Scripting小结

一 Painless scripting language：

Painless 是一种专为 Elasticsearch 设计的高性能、安全的脚本语言。我们可以使用 Painless 在 Elasticsearch 支持脚本的任何地方安全地编写内联和存储脚本。

Painless 提供了许多围绕以下核心原则的功能：

**安全**：确保集群的安全是最重要的。为此，Painless 使用细粒度的许可名单，其粒度细至类的成员。任何不属于许可名单的内容都会导致编译错误。有关每个脚本上下文的可用类、方法和字段的完整列表，请参阅 Painless API 参考。

**性能**：Painless 直接编译为 JVM 字节码，以利用 JVM 提供的所有可能的优化。此外，Painless 通常会避免在运行时需要额外慢速检查的功能。

**简单性**：Painless 实现的语法对于任何具有一些基本编码经验的人来说都是自然熟悉的。 Painless 使用 Java 语法的一个子集，并进行了一些额外的改进，以增强可读性并删除样板。

二 如何写脚本

只要 Elasticsearch API 支持脚本，语法都遵循相同的模式； 指定脚本的语言，提供脚本逻辑（或源，并添加传递到脚本中的参数：

"script": {

"lang": "...",

"source" | "id": "...",

"params": { ... }

}

**lang**

指定脚本所用的语言。Defaults to painless.

**source, id**

脚本本身，您将其指定为内联脚本的源或存储脚本的 id。 使用存储的脚本 API 来创建和管理存储的脚本。

**params**

指定作为变量传递到脚本中的任何命名参数。 使用参数而不是硬编码值来减少编译时间。

二 基本操作

1 如何构建一个对该字段进行操作的脚本？

首先，用单个字段索引一个文档：

PUT my-index-000001/\_doc/1

{

"my\_field": 5

}

然后，构建一个对该字段进行操作的脚本，并将该脚本作为查询的一部分运行。 以下查询使用搜索 API 的 script\_fields 参数来检索脚本估值。

GET my-index-000001/\_search

{

"script\_fields": {

"my\_doubled\_field": {

"script": {

"source": "doc['my\_field'].value \*params['multiplier']",

"params": {

"multiplier": 2

}

}

}

}

}

该脚本是一个标准的 JSON 对象，它定义了 Elasticsearch 中大多数 API 下的脚本。 这个对象需要 source 来定义脚本本身。 该脚本未指定语言，因此默认为 Painless。

2 在脚本中使用参数？

在上一个例子中，我们注意“source”字段，是这样的：

"source": "doc['my\_field'].value \*params['multiplier']",

我们可以直接把params['multiplier']变成固定的值：

"source": "return doc['my\_field'].value \* 2"

也可以这样写：

"source": "doc['my\_field'].value \* params['multiplier']",

"params": {

"multiplier": 2

}

3 编译脚本的格式更改？

对于大多数上下文，默认情况下每 5 分钟最多可以编译 75 个脚本。 对于摄取上下文，默认脚本编译率是无限的。

您可以通过设置***script.context.$CONTEXT.max\_compilations\_rate*** 动态更改这些设置。

例如，以下设置将字段上下文的脚本编译限制为每 10 分钟 100 个脚本：

script.context.field.max\_compilations\_rate=100/10m

\*\*\* 如果您在短时间内编译了太多独特的脚本，Elasticsearch 将拒绝新的动态脚本，并出现 circuit\_breaking\_exception 错误。

4 存储和检索脚本？

我们可以使用存储的脚本 API 从集群状态存储和检索脚本。 存储的脚本减少了编译时间并使搜索速度更快。

要创建脚本，使用创建存储脚本 API。 例如，以下请求创建一个名为 calculate-score 的存储脚本：

POST \_scripts/calculate-score

{

"script": {

"lang": "painless",

"source": "Math.log(\_score \* 2) + params['my\_modifier']"

}

}

检索脚本：

GET \_scripts/calculate-score

要在查询中使用存储的脚本，在脚本声明中包含脚本 ID：

GET my-index-000001/\_search

{

"query": {

"script\_score": {

"query": {

"match": {

"message": "some message"

