针（压缩 oops ）, 确切的值会有有所不同，但是接近 32G。你可以通过在日志中查找验证是否在阈值之下：

[1.9gb], compressed ordinary object pointers [true]

理想情况下将 Xmx 和 Xms 设置成不大于零压缩 oop 阈值，确切的阈值是变化的，但 26 GB 在大多数系统上是安全的，甚至在某些系统上可以达到 30 GB。您可以通过使用 JVM 选项 -XX：+ UnlockDiagnosticVMOptions -XX：+ PrintCompressedOopsMode 启动 Elasticsearch 并查找类似以下的行来验证您是否处于此阈值以下：

heap address: 0x000000011be00000, size: 27648 MB, zero based Compressed Oops

以上结果显示：启用了从零开始的压缩 oops 。如果没有启用从零开始的压缩 oops，那么您将看到下面这样的一行代码:

heap address: 0x0000000118400000, size: 28672 MB, Compressed Oops with base: 0x00000001183ff000

Elasticsearch 可用的堆越多，它可用于其内部缓存的内存就越多，但它留给操作系统用于文件系统缓存的内存就越少。此外，更大的堆可能导致更长的垃圾收集暂停。

下面是通过 jvm.options 文件设置堆大小的示例：

01 -Xms2g

02 -Xmx2g

将最小堆大小设置为 2g。

将最大堆大小设置为 2g。

你也可以通过环境变量设置堆大小。这可以通过在 jvm.options 文件，并通过 ES\_JAVA\_OPTS 设置以下值：

ES\_JAVA\_OPTS="-Xms2g -Xmx2g" ./bin/elasticsearch

ES\_JAVA\_OPTS="-Xms4000m -Xmx4000m" ./bin/elasticsearch

将最小和最大堆大小设置为 2 GB。

将最小和最大堆大小设置为 4000 MB

### JVM heap dump 文件路径

Elasticsearch 配置 JVM 的默认配置，将内存不足异常时生成的 heap dump 文件保存到 data 目录 ( 当以 RPM 和 Debian 的方式安装时，在 /var/lib/elasticsearch 目录下。 当以 tar 和 zip 压缩包的方式安装时，data 目录在 Elasticsearch 的根目录下)。

如果此路径不适合存储 heap dump 文件，你应该修改 jvm.options 的配置项 -XX:HeapDumpPath=... 。 当你指定的是一个目录，那么 JVM 将以正在运行实例的 PID 作为 heap dump 的文件名。如果你指定的是一个文件而不是一个目录，当出现内存不足异常时，必须确保这个文件不存在，否则 heap dump 将会生成失败。

### GC 日志

默认情况下，Elasticsearch 启用 GC 日志。这些是在 jvm.options 中配置的，默认为与 Elasticsearch 日志相同的默认位置。默认配置每 64 MB 轮换一次日志，最多可消耗 2 GB 磁盘空间。

### 临时文件

默认情况下，Elasticsearch 启动脚本在系统临时目录下立即创建私有临时目录。

在某些 Linux 发行版上，如果最近未访问过文件和目录，则系统程序将从 / tmp 中清除文件和目录。如果长时间不使用需要临时目录的功能，这可能会导致在运行时， Elasticsearch 删除私有临时目录。如果随后使用需要临时目录的功能，则将导致问题。

如果您使用.deb 或.rpm 软件包安装 Elasticsearch 并在 systemd 下运行它，则定期清理计划将排除 Elasticsearch 使用的私有临时目录。

但是，如果您打算长时间在 Linux 上运行.tar.gz 发行版，则应考虑为 Elasticsearch 创建专用的临时目录，该临时目录不在将清除旧文件和目录的路径下。该目录应该设置权限，以便只有运行 Elasticsearch 的用户才能访问它。然后在启动 Elasticsearch 之前将 $ ES\_TMPDIR 环境变量指向它。

### JVM 致命错误日志

**7.3**

默认情况下，Elasticsearch 的 JVM 配置为将致命错误日志写入默认日志记录目录（这是 [RPM](https://www.elastic.co/guide/en/elasticsearch/reference/7.3/rpm.html" \o "Install Elasticsearch with RPM)   和 [Debian](https://www.elastic.co/guide/en/elasticsearch/reference/7.3/deb.html" \o "使用Debian软件包) 的日志地址 /var/log/elasticsearch，[tar](https://www.elastic.co/guide/en/elasticsearch/reference/7.3/targz.html" \o "Install Elasticsearch from archive on Linux or MacOS) 和 [zip](https://www.elastic.co/guide/en/elasticsearch/reference/7.3/zip-windows.html" \o "Install Elasticsearch with .zip on Windows) 归档发行版的 Elasticsearch 安装在根目录下的 logs 目录 。 这些日志是 JVM 遇到致命错误时生成的 (例如，分段故障)。如果此路径不适合接收日志，则应将参数修改成备用路径 -XX:ErrorFile=... 在 [jvm.options](https://www.elastic.co/guide/en/elasticsearch/reference/7.3/jvm-options.html" \o "Setting JVM options)

### 重要的系统参数

**重要系统配置**

理想情况下，Elasticsearch 应该在服务器上单独运行并使用所有可用资源。为此，您需要配置操作系统，以允许运行 Elasticsearch 的用户访问比默认情况下允许更多的资源。

进入生产之前，必须考虑以下设置:

禁用交换

增加文件描述符

确保足够的虚拟内存

确保足够的线程数

JVM DNS 缓存设置

临时目录未安装 noexec

**开发模式与生产模式**

默认情况下，Elasticsearch 假定正在开发模式下工作。如果没有正确配置上述任何设置，那么将在日志文件中写入警告，但是您将能够启动和运行 Elasticsearch 节点。

一旦您配置了诸如 network.host 之类的网络设置，Elasticsearch 就会假设您即将投入生产并将上述警告升级为异常。这些异常将阻止您的 Elasticsearch 节点启动。这是一项重要的安全措施，可确保不会因服务器配置错误而丢失数据。

### 设置系统配置

哪里配置系统设置取决于您用于安装 Elasticsearch 的软件包以及所使用的操作系统：

当使用.zip 或.tar.gz 软件包时，可以这样配置系统设置

暂时使用 ulimit

永久存在于 /etc/security/limits.conf

使用 RPM 或 Debian 软件包时，大多数系统设置都在 系统配置文件. 但是，系统使用 systemd 要求配置在  systemd 配置文件.

ulimit

在 Linux 系统上，ulimit 可以用于临时更改资源限制。在切换到要运行 Elasticsearch 的用户之前，通常需要将限制设置为 root。例如，要将打开的文件句柄数 (ulimit -n) 设置为 65536，你可以执行以下操作：

sudo su

ulimit -n 65535

su elasticsearch

成为 root。

更改打开文件的最大数量。

成为 elasticsearch 用户，以启动 Elasticsearch。

新限制仅适用于当前会话。

您可以使用 ulimit -a 查询所有当前应用的限制。

/etc/security/limits.conf

在 Linux 系统上，可以通过编辑 /etc/security/limits.conf 文件为特定用户设置永久限制。要将 elasticsearch 用户的最大打开文件数设置为 65536，请将以下行添加到 limits.conf 文件中:

elasticsearch - nofile 65535

该更改仅在弹性搜索用户下次打开新会话时才生效。

注意: Ubuntu and limits.conf

对于由 init.d 启动的进程，Ubuntu 忽略 limits.conf 文件。要启用 limits.conf 文件，请编辑 /etc/pam.d/su 并取消注释以下行：

# session required pam\_limits.so

**系统配置文件**

使用 RPM 或 Debian 软件包时，可以在系统配置文件中指定系统设置和环境变量，该文件位于：

RPM /etc/sysconfig/elasticsearch

Debian /etc/default/elasticsearch

但是，对于使用 systemd 的系统，需要通过 systemd

**系统配置**

在使用 systemd 的系统上使用 RPM 或 Debian 软件包时，必须通过 systemd 指定系统限制。

systemd 服务文件 (/usr/lib/systemd/system/elasticsearch.service) 包含默认情况下应用的限制。

要覆盖它们，请添加一个名为 /etc/systemd/system/elasticsearch.service.d/override.conf 的文件 (或者，您可以运行 sudo systemctl edit elasticsearch 来打开该文件自动在您的默认编辑器中)。设置此文件中的所有更改，例如：

[Service]

LimitMEMLOCK=infinity

完成后，运行以下命令以重新加载单元：

sudo systemctl daemon-reload

### 禁止交换

大多数操作系统尝试为文件系统的缓存使用尽可能多的内存，并尽早换出未使用的应用程序内存。这可能导致 JVM 堆的一部分甚至其可执行页面换出到磁盘上。

交换对于性能，节点稳定性非常不利，应不惜一切代价避免交换。它可能导致垃圾回收持续分钟而不是毫秒，并且可能导致节点响应缓慢甚至断开与群集的连接。在弹性分布式系统中，甚至让操作系统杀死该节点更为有效。

有三种禁用交换的方法。首选选项是完全禁用交换。如果这不是一个选择，那么在你的环境中要尽量减少交换性而不是内存锁定。

**禁用所有交换文件**

通常，Elasticsearch 是在机器上运行的唯一服务，其内存使用量由 JVM 选项控制，应无需启用交换功能。

在 Linux 系统上，您可以通过运行以下命令去临时禁用交换:

sudo swapoff -a

不需要重启 Elasticsearch。

要永久禁用它，您将需要编辑 /etc/fstab 文件并注释包含 swap 单词的所有行。

在 Windows 上，可以通过以下方式完全实现此功能：系统属性 → 高级 → 性能 → 高级 → 虚拟内存。

**配置 swappiness**

在 Linux 系统上另一种可用的选择是确保将 sysctl 值 vm.swappiness 设置为 1。这可以减少内核的交换趋势，并且在正常情况下不导致交换，同时仍然允许整个系统在紧急情况下进行交换。

启用 bootstrap.memory\_lock

另一种选择是在 Linux / Unix 系统上使用 mlockall 或在 Windows 上使用 VirtualLock 尝试将进程地址空间锁定在 RAM 中，以防止任何 Elasticsearch 内存被换出。可以通过将以下行添加到 config / elasticsearch.yml 文件中来完成此操作：如果 mlockall 尝试分配的内存超过可用内存，则可能导致 JVM 或 Shell 会话退出！

启动 Elasticsearch 之后，您可以通过检查此请求的输出中的 mlockall 值来查看是否成功应用了此设置：

bootstrap.memory\_lock: true

如果 mlockall 尝试分配的内存超过可用内存，则可能导致 JVM 或 Shell 会话退出！

在启动 Elasticsearch 之后，您可以通过检查此请求的输出中的 mlockall 值来查看是否成功应用了此设置：

GET \_nodes?filter\_path=\*\*.mlockall

如果看到 mlockall 为 false ，则表示 mlockall 请求已失败。您还将在日志中看到一行包含更多信息的行，内容为 “无法锁定 JVM 内存”。

在 Linux / Unix 系统上，最可能的原因是运行 Elasticsearch 的用户无权锁定内存。可以授予以下权限：

.zip 和 .tar.gz

在启动 Elasticsearch 之前将 ulimit -l unlimited 设置为 root 或者在 /etc/security/limits.conf 中将 memlock 设置为 unlimited

RPM 和 Debian

在系统配置文件中将 MAX\_LOCKED\_MEMORY 设置为无限制 (对于使用 systemd 的系统，请参见下文)。

系统使用 systemd

在 systemd 配置中将 LimitMEMLOCK 设置为无穷大

mlockall 失败的另一个可能原因是 JNA 临时目录 (通常是 / tmp 的子.... 这可以通过使用 ES\_JAVA\_OPTS 环境变量为 JNA 指定新的临时目录来解决：

export ES\_JAVA\_OPTS="$ES\_JAVA\_OPTS -Djna.tmpdir=<path>"

./bin/elasticsearch

或在 jvm.options 配置文件中设置此 JVM 标志。

### 文件描述符

Elasticsearch 使用大量文件描述符或文件句柄。文件描述符用完可能是灾难性的，并且很可能导致数据丢失。请确保将运行 Elasticsearch 的用户打开文件描述符的数量限制增加到 65536 或更高。

对于.zip 和.tar.gz 包安装的，在启动 elasticsearch 之前，请将 ulimit -n 65535 设置为根目录，或者在 /etc/security/limits.conf 将 nofile 设置为 65535

在 macOS 上，您还必须将 JVM 选项 - XX：-MaxFDLimit 传递给 Elasticsearch ，使其可使用更高的文件描述符限制。

RPM 和 Debian 软件包已经默认将文件描述符的最大数量设置为 65535，并且不需要进一步配置。

您还可以使用 Nodes Stats API 为每个节点配置 max\_file\_descriptors, 如图：

GET \_nodes/stats/process?filter\_path=\*\*.max\_file\_descriptors

### 虚拟内存

Elasticsearch 默认使用  mmapfs 目录存储其索引。默认的操作系统对 mmap 计数的限制可能太低，这可能会导致内存不足异常。

在 Linux 上，您可以通过以 root 运行以下命令来增加限制：

sysctl -w vm.max\_map\_count=262144

要永久设置此值，请更新 /etc/sysctl.conf 中的 vm.max\_map\_count 设置。要在重启后进行验证，请运行 sysctl vm.max\_map\_count。

RPM 和 Debian 软件包将自动配置此设置。不需要进一步的配置。

### 线程数

Elasticsearch 对不同类型的操作使用许多线程池。能够在需要时创建新线程很重要。确保 Elasticsearch 用户可以创建的线程数至少为 4096。

可以通过设置 ulimit -u 4096 以 root 启动 Elasticsearch, 或者通过在 /etc/security/limits.conf 设置  nproc 为 4096

在 systemd 下作为服务运行时，程序包分发将自动为 Elasticsearch 进程配置线程数。无需其他配置。

### 设置 DNS 缓存

Elasticsearch 运行时有一个安全管理器。有了它， JVM 会默认无期限地缓存正主机名解析，并默认将负主机名解析缓存十秒。

Elasticsearch 用默认值覆盖此行为，以将正向查找缓存六十秒，并将负向查找缓存十秒，这些值应适用于大多数环境，包括 DNS 分辨率随时间变化的环境。如果没有，你可以编辑 es.networkaddress.cache.ttl 和 es.networkaddress.cache.negative.ttl 在 JVM 选项的值。

注意这些值

networkaddress.cache.ttl=<timeout... 和 networkaddress.cache.negative.ttl=&l... 在 Java 安全方针被 Elasticsearch 忽略，除非您删除 es.networkaddress.cache.ttl 和 es.networkaddress.cache.negative.ttl。

### JNA 临时目录禁止使用 noexec 装载

Elasticsearch 使用 Java 本机访问 (JNA) 库来执行一些平台相关的本机代码，在 Linux 上，在运行时从 JNA 存档中提取支持该库的本机代码。默认情况下，此代码被提取到 Elasticsearch 的临时目录中，该目录默认为 / tmp 的子目录。或者，可以使用 JVM 标志 -Djna.tmpdir = <path> 控制此位置。由于本机库以可执行文件的形式映射到 JVM 虚拟地址空间中，因此代码提取到的位置的底层挂载点必须

不能使用 noexec 装载，因为这会阻止 JVM 进程能够将此代码映射为可执行文件。在某些加载的 Linux 安装中，这是 / tmp 的默认安装选项。表示使用 noexec 挂载了基础挂载的一种迹象是，JNA 在启动时将无法加载，会导致 java.lang.UnsatisfiedLinkerError 异常，并带有一条类似 failed to map segment from shared object 映射段。请注意，在 JVM 版本之间，异常消息可能有所不同。此外，依赖于通过 JNA 执行本机代码的 Elasticsearch 组件将失败，并显示消息 because JNA is not available。如果看到此类错误消息，则必须重新挂载用于 JNA 的临时目录，以使其不通过 noexec 进行挂载。

# 六、Aggregations

### 聚合

聚合框架有助于基于搜索查询提供聚合数据。它基于称为聚合的简单构建块，可以进行组合以构建复杂的数据摘要。

聚合可以看作是在一组文档上建立分析信息的工作单元。执行的上下文定义此文档集是什么 (例如，在已执行的搜索请求的查询 / 过滤器的上下文中执行顶级聚合)。

有许多不同类型的聚合，每种聚合都有自己的目的和输出。为了更好地理解这些类型，通常更容易将它们分为四个主要系列：

**Bucketing**

生成存储桶的一组聚合，其中每个存储桶都与一个键和一个文档条件相关联。执行聚合时，将对上下文中的每个文档评估所有存储桶条件，并且当条件匹配时，该文档将被视为 “落入” 相关存储桶。在汇总过程结束时，我们将获得一个存储桶列表 - 每个存储桶都带有一组 “属于” 的文档。

**Metric**

用于跟踪和计算一组文档的指标的聚合。

**Matrix**

一组聚合，可在多个字段上进行操作，并根据从请求的文档字段中提取的值生成矩阵结果。与指标和存储桶聚合不同，此聚合系列尚不支持脚本。

**Pipeline**

聚合其他聚合的输出及其相关度量的聚合

接下来是有趣的部分。由于每个存储桶有效地定义了一个文档集 (所有属于该存储桶的文档)，可以潜在地关联 bucket 级别上的聚合，这些聚合将在该 bucket 的上下文中执行。这就是聚合的真正力量发挥作用的地方: 聚合可以嵌套！

注意： 存储桶聚合可以具有子聚合 (存储桶或指标)。子聚合将为其父聚合生成的存储桶进行计算。嵌套聚合的级别 / 深度没有硬性限制 (可以将一个聚合嵌套在 “父” 聚合下，该父聚合本身是另一个更高级别聚合的子聚合)。

注意： 聚合对数据的双重表示起作用。因此当在绝对值大于 2 ^ 53 的长度上运行时，结果可能是近似的。

**结构聚合**

下面的代码片段描述了聚合的基本结构:

"aggregations" : {

"<aggregation\_name>" : {

"<aggregation\_type>" : {

<aggregation\_body>

}

[,"meta" : { [<meta\_data\_body>] } ]?

[,"aggregations" : { [<sub\_aggregation>]+ } ]?

}

[,"<aggregation\_name\_2>" : { ... } ]\*

}

JSON 中的 aggregations 对象（也可以使用 aggs）保存要计算的聚合。每个聚合都与用户定义的逻辑名称相关联（例如，如果聚合计算平均价格，则将其命名为 avg\_price 是有意义的）。这些逻辑名称还将用于唯一地标识响应中的聚合。每个聚合都具有特定的类型 (上面的代码片段中为 <aggregation\_type>)，通常是命名聚合主体中的第一个键。每种聚合类型都取决于聚合的性质 (例如，特定字段上的 avg 聚合将定义要在其上计算平均值的字段)，定义自己的主体。在聚合类型定义的同一级别上，可以选择定义一组其他聚合，尽管仅当您定义的聚合具有存储特性时才有意义。在这种情况下，将为存储桶聚合构建的所有存储桶计算您在存储桶聚合级别上定义的子聚合。例如，如果您在 range 聚合下定义一组聚合，则将为定义的范围存储桶计算子聚合。

**聚合值的来源**

某些聚合对从聚合文档中提取的值起作用。通常这些值将从特定的文档字段中提取，该字段是使用聚合的 field 键设置的。它也可以定义一个 script去生成值 (每个文档)。

如果为聚合配置了 value 和 script 设置，则该脚本将被视为 value script。虽然普通脚本是在文档级别评估的 (即脚本可以访问与文档关联的所有数据)，但值脚本是在值级别评估的。在这种模式下，将从配置的 field 中提取值，并使用 script 对这些值进行 " 转换 ''。

注意：当使用脚本时，可以定义 lang 和 parmas 设置，前者定义了所使用的脚本语言（假设在 Elasticsearch 中有合适的语言，可以是默认的，也可以是插件）。后者支持将脚本中的所有 “动态” 表达式定义为参数，这使脚本能够在调用之间保持自身的静态 (这将确保在 Elasticsearch 中使用缓存的编译脚本)。

Elasticsearch 使用映射中的字段类型来确定如何运行聚合和格式化响应。但是有两种情况是 Elasticsearch 无法计算出这些信息：未映射的字段（例如，对于跨多个索引的搜索请求，其中只有一些具有字段映射）和纯脚本。在这种情况下，可以使用 value\_type 选项给 Elasticsearch 一个提示，接受以下列值：string， long（适用于所有整数类型）， double （适用于所有的十进制类型例如 float 或 scaled\_float)，date， ip 和 boolean.

