

# Lesson 1

Painless was designed and developed specifically for elasticsearch

- It is fast and secure
- It has a groovy like syntax
- It support all java data types
- It exposes many java classes(eg. MATH) and methods

As of Elasticsearch 6.0, painless is the only language supported Other languages, including Groovy, Javascript, and Python are no longer available

## Basics to Painless scripting

- In this lesson you will learn to write basic scripting using Painless:
- Scripting Syntax
- Single-Line Expression
- Script parameters
- Statements and blocks
- Primitive data types
- Variables
- Conditionals
- Methods

## Getting Started

- We can Start writing and running scripts using the `_scripts` API with the `_execute` end-point

```
POST _scripts/painless/_execute
{
  "script": {
    "lang": "painless",
    "source": "(2 + 1) * 4"
  }
}
```

- The source code in this case is simply a single-line numerical expression
- Note the use of JSON to encapsulate script - everything in Elasticsearch is in JSON

### Basics of Painless Scripting

- Writing complex code as an inline expression is not very practical
- We can write blocks of code using `"""` as the block delimiter. Using a code block, the previous script would now be:

```
POST
_scripts/painless/_execute
{
  "script": {
    "lang": "painless",
    "source": """
return (2 + 1) * 4; """
  }
}
```

- Variables in `painless` can be declared by specifying their type, name, and initial value

```
POST _scripts/painless/_execute
{
  "script": {
    "lang": "painless",
    "source": """ int x = 2; int y = 1;
return (x + y) * 4; """
  }
}
```

## Data types

- Painless has the same data types as Java, including
- Primitive types: byte, char, short, int, long, float, double and boolean
- Object wrappers for primitive types: Integer, Long, Float, Double and Boolean String
- Other data types including Date and other
- Data structures
- Arrays, Lists, Maps and others

## Expressions

### Expressions

- You can write numerical expressions using +, -, \*, / and %
  - Be aware that `4/3` is not the same expression as `4.0/3.0` (why?)
- You can also use bitwise operators & (and), | (or), ^ (xor), << (shift left), >> (shift right) — which will be very useful in the advanced course
- For boolean expressions use ==, >, <, >=, <=, &&, ||, and !
- For string expressions you can use + (concatenation)
  - Note that using + to concatenate a string with a non-string value will coerce this value into a string as part of the concatenation
- You can use parentheses to create sub-expressions and control the order of evaluation
- This is all familiar territory to you as a programmer!

## Maps and ArrayLists

In Painless, Maps and ArrayLists are particularly easy to build and use — so let's start working with them

- `m = ["a": 1, "b": 2]` creates a Map m containing two keys, "a" and "b" with values 1 and 2, respectively
- `m.get("a")` returns 1, the value of key "a"
- `m.put("c": 3)` adds a new key "c" to m with a value of 3
- `m.remove("a")` removes the entry in the Map with a key of "a"
- `a = [1, 2]` creates an ArrayList a containing two values, 1 and 2, in that order  
a [0] returns 1, while a [1] returns 2
- `a.add (3)` adds 3 to the end of a

- `a.remove(2)` removes the element at position 2 from `a`, shifting the remaining elements to the left

## Script Parameters

To make scripts more general and efficient, we can use script parameters

- Value that change from one execution to another should be passed as parameters
- The compiled version of the source will be cached by ES and can be reused with new data

## Conditional Statements

Painless supports if and if-else conditional statements

### The conditional operator

Instead of using an if-statement to set the value of a variable, we can use the conditional operator ?

```
POST _scripts/painless/_execute
{
  "script": {
    "lang": "painless",
    "source": """
      int score = params.score;
      String testResult;
      testResult = (score >= 60)? "Pass" : "Fail";
      return testResult;
    """,
    "params": {
      "score": 85
    }
  }
}
```