}

},

"script": {

**"id": "calculate-score",**

"params": {

"my\_modifier": 2

}

}

}

}

}

删除存储的脚本：

DELETE \_scripts/calculate-score

5 使用脚本更新文档？

使用更新 API 来更新具有指定脚本的文档。 脚本可以更新、删除或跳过修改文档。 更新 API 还支持传递部分文档，将其合并到现有文档中。

首先索引一个简单的文档：

PUT my-index-000001/\_doc/1

{

"counter" : 1,

"tags" : ["red"]

}

为了使计数器增加产生变化，使用以下脚本提交更新请求：

POST my-index-000001/\_update/1

{

"script" : {

"source": "ctx.\_source.counter += params.count",

"lang": "painless",

"params" : {

"count" : 4

}

}

}

同样，使用更新脚本将标签添加到标签列表中。 因为这只是一个列表，即使它存在，也会添加标签：

POST my-index-000001/\_update/1

{

"script": {

"source": "ctx.\_source.tags.add(params['tag'])",

"lang": "painless",

"params": {

"tag": "blue"

}

}

}

我们还可以在文档中添加和删除字段。 例如，此脚本添加字段 new\_field：

POST my-index-000001/\_update/1

{

"script" : "ctx.\_source.new\_field = 'value\_of\_new\_field'"

}

相反，此脚本删除字段 new\_field：

POST my-index-000001/\_update/1

{

"script" : "ctx.\_source.remove('new\_field')"

}

除了更新文档，我们还可以更改从脚本内执行的操作。 例如，如果标签字段包含绿色，则此请求将删除文档。 否则它什么都不做（noop）：

POST my-index-000001/\_update/1

{

"script": {

"source": "if (ctx.\_source.tags.contains(params['tag'])) { ctx.op = 'delete' } else { ctx.op = 'none' }",

"lang": "painless",

"params": {

"tag": "green"

}

}

}

6 脚本、缓存和搜索速度

Elasticsearch 执行了许多优化以尽可能快地使用脚本。 一个重要的优化是脚本缓存。 编译后的脚本被放置在缓存中，以便引用该脚本的请求不会招致编译损失。

缓存大小很重要。 我们的脚本缓存应该足够大以容纳用户需要同时访问的所有脚本。

如果在 node stats 中看到大量脚本缓存逐出和越来越多的编译，则缓存可能太小。

默认情况下，所有脚本都被缓存，因此只有在发生更新时才需要重新编译它们。 默认情况下，脚本没有基于时间的过期时间。 我们可以使用 script.context.$CONTEXT.cache\_expire 设置更改此行为。 使用 script.context.$CONTEXT.cache\_max\_size 设置来配置缓存的大小。

\*\*\*脚本的大小限制为 65,535 字节。 设置 script.max\_size\_in\_bytes 的值以增加该软限制。 如果脚本非常大，请考虑使用本机脚本引擎。

提高搜索速度：

脚本非常有用，但不能使用 Elasticsearch 的索引结构或相关优化。这种关系有时会导致搜索速度变慢。

如果经常使用脚本来转换索引数据，则可以通过在摄取期间转换数据来加快搜索速度。然而，这通常意味着较慢的索引速度。让我们看一个实际示例来说明如何提高搜索速度。

运行搜索时，通常按两个值的总和对结果进行排序。例如，考虑一个名为 my\_test\_scores 的索引，其中包含测试分数数据。该索引包括两个 long 类型的字段：

* math\_score
* verbal\_score

使用将这些值相加的脚本来运行查询。这种方法没有任何问题，**但查询会变慢，因为脚本评估是作为请求的一部分发生的**。以下请求返回 grad\_year 等于 2099 的文档，并根据脚本的估值对结果进行排序。

GET /my\_test\_scores/\_search

{

"query": {

"term": {

"grad\_year": "2099"

}

},

"sort": [

{

"\_script": {

"type": "number",

"script": {

"source": "doc['math\_score'].value + doc['verbal\_score'].value" –>当数据量大时，速度会很慢

},

"order": "desc"

}

}

]

}

**改进方法？**

首先，我们将向名为 total\_score 的索引添加一个新字段，该字段将包含 math\_score 和 verbal\_score 字段值的总和。

PUT /my\_test\_scores/\_mapping

{

"properties": {

"total\_score": {

"type": "long"