### 指标聚合

**指标汇总**

这个类型中的聚合基于以某种方式从被聚合的文档中提取的值来计算度量。这些值通常从文档的字段中提取 (使用字段数据)，但是也可以使用脚本生成。。

数值指标聚合是一种特殊类型的指标聚合，可输出数值。一些聚合输出单个数值指标 (平均值)，称为单值数值指标聚合，其他聚合则生成多个指标 (统计数据)，并称为多值数字指标聚合。单值和多值数值度量聚合之间的区别在这些聚合充当某些 bucket 聚合的直接子聚合时发挥了作用（一些桶聚合使您能够根据每个桶中的数值指标对返回的桶进行排序）。

### 平均聚合

单值指标聚合计算从聚合文档中提取的数值的平均值。这些值可以从文档中的特定数字字段提取，也可以由提供的脚本生成。

假设数据是由学生考试成绩 (0 到 100) 的文档组成，我们可以用以下方法平均他们的分数:

POST /exams/\_search?size=0

{

"aggs" : {

"avg\_grade" : { "avg" : { "field" : "grade" } }

}

}

以上汇总计算了所有文档的平均等级。聚合类型为 avg 和 field 设置定义了将计算平均值的文档的数字字段。将返回以下内容:

{

...

"aggregations": {

"avg\_grade": {

"value": 75.0

}

}

}

聚合的名称 (上面的 avg\_grade) 也是可以从返回的响应中检索聚合结果的键。

**脚本**

根据脚本计算平均成绩：

POST /exams/\_search?size=0

{

"aggs" : {

"avg\_grade" : {

"avg" : {

"script" : {

"source" : "doc.grade.value"

}

}

}

}

}

这将把 script 参数解释使用 painless 脚本语言且为没有脚本参数的 inline 脚本。如要使用存储的脚本，请使用以下语法:

POST /exams/\_search?size=0

{

"aggs" : {

"avg\_grade" : {

"avg" : {

"script" : {

"id": "my\_script",

"params": {

"field": "grade"

}

}

}

}

}

}

**脚本值**

事实证明，该考试远远超出了学生的水平，因此需要进行成绩更正。我们可以使用脚本值来获取新的平均值：:

POST /exams/\_search?size=0

{

"aggs" : {

"avg\_corrected\_grade" : {

"avg" : {

"field" : "grade",

"script" : {

"lang": "painless",

"source": "\_value \* params.correction",

"params" : {

"correction" : 1.2

}

}

}

}

}

}

**缺失值**

missing 参数定义应如何处理缺少值的文档。默认情况下，它们将被忽略，但也可以将它们视为具有值。

POST /exams/\_search?size=0

{

"aggs" : {

"grade\_avg" : {

"avg" : {

"field" : "grade",

"missing": 10

}

}

}

}

### 加权平均聚合

单值度量聚合，计算从聚合文档中提取的数值的加权平均值。这些值可以从文档中的特定数字字段中提取。

在计算常规平均值时，每个数据点都具有相等的权重... 它对最终值的贡献均等。另一方面，加权平均对每个数据点的加权不同。每个数据点对最终值的贡献量是从文档中提取的，或由脚本提供的。

作为一个公式，加权平均数是： ∑(value \* weight) / ∑(weight)

一个普通的平均值可以被认为是一个加权平均值，只是其中每个值都有一个隐含权值为 1。

表 3. 加权平均 参数

参数名 描述 是否必填 默认值

value 提供值的字段或脚本的配置 必填

weight 提供权重的字段或脚本的配置 必填

format 数字响应格式化 可选

value\_type 有关纯脚本或未映射字段的值的提示 可选

值和权重对象有每个字段特定的配置：

表 4. value 参数设置

参数名 描述 是否必填 默认值

field 应该从中提取值的字段 必填

missing 如果字段完全丢失可使用一个值 选填

表 5. weight 参数设置

参数名 描述 是否必填 默认值

field 应该从中提取值的字段 必填

missing 如果字段完全丢失可使用一个值 选填

**例子**

如果我们的文档有一个等级字段，该字段的得分为 0-100，而一个 weight 字段，该字段的得分为任意数值，则我们可以使用：

POST /exams/\_search

{

"size": 0,

"aggs" : {

"weighted\_grade": {

"weighted\_avg": {

"value": {

"field": "grade"

},

"weight": {

"field": "weight"

}

}

}

}

}

得到的响应如下:

{

...

"aggregations": {

"weighted\_grade": {

"value": 70.0

}

}

}

虽然每个字段允许多个值，但仅允许一个权重。如果汇总中遇到的文件的权重超过一个 (例如，权重字段是多值字段)，它将引发异常。如果遇到这种情况，则需要为权重字段指定脚本，然后使用脚本将多个值组合为一个要使用的值。

该单个权重将独立应用于从 value 字段提取的每个值。

此示例说明如何使用单个权重对具有多个值的单个文档进行平均：

POST /exams/\_doc?refresh

{

"grade": [1, 2, 3],

"weight": 2

}

POST /exams/\_search

{

"size": 0,

"aggs" : {

"weighted\_grade": {

"weighted\_avg": {

"value": {

"field": "grade"

},

"weight": {

"field": "weight"

}

}

}

}

}

三个值（1，2 和 3）将作为独立值包括在内，它们的权重均为 2：

{

...

"aggregations": {

"weighted\_grade": {

"value": 2.0

}

}

}

聚合结果返回 2.0，与我们手工计算时的预期相符： ((1\*2) + (2\*2) + (3\*2)) / (2+2+2) == 2

虽然每个字段允许多个值，但仅允许一个权重。如果汇总中遇到的文件的权重超过一个 (例如，权重字段是多值字段)，它将引发异常。如果遇到这种情况，则需要为权重字段指定脚本，然后使用脚本将多个值组合为一个要使用的值。

该单个权重将独立应用于从 value 字段提取的每个值。

此示例说明如何使用单个权重对具有多个值的单个文档进行平均：

POST /exams/\_doc?refresh

{

"grade": [1, 2, 3],

"weight": 2

}

POST /exams/\_search

{

"size": 0,

"aggs" : {

"weighted\_grade": {

"weighted\_avg": {

"value": {

"field": "grade"

},

"weight": {

"field": "weight"

}

}

}

}

}

三个值（1，2 和 3）将作为独立值包括在内，它们的权重均为 2：

{

...

"aggregations": {

"weighted\_grade": {

"value": 2.0

}

}

}

聚合结果返回 2.0，与我们手工计算时的预期相符： ((1\*2) + (2\*2) + (3\*2)) / (2+2+2) == 2

**脚本**

值和权重都可以从脚本而不是字段派生获取。作为一个简单的例子，下面将使用脚本为文档中的等级和权重添加一个:

POST /exams/\_search

{

"size": 0,

"aggs" : {

"weighted\_grade": {

"weighted\_avg": {

"value": {

"script": "doc.grade.value + 1"

},

"weight": {

"script": "doc.weight.value + 1"

}

}

}

}

}

**缺失值**

  missing 参数定义了当文档缺失默认值的时候应如何处理。 value 跟 weight 的默认行为是不同的:

默认情况下，如果缺少 value 字段，则会忽略该文档，并且聚合将继续到下一个文档。如果缺少 weight 字段，则假定其具有 1 的权重 (如正常平均值)。

可以使用 missing 参数覆盖这两个默认值：

POST /exams/\_search

{

"size": 0,

"aggs" : {

"weighted\_grade": {

"weighted\_avg": {

"value": {

"field": "grade",

"missing": 2

},

"weight": {

"field": "weight",

"missing": 3

}

}

}

}

}

### 基数聚合

单指标聚合，可以计算不同值的近似计数，值可以从文档中的特定字段提取，也可以由脚本生成。

假设您正在为商店的销售建立索引，并希望计算与查询匹配的已售产品唯一数量：

POST /sales/\_search?size=0

{

"aggs" : {

"type\_count" : {

"cardinality" : {

"field" : "type"

}

}

}

}

响应:

{

...

"aggregations" : {

"type\_count" : {

"value" : 3

}

}

}

**精准控制**

该聚合还支持 precision\_threshold 选项:

POST /sales/\_search?size=0

{

"aggs" : {

"type\_count" : {

"cardinality" : {

"field" : "type",

"precision\_threshold": 100

}

}

}

}

precision\_threshold 选项允许以内存交换精度，并定义了一个唯一的计数，低于该值时，期望计数将接近于准确。高于该值时，计数可能会变得模糊（误差变大）。最大计数是 40000，高于这个阈值跟 40000 效果一样，默认值是 30000。

**计数是近似值**

计算精确计数需要将值加载到哈希集中并返回其大小。当处理高基数集较大的值时，这不会自动扩容，所以所需的内存使用情况以及在节点之间传递每个分片集的需求会占用过多的群集资源。

 cardinality 聚合是基于  HyperLogLog++ 算法，基于具有一些有趣属性的值的哈希进行计数

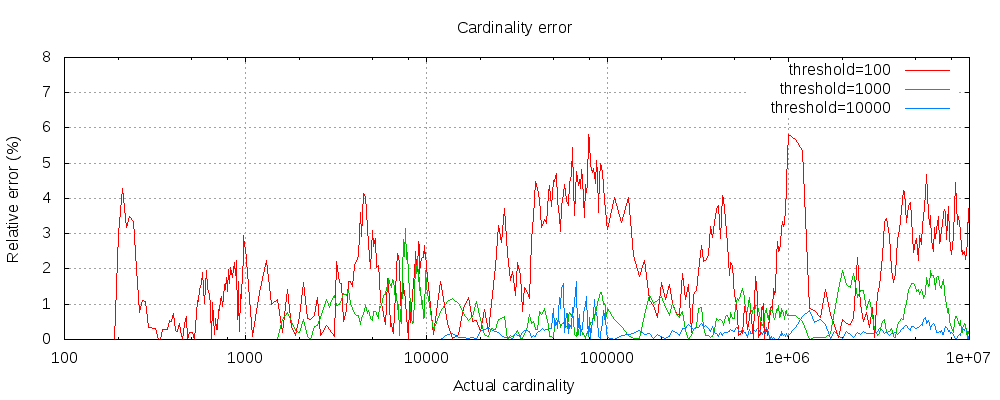
可配置的精度，该精度决定了如何以内存换取精度

在低基数集上具有出色的准确性

固定的内存使用量：无论是否有成百上千的唯一值，内存使用量仅取决于配置的精度。

对于 c 的精度阈值，我们正在使用的实现大约需要 c \* 8 个字节。

下表显示了阈值前后误差如何变化：



对于所有 3 个阈值，计数已精确到配置的阈值。虽然不能保证，但很可能是这样。实际的准确性取决于所讨论的数据集。一般来说，大多数数据集都显示出一贯良好的准确性。还要注意，即使阈值低至 100，在计算数百万个项目时，错误仍然非常低 (如上图所示为 1-6%)。

HyperLogLog++ 算法依赖于散列值的前导零，数据集中散列的精确分布会影响基数的准确性。

请注意，即使阈值低至 100，错误仍然很低，即使在计算数百万项时也是如此。

**预计算哈希**

在具有高基数的字符串字段上，将字段值的哈希存储在索引中然后在此字段上运行基数聚合可能会更快。这可以通过从客户端提供哈希值来完成，也可以让 Elasticsearch 通过使用 mapper-murmur3 插件完成。

注意: 预计算哈希通常仅在非常大或高基数的字段上有用，因为它可以节省 CPU 和内存。但是，在数字字段上哈希运算非常快，存储原始值所需的内存与存储哈希值所需的内存相同或更少。对于低基数的字符串字段也是如此，特别是考虑到这些字段已进行了优化，以确保每个段的每个唯一值最多计算一次哈希。

**脚本**

cardinality 指标支持脚本编写，但是由于需要实时计算哈希值，因此性能受到了明显的影响：

POST /sales/\_search?size=0

{

"aggs" : {

"type\_promoted\_count" : {

"cardinality" : {

"script": {

"lang": "painless",

"source": "doc['type'].value + ' ' + doc['promoted'].value"

}

}

}

}

}

这会将 scrpit 参数解释为具有 painless 脚本语言且没有脚本参数的 inline 脚本。要使用存储的脚本，请使用以下语法：

POST /sales/\_search?size=0

{

"aggs" : {

"type\_promoted\_count" : {

"cardinality" : {

"script" : {

"id": "my\_script",

"params": {

"type\_field": "type",

"promoted\_field": "promoted"

}

}

}

}

}

}

**缺失值**

missing 参数定义应如何处理缺少值的文档。默认情况下，它们将被忽略，但也可以将它们视为有默认值。

POST /sales/\_search?size=0

{

"aggs" : {

"tag\_cardinality" : {

"cardinality" : {

"field" : "tag",

"missing": "N/A"

}

}

}

}

标签字段中没有值的文档将与值是 N / A 的文档归入同一存储桶

### 扩展数据聚合

**扩展统计聚合**

多值度量聚合，用于根据从聚合文档中提取的数值计算统计信息。这些值可以从文档中的特定数字字段中提取，也可以由提供的脚本生成。

extended\_stats 聚合是统计聚合的扩展版本，其中添加了其他指标，例如 sum\_of\_squares，variance，std\_deviation 和 std\_deviation\_bounds。

假设数据由代表学生成绩（0~100）的文档组成：

GET /exams/\_search

{

"size": 0,

"aggs" : {

"grades\_stats" : {

"extended\_stats" :{

"field" : "grade"

}

}

}

}

上面的汇总计算了所有文档的成绩统计。聚合类型为 extended\_stats，并且 field 设置定义将要计算统计信息的文档的数字字段。

上面将返回以下内容：

{

...

"aggregations": {

"grades\_stats": {

"count": 2,

"min": 50.0,

"max": 100.0,

"avg": 75.0,

"sum": 150.0,

"sum\_of\_squares": 12500.0,

"variance": 625.0,

"std\_deviation": 25.0,

"std\_deviation\_bounds": {

"upper": 125.0,

"lower": 25.0

}

}

}

}

聚合的名称 (上面的 grades\_stats) 也是可以从返回的响应中检索聚合结果的键。

**标准偏差范围**

默认情况下 extended\_stats 指标将返回一个名为 std\_deviation\_bounds 的对象，该对象提供了平均值的正负两个标准偏差的间隔。这可能是可视化数据差异的有用一种方法。如果要使用其他边界，例如三个标准偏差，则可以在请求中设置 sigma：

GET /exams/\_search

{

"size": 0,

"aggs" : {

"grades\_stats" : {

"extended\_stats" : {

"field" : "grade",

"sigma" : 3

}

}

}

}

sigma 控制应该显示离均值的标准偏差的数量

注意: 准偏差和界限必须设置正常

默认情况下会显示标准差及其范围，但它们并不总是适用于所有数据集。您的数据必须是正态分布的，这样度量才有意义。标准差背后的统计假设数据是正态分布的，因此，如果数据向左或向右严重倾斜，则返回的值将产生误导。

**脚本**

根据脚本生成成绩统计：

GET /exams/\_search

{

"size": 0,

"aggs" : {

"grades\_stats" : {

"extended\_stats" : {

"script" : {

"source" : "doc['grade'].value",

"lang" : "painless"

}

}

}

}

}

这将把脚本参数解释为单的脚本语言且没有脚本参数的内联脚本。要使用存储的脚本，请使用以下语法:

GET /exams/\_search

{

"size": 0,

"aggs" : {

"grades\_stats" : {

"extended\_stats" : {

"script" : {

"id": "my\_script",

"params": {

"field": "grade"

}

}

}

}

}

}

**值脚本**

事实证明，该考试远远超出了学生的水平，因此需要进行成绩更正。我们可以使用值脚本来获取新的统计信息：

GET /exams/\_search

{

"size": 0,

"aggs" : {

"grades\_stats" : {

"extended\_stats" : {

"field" : "grade",

"script" : {

"lang" : "painless",

"source": "\_value \* params.correction",

"params" : {

"correction" : 1.2

}

}

}

}

}

}

**缺失值**

missing 参数定义应如何处理缺少值的文档。默认情况下，它们将被忽略，但也可以将它们视为有默认值

GET /exams/\_search

{

"size": 0,

"aggs" : {

"grades\_stats" : {

"extended\_stats" : {

"field" : "grade",

"missing": 0

}

}

}

}

标签 grade 字段中没有值的文档将与值为 0 的文档归入同一存储桶

### 地理边界聚合

**地理边界聚合**

计算包含字段的所有 geo\_point 值的边界框的度量聚合。

示例:

PUT /museums

{

"mappings": {

"properties": {

"location": {

"type": "geo\_point"

}

}

}

}

POST /museums/\_bulk?refresh

{"index":{"\_id":1}}

{"location": "52.374081,4.912350", "name": "NEMO Science Museum"}

{"index":{"\_id":2}}

{"location": "52.369219,4.901618", "name": "Museum Het Rembrandthuis"}

{"index":{"\_id":3}}

{"location": "52.371667,4.914722", "name": "Nederlands Scheepvaartmuseum"}

{"index":{"\_id":4}}

{"location": "51.222900,4.405200", "name": "Letterenhuis"}

{"index":{"\_id":5}}

{"location": "48.861111,2.336389", "name": "Musée du Louvre"}

{"index":{"\_id":6}}

{"location": "48.860000,2.327000", "name": "Musée d'Orsay"}

POST /museums/\_search?size=0

{

"query" : {

"match" : { "name" : "musée" }

},

"aggs" : {

"viewport" : {

"geo\_bounds" : {

"field" : "location",

"wrap\_longitude" : true

}

}

}

}

geo\_bounds 聚合指定用于获取边界的字段

wrap\_longitude 是一个可选参数，指定是否允许边界框与国际日期行重叠。默认值是 true

上面的聚合演示了如何计算具有业务类型 shop 的所有文档的 location 字段的边界框

上面的聚合的响应:

{

...

"aggregations": {

"viewport": {

"bounds": {

"top\_left": {

"lat": 48.86111099738628,

"lon": 2.3269999679178

},

"bottom\_right": {

"lat": 48.85999997612089,

"lon": 2.3363889567553997

}

}

}

}

}

### 地理重心聚合

**地理重心聚合**

有一种聚合，它根据地理点的所有坐标值计算加权的矩心字段。

示例:

PUT /museums

{

"mappings": {

"properties": {

"location": {

"type": "geo\_point"

}

}

}

}

POST /museums/\_bulk?refresh

{"index":{"\_id":1}}

{"location": "52.374081,4.912350", "city": "Amsterdam", "name": "NEMO Science Museum"}

{"index":{"\_id":2}}

{"location": "52.369219,4.901618", "city": "Amsterdam", "name": "Museum Het Rembrandthuis"}

{"index":{"\_id":3}}

{"location": "52.371667,4.914722", "city": "Amsterdam", "name": "Nederlands Scheepvaartmuseum"}

{"index":{"\_id":4}}

{"location": "51.222900,4.405200", "city": "Antwerp", "name": "Letterenhuis"}

{"index":{"\_id":5}}

{"location": "48.861111,2.336389", "city": "Paris", "name": "Musée du Louvre"}

{"index":{"\_id":6}}

{"location": "48.860000,2.327000", "city": "Paris", "name": "Musée d'Orsay"}

POST /museums/\_search?size=0

{

"aggs" : {

"centroid" : {

"geo\_centroid" : {

"field" : "location"

}

}

}

}

geo\_centroid 聚合指定用于计算质心的字段。 (注意：字段必须为地理点类型)

上面的汇总显示了如何计算所有犯罪类型盗窃文件的位置字段的质心

以上汇总的响应：

{

...