## Loops

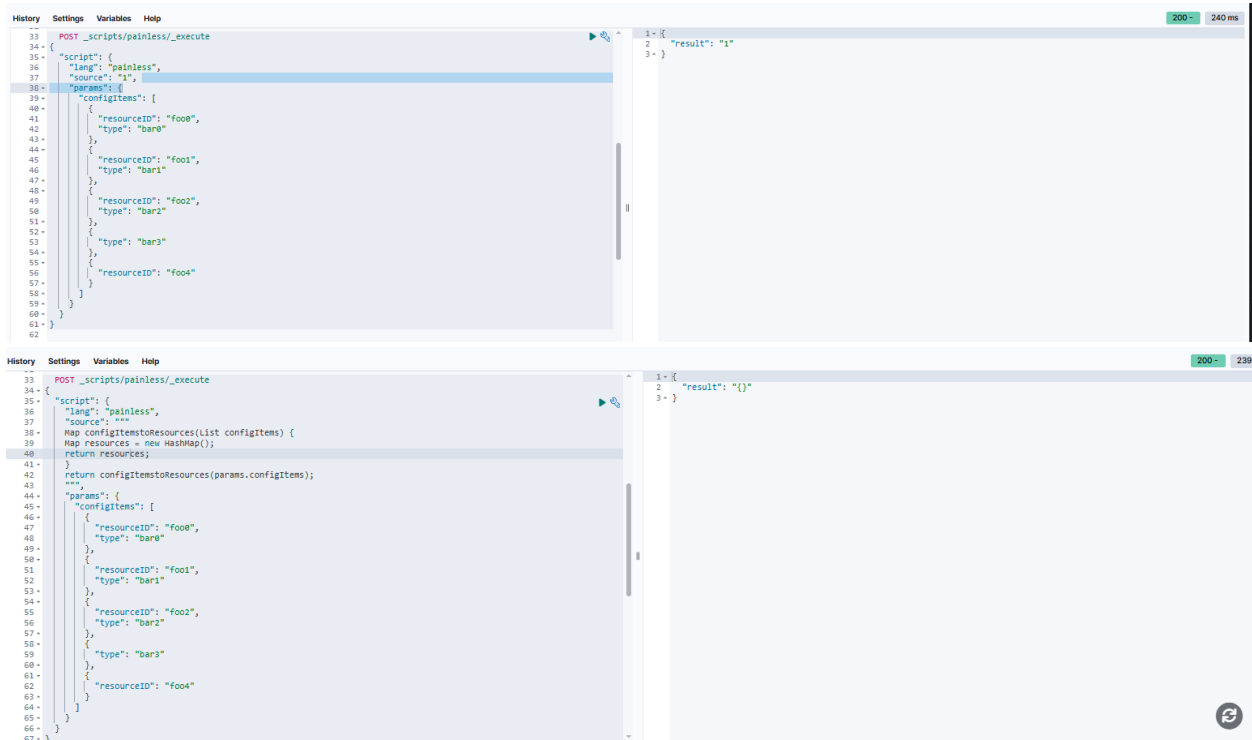
- Painless supports for, while, do-while and for-each loop
- In the do-while, the condition is evaluated at the end of each iteration - which means it will be run at least once

## Methods

- Methods and functions(methods that return value) are supported inside painless scripts
- This script contains a function that find the area of a circle of radius "r"
- Method allow us to write reusable painless code
- Unfortunately, methods cannot be saved outside a script and must be copy-pasted from script to script

Scripts can be stored in the cluster state and later on invoked by id and with new parameter values

## Solved Problem



The image shows two screenshots of a script editor interface. The top screenshot shows a script being edited, and the bottom screenshot shows the same script after execution, with the results displayed on the right.

**Top Screenshot:**

```
33 POST _scripts/painless/_execute
34 {
35   "script": {
36     "lang": "painless",
37     "source": "1",
38     "params": {
39       "configItems": [
40         {
41           "resourceID": "foo0",
42           "type": "bar0"
43         },
44         {
45           "resourceID": "foo1",
46           "type": "bar1"
47         },
48         {
49           "resourceID": "foo2",
50           "type": "bar2"
51         },
52         {
53           "type": "bar3"
54         },
55         {
56           "resourceID": "foo4"
57         }
58       ]
59     }
60   }
61 }
62
```

**Bottom Screenshot:**

```
33 POST _scripts/painless/_execute
34 {
35   "script": {
36     "lang": "painless",
37     "source": "",
38     "Map configItemstoResources(List configItems) {
39       Map resources = new HashMap();
40       return resources;
41     }
42     return configItemstoResources(params.configItems);
43     """,
44     "params": {
45       "configItems": [
46         {
47           "resourceID": "foo0",
48           "type": "bar0"
49         },
50         {
51           "resourceID": "foo1",
52           "type": "bar1"
53         },
54         {
55           "resourceID": "foo2",
56           "type": "bar2"
57         },
58         {
59           "type": "bar3"
60         },
61         {
62           "resourceID": "foo4"
63         }
64       ]
65     }
66   }
67 }
```

