JOLT Introduction

2013

What it is :

JSON to JSON transform library

Declarative

Transforms are written in JSON
Js**O**n **L**anguage for **T**ransform

Gets you 90% of what you need

Interface so you can get that last 10%

Testable / Good tooling

What it is Not :

Text Based

It operates on

Map<String,Object>

List<Object>

In JavaScript JSON is "data"

Streaming

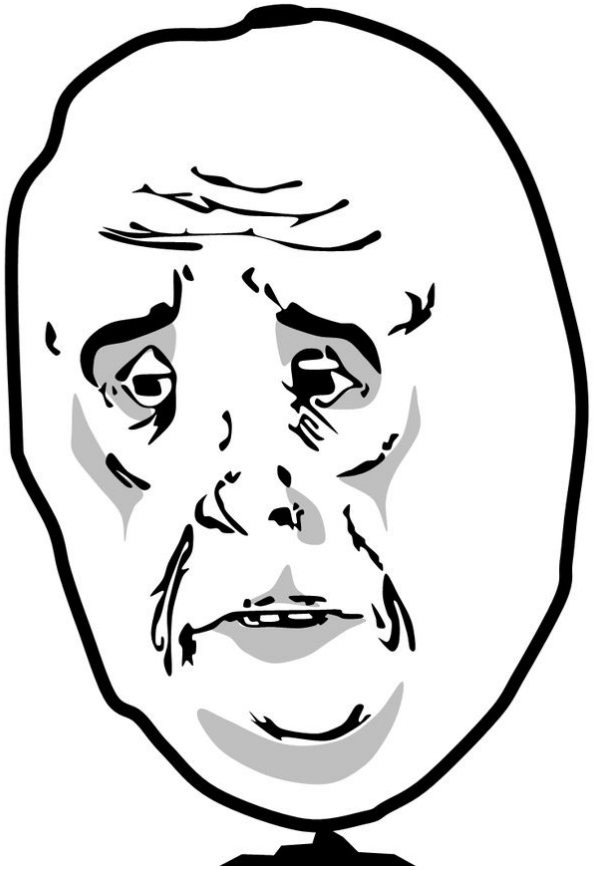
Operates on a fully in memory tree

Motivation 1:

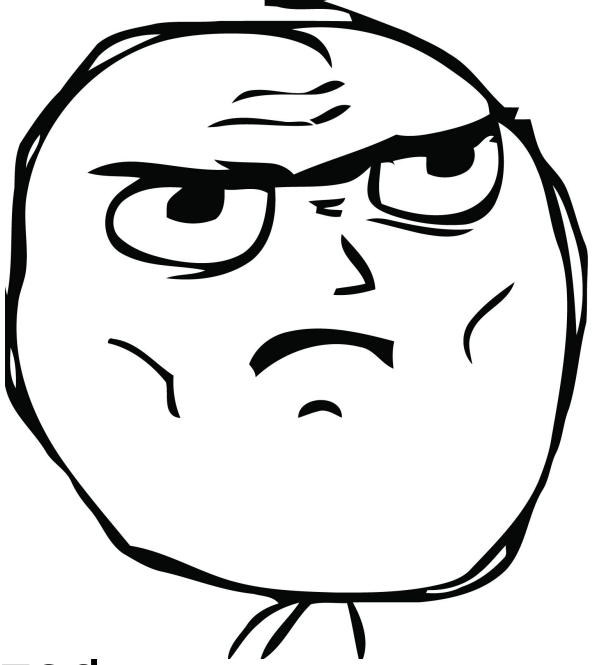
Cassandra, ElasticSearch, Mongo

Motivation 2 : No good Existing Tools

1) Json -> Xml -> Xslt/Stx -> Xml -> Json
2) Write a Template
3) Write custom Java



Opportunity and We Can Do Better :



Our initial transform needs were "simple"

Option 3.5)
Write custom Java, in a way that minimized the "Java change" when the "transform" changed