"aggregations": {

"centroid": {

"location": {

"lat": 51.00982965203002,

"lon": 3.9662131341174245

},

"count": 6

}

}

}

将 geo\_centroid 聚合作为子聚合合并到其他存储桶聚合时会更加有趣。

例如:

POST /museums/\_search?size=0

{

"aggs" : {

"cities" : {

"terms" : { "field" : "city.keyword" },

"aggs" : {

"centroid" : {

"geo\_centroid" : { "field" : "location" }

}

}

}

}

}

上面的示例使用 geo\_centroid 作为 terms 的子聚合存储桶聚合，用于查找每个城市的博物馆的中心位置。

以上汇总的响应：

{

...

"aggregations": {

"cities": {

"sum\_other\_doc\_count": 0,

"doc\_count\_error\_upper\_bound": 0,

"buckets": [

{

"key": "Amsterdam",

"doc\_count": 3,

"centroid": {

"location": {

"lat": 52.371655656024814,

"lon": 4.909563297405839

},

"count": 3

}

},

{

"key": "Paris",

"doc\_count": 2,

"centroid": {

"location": {

"lat": 48.86055548675358,

"lon": 2.3316944623366

},

"count": 2

}

},

{

"key": "Antwerp",

"doc\_count": 1,

"centroid": {

"location": {

"lat": 51.22289997059852,

"lon": 4.40519998781383

},

"count": 1

}

}

]

}

}

}

### 最大聚合

单值度量聚合，可跟踪并返回从聚合文档中提取的数值中的最大值。这些值可以从文档中的特定数字字段中提取，也可以由提供的脚本生成。

注意：min 和 max 聚合表示对 double 类型的数据进行操作，结果，当在绝对值大于 2 ^ 53 的多头上运行时，结果可能是近似的。

计算所有单据的最大价格值

POST /sales/\_search?size=0

{

"aggs" : {

"max\_price" : { "max" : { "field" : "price" } }

}

}

相应:

{

...

"aggregations": {

"max\_price": {

"value": 200.0

}

}

}

可以看出，聚合的名称 (上面的 max\_price) 也用作键，通过它可以从返回的响应中检索聚合结果。

**脚本**

max 聚合也可以计算脚本的最大值。下面的示例计算最高价格：

POST /sales/\_search

{

"aggs" : {

"max\_price" : {

"max" : {

"script" : {

"source" : "doc.price.value"

}

}

}

}

}

这将使用 Painless 脚本语言，并且不使用脚本参数。要使用存储的脚本，请使用以下语法：

POST /sales/\_search

{

"aggs" : {

"max\_price" : {

"max" : {

"script" : {

"id": "my\_script",

"params": {

"field": "price"

}

}

}

}

}

}

**值脚本**

假设我们索引中的文档价格以美元为单位，但是我们想以欧元为单位计算最大值 (在本例中，转换率为 1.2)。我们可以使用值脚本将转化率应用于每个值，然后再进行汇总：

POST /sales/\_search

{

"aggs" : {

"max\_price\_in\_euros" : {

"max" : {

"field" : "price",

"script" : {

"source" : "\_value \* params.conversion\_rate",

"params" : {

"conversion\_rate" : 1.2

}

}

}

}

}

}

**缺失值**

missing 参数定义应如何处理缺少值的文档。默认情况下，它们将被忽略，但也可以将它们视为具有值。

POST /sales/\_search

{

"aggs" : {

"grade\_max" : {

"max" : {

"field" : "grade",

"missing": 10

}

}

}

}

grade 字段中没有值的文档将与值为 10 的文档归入同一存储桶。

### 最小值聚合

单值度量聚合，可跟踪并返回从聚合文档中提取的数值中的最小值。这些值可以从文档中的特定数字字段中提取，也可以由提供的脚本生成。

注意:min 和 max 聚合表示对 double 的数据进行操作，结果当在绝对值大于 2 ^ 53 的多头上运行时，结果可能是近似的。

计算所有文档的最低价格值：

POST /sales/\_search?size=0

{

"aggs" : {

"min\_price" : { "min" : { "field" : "price" } }

}

}

相应：

{

...

"aggregations": {

"min\_price": {

"value": 10.0

}

}

}

可以看出，聚合的名称 (上面的 min\_price) 也用作键，通过该键可以从返回的响应中检索聚合结果。

**脚本**

min 聚合也可以计算脚本的最小值。下面的示例计算最低价格：

POST /sales/\_search

{

"aggs" : {

"min\_price" : {

"min" : {

"script" : {

"source" : "doc.price.value"

}

}

}

}

}

这将使用 Painless 脚本语言，并且不使用脚本参数。要使用存储的脚本，请使用以下语法：

POST /sales/\_search

{

"aggs" : {

"min\_price" : {

"min" : {

"script" : {

"id": "my\_script",

"params": {

"field": "price"

}

}

}

}

}

}

**值脚本**

假设我们索引中的文档价格以美元为单位，但是我们想以欧元为单位计算最小值 (在本例中，转化率为 1.2)。我们可以使用值脚本将转化率应用于每个值，然后再进行汇总：

POST /sales/\_search

{

"aggs" : {

"min\_price\_in\_euros" : {

"min" : {

"field" : "price",

"script" : {

"source" : "\_value \* params.conversion\_rate",

"params" : {

"conversion\_rate" : 1.2

}

}

}

}

}

}

**缺失值**

missing 参数定义应如何处理缺少值的文档。默认情况下，它们将被忽略，但也可以将它们视为具有值。

POST /sales/\_search

{

"aggs" : {

"grade\_min" : {

"min" : {

"field" : "grade",

"missing": 10

}

}

}

}

在 grade 字段中没有值的文档将与具有值 10 的文档归入同一存储桶。

### https://learnku.com/docs/elasticsearch73/7.3/percentiles-aggregation/7270

### 百分位数聚合

多值度量标准聚合，可对从聚合文档中提取的数值计算一个或多个百分位数。这些值可以从文档中的特定数字字段中提取，也可以由提供的脚本生成。

百分位数表示观察值出现一定百分比的点。例如，第 95 个百分位数是大于观察值的 95％的值。

百分位数通常用于查找异常值。在正态分布中，第 0.13 和第 99.87 个百分位数代表与平均值的三个标准差。任何超出三个标准偏差的数据通常被视为异常。

当检索到一定范围的百分位数时，它们可以用于估计数据分布并确定数据是否偏斜，双峰等。

假设您的数据包含网站加载时间。对于管理员来说，平均加载时间和中值加载时间并不是特别有用。最大值可能变的很出乎意料，它很容易被单一的缓慢响应所扭曲。

让我们看一下代表加载时间的百分比范围：

GET latency/\_search

{

"size": 0,

"aggs" : {

"load\_time\_outlier" : {

"percentiles" : {

"field" : "load\_time"

}

}

}

}

load\_time 字段类型必须是数字

默认情况下，百分比度量标准将生成一定范围的百分比：[1，5，25，50，75，95，99]。响应将如下所示：

{

...

"aggregations": {

"load\_time\_outlier": {

"values" : {

"1.0": 5.0,

"5.0": 25.0,

"25.0": 165.0,

"50.0": 445.0,

"75.0": 725.0,

"95.0": 945.0,

"99.0": 985.0

}

}

}

}

如您所见，聚合将返回默认范围内每个百分位的计算值。如果我们假设响应时间以毫秒为单位，那么很明显，该网页通常会在 10-725ms 内加载，但偶尔会激增至 945-985ms。

通常，管理员只对异常值 (极端百分比) 感兴趣。我们可以只指定我们感兴趣的百分比 (请求的百分比必须是介于 0 到 100 之间的一个值)：

GET latency/\_search

{

"size": 0,

"aggs" : {

"load\_time\_outlier" : {

"percentiles" : {

"field" : "load\_time",

"percents" : [95, 99, 99.9]

}

}

}

}

使用 percents 参数指定要计算的特定百分位数

**Keyed 控制**

默认情况下，keyed 标志设置为 true，该标志将唯一的字符串键与每个存储桶相关联，并将范围作为哈希而不是数组返回。将 keyed 标志设置为 false 将禁用此行为：

GET latency/\_search

{

"size": 0,

"aggs": {

"load\_time\_outlier": {

"percentiles": {

"field": "load\_time",

"keyed": false

}

}

}

}

响应:

{

...

"aggregations": {

"load\_time\_outlier": {

"values": [

{

"key": 1.0,

"value": 5.0

},

{

"key": 5.0,

"value": 25.0

},

{

"key": 25.0,

"value": 165.0

},

{

"key": 50.0,

"value": 445.0

},

{

"key": 75.0,

"value": 725.0

},

{

"key": 95.0,

"value": 945.0

},

{

"key": 99.0,

"value": 985.0

}

]

}

}

}

**脚本**

百分位数指标支持脚本。例如，如果我们的加载时间以毫秒为单位，但是我们希望以秒为单位计算百分位数，则可以使用脚本即时转换它们：

GET latency/\_search

{

"size": 0,

"aggs" : {

"load\_time\_outlier" : {

"percentiles" : {

"script" : {

"lang": "painless",

"source": "doc['load\_time'].value / params.timeUnit",

"params" : {

"timeUnit" : 1000

}

}

}

}

}

}

用脚本参数替换 field 参数，该脚本参数使用脚本生成在其上计算百分位数的值

脚本支持参数化输入，就像其他任何脚本一样

GET latency/\_search

{

"size": 0,

"aggs" : {

"load\_time\_outlier" : {

"percentiles" : {

"script" : {

"id": "my\_script",

"params": {

"field": "load\_time"

}

}

}

}

}

}

**百分位数 (通常) 是近似值**

有许多不同的算法可以计算百分位数。理想中实现只是将所有值存储在一个排序数组中。即要找到第 50 个百分位数，您只需找到 my\_array [count(my\_array)\* 0.5] 的值即可。

显然，理想中的实现是不可伸缩的 — 排序后的数组随数据集中值的数量线性增长。为了计算一个 Elasticsearch 集群中可能有数十亿个值的百分位数，需要计算近似百分位数。

百分位数度量使用的算法称为 TDigest (由 Ted Dunning 引入在 Computing Accurate Quantiles using T-Diges...)

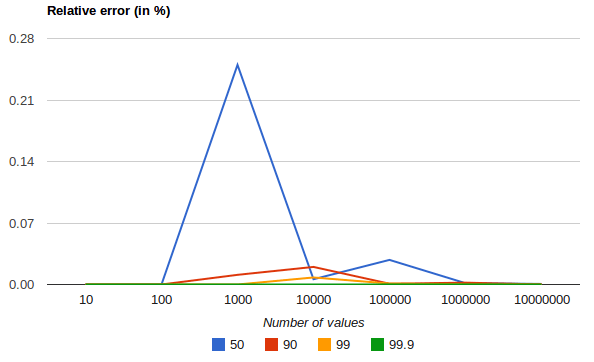
使用此指标时，需要牢记一些准则：

准确度与 q(1-q) 成正比。这意味着极端百分位数 (例如 99％) 比不那么极端百分位数 (例如中位数) 更准确

对于较小的一组值，百分位数非常准确 (如果数据足够小，则可能为 100％准确)。

随着存储桶中值数量的增加，算法开始近似百分位数。它有效地交易了准确性以节省内存。确切的准确度级别很难一概而论，因为它取决于您的数据分布和要聚合的数据量

下表显示了一个均匀分布的相对误差，它取决于收集值的数量和请求的百分位数:



它说明了对于极端百分位数来说，精度是如何更好。对于大量的值，误差减小的原因是大数定律使得值的分布越来越均匀，t-digest 树可以更好地对其进行汇总。对于更倾斜的分布，则不会出现这种情况。

**压缩**

近似算法必须在存储器利用率与估计精度之间取得平衡。可以使用 compression 参数来控制：

GET latency/\_search

{

"size": 0,

"aggs" : {

"load\_time\_outlier" : {

"percentiles" : {

"field" : "load\_time",

"tdigest": {

"compression" : 200

}

}

}

}

}

压缩控制内存使用和近似误差

TDigest 算法使用多个” 节点” 来近似百分位数 - 可用节点越多，与数据量成正比的精度 (以及较大的内存占用量) 就越高。compression 参数将最大节点数限制为 20 \* compression。

因此，通过增加压缩值，可以以更多内存为代价来提高百分位数的准确性。较大的压缩值也会使算法变慢，因为基础树数据结构的大小会增加，从而导致操作成本更高。默认压缩值为 100。

一个” 节点” 大约使用 32 个字节的内存，因此在最坏的情况下 (按顺序到达并排序的大量数据)，默认设置将产生大约 64KB 的 TDigest 大小。实际上，数据趋向于更加随机，并且 TDigest 将使用更少的内存。

### HDR 直方图

注意: 此设置公开了 HDR 直方图的内部实现，语法将来可能会更改。

HDR 直方图 (高动态范围直方图) 是一种替代实现，在计算延迟测量的百分位数时会很有用，因为它比使用 t-digest 实现的速度更快在更大的内存占用量之间进行权衡。此实现维护一个固定的最坏情况百分比错误 (指定为多个有效数字)。这意味着，如果在设置为 3 个有效数字的直方图中以 1 微秒至 1 小时 (3,600,000,000 微秒) 的值记录数据，则对于 1 毫秒和 3.6 秒 (或更佳) 的值，它将保持 1 微秒的值分辨率) 以获取最大跟踪值 (1 小时)。

可以通过在请求中指定 method 参数来使用 HDR 直方图：

GET latency/\_search

{

"size": 0,

"aggs" : {

"load\_time\_outlier" : {

"percentiles" : {

"field" : "load\_time",

"percents" : [95, 99, 99.9],

"hdr": {

"number\_of\_significant\_value\_digits" : 3

}

}

}

}

}

hdr 对象指示应使用 HDR 直方图来计算百分位数，并且可以在对象内部指定此算法的特定设置

number\_of\_significant\_value\_digits 指定有效位数的直方图值的分辨率

HDRHistogram 仅支持正值，如果传递负值，则将出错。如果值的范围未知，则使用 HDRHistogram 也不是一个好主意，因为这可能会导致大量内存使用。

**缺失值**

missing 参数定义应如何处理缺少值的文档。默认情况下，它们将被忽略，但也可以将它们视为具有值。

GET latency/\_search

{

"size": 0,

"aggs" : {

"grade\_percentiles" : {

"percentiles" : {

"field" : "grade",

"missing": 10

}

}

}

}

在 grade 字段中没有值的文档将与具有值 10 的文档归入同一存储桶。

### 百分数排名聚合

**百分数排名聚合**

一种多值度量标准聚合，该聚合针对从聚合文档中提取的数值计算一个或多个百分数等级。这些值可以从文档中的特定数字字段中提取，也可以由提供的脚本生成。

注意：请参阅 (通常) 百分位数和压缩以获取有关百分位数秩聚合的近似值和内存使用的建议

百分数等级显示低于特定值的观察值的百分比。例如，如果某个值大于或等于观察值的 95％，则它被称为第 95 个百分位等级。

假设您的数据包含网站加载时间。您可能有一项服务协议，其中 95％的页面加载在 500ms 内完成，而 99％的页面加载在 600ms 内完成。

让我们看一下代表加载时间的百分比范围：

GET latency/\_search

{

"size": 0,

"aggs" : {

"load\_time\_ranks" : {

"percentile\_ranks" : {

"field" : "load\_time",

"values" : [500, 600]

}

}

}

}

load\_time 字段必须为数字字段

响应将如下所示：

{

...

"aggregations": {

"load\_time\_ranks": {

"values" : {

"500.0": 55.00000000000001,

"600.0": 64.0

}

}

}

}

根据此信息，您可以确定达到了 99％的 600ms 加载时间目标，但还没有达到 95％ 500ms 的加载时间目标

**Keyed 响应**

默认情况下，keyed 标志设置为 true，将唯一的字符串键与每个存储桶相关联，并将范围作为哈希而不是数组返回。将 keyed 标志设置为 false 将禁用此行为：

GET latency/\_search

{

"size": 0,

"aggs": {

"load\_time\_ranks": {

"percentile\_ranks": {

"field": "load\_time",

"values": [500, 600],

"keyed": false

}

}

}

}

相应：

{

...

"aggregations": {

"load\_time\_ranks": {

"values": [

{

"key": 500.0,

"value": 55.00000000000001

},

{

"key": 600.0,

"value": 64.0

}

]

}

}

}

**脚本**

百分数等级度量标准支持脚本编写。例如，如果我们的加载时间以毫秒为单位，但是我们想以秒为单位指定值，则可以使用脚本即时转换它们：

GET latency/\_search

{

"size": 0,

"aggs" : {

"load\_time\_ranks" : {

"percentile\_ranks" : {

"values" : [500, 600],

"script" : {

"lang": "painless",

"source": "doc['load\_time'].value / params.timeUnit",

"params" : {

"timeUnit" : 1000

}

}

}

}

}

}

将 field 参数替换为 script 参数，该参数使用脚本生成在其上计算百分等级的值

脚本支持参数化输入，就像其他任何脚本一样

这会将 script 参数解释为具有 painless 脚本语言且没有脚本参数的内联脚本。要使用存储的脚本，请使用以下语法：

GET latency/\_search

{

"size": 0,

"aggs" : {

"load\_time\_ranks" : {

"percentile\_ranks" : {

"values" : [500, 600],

"script" : {

"id": "my\_script",

"params": {

"field": "load\_time"

}

}

}

}

}

}

**HDR 直方图**

此设置公开了 HDR 直方图的内部实现，语法将来可能会更改。

HDR 直方图 (高动态范围直方图) 是一种替代实现，在计算延迟测量的百分位等级时可能会很有用，因为它可以比使用 t-digest 实现更快更大内存占用的权衡。此实现维护一个固定的最坏情况百分比错误 (指定为多个有效数字)。这意味着，如果在设置为 3 个有效数字的直方图中以 1 微秒至 1 小时 (3,600,000,000 微秒) 的值记录数据，则对于 1 毫秒和 3.6 秒 (或更佳) 的值，它将保持 1 微秒的值分辨率) 以获取最大跟踪值 (1 小时)。

可以通过在请求中指定方法参数来使用 HDR 直方图：

GET latency/\_search

{

"size": 0,

"aggs" : {

"load\_time\_ranks" : {

"percentile\_ranks" : {

"field" : "load\_time",

"values" : [500, 600],

"hdr": {

"number\_of\_significant\_value\_digits" : 3

}

}

}

}

}

hdr 对象指示应使用 HDR 直方图来计算百分位数，并且可以在对象内部指定此算法的特定设置

number\_of\_significant\_value\_digits s 指定有效位数的直方图值的分辨率

HDRHistogram 仅支持正值，如果传递负值，则将出错。如果值的范围未知，则使用 HDRHistogram 也不是一个好主意，因为这可能会导致大量内存使用。

**缺失值**

missing 参数定义应如何处理缺少值的文档。默认情况下，它们将被忽略，但也可以将它们视为具有值。

GET latency/\_search

{

"size": 0,

"aggs" : {

"load\_time\_ranks" : {

"percentile\_ranks" : {

"field" : "load\_time",

"values" : [500, 600],

"missing": 10

}

}

}

}

在 load\_time 字段中没有值的文档将与值为 10 的文档归入同一存储桶。

### https://learnku.com/docs/elasticsearch73/7.3/5111-scripted-metric-aggregation/7319

### 11 脚本式聚合

使用脚本执行以提供度量标准输出的度量标准聚合。

例子:

POST ledger/\_search?size=0

{

"query" : {

"match\_all" : {}

},

"aggs": {

"profit": {

"scripted\_metric": {

"init\_script" : "state.transactions = []",

"map\_script" : "state.transactions.add(doc.type.value == 'sale' ? doc.amount.value : -1 \* doc.amount.value)",

"combine\_script" : "double profit = 0; for (t in state.transactions) { profit += t } return profit",

"reduce\_script" : "double profit = 0; for (a in states) { profit += a } return profit"

}

}

}

}

init\_script 是可选参数，所有其他脚本都是必需的。

以上聚合演示了如何使用脚本汇总来计算销售和成本交易的总利润。

以上聚合的响应：

{

"took": 218,

...