**Execution Results (Top Screenshot):**

```
1- {
2  "result": "1"
3 }
```

**Execution Results (Bottom Screenshot):**

```
1- {
2  "result": "{}"
3 }
```

POST \_scripts/painless/\_execute

```
{
  "script": {
    "lang": "painless",
    "source": ""
    Map configItemstoResources(List configItems) {
      Map resources = new HashMap();
      for (configItem in configItems){
        resources.put(
          configItem.getDefault("resourceID", "Unknow_resouce_ID"),
          configItem.getDefault("type", "Unkwon_type")
        )
      }
    }
  }
}
```

```

    }
    return resources;
}
return configItemstoResources(params.configItems);
"""
,
"params": {
  "configItems": [
    {
      "resourceID": "foo0",
      "type": "bar0"
    },
    {
      "resourceID": "foo1",
      "type": "bar1"
    },
    {
      "resourceID": "foo2",
      "type": "bar2"
    },
    {
      "type": "bar3"
    },
    {
      "resourceID": "foo4"
    }
  ]
}
}
}

```

The screenshot shows a web browser window with a REST client interface. The top bar includes tabs for History, Settings, Variables, and Help. The main area displays a POST request to the endpoint `POST _scripts/painless/_execute`. The request body is a JSON object with a `script` field containing a Groovy script and a `params` field containing a list of configuration items. The script defines a `resources` map and returns the result of `configItemstoResources`. The response, shown on the right, is a JSON object with a `result` field containing a list of strings representing the mapped configuration items.

```

28 * }
29 *
30 *
31 *
32 *
33 * POST _scripts/painless/_execute
34 * {
35 *   "script": {
36 *     "lang": "painless",
37 *     "source": ""
38 *   }
39 *   Map configItemstoResources(List configItems) {
40 *     Map resources = new HashMap();
41 *     for (configItem in configItems){
42 *       resources.put(
43 *         configItem.getDefault("resourceID", "unknown_resource_ID"),
44 *         configItem.getDefault("type", "unknown_type")
45 *       )
46 *     }
47 *     return resources;
48 *   }
49 *   return configItemstoResources(params.configItems);
50 * }
51 *
52 * "params": {
53 *   "configItems": [
54 *     {
55 *       "resourceID": "foo0",
56 *       "type": "bar0"
57 *     },
58 *     {
59 *       "resourceID": "foo1",
60 *       "type": "bar1"
61 *     },
62 *     {
63 *       "resourceID": "foo2",
64 *       "type": "bar2"
65 *     },
66 *     {
67 *       "type": "bar3"
68 *     },
69 *     {
70 *       "resourceID": "foo4"
71 *     }
72 *   ]
73 * }
74 *
75 *
76 *

```

```

1 = {
2   "result": ["foo0=bar0, foo1=bar1, foo2=bar2, unknown_resource_ID=bar3, foo4=unknown_type"]
3 }

```

```

{
  "error": {
    "root_cause": [
      {
        "type": "script_exception",
        "reason": "compile error",
        "script_stack": [
          ""... dObjects){
entries.put(
nestedObject ..."",
          "          ^---- HERE"
        ],
        "script": ""
      }
    ]
  }
}

Map listtoMap(List nestedObjects) {
  Map entreis = new HashMap();
  for (nestedObject in nestedObjects){
    entries.put(
      nestedObject.getDefault("resourceID", "Unknow_resouce_ID"),
      nestedObject.getDefault("type", "Unkwon_type")
    )
  }
  return entreis;
}

return listtoMap(params.configItems);
"",
  "lang": "painless",
  "position": {
    "offset": 132,
    "start": 107,
    "end": 157
  }
}
],
"type": "script_exception",
"reason": "compile error",
"script_stack": [
  ""... dObjects){
entries.put(
nestedObject ..."",
  "          ^---- HERE"
],
"script": ""