}

}

}

接下来，使用包含脚本处理器的摄取管道来计算 math\_score 和 verbal\_score 的总和，并将其索引到 total\_score 字段中。

PUT \_ingest/pipeline/my\_test\_scores\_pipeline

{

"description": "Calculates the total test score",

"processors": [

{

"script": {

"source": "ctx.total\_score = (ctx.math\_score + ctx.verbal\_score)"

}

}

]

}

要更新现有数据，使用此管道将 my\_test\_scores 中的任何文档重新索引到名为 my\_test\_scores\_2 的新索引。

POST /\_reindex

{

"source": {

"index": "my\_test\_scores"

},

"dest": {

"index": "my\_test\_scores\_2",

"pipeline": "my\_test\_scores\_pipeline"

}

}

继续使用管道将任何新文档索引到 my\_test\_scores\_2。

POST /my\_test\_scores\_2/\_doc/?pipeline=my\_test\_scores\_pipeline

{

"student": "kimchy",

"grad\_year": "2099",

"math\_score": 1200,

"verbal\_score": 800

}

这些更改减慢了索引过程，但允许更快的搜索。 您可以使用 total\_score 字段对在 my\_test\_scores\_2 上进行的搜索进行排序，而不是使用脚本。 响应接近实时！ 尽管此过程会减慢摄取时间，但会大大增加搜索时的查询量。

GET /my\_test\_scores\_2/\_search

{

"query": {

"term": {

"grad\_year": "2099"

}

},

"sort": [

{

"total\_score": {

"order": "desc"

}

}

]

}

7 脚本中的模式匹配

Dissect 根据定义的模式匹配单个文本字段。 解析模式由您要丢弃的字符串部分定义。 特别注意字符串的每个部分有助于构建成功的剖析模式。

如果不需要正则表达式的强大功能，则使用解剖模式而不是 grok。 Dissect 使用比 grok 更简单的语法，并且总体上通常更快。 dissect 的语法是透明的：告诉 dissect 你想要什么，它会将这些结果返回给你。

解剖模式由变量和分隔符组成。 由百分号和大括号 %{} 定义的任何内容都被视为变量，例如 %{clientip}。 您可以将变量分配给字段中数据的任何部分，然后只返回您想要的部分。 分隔符是变量之间的任何值，可以是空格、破折号或其他分隔符。

例如，假设存在一个消息字段的日志数据，如下所示：

"message" : "247.37.0.0 - - [30/Apr/2020:14:31:22 -0500] \"GET /images/hm\_nbg.jpg HTTP/1.0\" 304 0"

您可以为数据的每个部分分配变量以构建成功的dissect模式。 请记住，准确地告诉解剖您想要匹配的内容。

数据的第一部分看起来像一个 IP 地址，因此您可以分配一个像 %{clientip} 这样的变量。 接下来的两个字符是破折号，两边各有一个空格。 您可以为每个破折号分配一个变量，也可以为单个变量分配一个变量来表示破折号和空格。 接下来是一组包含时间戳的括号。 括号是一个分隔符，因此您可以将它们包含在解剖模式中。 到目前为止，数据和匹配的解剖模式如下所示：

247.37.0.0 - - [30/Apr/2020:14:31:22 -0500]

%{clientip} %{ident} %{auth} [%{@timestamp}]

使用相同的逻辑，您可以为剩余的数据块创建变量。 双引号是分隔符，因此将它们包含在您的剖析模式中。 该模式将 GET 替换为 %{verb} 变量，但将 HTTP 作为该模式的一部分。

\"GET /images/hm\_nbg.jpg HTTP/1.0\" 304 0

"%{verb} %{request} HTTP/%{httpversion}" %{response} %{size}

结合这两种模式会产生如下所示的dissect模式：

%{clientip} %{ident} %{auth} [%{@timestamp}] \"%{verb} %{request} HTTP/%{httpversion}\" %{status} %{size}

**在运行时字段中使用dissect模式和脚本：**

如果您有功能剖析模式，则可以将其添加到运行时字段以操作数据。 由于运行时字段不需要您对字段进行索引，因此您可以非常灵活地修改脚本及其运行方式。 如果您已经使用 Painless 执行 API 测试了您的解剖模式，您可以在您的运行时字段中使用该确切的 Painless 脚本。

首先，将消息字段添加为通配符（wildcard）类型，但还要添加@timestamp 作为日期，以防您想对该字段进行操作以用于其他用例：

PUT /my-index/

{

"mappings": {

"properties": {

"@timestamp": {

"format": "strict\_date\_optional\_time||epoch\_second",

"type": "date"