"aggregations": {

"profit": {

"value": 240.0

}

}

}

也可以使用存储的脚本指定上述示例，如下所示：

POST ledger/\_search?size=0

{

"aggs": {

"profit": {

"scripted\_metric": {

"init\_script" : {

"id": "my\_init\_script"

},

"map\_script" : {

"id": "my\_map\_script"

},

"combine\_script" : {

"id": "my\_combine\_script"

},

"params": {

"field": "amount"

},

"reduce\_script" : {

"id": "my\_reduce\_script"

}

}

}

}

}

必须在全局 params 对象中指定 init，map 和合并脚本的脚本参数，以便可以在脚本之间共享。

**允许的返回类型**

虽然任何有效的脚本对象都可以在一个脚本中使用，但脚本必须返回或存储在 state 对象中只有以下类型:

基本类型

字符串

映射 (只包含此处列出的类型的键和值)

数组 (仅包含此处列出的类型的元素)

**脚本范围**

脚本度量聚合在执行的 4 个阶段使用脚本：

init\_script

在收集任何文件之前执行。允许聚合设置任何初始状态。

在上面的示例中，init\_script 在 state 对象中创建一个数组 transactions。

map\_script

每个收集的文档执行一次。这是必需的脚本。如果未指定 combine\_script，则需要将结果状态存储在 state 对象中。

在上面的示例中，map\_script 检查类型字段的值。如果值为 sale，则金额字段的值将添加到交易数组中。如果类型字段的值不是 sale，则金额字段的取反值将添加到交易中。

combine\_script

文档收集完成后，对每个分片执行一次。这是必需的脚本。允许聚合合并从每个分片返回的状态。

在上面的示例中，combine\_script 遍历所有存储的交易，对 profit 变量中的值求和，最后返回 profit。

reduce\_script

所有分片均返回其结果后，在协调节点上执行一次。这是必需的脚本。该脚本可以访问变量 states，该变量是每个分片上 combine\_script 结果的数组。

在上面的示例中，reduce\_script 通过每个分片返回的 profit 进行迭代，然后在返回最终的合并利润之前返回每个合并的值，该合并后的利润将被返回。

**工作示例**

想象一下您将以下文档编入带有 2 个分片的索引的情况

PUT /transactions/\_bulk?refresh

{"index":{"\_id":1}}

{"type": "sale","amount": 80}

{"index":{"\_id":2}}

{"type": "cost","amount": 10}

{"index":{"\_id":3}}

{"type": "cost","amount": 30}

{"index":{"\_id":4}}

{"type": "sale","amount": 130}

可以说文档 1 和 3 最终位于分片 A 上，而文档 2 和 4 最终位于分片 B 上。以下是上述示例各阶段汇总结果的细目分类。

初始化 init\_script

state 被初始化为新的空对象。

"state" : {}

初始化 init\_script 后

在执行任何文档收集之前，此操作在每个分片上运行一次，因此我们在每个分片上都有一个副本：

分片 A

"state" : { "transactions" : [] }

Copy

分片 B

"state" : { "transactions" : [] }

map\_script 后

每个分片都会收集其文档，并在收集到的每个文档上运行 map\_script：

分片 A

"state" : { "transactions" : [ 80, -30 ] }

分片 B

"state" : { "transactions" : [ -10, 130 ] }

combine\_script 之后

在文档收集完成之后，对每个分片执行 combine\_script，并将每个分片的所有交易降低为单个利润数字 (通过对交易数组中的值求和)，然后将其传递回协调节点：

分片 A

50

分片 B

120

reduce\_script 之后

reduce\_script 接受另一个 states 数组，其中包含每个分片的合并脚本的结果：

"states" : [

50,

120

]

它将分片的响应降低到最终的总利润数字 (通过对值求和)，并将其作为聚合结果返回以产生响应：

{

...

"aggregations": {

"profit": {

"value": 170

}

}

}

其他参数

params

可选的。一个对象，其内容将作为变量传递到 init\_script，map\_script 和 combine\_script。 这对于允许用户控制聚合的行为以及在脚本之间存储状态很有用。如果未指定，则默认等效于提供：

"params" : {}

空桶

如果脚本化指标聚合的父桶未收集任何文档，则分片将返回空聚合响应，并带有 null 值。在这种情况下，reduce\_script 的 status 变量将包含 null 作为该分片的响应。因此，reduce\_script 应该期望并处理分片的 null 响应。

### 12 统计聚合

统计聚合

多值指标聚合，可根据从聚合文档中提取的数值计算统计信息。这些值可以从文档中的特定数字字段中提取，也可以由提供的脚本生成。

返回的统计信息包括：min，max，sum，count 和 avg。

假设数据由代表学生考试成绩 (0 至 100) 的文档组成

POST /exams/\_search?size=0

{

"aggs" : {

"grades\_stats" : { "stats" : { "field" : "grade" } }

}

}

上面的汇总计算了所有文档的成绩统计。聚合类型为统计信息，并且字段设置定义了计算统计信息所依据的文档的数字字段。上面将返回以下内容：

{

...

"aggregations": {

"grades\_stats": {

"count": 2,

"min": 50.0,

"max": 100.0,

"avg": 75.0,

"sum": 150.0

}

}

}

聚合的名称 (上面的 grades\_stats ) 还用作可以从返回的响应中检索聚合结果的键。

脚本

根据脚本计算成绩统计信息：

POST /exams/\_search?size=0

{

"aggs" : {

"grades\_stats" : {

"stats" : {

"script" : {

"lang": "painless",

"source": "doc['grade'].value"

}

}

}

}

}

这会将脚本参数解释为具有轻松且没有脚本参数的嵌入式脚本语言。要使用存储的脚本，请使用以下语法：

POST /exams/\_search?size=0

{

"aggs" : {

"grades\_stats" : {

"stats" : {

"script" : {

"id": "my\_script",

"params" : {

"field" : "grade"

}

}

}

}

}

}

值脚本

事实证明，该考试远远超出了学生的水平，因此需要进行成绩更正。我们可以使用值脚本来获取新的统计信息：

POST /exams/\_search?size=0

{

"aggs" : {

"grades\_stats" : {

"stats" : {

"field" : "grade",

"script" : {

"lang": "painless",

"source": "\_value \* params.correction",

"params" : {

"correction" : 1.2

}

}

}

}

}

}

缺失值

missing 参数定义应如何处理缺少值的文档。默认情况下，它们将被忽略，但也可以将它们视为具有值

POST /exams/\_search?size=0

{

"aggs" : {

"grades\_stats" : {

"stats" : {

"field" : "grade",

"missing": 0

}

}

}

}

在 grade 字段中没有值的文档将与具有值 0 的文档归入同一存储桶。

### 13 总数聚合

单值指标聚合，汇总从聚合文档中提取的数值。这些值可以从文档中的特定数字字段中提取，也可以由提供的脚本生成。

假设数据由代表销售记录的文档组成，我们可以对所有帽子的销售价格求和：

POST /sales/\_search?size=0

{

"query" : {

"constant\_score" : {

"filter" : {

"match" : { "type" : "hat" }

}

}

},

"aggs" : {

"hat\_prices" : { "sum" : { "field" : "price" } }

}

}

结果：

{

...

"aggregations": {

"hat\_prices": {

"value": 450.0

}

}

}

聚合的名称 (上面的 hat\_prices) 还用作可以从返回的响应中检索聚合结果的键。

脚本

我们还可以使用脚本来获取销售价格

POST /sales/\_search?size=0

{

"query" : {

"constant\_score" : {

"filter" : {

"match" : { "type" : "hat" }

}

}

},

"aggs" : {

"hat\_prices" : {

"sum" : {

"script" : {

"source": "doc.price.value"

}

}

}

}

}

这会将 script 参数解释为具有 painless 脚本语言且没有脚本参数的 inline 脚本。要使用存储的脚本，请使用以下语法：

POST /sales/\_search?size=0

{

"query" : {

"constant\_score" : {

"filter" : {

"match" : { "type" : "hat" }

}

}

},

"aggs" : {

"hat\_prices" : {

"sum" : {

"script" : {

"id": "my\_script",

"params" : {

"field" : "price"

}

}

}

}

}

}

值脚本

也可以使用\_value 从脚本访问字段值。例如，这将求和所有帽子的价格的平方：

POST /sales/\_search?size=0

{

"query" : {

"constant\_score" : {

"filter" : {

"match" : { "type" : "hat" }

}

}

},

"aggs" : {

"square\_hats" : {

"sum" : {

"field" : "price",

"script" : {

"source": "\_value \* \_value"

}

}

}

}

}

缺失值

missing 参数定义应如何处理缺少值的文档。默认情况下，缺少该值的文档将被忽略，但也可以将它们视为具有值的文档。例如，这将所有不带价格的帽子销售视为 100。

POST /sales/\_search?size=0

{

"query" : {

"constant\_score" : {

"filter" : {

"match" : { "type" : "hat" }

}

}

},

"aggs" : {

"hat\_prices" : {

"sum" : {

"field" : "price",

"missing": 100

}

}

}

}

### 14 热门聚合

最热点聚合

top\_hits 度量聚合器跟踪要聚合的最相关文档。该聚合器旨在用作子聚合器，以便可以按存储分区汇总最匹配的文档。

top\_hits 聚合器可以有效地用于通过存储桶聚合器按某些字段对结果集进行分组。一个或多个存储桶聚合器确定将结果集切成哪些属性。

选项

from - 与要获取的第一个结果的偏移量。

size - 每个存储区要返回的最匹配匹配项的最大数目。默认情况下，返回前三个匹配项。

sort - 应该如何对最匹配的匹配进行排序。默认情况下，命中按主要查询的分数排序。

支持每次点击功能

top\_hits 聚合返回常规搜索命中，因为可以支持许多每次命中功能：

高亮

解析

命名过滤器和查询

来源

来源字段

脚本字段

文档值字段

包含版本

包含序列号和主要术语

示例

在以下示例中，我们按类型对销售进行分组，并且按类型将最后的销售显示。对于每个销售，源中仅包含日期和价格字段。

POST /sales/\_search?size=0

{

"aggs": {

"top\_tags": {

"terms": {

"field": "type",

"size": 3

},

"aggs": {

"top\_sales\_hits": {

"top\_hits": {

"sort": [

{

"date": {

"order": "desc"

}

}

],

"\_source": {

"includes": [ "date", "price" ]

},

"size" : 1

}

}

}

}

}

}

可能的响应：

{

...

"aggregations": {

"top\_tags": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "hat",

"doc\_count": 3,

"top\_sales\_hits": {

"hits": {

"total" : {

"value": 3,

"relation": "eq"

},

"max\_score": null,

"hits": [

{

"\_index": "sales",

"\_type": "\_doc",

"\_id": "AVnNBmauCQpcRyxw6ChK",

"\_source": {

"date": "2015/03/01 00:00:00",

"price": 200

},

"sort": [

1425168000000

],

"\_score": null

}

]

}

}

},

{

"key": "t-shirt",

"doc\_count": 3,

"top\_sales\_hits": {

"hits": {

"total" : {

"value": 3,

"relation": "eq"

},

"max\_score": null,

"hits": [

{

"\_index": "sales",

"\_type": "\_doc",

"\_id": "AVnNBmauCQpcRyxw6ChL",

"\_source": {

"date": "2015/03/01 00:00:00",

"price": 175

},

"sort": [

1425168000000

],

"\_score": null

}

]

}

}

},

{

"key": "bag",

"doc\_count": 1,

"top\_sales\_hits": {

"hits": {

"total" : {

"value": 1,

"relation": "eq"

},

"max\_score": null,

"hits": [

{

"\_index": "sales",

"\_type": "\_doc",

"\_id": "AVnNBmatCQpcRyxw6ChH",

"\_source": {

"date": "2015/01/01 00:00:00",

"price": 150

},

"sort": [

1420070400000

],

"\_score": null

}

]

}

}

}

]

}

}

}

字段折叠示例

字段折叠或结果分组是一项功能，可将结果集按逻辑分组，然后按组返回顶级文档。组的顺序由组中第一个文档的相关性确定。在 Elasticsearch 中，这可以通过将 top\_hits 聚合器包装为子聚合器的存储桶聚合器来实现。

在下面的示例中，我们搜索了所有已爬的网页。对于每个网页，我们存储该网页所属的正文和域名。通过在 domain 字段中定义 terms 聚合器，我们可以按域对网页结果集进行分组。然后将 top\_hits 聚合器定义为子聚合器，以便按存储桶收集最匹配的匹配项。

还定义了 max 聚合器，terms 聚合器的排序功能使用该聚合器根据存储桶中最相关文档的相关性顺序返回存储桶。

POST /sales/\_search

{

"query": {

"match": {

"body": "elections"

}

},

"aggs": {

"top\_sites": {

"terms": {

"field": "domain",

"order": {

"top\_hit": "desc"

}

},

"aggs": {

"top\_tags\_hits": {

"top\_hits": {}

},

"top\_hit" : {

"max": {

"script": {

"source": "\_score"

}

}

}

}

}

}

}

目前需要最大 (或最小) 聚合器，以确保根据每个域中最相关网页的分数对术语聚合器中的存储桶进行排序。遗憾的是，top\_hits 聚合器尚不能在术语聚合器的 order 选项中使用。

top\_hits 支持嵌套或反向嵌套聚合器

如果 top\_hits 聚合包在 nested 或 reverse\_nested 聚合中，则返回嵌套的匹配数据。从某种意义上说，嵌套命中是隐藏的迷你文档，它们是常规文档的一部分，在映射中已配置了嵌套字段类型。如果 top\_hits 聚合包装在 nested 或 reverse\_nested 聚合中，则可以取消隐藏这些文档。在嵌套类型映射中了解有关嵌套的更多信息。

如果已配置嵌套类型，则单个文档实际上将被索引为多个 Lucene 文档，并且它们共享相同的 ID。为了确定嵌套匹配的身份，除了 ID 之外，还需要更多的信息，因此这就是嵌套匹配还包括其嵌套身份的原因。嵌套标识保留在搜索匹配的\_nested 字段下，并包括数组字段和嵌套匹配所属的数组字段中的偏移量。偏移量基于零。

让我们看看它如何与真实例子一起工作。考虑以下映射：

PUT /sales

{

"mappings": {

"properties" : {

"tags" : { "type" : "keyword" },

"comments" : {

"type" : "nested",

"properties" : {

"username" : { "type" : "keyword" },

"comment" : { "type" : "text" }

}

}

}

}

}

注释是一个数组，用于在产品对象下保存嵌套文档 .

一些文档

PUT /sales/\_doc/1?refresh

{

"tags": ["car", "auto"],

"comments": [

{"username": "baddriver007", "comment": "This car could have better brakes"},

{"username": "dr\_who", "comment": "Where's the autopilot? Can't find it"},

{"username": "ilovemotorbikes", "comment": "This car has two extra wheels"}

]

}

现在可以执行以下顶级点击聚合 (包装在嵌套聚合中)：

POST /sales/\_search

{

"query": {

"term": { "tags": "car" }

},

"aggs": {

"by\_sale": {

"nested" : {

"path" : "comments"

},

"aggs": {

"by\_user": {

"terms": {

"field": "comments.username",

"size": 1

},

"aggs": {

"by\_nested": {

"top\_hits":{}

}

}

}

}

}

}

}

具有嵌套匹配项的热门匹配项摘要，位于数组字段 comments 的第一个插槽中：

{

...

"aggregations": {

"by\_sale": {

"by\_user": {

"buckets": [

{

"key": "baddriver007",

"doc\_count": 1,

"by\_nested": {

"hits": {

"total" : {

"value": 1,

"relation": "eq"

},

"max\_score": 0.3616575,

"hits": [

{

"\_index": "sales",

"\_type" : "\_doc",

"\_id": "1",

"\_nested": {

"field": "comments",

"offset": 0

},

"\_score": 0.3616575,

"\_source": {

"comment": "This car could have better brakes",

"username": "baddriver007"

}

}

]

}

}

}

...

]

}

}

}

}

包含嵌套匹配项的数组字段的名称

如果嵌套命中在包含数组中

嵌套匹配的来源

如果请求\_source，则返回嵌套对象源的一部分，而不是文档的整个源。还可以通过位于 nested 或 reverse\_nested 聚合中的 top\_hits 聚合访问嵌套对象内部级别上的存储字段。

仅嵌套匹配在匹配中具有\_nested 字段，非嵌套 (常规) 匹配将没有\_nested 字段。

如果未启用\_source，则\_nested 中的信息还可用于解析其他地方的原始源。

在下面的示例中，嵌套匹配项位于 nested\_grand\_child\_field 字段的第一个插槽中，然后位于 nested\_child\_field 字段的第二个慢速位置中：

...

"hits": {

"total" : {

"value": 2565,

"relation": "eq"

},

"max\_score": 1,

"hits": [

{

"\_index": "a",

"\_type": "b",

"\_id": "1",

"\_score": 1,

"\_nested" : {

"field" : "nested\_child\_field",

"offset" : 1,

"\_nested" : {

"field" : "nested\_grand\_child\_field",

"offset" : 0

}

}

"\_source": ...

},

...

]

}

...

### 15 值计数聚合

值计数聚合

单值度量标准聚合，用于计算从聚合文档中提取的值的数量。这些值可以从文档中的特定字段中提取，也可以由提供的脚本生成。通常，此聚合器将与其他单值聚合一起使用。例如，在计算 avg 时，可能会对计算平均值的值数量感兴趣。

POST /sales/\_search?size=0

{

"aggs" : {

"types\_count" : { "value\_count" : { "field" : "type" } }

}

}

响应结果：

{

...

"aggregations": {

"types\_count": {

"value": 7

}

}

}

聚合的名称 (上面的 types\_count) 也是可以从返回的响应中检索聚合结果的键。

脚本

计算脚本生成的值：

POST /sales/\_search?size=0

{

"aggs" : {

"type\_count" : {

"value\_count" : {

"script" : {

"source" : "doc['type'].value"

}

}

}

}

}

这会将脚本参数解释为具有轻松脚本语言且没有脚本参数的嵌入式脚本。要使用存储的脚本，请使用以下语法：

POST /sales/\_search?size=0

{

"aggs" : {

"types\_count" : {

"value\_count" : {

"script" : {

"id": "my\_script",

"params" : {

"field" : "type"

}

}

}

}

}

}

### 16 中位数绝对偏差聚合

中位数绝对偏差聚合

此单值聚合近似于其搜索结果的中位数绝对偏差。

中位数绝对偏差是变异性的量度。这是一个可靠的统计信息，这意味着它对于描述可能具有异常值或未正常分布的数据很有用。对于此类数据，它比标准偏差更具描述性。

计算为每个数据点与整个样本的中值的偏差的中值。即，对于随机变量 X，中值绝对偏差为中值 (| median (X)-X〜i〜|)。

示例

假设我们的数据代表一到五星级的产品评论。此类评论通常被概括为平均值，这很容易理解，但却无法描述评论的可变性。估计中值绝对偏差可以洞悉彼此之间有多少差异。

在此示例中，我们的产品平均评分为 3 星。让我们看一下其评分的中位数绝对偏差，以确定它们之间的差异

GET reviews/\_search

{

"size": 0,

"aggs": {

"review\_average": {

"avg": {

"field": "rating"

}

},

"review\_variability": {

"median\_absolute\_deviation": {

"field": "rating"

}

}

}

}

评分必须是数字字段

所得的 2 的绝对中位数绝对值告诉我们，评级中存在相当大的可变性。评论者必须对此产品有不同的意见。

{

...