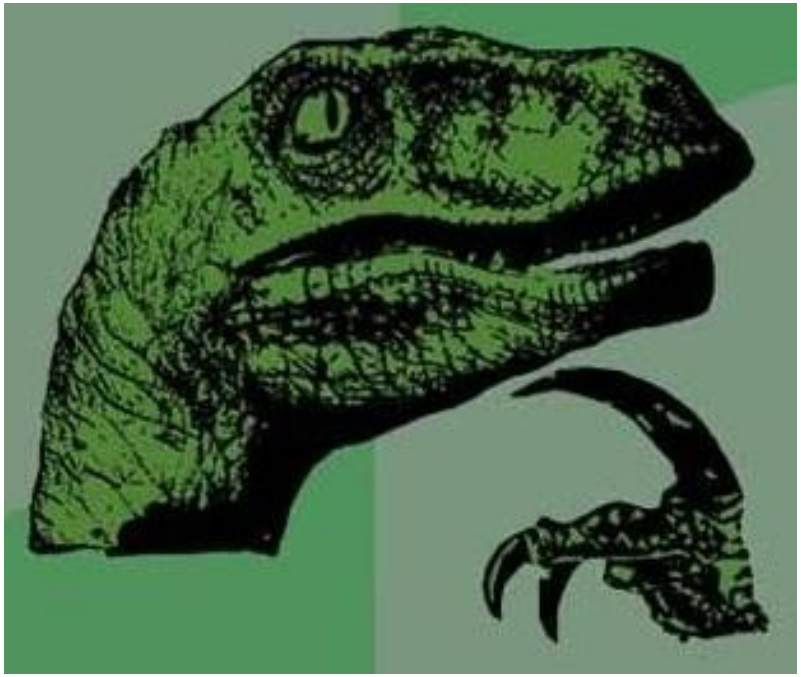
Transform difficulty ramped nicely (lucky)

L1) DevApi to Custom ElasticSearch format
L2) Cassandra to DevApi
L3) ElasticSearch Facet results to DevApi
L4) Transform/Extract info from Config docs

Philosophy :

What is a Transform?

Dunno... I am ISTP, so I wrote a Transform as a FreeMarker template and marinated in it.



It is not one thing. It is several different concerns.

1) Find a home for all the input data (maybe /dev/null)

2) Make sure the Output looks ok. (Output format)

3) Make it palatable to the machine. (**,] }</xmlTag>**)

Template approach sucks, because all those concerns are mashed up in a single "step".

Transform Separable Concerns 1:

For each Input value, where does it go in the Output?

INPUT

```
{
  "rating": {
    "quality": {
      "value": 3,
    },
    "primary": {
      "value": 4
    }
  }
}
```

OUTPUT

```
{
  "Rating": 4,
  "SecondaryRatings": {
    "quality": {
      "Value": 3
    }
  }
}
```

```
3 -> "Rating.SecondaryRatings.quality.Value"
5 -> "Rating.SecondaryRatings.quality.Range"
```

Transform Separable Concerns 2:

Maintain Output “format”.

Input : Half DevApi

```
{
  "Rating": 4,
  "SecondaryRatings": {
    "quality": {
      "Value": 3,
} } }
```

New Output

```
{
  "Rating": 4,
  "RatingRange": 5,
  "SecondaryRatings": {
    "quality": {
      "Value": 3,
      "Range": 5
} } }
```

Would be nice...

```
{
  "RatingRange": 5,
  "SecondaryRatings" : {
    "*" : {
      "Range" : 5,
```

Transform Separable Concerns 3: Machine Format

Defaultr Output

```
{
  "Rating": 4,
  "RatingRange": 5,
  "SecondaryRatings": {
    "quality": {
      "Value": 3,
      "Range": 5
    }
  }
}
```

Operate on

Map<String,Object> and List<Object>

and let

handle it.

Recap :