},

"message": {

"type": "wildcard"

}

}

}

}

如果您想使用解析模式提取 HTTP 响应代码，您可以创建一个类似 http.response 的运行时字段：

PUT my-index/\_mappings

{

"runtime": {

"http.response": {

"type": "long",

"script": """

String response=dissect('%{clientip} %{ident} %{auth} [%{@timestamp}] "%{verb} %{request} HTTP/%{httpversion}" %{response} %{size}').extract(doc["message"].value)?.response;

if (response != null) emit(Integer.parseInt(response));

"""

}

}

}

映射您要检索的字段后，将日志数据中的一些记录索引到 Elasticsearch 中。 以下请求使用批量 API 将原始日志数据索引到 my-index 中：

POST /my-index/\_bulk?refresh=true

{"index":{}}

{"timestamp":"2020-04-30T14:30:17-05:00","message":"40.135.0.0 - - [30/Apr/2020:14:30:17 -0500] \"GET /images/hm\_bg.jpg HTTP/1.0\" 200 24736"}

{"index":{}}

{"timestamp":"2020-04-30T14:30:53-05:00","message":"232.0.0.0 - - [30/Apr/2020:14:30:53 -0500] \"GET /images/hm\_bg.jpg HTTP/1.0\" 200 24736"}

{"index":{}}

{"timestamp":"2020-04-30T14:31:12-05:00","message":"26.1.0.0 - - [30/Apr/2020:14:31:12 -0500] \"GET /images/hm\_bg.jpg HTTP/1.0\" 200 24736"}

{"index":{}}

{"timestamp":"2020-04-30T14:31:19-05:00","message":"247.37.0.0 - - [30/Apr/2020:14:31:19 -0500] \"GET /french/splash\_inet.html HTTP/1.0\" 200 3781"}

{"index":{}}

{"timestamp":"2020-04-30T14:31:22-05:00","message":"247.37.0.0 - - [30/Apr/2020:14:31:22 -0500] \"GET /images/hm\_nbg.jpg HTTP/1.0\" 304 0"}

{"index":{}}

{"timestamp":"2020-04-30T14:31:27-05:00","message":"252.0.0.0 - - [30/Apr/2020:14:31:27 -0500] \"GET /images/hm\_bg.jpg HTTP/1.0\" 200 24736"}

{"index":{}}

{"timestamp":"2020-04-30T14:31:28-05:00","message":"not a valid apache log"}

您可以定义一个简单的查询来搜索特定的 HTTP 响应并返回所有相关字段。 使用搜索 API 的 fields 参数来检索 http.response 运行时字段：

GET my-index/\_search

{

"query": {

"match": {

"http.response": "304"

}

},

"fields" : ["http.response"]

}

8 Grok

Grok 是一种支持可重用别名表达式的正则表达式方言。 Grok 非常适用于 syslog 日志、Apache 和其他网络服务器日志、mysql 日志以及通常为人类而非计算机使用编写的任何日志格式。

Grok 位于 Oniguruma 正则表达式库之上，因此任何正则表达式在 grok 中都是有效的。 Grok 使用这种正则表达式语言来允许命名现有模式并将它们组合成与您的字段匹配的更复杂的模式。

Grok模式编辑

Elastic Stack 附带了许多预定义的 grok 模式，可以简化 grok 的使用。 重用 grok 模式的语法采用以下形式之一：

|  |  |  |
| --- | --- | --- |
| %{SYNTAX} | %{SYNTAX:ID} | %{SYNTAX:ID:TYPE} |

**SYNTAX**

将匹配您的文本的模式的名称。 例如，NUMBER 和 IP 都是默认模式集中提供的模式。 NUMBER 模式匹配 3.44 之类的数据，IP 模式匹配 55.3.244.1 之类的数据。