"aggregations": {

"review\_average": {

"value": 3.0

},

"review\_variability": {

"value": 2.0

}

}

}

近似值

计算中值绝对偏差的幼稚实现将整个样本存储在内存中，因此此聚合计算出一个近似值。 它使用 TDigest 数据结构 来近似估计样本中位数和样本中位数的偏差中位数，有关 TDigests 近似特性的更多信息，请参见 百分位数通常是近似值 。

资源使用率与 TDigest 分位数逼近度的精度之间的权衡，以及由此聚合的中位数绝对偏差近似值的精度，由 compression 参数控制。较高的 compression 设置以较高的内存使用量为代价提供了更准确的近似值。有关 TDigestcompression 参数的特征的更多信息，请参见压缩。

GET reviews/\_search

{

"size": 0,

"aggs": {

"review\_variability": {

"median\_absolute\_deviation": {

"field": "rating",

"compression": 100

}

}

}

}

此聚合的默认压缩值为 1000。在此压缩级别，此聚合通常在精确结果的 5％以内，但是观察到的性能将取决于样本数据。

脚本

此度量标准聚合支持脚本。在上面的示例中，产品评论的规模从 1 到 5。如果我们想将它们修改为 1 到 10 的比例，则可以使用脚本。

要提供内联脚本：

GET reviews/\_search

{

"size": 0,

"aggs": {

"review\_variability": {

"median\_absolute\_deviation": {

"script": {

"lang": "painless",

"source": "doc['rating'].value \* params.scaleFactor",

"params": {

"scaleFactor": 2

}

}

}

}

}

}

提供储存脚本

GET reviews/\_search

{

"size": 0,

"aggs": {

"review\_variability": {

"median\_absolute\_deviation": {

"script": {

"id": "my\_script",

"params": {

"field": "rating"

}

}

}

}

}

}

缺失值

missing 参数定义应如何处理缺少值的文档。默认情况下，它们将被忽略，但也可以将它们视为具有值。

让我们保持乐观，假设有些评论者非常喜欢该产品，以至于他们忘记给它打分了。我们给他们分配五颗星

GET reviews/\_search

{

"size": 0,

"aggs": {

"review\_variability": {

"median\_absolute\_deviation": {

"field": "rating",

"missing": 5

}

}

}

}

### 20 桶聚合

桶聚合不像度量聚合那样计算字段的度量，而是创建文档存储桶。每个存储桶都与一个标准 (取决于聚合类型) 相关联，该标准确定当前上下文中的文档是否 “落入” 其中。换句话说，存储桶有效地定义了文档集。除了存储桶本身之外，存储桶聚合还计算并返回 “落入” 每个存储桶的文档数。

桶聚合与 metrics 聚合相反，可以保存子聚合。这些子聚合将针对其” 父”bucket 聚合创建的 bucket 进行聚合。

有不同的存储桶聚合器，每个聚合器都有不同的 “存储桶” 策略。一些定义单个存储桶，一些定义固定数量的多个存储桶，另一些定义在聚合过程中动态创建存储桶。

注意:

单个响应中允许的最大存储桶数受 search.max\_buckets 的动态集群设置的限制。它的默认值为 10,000，尝试返回大于限制的请求将失败，并发生异常。

### 21 邻接矩阵聚合

邻接矩阵聚合

桶聚合返回一种形式的邻接矩阵。 该请求提供了一个命名过滤器表达式的集合，类似于 filters 聚合请求。响应中的每个存储桶代表相交过滤器矩阵中的一个非空单元。

给定名为 A，B 和 C 的过滤器，响应将返回具有以下名称的存储桶：

A B C

A A A&B A&C

B B B&C

C C

相交的存储桶 (例如 A＆C) 使用两个与号字符分隔的过滤器名称的组合来标记。请注意，响应中也不包含 “C＆A” 存储桶，因为这与 “ A＆C” 属于同一组文档。据说矩阵是对称的，所以我们只返回一半。为此，我们对过滤器名称字符串进行排序，并始终使用一对中的最低值作为 “＆” 分隔符左侧的值。

如果客户端希望使用除与号默认值以外的分隔符字符串，则可以在请求中传递替代的分隔符参数。

例：

PUT /emails/\_bulk?refresh

{ "index" : { "\_id" : 1 } }

{ "accounts" : ["hillary", "sidney"]}

{ "index" : { "\_id" : 2 } }

{ "accounts" : ["hillary", "donald"]}

{ "index" : { "\_id" : 3 } }

{ "accounts" : ["vladimir", "donald"]}

GET emails/\_search

{

"size": 0,

"aggs" : {

"interactions" : {

"adjacency\_matrix" : {

"filters" : {

"grpA" : { "terms" : { "accounts" : ["hillary", "sidney"] }},

"grpB" : { "terms" : { "accounts" : ["donald", "mitt"] }},

"grpC" : { "terms" : { "accounts" : ["vladimir", "nigel"] }}

}

}

}

}

}

在上面的示例中，我们分析了电子邮件，以查看哪些个人已交换了消息。我们将分别获取每个组的计数，以及已记录交互的成对组的消息计数。

响应:

{

"took": 9,

"timed\_out": false,

"\_shards": ...,

"hits": ...,

"aggregations": {

"interactions": {

"buckets": [

{

"key":"grpA",

"doc\_count": 2

},

{

"key":"grpA&grpB",

"doc\_count": 1

},

{

"key":"grpB",

"doc\_count": 2

},

{

"key":"grpB&grpC",

"doc\_count": 1

},

{

"key":"grpC",

"doc\_count": 1

}

]

}

}

}

用法

这种聚合本身可以提供创建无向加权图所需的所有数据。但是，当与 date\_histogram 之类的子聚合一起使用时，结果可以提供执行动态网络分析所需的附加数据级别检查互动随着时间的推移变得重要。

限制

对于 N 个过滤器，所产生的存储桶矩阵可以为 N²/ 2，因此默认情况下最大限制为 100 个过滤器。可以使用 index.max\_adjacency\_matrix\_filters 索引级别设置更改此设置。

### 22 自动间隔日期直方图聚合

自动间隔日期直方图聚合

与日期聚合直方图除了提供一个间隔来用作每个存储桶的宽度外，还提供了一个目标存储桶数来指示所需的存储桶数，并自动选择存储桶的间隔以最佳地实现该目标。返回的存储桶数将始终小于或等于此目标数。

存储桶字段是可选的，如果未指定，则默认为 10 个存储桶。

要求目标为 10 个存储桶。

POST /sales/\_search?size=0

{

"aggs" : {

"sales\_over\_time" : {

"auto\_date\_histogram" : {

"field" : "date",

"buckets" : 10

}

}

}

}

键

在内部，日期表示为一个 64 位数字，表示自该时间点以来的毫秒数。这些时间戳作为存储桶键返回。 key\_as\_string 是使用 format 参数指定的格式转换为格式化日期字符串的相同时间戳：

提示

如果未指定格式，则它将使用在字段映射中指定的第一个日期格式。

POST /sales/\_search?size=0

{

"aggs" : {

"sales\_over\_time" : {

"auto\_date\_histogram" : {

"field" : "date",

"buckets" : 5,

"format" : "yyyy-MM-dd"

}

}

}

}

支持富有表现力的日期格式模式

响应：

{

...

"aggregations": {

"sales\_over\_time": {

"buckets": [

{

"key\_as\_string": "2015-01-01",

"key": 1420070400000,

"doc\_count": 3

},

{

"key\_as\_string": "2015-02-01",

"key": 1422748800000,

"doc\_count": 2

},

{

"key\_as\_string": "2015-03-01",

"key": 1425168000000,

"doc\_count": 2

}

],

"interval": "1M"

}

}

}

时间间隔

根据聚合收集的数据选择返回的存储桶的间隔，以使返回的存储桶数小于或等于请求的数量。返回的可能间隔是：

秒 | 1, 5, 10 和 30 的倍数

分 | 1, 5, 10 和 30 的倍数

时 | 1, 3 和 12 的倍数

日 | 1, 和 7 的倍数

月 | 1, 和 3 的倍数

年 | 1, 5, 10, 20, 50 和 100 的倍数

在最坏的情况下，如果每天的存储桶数超出了请求的存储桶数，则返回的存储桶数将是请求的存储桶数的 1/7。

时区

日期时间存储在 UTC 中的 Elasticsearch 中。默认情况下，所有存储和取整也是在 UTC 中完成的。time\_zone 参数可用于指示存储段应使用不同的时区。

时区可以指定为 ISO 8601 UTC 偏移量 (例如 +01：00 或 -08：00)，也可以指定为时区 ID ，例如 TZ 数据库中使用的标识符 America / Los\_Angeles。

考虑以下示例：

PUT my\_index/log/1?refresh

{

"date": "2015-10-01T00:30:00Z"

}

PUT my\_index/log/2?refresh

{

"date": "2015-10-01T01:30:00Z"

}

PUT my\_index/log/3?refresh

{

"date": "2015-10-01T02:30:00Z"

}

GET my\_index/\_search?size=0

{

"aggs": {

"by\_day": {

"auto\_date\_histogram": {

"field": "date",

"buckets" : 3

}

}

}

}

如果未指定时区，则使用 UTC，从 UTC 2015 年 10 月 1 日午夜开始返回三个 1 小时时段：

{

...

"aggregations": {

"by\_day": {

"buckets": [

{

"key\_as\_string": "2015-10-01T00:00:00.000Z",

"key": 1443657600000,

"doc\_count": 1

},

{

"key\_as\_string": "2015-10-01T01:00:00.000Z",

"key": 1443661200000,

"doc\_count": 1

},

{

"key\_as\_string": "2015-10-01T02:00:00.000Z",

"key": 1443664800000,

"doc\_count": 1

}

],

"interval": "1h"

}

}

}

如果指定了 -01：00 的 time\_zone，则午夜在 UTC 午夜前一小时开始：

GET my\_index/\_search?size=0

{

"aggs": {

"by\_day": {

"auto\_date\_histogram": {

"field": "date",

"buckets" : 3,

"time\_zone": "-01:00"

}

}

}

}

现在，仍返回三个 1 小时时段，但第一个时段从 2015 年 9 月 30 日晚上 11:00 开始，因为这是该时段在指定时区的本地时间。

{

...

"aggregations": {

"by\_day": {

"buckets": [

{

"key\_as\_string": "2015-09-30T23:00:00.000-01:00",

"key": 1443657600000,

"doc\_count": 1

},

{

"key\_as\_string": "2015-10-01T00:00:00.000-01:00",

"key": 1443661200000,

"doc\_count": 1

},

{

"key\_as\_string": "2015-10-01T01:00:00.000-01:00",

"key": 1443664800000,

"doc\_count": 1

}

],

"interval": "1h"

}

}

}

key\_as\_string 值表示指定时区中每一天的午夜。

警告

当使用 DST (夏令时) 更改之后的时区时，接近更改发生时的存储区的大小可能会与相邻存储区的大小略有不同。例如，假设某个夏令时开始于 CET 时区：2016 年 3 月 27 日凌晨 2 点，时钟被调到本地时间 1 小时到凌晨 3 点。如果聚合的结果是每日存储桶，则当天的存储桶将仅保留 23 小时的数据，而不是其他存储桶通常的 24 小时。对于较短的时间间隔 (例如 12 小时在这里，DST 轮换发生的 3 月 27 日上午，我们只有 11 小时的时段。

脚本

像普通的 date\_histogram，同时支持文档级脚本和值级脚本。但是，此聚合不支持 min\_doc\_count，extended\_bounds 和 order 参数。

最小间隔参数

minimum\_interval 允许调用方指定应使用的最小舍入间隔。这可以使收集过程更有效率，因为聚合不会尝试以小于 minimum\_interval 的任何时间间隔取整。

minimum\_interval 的可接受单位为：

年

月

日

时

分

秒

POST /sales/\_search?size=0

{

"aggs" : {

"sale\_date" : {

"auto\_date\_histogram" : {

"field" : "date",

"buckets": 10,

"minimum\_interval": "minute"

}

}

}

}

缺失值

missing 参数定义应如何处理缺少值的文档。默认情况下，它们将被忽略，但也可以将它们视为具有值。

POST /sales/\_search?size=0

{

"aggs" : {

"sale\_date" : {

"auto\_date\_histogram" : {

"field" : "date",

"buckets": 10,

"missing": "2000/01/01"

}

}

}

}

在 publish\_date 字段中没有值的文档将与值为 2000-01-01 的文档归入同一存储桶。

### 23 子聚合

一种特殊的单存储桶聚合，用于选择具有指定类型的子文档，如 join 字段。

该聚合只有一个选项：

type- 应选择的子类型。

例如，假设我们有一个问题和答案的索引。答案类型在映射中具有以下 join 字段：

PUT child\_example

{

"mappings": {

"properties": {

"join": {

"type": "join",

"relations": {

"question": "answer"

}

}

}

}

}

question 文档包含一个标记字段，而 answer 文档包含一个所有者字段。使用子级聚合，即使两个字段存在两种不同的文档中，也可以在单个请求中将标签存储桶映射到所有者存储桶。

问题文档的示例:

PUT child\_example/\_doc/1

{

"join": {

"name": "question"

},

"body": "<p>I have Windows 2003 server and i bought a new Windows 2008 server...",

"title": "Whats the best way to file transfer my site from server to a newer one?",

"tags": [

"windows-server-2003",

"windows-server-2008",

"file-transfer"

]

}

answer 文档的示例：

PUT child\_example/\_doc/2?routing=1

{

"join": {

"name": "answer",

"parent": "1"

},

"owner": {

"location": "Norfolk, United Kingdom",

"display\_name": "Sam",

"id": 48

},

"body": "<p>Unfortunately you're pretty much limited to FTP...",

"creation\_date": "2009-05-04T13:45:37.030"

}

PUT child\_example/\_doc/3?routing=1&refresh

{

"join": {

"name": "answer",

"parent": "1"

},

"owner": {

"location": "Norfolk, United Kingdom",

"display\_name": "Troll",

"id": 49

},

"body": "<p>Use Linux...",

"creation\_date": "2009-05-05T13:45:37.030"

}

可以建立以下将两者连接在一起的请求：

POST child\_example/\_search?size=0

{

"aggs": {

"top-tags": {

"terms": {

"field": "tags.keyword",

"size": 10

},

"aggs": {

"to-answers": {

"children": {

"type" : "answer"

},

"aggs": {

"top-names": {

"terms": {

"field": "owner.display\_name.keyword",

"size": 10

}

}

}

}

}

}

}

}

type 指向名称为 answer 的类型 / 映射。

上面的示例返回顶部的问题标签，每个标签返回顶部的答案所有者。

可能的响应：

{

"took": 25,

"timed\_out": false,

"\_shards": {

"total": 1,

"successful": 1,

"skipped" : 0,

"failed": 0

},

"hits": {

"total" : {

"value": 3,

"relation": "eq"

},

"max\_score": null,

"hits": []

},

"aggregations": {

"top-tags": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "file-transfer",

"doc\_count": 1,

"to-answers": {

"doc\_count": 2,

"top-names": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "Sam",

"doc\_count": 1

},

{

"key": "Troll",

"doc\_count": 1

}

]

}

}

},

{

"key": "windows-server-2003",

"doc\_count": 1,

"to-answers": {

"doc\_count": 2,

"top-names": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "Sam",

"doc\_count": 1

},

{

"key": "Troll",

"doc\_count": 1

}

]

}

}

},

{

"key": "windows-server-2008",

"doc\_count": 1,

"to-answers": {

"doc\_count": 2,

"top-names": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "Sam",

"doc\_count": 1

},

{

"key": "Troll",

"doc\_count": 1

}

]

}

}

}

]

}

}

}

1. 标签为 file-transfer，windows-server-2003 等的问题文档的数量。

2. 与标记为 file-transfer，windows-server-2003 等的问题文档相关的回答文档的数量。

### 24 复合聚合

多桶聚合是指从不同的源创建的复合桶.

与其他 多桶 聚合不同的是，复合 聚合可用于高效地对多层聚合中的 所有 桶进行分页。 这种聚合提供了一种流特定聚合的所有桶的方法，类似于 scroll 对文档做的操作。

复合桶是根据每个文档提取 / 创建的值的组合构建的，每个组合都被视为一个复合桶。

例如下面的文档：

{

"keyword": ["foo", "bar"],

"number": [23, 65, 76]

}

… 创建下面的复合桶， 使用 keyword 和 number 两个字段的值进行聚合：

{ "keyword": "foo", "number": 23 }

{ "keyword": "foo", "number": 65 }

{ "keyword": "foo", "number": 76 }

{ "keyword": "bar", "number": 23 }

{ "keyword": "bar", "number": 65 }

{ "keyword": "bar", "number": 76 }

Values source

sources 参数控制用于构建复合桶的源。 定义 sources 的顺序很重要，因为它还控制着键的返回顺序。

每个 sources 的名称必须是唯一的.

下面是三种不同类型的 values source:

Terms

terms 的 value source 相当于一个简单的 terms 聚合。 这些值是从 field 或 script 中提取的，与 terms 聚合完全一样。

使用 field 为复合桶创建值:

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"sources" : [

{ "product": { "terms" : { "field": "product" } } }

]

}

}

}

}

也可以使用 script 为复合桶创建值:

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"sources" : [

{

"product": {

"terms" : {

"script" : {

"source": "doc['product'].value",

"lang": "painless"

}

}

}

}

]

}

}

}

}

Histogram

histogram 值源可应用于数值，以在值上构建固定大小的间隔。 interval 参数定义如何转换数值。 例如，将 interval 的值设为 5，会将任何数值转换到最接近的间隔， 值 101 将会被转换为 100 ，这是这个键的间隔在 100 到 105 之间。

Example:

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"sources" : [

{ "histo": { "histogram" : { "field": "price", "interval": 5 } } }

]

}

}

}

}

这些值是从数字字段或返回数字值的脚本中构建的:

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"sources" : [

{

"histo": {

"histogram" : {

"interval": 5,

"script" : {

"source": "doc['price'].value",

"lang": "painless"

}

}

}

}

]

}

}

}

}

Date Histogram

date\_histogram 与 histogram 值源相似，但不同之处是由日期 / 时间表达式指定的时间间隔:：

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"sources" : [

{ "date": { "date\_histogram" : { "field": "timestamp", "calendar\_interval": "1d" } } }

]

}

}

}

}

上面的示例每天创建一个间隔，并将所有 timestamp 值转换为其最接近的间隔的开始值。间隔的可用表达式：年，季度，月，周，天，hour，分钟，秒

时间值也可以通过时间单位支持的缩写来指定。 请注意，不支持小数时间值，但是您可以通过转移到其他时间单位来解决此问题 (例如，可以将 1.5h 指定为 90 分钟)。

Format

在内部，日期表示为一个 64 位的数字，表示自该时间点以来的毫秒数。这些时间戳作为存储桶键返回。可以使用格式化参数指定的格式来返回格式化的日期字符串：

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"sources" : [

{

"date": {

"date\_histogram" : {

"field": "timestamp",

"calendar\_interval": "1d",

"format": "yyyy-MM-dd"

}

}

}

]

}

}

}

}

支持表示日期格式模式

时区

Elasticsearch 使用世界标准时间（UTC）存储日期时间。 默认情况下，所有存储和取整也是使用世界标准时间（UTC）。 time\_zone 参数用于指定存储时所使用不同的时区

时区可以指定为 ISO 8601 UTC 偏移 (例如 +01:00 或者 -08:00) , 也可以指定为时区数据库（TimeZoneDB）中的时区编号，例如 America/Los\_Angeles.