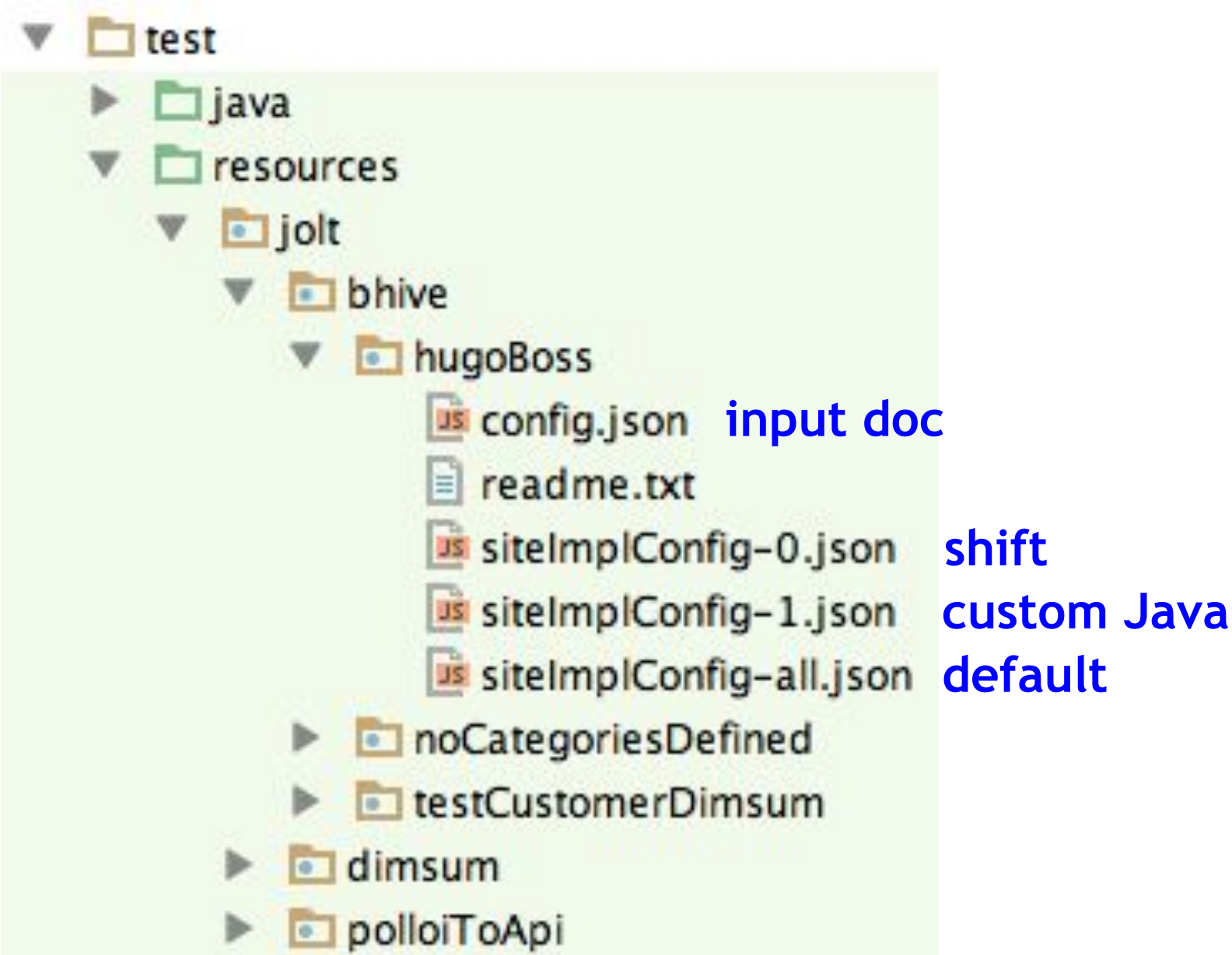
- Operate on Maps-of-Maps
- Small JSON based DSL for each transform "concern"
- Chain them together

```
[
  {
    "operation" : "shift",
    "spec": { ... }
  },
  {
    "operation" : "java",
    "classname" : "com.bazaar.."
    "spec": { ... } // optional
  },
  {
    "operation" : "default"
    "spec" : { ... }
  }
]
```

Valid Operations:

```
"shift",
"default",
"remove",
"sort",
"java"
```

A Note on Testing :



Shiftr Basics :

```
INPUT {
  "rating": {
    "quality": {
      "value": 3,
    },
    "primary": {
      "value": 4
} } }
```

```
OUTPUT {
  "Rating": 4,
  "SecondaryRatings": {
    "quality": {
      "Value": 3
    }
  }
}
```

```
SPEC (Starts out as a copy of the INPUT {
  "rating": {
    "quality": {
      "value": "SecondaryRatings.quality.Value",
    },
    "primary": {
      "value": "Rating"
} } }
```

Shiftr Basics 2 : Send input to two places

```
INPUT {
  "rating": {
    "quality": {
      "value": 3,
    },
    "primary": {
      "value": 4
} } }
```

```
OUTPUT {
  "Rating": 4
  "PrimaryRating" : 4
  "SecondaryRatings": {
    "quality": {
      "Value": 3
    }
  }
}
```

```
SPEC {
  "rating": {
    "quality": {
      "value": "SecondaryRatings.quality.Value",
    },
    "primary": {
      "value": [ "Rating", "PrimaryRating" ]
} } }
```