**ID**

您为要匹配的文本提供的标识符。 例如，3.44 可能是事件的持续时间，因此您可以将其称为持续时间。 字符串 55.3.244.1 可能标识发出请求的客户端。

**TYPE**

要转换命名字段的数据类型。 int、long、double、float 和 boolean 是支持的类型。

例如，假设您有如下所示的消息数据：

3.44 55.3.244.1

第一个值是一个数字，后面是一个 IP 地址。 您可以使用以下 grok 表达式匹配此文本：

%{NUMBER:duration} %{IP:client}

在painless脚本里面使用grok patterns：

您可以将预定义的 grok 模式合并到 Painless 脚本中以提取数据。 要测试您的脚本，请使用 Painless execute API 的字段上下文或创建包含脚本的运行时字段。 运行时字段提供更大的灵活性并接受多个文档，但如果您在测试脚本的集群上没有写访问权限，则无痛执行 API 是一个很好的选择。

例如，如果您正在处理 Apache 日志数据，则可以使用 %{COMMONAPACHELOG} 语法，它了解 Apache 日志的结构。 示例文档可能如下所示：

"timestamp":"2020-04-30T14:30:17-05:00","message":"40.135.0.0 - -

[30/Apr/2020:14:30:17 -0500] \"GET /images/hm\_bg.jpg HTTP/1.0\" 200 24736"

要从消息字段中提取 IP 地址，您可以编写一个包含 %{COMMONAPACHELOG} 语法的painless脚本。 您可以使用 Painless 执行 API 的 ip 字段上下文来测试此脚本，但让我们改用运行时字段。

根据示例文档，索引 @timestamp 和 message 字段。 为了保持灵活性，请使用通配符作为消息的字段类型：

PUT /my-index/

{

"mappings": {

"properties": {

"@timestamp": {

"format": "strict\_date\_optional\_time||epoch\_second",

"type": "date"

},

"message": {

"type": "wildcard"

}

}

}

}

接下来，使用批量 API 将一些日志数据索引到 my-index 中。

POST /my-index/\_bulk?refresh

{"index":{}}

{"timestamp":"2020-04-30T14:30:17-05:00","message":"40.135.0.0 - - [30/Apr/2020:14:30:17 -0500] \"GET /images/hm\_bg.jpg HTTP/1.0\" 200 24736"}

{"index":{}}

{"timestamp":"2020-04-30T14:30:53-05:00","message":"232.0.0.0 - - [30/Apr/2020:14:30:53 -0500] \"GET /images/hm\_bg.jpg HTTP/1.0\" 200 24736"}

{"index":{}}

{"timestamp":"2020-04-30T14:31:12-05:00","message":"26.1.0.0 - - [30/Apr/2020:14:31:12 -0500] \"GET /images/hm\_bg.jpg HTTP/1.0\" 200 24736"}

{"index":{}}

{"timestamp":"2020-04-30T14:31:19-05:00","message":"247.37.0.0 - - [30/Apr/2020:14:31:19 -0500] \"GET /french/splash\_inet.html HTTP/1.0\" 200 3781"}

{"index":{}}

{"timestamp":"2020-04-30T14:31:22-05:00","message":"247.37.0.0 - - [30/Apr/2020:14:31:22 -0500] \"GET /images/hm\_nbg.jpg HTTP/1.0\" 304 0"}

{"index":{}}

{"timestamp":"2020-04-30T14:31:27-05:00","message":"252.0.0.0 - - [30/Apr/2020:14:31:27 -0500] \"GET /images/hm\_bg.jpg HTTP/1.0\" 200 24736"}

{"index":{}}

{"timestamp":"2020-04-30T14:31:28-05:00","message":"not a valid apache log"}

9 字段提取

字段提取的目标很简单； 您的数据中有包含大量信息的字段，但您只想提取部分和部分。

两种选择：

Grok 是一种正则表达式方言，支持可以重用的别名表达式。 因为 Grok 位于正则表达式 (regex) 之上，所以任何正则表达式在 grok 中也是有效的。

Dissect 从文本中提取结构化字段，使用分隔符定义匹配模式。 与 grok 不同，dissect 不使用正则表达式。

10 访问文档字段和特殊变量编辑

根据使用脚本的位置，它可以访问某些特殊变量和文档字段。

更新脚本编辑

在更新、按查询更新或重新索引 API 中使用的脚本将可以访问 ctx 变量，该变量公开：

ctx.\_source：访问文档 \_source 字段。

ctx.op:应用于文档的操作：索引或删除。

ctx.\_index 等:访问文档元数据字段，其中一些可能是只读的。

10 搜索和聚合脚本编辑

除了每次搜索命中执行一次的脚本字段外，搜索和聚合中使用的脚本将针对可能与查询或聚合匹配的每个文档执行一次。 根据您拥有的文档数量，这可能意味着数百万或数十亿次执行：这些脚本需要很快！