混合不同的值源

sources 参数数接受源数组的值。可以混合使用不同的值源来创建复合存储桶。例如：

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"sources" : [

{ "date": { "date\_histogram": { "field": "timestamp", "calendar\_interval": "1d" } } },

{ "product": { "terms": {"field": "product" } } }

]

}

}

}

}

这将从两个值源创建的值中（ date\_histogram 和 terms）创建复合桶。

每个桶均由两个值组成，其中每一个值源都在聚合中定义。 允许任何类型的组合，并且在存储桶中，数组中的排序将被保留。

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"sources" : [

{ "shop": { "terms": {"field": "shop" } } },

{ "product": { "terms": { "field": "product" } } },

{ "date": { "date\_histogram": { "field": "timestamp", "calendar\_interval": "1d" } } }

]

}

}

}

}

排序

默认情况下，复合桶按其自然顺序排序。 值按其值的升序排序。 当请求多个值源时，将对每个值源进行排序， 将复合桶的第一个值与另一个复合桶的第一个值进行比较 ，如果它们相等，将比较复合桶中的下一个值。 这就意味着复合桶 [foo, 100] 被认为小于 [foobar, 0] ，因为 foo 被认为小于 foobar。 可以通过直接将 order 设置为 asc (默认的) 或者 desc (降序) 来定义每个值源的排序方式。例如:

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"sources" : [

{ "date": { "date\_histogram": { "field": "timestamp", "calendar\_interval": "1d", "order": "desc" } } },

{ "product": { "terms": {"field": "product", "order": "asc" } } }

]

}

}

}

}

… 当比较源为 date\_histogram 的值时，将按降序对复合桶进行排序；当比较源 terms 的值时，将按升序对复合桶进行排序。

缺失桶

默认情况下，没有给定源值的文档将被忽略。 可以通过将 missing\_bucket 设置为 true (默认值是 false)，把它们包括在响应中。例如 :

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"sources" : [

{ "product\_name": { "terms" : { "field": "product", "missing\_bucket": true } } }

]

}

}

}

}

在上面的示例中，当 product 字段没有值时，源 product\_name 将会被设置一个显式的 null 值 。 通过源 order 参数指定 null 值应该排在首位 (升序，asc) 或者是末尾 (降序，desc).

设置返回数量

通过设置 size 参数指定返回复合桶的数量。 每个复合桶都被视为单个桶， 当我们将 size 指定为 10 时，将返回从源值创建的前 10 个复合桶。 响应包含数组中每个复合桶的值，该数组包含从每个值源提取的值。

分页

如果复合桶的数量过多 (或者不知其数量) ，从而无法在单个响应中返回，那么可以将检索拆分为多个请求。 由于复合存储桶在本质上是扁平的，所以请求的 size 恰好是将在响应中返回的复合存储桶的数量 (假设它们能够返回 size 的复合桶). 如果要检索所有复合存储桶，最好使用较小的 size (100 或者 1000) ，然后使用 after 参数去检索余下的结果。 例如:

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"size": 2,

"sources" : [

{ "date": { "date\_histogram": { "field": "timestamp", "calendar\_interval": "1d" } } },

{ "product": { "terms": {"field": "product" } } }

]

}

}

}

}

… 返回的结果:

{

...

"aggregations": {

"my\_buckets": {

"after\_key": { ①

"date": 1494288000000,

"product": "mad max"

},

"buckets": [

{

"key": {

"date": 1494201600000,

"product": "rocky"

},

"doc\_count": 1

},

{

"key": {

"date": 1494288000000,

"product": "mad max"

},

"doc\_count": 2

}

]

}

}

}

①. 查询最后一个复合桶。

注意

after\_key 等于 Pipeline aggregations. 执行任何过滤前返回的最后一个桶 。如果通过 pipeline aggregation（管道聚合）过滤 / 删除全部的桶，那么 after\_key 将包含过滤前的最后一个桶.

after 参数可以被用于检索在前一个检索中返回的最后一个复合桶。 下面的示例中， 可以在 after\_key 值中找到前一个检索中返回的最后一个桶，并且根据该值，使用下面的方法进行分页检索:

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"size": 2,

"sources" : [

{ "date": { "date\_histogram": { "field": "timestamp", "calendar\_interval": "1d", "order": "desc" } } },

{ "product": { "terms": {"field": "product", "order": "asc" } } }

],

"after": { "date": 1494288000000, "product": "mad max" } ①

}

}

}

}

① 聚合中的排序字段应该使用 after 的字段.

子聚合

像 multi-bucket （多桶）聚合一样， composite （复合）聚合也可以包含子聚合。这些子聚合可用于计算其他存储桶或有关此父聚合创建的每个复合桶的统计信息。 如，以下示例计算每个复合桶字段的平均值:

GET /\_search

{

"aggs" : {

"my\_buckets": {

"composite" : {

"sources" : [

{ "date": { "date\_histogram": { "field": "timestamp", "calendar\_interval": "1d", "order": "desc" } } },

{ "product": { "terms": {"field": "product" } } }

]

},

"aggregations": {

"the\_avg": {

"avg": { "field": "price" }

}

}

}

}

}

… returns:

{

...

"aggregations": {

"my\_buckets": {

"after\_key": {

"date": 1494201600000,

"product": "rocky"

},

"buckets": [

{

"key": {

"date": 1494460800000,

"product": "apocalypse now"

},

"doc\_count": 1,

"the\_avg": {

"value": 10.0

}

},

{

"key": {

"date": 1494374400000,

"product": "mad max"

},

"doc\_count": 1,

"the\_avg": {

"value": 27.0

}

},

{

"key": {

"date": 1494288000000,

"product" : "mad max"

},

"doc\_count": 2,

"the\_avg": {

"value": 22.5

}

},

{

"key": {

"date": 1494201600000,

"product": "rocky"

},

"doc\_count": 1,

"the\_avg": {

"value": 10.0

}

}

]

}

}

}

Pipeline aggregations（管道聚合）

当前复合聚合与管道聚合并不兼容，在大多数情况下也没有意义。 例如， 由于复合聚合的分页特性，单个逻辑分区 (例如一天) 可能分布在多个页面上。 由于管道聚合纯粹是 对桶的最终列表的后处理，因此在复合页面上运行类似派生的操作可能会导致不准确的结果，因为它只考虑了该页面上的 “部分” 结果。

将来可能会支持自我包含在单个桶中的管道聚合 (例如 bucket\_selector)

### 25 日期直方图聚合

日期直方图聚合

这种多桶聚合类似于普通的 histogram, 但只能与日期值一起使用。由于日期在 Elasticsearch 中内部以长值表示，因此在日期中使用普通的 histogram 可能并不准确。 两种 API 的主要区别在于，可以使用日期 / 时间表达式指定间隔。基于时间的数据需要特殊的支持，因为基于时间的间隔并不总是固定长度。

日历和固定间隔

当我们配置日期直方图聚合时，可以通过两种方式指定时间间隔：日历感知时间间隔和固定时间间隔。

日历感知间隔在夏令时会更改特定日期的时间长度，月份具有不同的天数，并且可以将闰秒附加到特定年份。

相比之下，固定间隔始终是 SI 单位的倍数，并且不会根据日历环境而变化。

注意

组合 interval 字段已被弃用

在 [7.2] 中 interval 字段已被弃用

历史上，日历和固定间隔都是在单个 interval 字段中配置的，这导致语义混乱。指定 1d 将假定为日历感知时间，而 2d 将其解释为固定时间。 为了获得固定时间的 “一天” ，用户需要指定下一个较小的单位 (在这种情况下为，24h).

这种组合行为通常对于用户来说是未知的，即使知道该行为，也很难使用和混淆。

不建议使用此行为，而推荐使用两个新的显式字段: calendar\_interval 和 fixed\_interval.

通过在日历和时间间隔之间强制进行选择，时间间隔的语义对用户而言立即清晰可见，并且没有歧义。 旧的 interval 字段将在以后的版本中删除。

日历间隔

日历感知间隔使用 calendar\_interval 参数进行配置。 日历间隔只能以 “单数” 为单位指定 (1d, 1M, 等等)。 不支持倍数，如 2d, 将引发异常。

日历感知间隔可接受的单位为：

分 (m, 1m)

所有分钟数从 00 秒开始。

一分钟是指定时区中第一分钟的 00 秒与后一分钟的 00 秒之间的时间间隔，补偿任何中间的闰秒，因此，每小时的分钟数和秒数在开始和结束时相同。

小时 (h, 1h)

所有小时都始于 00 分 00 秒。

一小时 (1h) 是在指定时区的第一个小时的 00:00 分钟与第二个小时的下一个小时 00:00 分钟之间的时间间隔，可补偿任何中间的润秒。因此，每小时的分钟数和秒数在开始和结束时相同。

天 (d, 1d)

所有天都是从最早的时间开始，通常是 00:00:00 (凌晨).

一天 (1d) 是指定时区中一天的开始与第二天的开始之间的间隔，补偿任何中间的时间变化。

周 (w, 1w)

一周是从指定的时区以一周的同一天为开始到下一周的时间之间的间隔。

月 (M, 1M)

一个月是指在指定时区中，月份的开始日期和时间与月份的同一天和时间之间的间隔，月份中日期和时间在开始和结束时相同。

季度 (q, 1q)

一个季度 (1q) 是指月份开始日和当天的时间到三个月后同一天相同时间的时间间隔。

年 (y, 1y)

一年 (1y) 是指在指定时区中， 从开始月份的日期和时间与下一年同一天和相同时间之间的间隔，日期和时间在开始和结束时相同。

日历间隔示例

例如，下面是一个以月为时间间隔的聚合:

POST /sales/\_search?size=0

{

"aggs" : {

"sales\_over\_time" : {

"date\_histogram" : {

"field" : "date",

"calendar\_interval" : "month"

}

}

}

}

如果尝试使用倍数日历单位，则聚合将失败，因为日历间隔仅支持单个日历单位:

POST /sales/\_search?size=0

{

"aggs" : {

"sales\_over\_time" : {

"date\_histogram" : {

"field" : "date",

"calendar\_interval" : "2d"

}

}

}

}

{

"error" : {

"root\_cause" : [...],

"type" : "x\_content\_parse\_exception",

"reason" : "[1:82] [date\_histogram] failed to parse field [calendar\_interval]",

"caused\_by" : {

"type" : "illegal\_argument\_exception",

"reason" : "The supplied interval [2d] could not be parsed as a calendar interval.",

"stack\_trace" : "java.lang.IllegalArgumentException: The supplied interval [2d] could not be parsed as a calendar interval."

}

}

}

固定间隔

固定间隔使用 fixed\_interval 参数进行配置。

与日历感知间隔相比，固定间隔是固定数量的 SI 单位，无论它们位于日历上的什么位置，它们都不会偏离。就像一秒始终由 1000 毫秒组成。这样在任何多个受支持单位上都可以指定固定间隔。

然而，这也意味着固定间隔不能表示像月份这样的单位，因为一个月的时间并不是一个确切的值。尝试指定一个像月份或者季度这样的日历间隔会引发异常。

固定间隔可接受的单位有：

毫秒 (ms)

秒 (s)

定义为 1000 毫秒每秒

分钟 (m)

每分钟都从 00 秒开始。

定义为 60 秒每分钟 (60,000 毫秒)

小时 (h)

每小时都从 00 分 00 秒开始。定义为 60 分钟每小时 (3,600,000 毫秒)

天 (d)

每天都从最早的 00:00:00 (午夜) 开始。

定义为 24 小时每天 (86,400,000 毫秒)

固定间隔示例

如果我们重新创建之前的 “月份” calendar\_interval，我们可以用 30 天估算：

POST /sales/\_search?size=0

{

"aggs" : {

"sales\_over\_time" : {

"date\_histogram" : {

"field" : "date",

"fixed\_interval" : "30d"

}

}

}

}

但是如果我们尝试使用一个如星期这样的不支持的日历单位就会引发异常：

POST /sales/\_search?size=0

{

"aggs" : {

"sales\_over\_time" : {

"date\_histogram" : {

"field" : "date",

"fixed\_interval" : "2w"

}

}

}

}

{

"error" : {

"root\_cause" : [...],

"type" : "x\_content\_parse\_exception",

"reason" : "[1:82] [date\_histogram] failed to parse field [fixed\_interval]",

"caused\_by" : {

"type" : "illegal\_argument\_exception",

"reason" : "failed to parse setting [date\_histogram.fixedInterval] with value [2w] as a time value: unit is missing or unrecognized",

"stack\_trace" : "java.lang.IllegalArgumentException: failed to parse setting [date\_histogram.fixedInterval] with value [2w] as a time value: unit is missing or unrecognized"

}

}

}

笔记

任何情况下，当指定的结束时间不存在时，实际结束时间是最接近指定结束时间的可用时间。

分布广泛的应用程序还应该考虑多变场景，比如某些国家夏令时间开始 / 结束于凌晨 12:01 ，因此每年结束都伴随着 59 分钟周六 和 1 分钟周日，再比如一些国家是被国际换日线穿过。上述情况很容易导致不规则时区偏移。

通常，严格的测试能够确保你的时间间隔规范符合你的预期，尤其是关于时间改变事件的测试。

警告：为了避免意外结果，所有建立连接的客户端和服务器，都应该和一个可靠的网络时间服务保持同步。

注意

不支持小数时间，但是你可以通过转换成其它时间单位解决（例如： 1.5h 可以用 90m 代替）。

注意

你可以使用 时间单位 支持的缩写来指定时间。

键

在内部，日期被表示为一个从纪元时间（1970.01.01 00:00:00 UTC）开始的 64 位的毫秒时间戳。时间戳被作为桶的 key 键名返回。 key\_as\_string 是一个符合 format 参数规范的格式化日期字符串，由同一时间戳转换来。

提示

如果你未指定 format，则使用字段映射中指定的第一个日期格式。

POST /sales/\_search?size=0

{

"aggs" : {

"sales\_over\_time" : {

"date\_histogram" : {

"field" : "date",

"calendar\_interval" : "1M",

"format" : "yyyy-MM-dd"

}

}

}

}

1. 支持丰富的日期格式

Response:

{

...

"aggregations": {

"sales\_over\_time": {

"buckets": [

{

"key\_as\_string": "2015-01-01",

"key": 1420070400000,

"doc\_count": 3

},

{

"key\_as\_string": "2015-02-01",

"key": 1422748800000,

"doc\_count": 2

},

{

"key\_as\_string": "2015-03-01",

"key": 1425168000000,

"doc\_count": 2

}

]

}

}

}

时区

日期时间在 Elasticsearch 中使用 UTC 存储。默认所有存桶和范围查询也是使用 UTC 完成。如果想要使用 time\_zone 参数来区分桶，应该设置一个不同的时区。

你既可以将时区指定为带偏移量的 ISO 8601 UTC (例如 +01:00 或 -08:00)，也可以指定为 IANA 时区中的一个时区 ID，例如 America/Los\_Angeles 。

看一下下面的例子：

PUT my\_index/\_doc/1?refresh

{

"date": "2015-10-01T00:30:00Z"

}

PUT my\_index/\_doc/2?refresh

{

"date": "2015-10-01T01:30:00Z"

}

GET my\_index/\_search?size=0

{

"aggs": {

"by\_day": {

"date\_histogram": {

"field": "date",

"calendar\_interval": "day"

}

}

}

}

如果你未指定时区，则默认使用 UTC。

下面命令会将两个文件放到同一天的桶中，这些桶从 2015 年 10 月 1 日午夜开始。

{

...

"aggregations": {

"by\_day": {

"buckets": [

{

"key\_as\_string": "2015-10-01T00:00:00.000Z",

"key": 1443657600000,

"doc\_count": 2

}

]

}

}

}

如果您将 time\_zone 指定为 -01:00，则该时区中的午夜比 UTC 时区的午夜提前 1 小时：

GET my\_index/\_search?size=0

{

"aggs": {

"by\_day": {

"date\_histogram": {

"field": "date",

"calendar\_interval": "day",

"time\_zone": "-01:00"

}

}

}

}

第一个文档存入时间为 2015 年 9 月 30 日的桶，而第二个文档存入时间为 2015 年 10 月 1 日的桶：

{

...

"aggregations": {

"by\_day": {

"buckets": [

{

"key\_as\_string": "2015-09-30T00:00:00.000-01:00",

"key": 1443574800000,

"doc\_count": 1

},

{

"key\_as\_string": "2015-10-01T00:00:00.000-01:00",

"key": 1443661200000,

"doc\_count": 1

}

]

}

}

}

key\_as\_string 值表示指定时区中每天午夜时间。

警告

使用会随着 DST (夏令时) 更改的时区时，这些靠近发生变化的时间的桶的存储区大小可能与你使用的时间间隔所期望的略有不同。举例说明，假设一个 DST 在 CET 时区开始：在 2016 年 3 月 27 日凌晨 2 点，时钟被调后 1 小时到当地时间凌晨 3 点。如果你以天为间隔，则当天的桶只有 23 个小时的数据，而不是其它桶存储的 24 个小时的数据。对于较短的时间间隔也是如此，比如 12 小时，在 3 月 27 日凌晨 DST 发生变化时，桶中也只有 11 个小时的数据。

偏移量

offset 参数通过指定正（+）或负（-）偏移持续时间来更改每个桶的起始值，比如 1h 代表 1 小时，1d 代表 1 天，更多可能的持续时间选项，请参见 时间单位。

例如，当使用 天 作为间隔时，每个桶从午夜持续到午夜。将 偏移量 设置为 +6h ，每个桶的运行时间就变为从早上 6 点到第二天 早上 6 点。

PUT my\_index/\_doc/1?refresh

{

"date": "2015-10-01T05:30:00Z"

}

PUT my\_index/\_doc/2?refresh

{

"date": "2015-10-01T06:30:00Z"

}

GET my\_index/\_search?size=0

{

"aggs": {

"by\_day": {

"date\_histogram": {

"field": "date",

"calendar\_interval": "day",

"offset": "+6h"

}

}

}

}

上面的请求会将文档分组到开始于上午 6 点的桶，而不是从午夜开始的一个单独的桶。

{

...

"aggregations": {

"by\_day": {

"buckets": [

{

"key\_as\_string": "2015-09-30T06:00:00.000Z",

"key": 1443592800000,

"doc\_count": 1

},

{

"key\_as\_string": "2015-10-01T06:00:00.000Z",

"key": 1443679200000,

"doc\_count": 1

}

]

}

}

}

注意

每个桶的开始 offset 是在 time\_zone 调整完之后计算来的。

键响应

将 keyed 设置为 true 会将每个桶和一个唯一的字符串关联起来，并将范围作为散列值而不是数组返回。

POST /sales/\_search?size=0

{

"aggs" : {

"sales\_over\_time" : {

"date\_histogram" : {

"field" : "date",

"calendar\_interval" : "1M",

"format" : "yyyy-MM-dd",

"keyed": true

}

}

}

}

响应结果：

{

...

"aggregations": {

"sales\_over\_time": {

"buckets": {

"2015-01-01": {

"key\_as\_string": "2015-01-01",

"key": 1420070400000,

"doc\_count": 3

},

"2015-02-01": {

"key\_as\_string": "2015-02-01",

"key": 1422748800000,

"doc\_count": 2

},

"2015-03-01": {

"key\_as\_string": "2015-03-01",

"key": 1425168000000,

"doc\_count": 2

}

}

}

}

}

脚本

与正常的 直方图 一样，支持文档级别脚本和值级别脚本。你还可以使用 order 设置来控制返回桶的顺序，并可以通过 min\_doc\_count 设置对返回的桶进行过滤（默认情况下，与文档匹配的第一个桶和最后一个桶之间的所有桶都会被返回）。该直方图还支持 extended\_bounds 设置，这使得直方图的范围扩展到数据本身之外（更多信息，请查看 扩展范围 ）。

缺失值

missing 参数对如何处理文档缺失值进行了定义。默认情况下，它们会被忽略，但也可以认为它们有值。

POST /sales/\_search?size=0

{

"aggs" : {

"sale\_date" : {

"date\_histogram" : {

"field" : "date",

"calendar\_interval": "year",

"missing": "2000/01/01"

}

}

}

在 publish\_date 字段中没有值的文档将与值为 2000-01-01 的文档落入同一桶中。

排序

默认情况下，返回的桶按照他们的 key 升序排序，但是你可以通过 order 设置控制返回的顺序。该设置支持和 Terms Aggregation 同样的 order 功能。

使用脚本来按照星期几聚合

当你需要按照星期几来聚合时，使用如下脚本来返回结果：

POST /sales/\_search?size=0

{

"aggs": {

"dayOfWeek": {

"terms": {

"script": {

"lang": "painless",

"source": "doc['date'].value.dayOfWeekEnum.value"

}

}

}

}

}

响应结果：

{

...