```

```

Map listtoMap(List nestedObjects) {
Map entreis = new HashMap();
for (nestedObject in nestedObjects){
    entreis.put(
        nestedObject.getDefault("resourceID", "Unknow_resouce_ID"),
        nestedObject.getDefault("type", "Unkwon_type")
    )
}
return entreis;
}
return listtoMap(params.configItems);
}
{
    "lang": "painless",
    "position": {
        "offset": 132,
        "start": 107,
        "end": 157
    },
    "caused_by": {
        "type": "illegal_argument_exception",
        "reason": "cannot resolve symbol [entries]"
    }
},
"status": 400
}

```

The screenshot shows an IDE with a script editor on the left and a console on the right. The script editor contains a Java script that defines a function `listtoMap` and calls it with a list of objects. The console shows the result of the script execution, which is a JSON object containing the result of the `listtoMap` function call.

```

History Settings Variables Help
30
31
32
33 POST _scripts/painless/_execute
34 {
35   "script": {
36     "lang": "painless",
37     "source": """
38       Map listtoMap(List nestedObjects) {
39         Map entreis = new HashMap();
40         for (nestedObject in nestedObjects){
41           entreis.put(
42             nestedObject.getDefault("resourceID", "Unknow_resouce_ID"),
43             nestedObject.getDefault("type", "Unkwon_type")
44           )
45         }
46         return entreis;
47       }
48       return listtoMap(params.configItems);
49     }
50     "params": {
51       "configItems": [
52         {
53           "resourceID": "foo0",
54           "type": "bar0"
55         },
56         {
57           "resourceID": "foo1",
58           "type": "bar1"
59         },
60         {
61           "resourceID": "foo2",
62           "type": "bar2"
63         },
64         {
65           "type": "bar3"
66         },
67         {
68           "resourceID": "foo4"
69         }
70       ]
71     }
72   }
73 }
74
75
76
77 }

```

```

1 - {
2   "result": "{foo0=bar0, foo1=bar1, foo2=bar2, Unknow_resouce_ID=bar3, foo4=Unkwon_type}"
3 }

```

**FINAL CODE OF PROBLEM**



POST \_scripts/painless/\_execute

```
{
  "script": {
    "lang": "painless",
    "source": ""
    Map listtoMap(List nestedObjects, String keyField, String valueField) {
      Map entries = new HashMap();
      for (nestedObject in nestedObjects){
        entries.put(
          nestedObject.getDefault(keyField, "Unknow_key"),
          nestedObject.getDefault(valueField, "Unkwon_value")
        )
      }
      return entries;
    }
    return listtoMap(params.configItems, params.keyField, params.valueField);
    "",
    "params": {
      "keyField" : "resourceID",
      "valueField" : "type",
      "configItems": [
        {
          "resourceID": "foo0",
          "type": "bar0"
        },
        {
          "resourceID": "foo1",
          "type": "bar1"
        },
        {
          "resourceID": "foo2",
          "type": "bar2"
        },
        {
          "type": "bar3"
        },
        {
          "resourceID": "foo4"
        }
      ]
    }
  }
}
```

```
}
```

```
POST _scripts/painless/_execute
```

```
{
  "script": {
    "lang": "painless",
    "source": ""
    Map listtoMap(List nestedObjects, String keyField, String valueField) {
      Map entries = new HashMap();
      for (nestedObject in nestedObjects){
        entries.put(
          nestedObject.getDefault(keyField, "Unknow_key"),
          nestedObject.getDefault(valueField, "Unkwon_value")
        )
      }
      return entries;
    }
    return listtoMap(params.configItems, params.keyField, params.valueField);
  },
  "params": {
    "keyField" : "resourceID",
    "valueField" : "type",
    "configItems": [
      {
        "resourceID": "foo0",
        "type": "bar0"
      },
      {
        "resourceID": "foo1",
        "type": "bar1"
      },
      {
        "resourceID": "foo2",
        "type": "bar2"
      },
      {
        "type": "bar3"
      },
      {
        "resourceID": "foo4"
      }
    ]
  }
}
```

OUTPUT

200 - 685 ms

```
1 {  
2   "result": "{foo0=bar0, foo1=bar1, foo2=bar2, Unknow_key=bar3, foo4=Unkwon_value}"  
3 }
```