可以使用 doc-values、\_source 字段或存储的字段从脚本中访问字段值，下面对每个字段进行说明。

**在脚本中访问文档的分数:**

在 function\_score 查询、基于脚本的排序或聚合中使用的脚本可以访问 \_score 变量，该变量表示文档的当前相关性分数。

这是在 function\_score 查询中使用脚本来改变每个文档的相关性 \_score 的示例：

PUT my-index-000001/\_doc/1?refresh

{

"text": "quick brown fox",

"popularity": 1

}

PUT my-index-000001/\_doc/2?refresh

{

"text": "quick fox",

"popularity": 5

}

GET my-index-000001/\_search

{

"query": {

"function\_score": {

"query": {

"match": {

"text": "quick brown fox"

}

},

"script\_score": {

"script": {

"lang": "expression",

"source": "\_score \* doc['popularity']"

}

}

}

}

}

**Doc values:**

到目前为止，从脚本中访问字段值最快最有效的方法是使用 doc['field\_name'] 语法，它从 doc 值中检索字段值。 Doc 值是一个列式字段值存储，默认情况下在所有字段上启用，分析的文本字段除外。

PUT my-index-000001/\_doc/1?refresh

{

"cost\_price": 100

}

GET my-index-000001/\_search

{

"script\_fields": {

"sales\_price": {

"script": {

"lang": "expression",

"source": "doc['cost\_price'] \* markup",

"params": {

"markup": 0.2

}

}

}

}

}

如果字段是多值的，文档值只能返回“简单”字段值，如数字、日期、地理点、术语等，或这些值的数组。 它不能返回 JSON 对象。

**The document\_source:**

可以使用 \_source.field\_name 语法访问文档 \_source。 \_source 作为 map-of-maps 加载，因此可以访问对象字段中的属性，例如 \_source.name.first。

For instance:

PUT my-index-000001

{

"mappings": {

"properties": {

"first\_name": {

"type": "text"

},

"last\_name": {

"type": "text"

}

}

}

}

PUT my-index-000001/\_doc/1?refresh

{

"first\_name": "Barry",

"last\_name": "White"

}

GET my-index-000001/\_search

{

"script\_fields": {

"full\_name": {

"script": {

"lang": "painless",

"source": "params.\_source.first\_name + ' ' + params.\_source.last\_name"

}

}

}

}

**Stored field:**

存储的字段 — 在映射中明确标记为“存储”的字段 — 可以使用 \_fields['field\_name'].value 或 \_fields['field\_name'] 语法访问：

PUT my-index-000001

{

"mappings": {

"properties": {

"full\_name": {

"type": "text",

"store": true

},

"title": {

"type": "text",

"store": true

}

}

}

}

PUT my-index-000001/\_doc/1?refresh

{

"full\_name": "Alice Ball",

"title": "Professor"

}

GET my-index-000001/\_search

{

"script\_fields": {

"name\_with\_title": {

"script": {

"lang": "painless",

"source": "params.\_fields['title'].value + ' ' + params.\_fields['full\_name'].value"

}

}

}

}

11 脚本和安全

Painless 和 Elasticsearch 实施了安全层来构建深度防御策略，以安全地运行脚本。

Painless 使用细粒度的许可名单。任何不属于许可名单的内容都会导致编译错误。此功能是脚本纵深防御策略中的第一层安全性。

第二层安全是 Java 安全管理器。作为其启动序列的一部分，Elasticsearch 使 Java 安全管理器能够限制部分代码可以采取的操作。 Painless 使用 Java 安全管理器作为额外的防御层，以防止脚本执行诸如写入文件和侦听套接字之类的操作。

Elasticsearch 在 Linux 中使用 seccomp，在 macOS 中使用 Seatbelt，在 Windows 上使用 ActiveProcessLimit 作为额外的安全层，以防止 Elasticsearch 分叉或运行其他进程。

您可以修改以下脚本设置以限制允许运行的脚本类型，并控制可以运行脚本的可用上下文。要在纵深防御策略中实施附加层，请遵循 Elasticsearch 安全原则。