"aggregations": {

"dayOfWeek": {

"doc\_count\_error\_upper\_bound": 0,

"sum\_other\_doc\_count": 0,

"buckets": [

{

"key": "7",

"doc\_count": 4

},

{

"key": "4",

"doc\_count": 3

}

]

}

}

}

响应中包含所有桶，它们使用每星期中的相对日作为键：1 代表星期一，2 代表星期二…… 7 代表星期日。

### 26 日期范围聚合

日期范围聚合

日期范围聚合是专用于日期值的范围聚合。该聚合和正常的 范围 聚合的区别主要在于：该聚合可以用 日期数学 表达式表示 from 值 和 to 值，还可以指定 返回 from 和 to 响应字段的日期格式。注意，该聚合包含 from 值，但不包含 to 值。

示例：

POST /sales/\_search?size=0

{

"aggs": {

"range": {

"date\_range": {

"field": "date",

"format": "MM-yyyy",

"ranges": [

{ "to": "now-10M/M" }, ①

{ "from": "now-10M/M" } ②

]

}

}

}

}

① < 现在减去 10 个月，向下舍入到月初

② >= 现在减去 10 个月，向下舍入到月初

在上面的例子中，我们创建了两个范围桶，第一个桶会将早于 10 个月之前的所有文档存储，第二个桶会将从 10 月之前开始的文档存储。

响应结果：

{

...

"aggregations": {

"range": {

"buckets": [

{

"to": 1.4436576E12,

"to\_as\_string": "10-2015",

"doc\_count": 7,

"key": "\*-10-2015"

},

{

"from": 1.4436576E12,

"from\_as\_string": "10-2015",

"doc\_count": 0,

"key": "10-2015-\*"

}

]

}

}

}

缺失值

missing 参数对文档缺少值的情况进行了定义。默认情况下，它们会被忽略，但也可以认为它们有值。可以通过添加一组 字段名：值 的映射来指定每个字段的默认值。

POST /sales/\_search?size=0

{

"aggs": {

"range": {

"date\_range": {

"field": "date",

"missing": "1976/11/30",

"ranges": [

{

"key": "Older",

"to": "2016/02/01"

}, ①

{

"key": "Newer",

"from": "2016/02/01",

"to" : "now/d"

}

]

}

}

}

}

① 在 date 字段中没有值的文档会被添加到 “Older” 桶中，仿佛他们有一个日期值 “1976-11-30”。

日期格式 / 样式

注意：这些信息从 日期时间格式化 复制而来。

所有 ASCII 字母都保留为格式化模式字母，其定义如下：

符号 含义 展示方式 示例

G era text AD; Anno Domini; A

u year year 2004; 04

y year-of-era year 2004; 04

D day-of-year number 189

M/L month-of-year number/text 7; 07; Jul; July; J

d day-of-month number 10

Q/q quarter-of-year number/text 3; 03; Q3; 3rd quarter

Y week-based-year year 1996; 96

w week-of-week-based-year number 27

W week-of-month number 4

E day-of-week text Tue; Tuesday; T

e/c localized day-of-week number/text 2; 02; Tue; Tuesday; T

F week-of-month number 3

a am-pm-of-day text PM

h clock-hour-of-am-pm (1-12) number 12

K hour-of-am-pm (0-11) number 0

k clock-hour-of-am-pm (1-24) number 0

H hour-of-day (0-23) number 0

m minute-of-hour number 30

s second-of-minute number 55

S fraction-of-second fraction 978

A milli-of-day number 1234

n nano-of-second number 987654321

N nano-of-day number 1234000000

V time-zone ID zone-id America/Los\_Angeles; Z; -08:30

z time-zone name zone-name Pacific Standard Time; PST

O localized zone-offset offset-O GMT+8; GMT+08:00; UTC-08:00;

X zone-offset Z for zero offset-X Z; -08; -0830; -08:30; -083015; -08:30:15;

x zone-offset offset-x +0000; -08; -0830; -08:30; -083015; -08:30:15;

Z zone-offset offset-Z +0000; -0800; -08:00;

p pad next pad modifier 1

‘ escape for text delimiter ‘’

single quote literal ‘ [

optional section start ] optional section end #

reserved for future use { reserved for future use }

模式字母的数量决定了格式。

文本

文本样式由使用的模式字母确定。少于 4 个模式字母将使用缩写格式。恰好 4 个模式字母将使用完整格式。正好 5 个模式字母将使用窄边格式。模式字母 L，c 和 q 指定文本样式的独立格式。

数字

如果字母数量为 1，则使用最少的位数输出该值，并且不进行填充。其他情况，则使用数字位数作为输出字母的宽度，并且必要时用 0 进行填充。以下模式字符对字母数量有约束。字母 c 和 F 只能指定一个。字母 d，H，h，K，k，m 和 s 最多可以指定两个。字母 D 最多可以指定三个。

数字 / 文本

如果模式字母的数量是 3 个或者 3 个以上，请使用上面的文本规则。 否则使用上面的数字规则。

分数

用纳秒输出几分之一秒。纳秒值有 9 位数，因此模式字母的数量为从 1 到 9，如果模式字母的数量少于 9 个，则纳秒的值将被截断，仅输出最高有效位。

年份

字母的数量决定了使用填充的最小字段宽度。如果字母数量为 2，则使用简化的两位数格式。对于打印，则输出最右边的两位数字。对于解析，则使用 2000 的基值进行解析，从而产生 2000 到 2099（包括 2000 和 2099）范围内的一年。如果字母数少于 4 个（但不是 2 个），则按照 SignStyle.NORMAL 符号风格，仅负年数输出符号。其他情况，则按照 SignStyle.EXCEEDS\_PAD 符号风格，只输出超出宽度限制的符号。

时区 ID

输出时区 ID，例如 Europe/Paris。如果字母数为 2，则输出时区 ID。其他个数则抛出 IllegalArgumentException 异常。

时区名称

输出时区 ID 的展示名称。如果字母的数量为 1，2，或 3，则输出短名称。如果字母数量为 4 个，则输出完整名称。5 个及以上的字母则抛出 IllegalArgumentException 异常。

偏移 X 和 x

根据模式字母的数量格式化偏移。一个字母仅输出小时，例如 +01，分钟非零情况下也会被输出，例如 +0130。两个字母输出没有冒号的小时和分钟，例如 +0130。三个字母输出带有冒号的小时和分钟，例如 +01:30。四个字母输出没有冒号的小时、分钟和可选秒，例如 +013015。五个字母输出带有冒号的小时、分钟和可选秒，例如 +01:30:15。留个字母会抛出 IllegalArgumentException 异常。当要输出的偏移量为零时，模式字母 X（大写）将输出 Z，而模式字母 x（消息）将会输出 +00，+0000 或 +00:00。

偏移 O

根据模式字母的数量格式化局部偏移。一个字母输出局部偏移的 short 格式，这是局部偏移文本，例如 GMT，有无前导零的小时，2 位可选分秒（如果非零）和冒号，如 GMT+8。4 个模式字母输出 full 格式，这是一个局部偏移文本，例如 GMT，具有 2 位小时和分钟字段、可选秒字段（如果非零）和冒号，例如 GMT+08:00。其他字母数会抛出 IllegalArgumentException 异常。

偏移 Z

根据模式字母的数量格式化偏移量。一个，两个或者三个字母输出小时和分钟，没有冒号，例如，+0130。当偏移量为零时，输出 +0000。四个字母输出 full 格式的局部偏移量，相当于 偏移 O 的四个字母。如果偏移为零，输出相应的局部偏移文本。五个字母输出小时，分钟，可选秒（如果非零），冒号。如果偏移为零，则输出 Z。六位或者更多的字母抛出 IllegalArgumentException 异常。

可选部分

可选部分标记的工作原理与调用 DateTimeFormatterBuilder.optionalStart() 和 DateTimeFormatterBuilder.optionalEnd() 完全相同。

填充修饰符

修改模式，模式后面用空格填充。填充宽度由图案字母的数量决定。这与调用 DateTimeFormatterBuilder.padNext(int) 相同。

例如，ppH 输出小时，小时左边由宽度为 2 的空格填充。

任何无法识别的字母都是错误的。 除 [，]，{，}，＃ 和单引号之外的任何非字母字符都将直接输出。 尽管如此，仍然建议您对要直接输出的所有字符使用单引号，以确保将来的更改不会破坏您的应用程序。

日期范围聚合中的时区

通过指定 time\_zone 参数可以将日期从另一个时区转化成 UTC 时区。

时区也可以指定为 ISO 8601 UTC 偏移量（例如 +01:00 或 -08:00），也可以指定为 TZ 数据库中的一个时区 ID。

time\_zone 参数也适用日期数学表达式中的四舍五入。例如，要在 CET 时区四舍五入到一天的开始，您可以执行如下操作：

POST /sales/\_search?size=0

{

"aggs": {

"range": {

"date\_range": {

"field": "date",

"time\_zone": "CET",

"ranges": [

{ "to": "2016/02/01" },

{ "from": "2016/02/01", "to" : "now/d" },

{ "from": "now/d" }

]

}

}

}

}

日期将会被转化为 2016-02-01T00:00:00.000+01:00。

now/d 将在 CET 时区四舍五入到一天的开始。

Keyed 响应结果

将 keyed 标志设置为 true 会将唯一的 key 与每个桶关联起来，并且返回每个桶的范围，该范围是哈希格式而不是数组格式。

POST /sales/\_search?size=0

{

"aggs": {

"range": {

"date\_range": {

"field": "date",

"format": "MM-yyy",

"ranges": [

{ "to": "now-10M/M" },

{ "from": "now-10M/M" }

],

"keyed": true

}

}

}

}

响应结果：

{

...

"aggregations": {

"range": {

"buckets": {

"\*-10-2015": {

"to": 1.4436576E12,

"to\_as\_string": "10-2015",

"doc\_count": 7

},

"10-2015-\*": {

"from": 1.4436576E12,

"from\_as\_string": "10-2015",

"doc\_count": 0

}

}

}

}

}

也可以为每个区间自定义 key：

POST /sales/\_search?size=0

{

"aggs": {

"range": {

"date\_range": {

"field": "date",

"format": "MM-yyy",

"ranges": [

{ "from": "01-2015", "to": "03-2015", "key": "quarter\_01" },

{ "from": "03-2015", "to": "06-2015", "key": "quarter\_02" }

],

"keyed": true

}

}

}

}

响应结果：

{

...

"aggregations": {

"range": {

"buckets": {

"quarter\_01": {

"from": 1.4200704E12,

"from\_as\_string": "01-2015",

"to": 1.425168E12,

"to\_as\_string": "03-2015",

"doc\_count": 5

},

"quarter\_02": {

"from": 1.425168E12,

"from\_as\_string": "03-2015",

"to": 1.4331168E12,

"to\_as\_string": "06-2015",

"doc\_count": 2

}

}

}

}

}

### 27 多元化采集聚合

多元化采集聚合

同 sampler 聚合一样，这是一个过滤聚合，用于把子聚合的处理限制为一个得分最高的文档的样本。diversified\_sampler 聚合增加了限制共有一个值的匹配数量的能力，例如 “作者”。

注意

任何优秀的市场研究人员都会告诉你，处理数据样本时，相比只能反映单一观点的样本来说，健康的、能反映多种观点的样本更加重要。聚合也是如此。采用多样化的设置进行抽样可以提供一种方法来消除内容偏差（人口过多的地区，时间轴上的峰值或者活跃的论坛邮件发送者）。

用例示例：

将分析的重点放在高相关性匹配上，而不是低质量的长尾词的匹配上

确保不同来源的内容能够被公平的展示，以便消除分析中的偏差

significant\_terms 能通过单一样本就产生有用的结果，减少了聚合的成本

使用 field 或者 script 设置可以提供用来删除重复数据的值。max\_docs\_per\_value 设置了在任何共享同一值的分片上收集的最大文档数。max\_docs\_per\_value 默认设置为 1。

当 选择 field 或 script 后，在同一个文档中产生了多个值，聚合会抛出一个错误（由于效率问题，不支持使用多值字段进行重复数据删除）。

示例：

我们可能很想看看在 StackOverflow 论坛上哪些标签和 #elasticsearch 有很强的关联，但是会忽略多产用户的影响，这些用户倾向于把 #Kibana 误拼写为 #Cabana。

POST /stackoverflow/\_search?size=0

{

"query": {

"query\_string": {

"query": "tags:elasticsearch"

}

},

"aggs": {

"my\_unbiased\_sample": {

"diversified\_sampler": {

"shard\_size": 200,

"field" : "author"

},

"aggs": {

"keywords": {

"significant\_terms": {

"field": "tags",

"exclude": ["elasticsearch"]

}

}

}

}

}

}

响应结果：

{

...

"aggregations": {

"my\_unbiased\_sample": {

"doc\_count": 151,①

"keywords": {②

"doc\_count": 151,

"bg\_count": 650,

"buckets": [

{

"key": "kibana",

"doc\_count": 150,

"score": 2.213,

"bg\_count": 200

}

]

}

}

}

}

① 总共采样了 151 个文档。

② significant\_terms 聚合的结果不会因为任何一个作者的坏习惯而产生偏差，因为我们要求样本中的任何一位作者最多有一篇帖子。

脚本示例：

在这种情况下，我们可能希望使字段值的组合更加多样化。我们可以使用一个脚本来产生一个标签字段中多个值的哈希值，以确保我们没有由同样标签组成的样本。

POST /stackoverflow/\_search?size=0

{

"query": {

"query\_string": {

"query": "tags:kibana"

}

},

"aggs": {

"my\_unbiased\_sample": {

"diversified\_sampler": {

"shard\_size": 200,

"max\_docs\_per\_value" : 3,

"script" : {

"lang": "painless",

"source": "doc['tags'].hashCode()"

}

},

"aggs": {

"keywords": {

"significant\_terms": {

"field": "tags",

"exclude": ["kibana"]

}

}

}

}

}

}

响应结果：

{

...

"aggregations": {

"my\_unbiased\_sample": {

"doc\_count": 6,

"keywords": {

"doc\_count": 6,

"bg\_count": 650,

"buckets": [

{

"key": "logstash",

"doc\_count": 3,

"score": 2.213,

"bg\_count": 50

},

{

"key": "elasticsearch",

"doc\_count": 3,

"score": 1.34,

"bg\_count": 200

}

]

}

}

}

}

shard\_size

shard\_size 参数限制了在每个分片处理的样本中收集的顶级评分文档数量。默认值为 100。

max\_docs\_per\_value

max\_docs\_per\_value 是一个可选参数，该参数限制了每次数据去重时允许的文档数量。默认设置为 “1”。

execution\_hint

可选设置 execution\_hint 可以影响用于数据去重的值的管理。每个选项在执行数据去重时都会在内存中保存 shard\_size 值，但是可以按以下方式控制所保存的值的类型：

直接保存字段值（map）

保存根据 Lucene 索引确定的字段的序值（global\_ordinals）

保存字段值的哈希值 - 具有哈希冲突的可能（bytes\_hash）

如果信息能够从 Lucene 索引中获取，则使用默认设置 global\_ordinals，否则的话则使用 map。 在某些情况下，设置 bytes\_hash 被证明更快，但是由于存在哈希冲突的可能性，在数据去重逻辑中可能会产生误报。要注意的是，如果 execution\_hint 参数指定的方式不存在 Elasticsearch 将会无视，而且这些都没有保证向后兼容。

局限性

无法嵌套在 breadth\_first 聚合下

作为基于质量的过滤器，diversified\_sampler 聚合需要获取每个文档生成的相关性得分。因此，它不能嵌套在 collect\_mode 从默认的 depth\_first 模式切换成 breadth\_first 模式的 terms 聚合下，因为这丢弃了分数。在这种情况下，将引发错误。

数据去重逻辑受限

数据去重逻辑仅适用于一个分片级别，因此不支持于分片之间。

没有为 地理位置 / 日期 提供专门的语法

目前定义多样化值的语法由 field 或 script 决定，因此没有添加语法糖来表达地理位置或日期单位，例如 “7d”（七天）。这种支持可能会在以后的版本中添加，用户当前必须使用 script 来创建这些类型的值。

### 28 过滤器聚合

过滤器聚合

定义了一个包含当前文档集上下文中的所有文档的桶，该桶匹配一个特殊的过滤器。通常情况下，过滤器聚合被用于把当前聚合上下文缩小到一组特定的文档。

示例：

POST /sales/\_search?size=0

{

"aggs" : {

"t\_shirts" : {

"filter" : { "term": { "type": "t-shirt" } },

"aggs" : {

"avg\_price" : { "avg" : { "field" : "price" } }

}

}

}

}

在上面的例子中，我们计算了类型为 t-shirt 的所有产品的平均价格。

响应结果：

{

...

"aggregations" : {

"t\_shirts" : {

"doc\_count" : 3,

"avg\_price" : { "value" : 128.33333333333334 }

}

}

}

### 29 多过滤器聚合

多过滤器聚合

定义了多桶聚合，其中每个桶都关联一个过滤器。每个桶将收集与其关联的过滤器匹配的所有文档。

PUT /logs/\_bulk?refresh

{ "index" : { "\_id" : 1 } }

{ "body" : "warning: page could not be rendered" }

{ "index" : { "\_id" : 2 } }

{ "body" : "authentication error" }

{ "index" : { "\_id" : 3 } }

{ "body" : "warning: connection timed out" }

GET logs/\_search

{

"size": 0,

"aggs" : {

"messages" : {

"filters" : {

"filters" : {

"errors" : { "match" : { "body" : "error" }},

"warnings" : { "match" : { "body" : "warning" }}

}

}

}

}

}

在上述例子中，我们分析日志信息。聚合将构建包含日志信息的两个集合（桶）—— 一个用于包含 error 的所有消息，另一个用于包含 warning 的所有消息。

响应结果：

{

"took": 9,

"timed\_out": false,

"\_shards": ...,

"hits": ...,

"aggregations": {

"messages": {

"buckets": {

"errors": {

"doc\_count": 1

},

"warnings": {

"doc\_count": 2

}

}

}

}

}

匿名过滤器

过滤器字段也可以作为过滤器数组提供，像下面的请求一样：

GET logs/\_search

{

"size": 0,

"aggs" : {

"messages" : {

"filters" : {

"filters" : [

{ "match" : { "body" : "error" }},

{ "match" : { "body" : "warning" }}

]

}

}

}

}

过滤的桶按照请求中提供的顺序返回，这个例子的响应结果是：

{

"took": 4,

"timed\_out": false,

"\_shards": ...,

"hits": ...,

"aggregations": {

"messages": {

"buckets": [

{

"doc\_count": 1

},

{

"doc\_count": 2

}

]

}

}

}

other 桶

other\_bucket 参数可以给响应结果添加一个桶，该桶将容纳与任何给定过滤器都不匹配的所有文档。该参数的值如下：

false

不计算 other 桶

true

返回一个 other 桶。如果使用了命名过滤器，则返回这些桶中的一个桶（默认名称为 \_other\_），如果使用了匿名过滤器，则返回最后一个存储桶。

other\_bucket\_key 参数可用于将 other 桶的键设置为除默认值 \_other\_ 之外的值。设置该参数会将 other\_bucket 参数隐式地设置为 true。

下面的代码片段返回将一个 other 桶命名为 other\_messages 桶的响应结果。

PUT logs/\_doc/4?refresh

{

"body": "info: user Bob logged out"

}

GET logs/\_search

{

"size": 0,

"aggs" : {

"messages" : {

"filters" : {

"other\_bucket\_key": "other\_messages",

"filters" : {

"errors" : { "match" : { "body" : "error" }},

"warnings" : { "match" : { "body" : "warning" }}

}

}

}

}

}

响应结果如下所示：

{

"took": 3,

"timed\_out": false,

"\_shards": ...,

"hits": ...,

"aggregations": {

"messages": {

"buckets": {

"errors": {

"doc\_count": 1

},

"warnings": {

"doc\_count": 2

},

"other\_messages": {

"doc\_count": 1

}

}

}

}

}

### 30 地理距离聚合

地理距离聚合

对 geo\_point 字段操作的多桶聚合，在概念上和范围聚合非常相似。用户可以定义一个原点和一组距离范围的桶。聚合计算每个文档值到原点的距离，并根据范围确定其所属的桶（如果文档和原点之间的距离落在桶内，则文档属于该桶）。

PUT /museums

{

"mappings": {

"properties": {

"location": {

"type": "geo\_point"

}

}

}

}

POST /museums/\_bulk?refresh

{"index":{"\_id":1}}

{"location": "52.374081,4.912350", "name": "NEMO Science Museum"}

{"index":{"\_id":2}}

{"location": "52.369219,4.901618", "name": "Museum Het Rembrandthuis"}

{"index":{"\_id":3}}

{"location": "52.371667,4.914722", "name": "Nederlands Scheepvaartmuseum"}

{"index":{"\_id":4}}

{"location": "51.222900,4.405200", "name": "Letterenhuis"}

{"index":{"\_id":5}}

{"location": "48.861111,2.336389", "name": "Musée du Louvre"}

{"index":{"\_id":6}}

{"location": "48.860000,2.327000", "name": "Musée d'Orsay"}

POST /museums/\_search?size=0

{

"aggs" : {

"rings\_around\_amsterdam" : {

"geo\_distance" : {

"field" : "location",

"origin" : "52.3760, 4.894",

"ranges" : [

{ "to" : 100000 },

{ "from" : 100000, "to" : 300000 },

{ "from" : 300000 }

]

}

}

}

}

响应结果：

{

...

"aggregations": {

"rings\_around\_amsterdam" : {

"buckets": [

{

"key": "\*-100000.0",

"from": 0.0,

"to": 100000.0,

"doc\_count": 3

},

{

"key": "100000.0-300000.0",

"from": 100000.0,

"to": 300000.0,

"doc\_count": 1

},

{

"key": "300000.0-\*",

"from": 300000.0,

"doc\_count": 2

}

]

}

}

}

指定的字段必须是 geo\_pint 类型（只能在映射中显式设置）。多个 geo\_pint 字段可以保存为一个数组，该情况下，在聚合期间将考虑该数组中所有字段。原点可以接受 geo\_pint 类型支持的所有格式：

对象格式：{ "lat" : 52.3760, "lon" : 4.894 } - 这是最安全的格式，因为它明确的表示了 lat 和 lon 的值

字符串格式："52.3760, 4.894" - 第一个数字是 lat，第二个数字是 lon

数组格式：[4.894, 52.3760] - 数组格式基于 GeoJson 标准，第一个数字为 lon，第二个数字为 lat。

在默认情况下，距离单位是 m（米），其它单位也可以，如：mi（英里），in（英寸），yd（码），km（公里），cm（厘米），mm（毫米）。

POST /museums/\_search?size=0

{

"aggs" : {

"rings" : {

"geo\_distance" : {

"field" : "location",

"origin" : "52.3760, 4.894",

"unit" : "km", ①

"ranges" : [

{ "to" : 100 },

{ "from" : 100, "to" : 300 },

{ "from" : 300 }

]

}

}

}

}

① 距离将以公里计算

有两种距离计算模式：arc（默认）和 plane。arc 计算模式是最准确的，plane 计算模式是最快的，却是最不准确的。当你搜索的上下文比较小，地理区域跨越较小（~5km），可以考虑使用 plane。当搜索跨越很大区域时（如跨大陆搜索），plane 返回结果的误差幅度会更大。距离计算类型可以使用 distance\_type 参数设置：

POST /museums/\_search?size=0

{

"aggs" : {

"rings" : {

"geo\_distance" : {

"field" : "location",

"origin" : "52.3760, 4.894",

"unit" : "km",

"distance\_type" : "plane",

"ranges" : [

{ "to" : 100 },

{ "from" : 100, "to" : 300 },

{ "from" : 300 }

]

}

}

}

}

Keyed Response

将 keyed 标志设置为 true 会给每个桶关联一个唯一的字符串键，并将范围作为哈希而不是数组返回:

POST /museums/\_search?size=0

{

"aggs" : {

"rings\_around\_amsterdam" : {

"geo\_distance" : {

"field" : "location",

"origin" : "52.3760, 4.894",

"ranges" : [

{ "to" : 100000 },

{ "from" : 100000, "to" : 300000 },

{ "from" : 300000 }

],

"keyed": true

}

}

}

}

响应结果：

{

...

"aggregations": {

"rings\_around\_amsterdam" : {

"buckets": {

"\*-100000.0": {

"from": 0.0,

"to": 100000.0,

"doc\_count": 3

},

"100000.0-300000.0": {

"from": 100000.0,

"to": 300000.0,

"doc\_count": 1

},

"300000.0-\*": {

"from": 300000.0,

"doc\_count": 2

}

}

}

}

}

也可以为每个范围自定义 key：

POST /museums/\_search?size=0

{

"aggs" : {

"rings\_around\_amsterdam" : {

"geo\_distance" : {

"field" : "location",

"origin" : "52.3760, 4.894",

"ranges" : [

{ "to" : 100000, "key": "first\_ring" },

{ "from" : 100000, "to" : 300000, "key": "second\_ring" },

{ "from" : 300000, "key": "third\_ring" }

],

"keyed": true

}

}

}

}

响应结果：

{

...

"aggregations": {

"rings\_around\_amsterdam" : {

"buckets": {

"first\_ring": {

"from": 0.0,

"to": 100000.0,

"doc\_count": 3

},

"second\_ring": {

"from": 100000.0,

"to": 300000.0,

"doc\_count": 1

},

"third\_ring": {

"from": 300000.0,

"doc\_count": 2

}

}

}

}

}

### 31 Geohash网格聚合

Geohash 网格聚合

一种对 geo\_point 字段操作的多桶聚合，并将点分组到代表网格中单元的桶中。生成的网格可能是稀疏的，它只包含匹配到数据的单元。每个单元使用用户可自定义精度的 geohash 进行标记。

高精度的 geohashes 具有较长的字符串长度，表示仅覆盖很小面积的单元。

低精度的 geohashes 具有较短的字符串长度，表示覆盖面积很大的单元。

在该聚合中 geohashes 精度可选范围为 1 到 12。

警告

长度为 12 的最高精度的 geohash 产生的单元覆盖的土地不到一平方米，因此，就 RAM 和结果大小而言，高精度请求可能非常昂贵。请参考下面的示例，了解一下在请求高级信息之前，如何将聚合过滤到较小的地理区域。

指定字段必须为 geo\_point 类型（只能在映射中显式设置），多个 geo\_point 字段可以保存为一个数组，该情况下，在聚合期间将考虑该数组中所有点。

简单的低精度请求

PUT /museums

{

"mappings": {

"properties": {

"location": {

"type": "geo\_point"

}

}

}

}

POST /museums/\_bulk?refresh

{"index":{"\_id":1}}

{"location": "52.374081,4.912350", "name": "NEMO Science Museum"}

{"index":{"\_id":2}}

{"location": "52.369219,4.901618", "name": "Museum Het Rembrandthuis"}

{"index":{"\_id":3}}

{"location": "52.371667,4.914722", "name": "Nederlands Scheepvaartmuseum"}

{"index":{"\_id":4}}

{"location": "51.222900,4.405200", "name": "Letterenhuis"}

{"index":{"\_id":5}}

{"location": "48.861111,2.336389", "name": "Musée du Louvre"}

{"index":{"\_id":6}}

{"location": "48.860000,2.327000", "name": "Musée d'Orsay"}

POST /museums/\_search?size=0

{

"aggregations" : {

"large-grid" : {

"geohash\_grid" : {

"field" : "location",

"precision" : 3

}

}

}

}

响应结果：

{

...

"aggregations": {

"large-grid": {

"buckets": [

{

"key": "u17",

"doc\_count": 3

},

{

"key": "u09",

"doc\_count": 2

},

{

"key": "u15",

"doc\_count": 1

}

]

}

}

}

高精度要求

当请求详细的存储库（通常用于显示 “放大” 的地图）时，需要使用 [geo\_bounding\_box]（www.elastic.co/guide/en/elasticsea...) 应用于缩小主题范围，否则可能会创建并返回数百万个存储库。

POST /museums/\_search?size=0

{

"aggregations" : {

"zoomed-in" : {

"filter" : {

"geo\_bounding\_box" : {

"location" : {

"top\_left" : "52.4, 4.9",

"bottom\_right" : "52.3, 5.0"

}

}

},

"aggregations":{

"zoom1":{

"geohash\_grid" : {

"field": "location",

"precision": 8

}

}

}

}

}

}

也可以使用”geohash\_grid” 聚合返回的 geohash 进行放大。要放大上一示例中返回的第一个 geohash u17，应将其同时指定为”top\_left” 和”bottom\_right”

POST /museums/\_search?size=0

{

"aggregations" : {

"zoomed-in" : {

"filter" : {

"geo\_bounding\_box" : {

"location" : {

"top\_left" : "u17",

"bottom\_right" : "u17"

}

}

},

"aggregations":{

"zoom1":{

"geohash\_grid" : {

"field": "location",

"precision": 8

}

}

}

}

}

}

{

...

"aggregations" : {

"zoomed-in" : {

"doc\_count" : 3,

"zoom1" : {

"buckets" : [

{

"key" : "u173zy3j",

"doc\_count" : 1

},

{

"key" : "u173zvfz",

"doc\_count" : 1

},

{

"key" : "u173zt90",

"doc\_count" : 1

}

]

}

}

}

}

为了在不支持 geohash 的系统上 “放大”，应使用可用的 geohash 库之一将存储桶键转换为边界框。 例如，对于 javascript，可以使用 node-geohash 库

var geohash = require('ngeohash');

// bbox 包含 [ 52.03125, 4.21875, 53.4375, 5.625 ]

// [ minlat, minlon, maxlat, maxlon]

var bbox = geohash.decode\_bbox('u17');

赤道上的单元尺寸

下表显示了由不同长度的 geohash 字符串覆盖的单元的度量尺寸。单元格大小随纬度变化，因此该表适用于赤道最坏情况。

地理哈希长度 区域宽度 x 高度

1 5,009.4km x 4,992.6km

2 1,252.3km x 624.1km

3 156.5km x 156km

4 39.1km x 19.5km

5 4.9km x 4.9km

6 1.2km x 609.4m

7 152.9m x 152.4m

8 38.2m x 19m

9 4.8m x 4.8m

10 1.2m x 59.5cm

11 14.9cm x 14.9cm

12 3.7cm x 1.9cm

选项

字段 必须填写。用地质点索引的字段的名称。

精度 可选。用于定义结果中的单元格 / 存储桶的 geohash 的字符串长度。默认为 5。精度可以根据上述整数精度级别定义。[1,12] 以外的值将被拒绝。或者，精度水平可以从 “1km”、“10m” 等距离测量中近似。计算精度级别时，单元格将不会超过所需精度的指定大小 (对角线)。如果这会导致精度水平高于支持的 12 个水平 (例如，距离 < 5.6cm)，则拒绝该值。

尺寸 可选。要返回的最大 geohash 存储桶数 (默认为 10000)。当结果被修剪时，bucket 将根据其包含的文档量进行优先级排序。

碎片大小 可选。为了对最终结果中返回的顶部单元格进行更精确的计数，聚合默认为从每个碎片返回 max(10，(大小x碎片数))bucket。如果这种启发式方法是不可取的，则可以使用此参数对每个碎片考虑的数量进行重载。

### 32 GeoTile Grid Aggregation

PUT /museums

{

"mappings": {

"properties": {

"location": {

"type": "geo\_point"

}

}

}

}

POST /museums/\_bulk?refresh

{"index":{"\_id":1}}

{"location": "52.374081,4.912350", "name": "NEMO Science Museum"}

{"index":{"\_id":2}}

{"location": "52.369219,4.901618", "name": "Museum Het Rembrandthuis"}

{"index":{"\_id":3}}

{"location": "52.371667,4.914722", "name": "Nederlands Scheepvaartmuseum"}

{"index":{"\_id":4}}

{"location": "51.222900,4.405200", "name": "Letterenhuis"}

{"index":{"\_id":5}}

{"location": "48.861111,2.336389", "name": "Musée du Louvre"}

{"index":{"\_id":6}}

{"location": "48.860000,2.327000", "name": "Musée d'Orsay"}

POST /museums/\_search?size=0

{

"aggregations" : {

"large-grid" : {

"geotile\_grid" : {

"field" : "location",

"precision" : 8

}

}

}

}

{

...

"aggregations": {

"large-grid": {

"buckets": [

{

"key" : "8/131/84",

"doc\_count" : 3

},

{

"key" : "8/129/88",

"doc\_count" : 2

},

{

"key" : "8/131/85",

"doc\_count" : 1

}

]

}

}

}

POST /museums/\_search?size=0

{

"aggregations" : {

"zoomed-in" : {

"filter" : {

"geo\_bounding\_box" : {

"location" : {

"top\_left" : "52.4, 4.9",

"bottom\_right" : "52.3, 5.0"

}

}

},

"aggregations":{

"zoom1":{

"geotile\_grid" : {

"field": "location",

"precision": 22

}

}

}

}

}

}

{

...

"aggregations" : {

"zoomed-in" : {

"doc\_count" : 3,

"zoom1" : {

"buckets" : [

{

"key" : "22/2154412/1378379",

"doc\_count" : 1

},

{

"key" : "22/2154385/1378332",

"doc\_count" : 1

},

{

"key" : "22/2154259/1378425",

"doc\_count" : 1

}

]

}

}

}

}

### 33 Global Aggresgation

POST /sales/\_search?size=0

{

"query" : {

"match" : { "type" : "t-shirt" }

},

"aggs" : {

"all\_products" : {

"global" : {},

"aggs" : {

"avg\_price" : { "avg" : { "field" : "price" } }

}

},

"t\_shirts": { "avg" : { "field" : "price" } }

}

}

{

...

"aggregations" : {

"all\_products" : {

"doc\_count" : 7,①

"avg\_price" : {

"value" : 140.71428571428572 ②

}

},

"t\_shirts": {

"value" : 128.33333333333334 ③

}

}

}

### 34 Histogram Aggresgation

bucket\_key = Math.floor((value - offset) / interval) \* interval + offset

POST /sales/\_search?size=0

{

"aggs" : {

"prices" : {

"histogram" : {

"field" : "price",

"interval" : 50

}

}

}

}

{

...

"aggregations": {

"prices" : {

"buckets": [

{

"key": 0.0,

"doc\_count": 1

},

{

"key": 50.0,

"doc\_count": 1

},

{

"key": 100.0,

"doc\_count": 0

},

{

"key": 150.0,

"doc\_count": 2

},

{

"key": 200.0,

"doc\_count": 3

}

]

}

}

}

POST /sales/\_search?size=0

{

"aggs" : {

"prices" : {

"histogram" : {

"field" : "price",

"interval" : 50,

"min\_doc\_count" : 1

}

}

}

}

{

...

"aggregations": {

"prices" : {

"buckets": [

{

"key": 0.0,

"doc\_count": 1

},

{

"key": 50.0,

"doc\_count": 1

},

{

"key": 150.0,

"doc\_count": 2

},

{

"key": 200.0,

"doc\_count": 3

}

]

}

}

}

POST /sales/\_search?size=0

{

"query" : {

"constant\_score" : { "filter": { "range" : { "price" : { "to" : "500" } } } }

},

"aggs" : {

"prices" : {

"histogram" : {

"field" : "price",

"interval" : 50,

"extended\_bounds" : {

"min" : 0,

"max" : 500

}

}

}

}

}

POST /sales/\_search?size=0

{

"aggs" : {

"prices" : {

"histogram" : {

"field" : "price",

"interval" : 50,

"keyed" : true

}

}

}

}

{

...

"aggregations": {

"prices": {

"buckets": {

"0.0": {

"key": 0.0,

"doc\_count": 1

},

"50.0": {

"key": 50.0,

"doc\_count": 1

},

"100.0": {

"key": 100.0,

"doc\_count": 0

},

"150.0": {

"key": 150.0,

"doc\_count": 2

},

"200.0": {

"key": 200.0,

"doc\_count": 3

}

}

}

}

}

POST /sales/\_search?size=0

{

"aggs" : {

"quantity" : {

"histogram" : {

"field" : "quantity",

"interval": 10,

"missing": 0 ①

}

}

}

}

### 35 IP Range Aggresgtion

GET /ip\_addresses/\_search

{

"size": 10,

"aggs" : {

"ip\_ranges" : {

"ip\_range" : {

"field" : "ip",

"ranges" : [

{ "to" : "10.0.0.5" },

{ "from" : "10.0.0.5" }

]

}

}

}

}

{

...

"aggregations": {

"ip\_ranges": {

"buckets" : [

{

"key": "\*-10.0.0.5",

"to": "10.0.0.5",

"doc\_count": 10

},

{

"key": "10.0.0.5-\*",

"from": "10.0.0.5",

"doc\_count": 260

}

]

}

}

}

GET /ip\_addresses/\_search

{

"size": 0,

"aggs" : {

"ip\_ranges" : {

"ip\_range" : {

"field" : "ip",

"ranges" : [

{ "mask" : "10.0.0.0/25" },

{ "mask" : "10.0.0.127/25" }

]

}

}

}

}

{

...

"aggregations": {

"ip\_ranges": {

"buckets": [

{

"key": "10.0.0.0/25",

"from": "10.0.0.0",

"to": "10.0.0.128",

"doc\_count": 128

},

{

"key": "10.0.0.127/25",

"from": "10.0.0.0",

"to": "10.0.0.128",

"doc\_count": 128

}

]

}

}

}

GET /ip\_addresses/\_search

{

"size": 0,

"aggs": {

"ip\_ranges": {

"ip\_range": {

"field": "ip",

"ranges": [

{ "to" : "10.0.0.5" },

{ "from" : "10.0.0.5" }

],

"keyed": true

}

}

}

}

{

...

"aggregations": {

"ip\_ranges": {

"buckets": {

"\*-10.0.0.5": {

"to": "10.0.0.5",

"doc\_count": 10

},

"10.0.0.5-\*": {

"from": "10.0.0.5",

"doc\_count": 260

}

}

}

}

}

GET /ip\_addresses/\_search

{

"size": 0,

"aggs": {

"ip\_ranges": {

"ip\_range": {

"field": "ip",

"ranges": [

{ "key": "infinity", "to" : "10.0.0.5" },

{ "key": "and-beyond", "from" : "10.0.0.5" }

],

"keyed": true

}

}

}

}

{

...

"aggregations": {

"ip\_ranges": {

"buckets": {

"infinity": {

"to": "10.0.0.5",

"doc\_count": 10

},

"and-beyond": {

"from": "10.0.0.5",

"doc\_count": 260

}

}

}

}

}

### 36 Missing Aggresgtion

POST /sales/\_search?size=0

{

"aggs" : {

"products\_without\_a\_price" : {

"missing" : { "field" : "price" }

}

}

}

{

...

"aggregations" : {

"products\_without\_a\_price" : {

"doc\_count" : 00

}

}

}