Shiftr Basics 3 : Two inputs to the same place

```
INPUT {
  "rating": {
    "quality": {
      "value": 3,
    },
    "primary": {
      "value": 4
} } }
```

```
OUTPUT {
  "allRatings":[ 3, 4 ]
  }
}
```

Order of array not Guaranteed.

```
SPEC {
  "rating": {
    "quality": {
      "value": "allRatings",
    },
    "primary": {
      "value": "allRatings"
} } }
```

Shiftr WildCards 101 : * and &

```
INPUT {
  "rating": {
    "quality": {
      "value": 3,
    },
    "colour": {
      "value": 4
} } }
```

```
OUTPUT {
  "SecondaryRatings": {
    "quality": {
      "Value": 3
    },
    "colour" : {
      "Value" : 4
} } }
```

~~"SecondaryRatings.quality.Value"~~
~~"SecondaryRatings.colour.Value"~~

```
SPEC {
  "rating": {
    "*": {
      "value": "SecondaryRatings.&1.Value"
    }
} }
```

Shiftr WildCards 101 : & Explained

```
INPUT {
  "rating": {
    "quality": {
      "value": 3,
    },
    "colour": {
      "value": 4
} } }
```

MVP

What goes in the green box?

```
&0 = "value"
&  = "value"  (sugar)
&1 = "quality" or "colour"
&2 = "rating"
&3 = Fail
```

```
SPEC {
  "rating": {
    "*": {
      "value": "SecondaryRatings.[ ].Value"
    }
} }
```

Shiftr Hangover : Precedence

INPUT

```
{
  "rating": {
    "quality": {
      "value": 3,
    },
    "primary": {
      "value": 4
} } }
```

OUTPUT

```
{
  "Rating": 4,
  "SecondaryRatings": {
    "quality": {
      "Value": 3
    }
  }
}
```

SPEC

```
{
  "rating": {
    "*": {
      "value": "SecondaryRatings.&1.Value"
    },
    "primary" : {
      "value" : "Rating"
} }
```

Has Precedence

Shiftr Hangover : Moving SubTrees

```
INPUT {
  "rating": {
    "quality": {
      "value": 3,
    },
    "primary": {
      "value": 4
} } }
```

```
OUTPUT {
  "Ratings": {
    "quality": {
      "value": 3,
    },
    "primary": {
      "value": 4
} } }
```

```
SPEC
{
  "rating": "Ratings"
}
```

The leaf of the parallel tree walk is determined by the spec

Shiftr Hangover : Moving SubTrees Problems

```
INPUT {
  "rating": {
    "quality": {
      "value": 3,
    },
    "primary": {
      "value": 4
} } }
```

```
OUTPUT {
  "PrimaryRating" : 4,
  "Ratings": {
    "quality": {
      "value": 3,
    },
    "primary": {
      "value": 4
} } }
```

```
SPEC
{
  "rating": {
    "@" : "Ratings",
    "primary" : {
      "value" : "PrimaryRating"
} } }
```

Shiftr Hangover : @ to be clear

```
SPEC @
{
  "rating": {
    "@" : "Ratings"
  }
}
```

Equivalent To :

```
SPEC Normal
{
  "rating": "Ratings"
}
```

Shiftr WildCards 201 : Handling Prefixes

```
INPUT from EMO
{
  "rating-quality": 3,
  "rating-colour": 4
}
```

```
OUTPUT {
  "SecondaryRatings": {
    "quality": {
      "Value": 3
    },
    "colour" : {
      "Value" : 4
} } }
```

```
SPEC {
  "rating-*": "SecondaryRatings.&(0,1).Value",
}
```

What goes in the green box?

```
 &0 = "rating-quality" or "rating-colour"
 &  = (sugar) same as above
 &(0,0) = (canonical form) same as above
 &(0,1) = (first Star) "quality" or "colour"
 &(0,2) = Fail
```

Shiftr WildCards 201 : Prefixes to be Clear

```
INPUT: Avg of SecondaryRating for product Id
196
{
  "stats--colour--196--avgRating" : 4.65
}
SPEC {
  "stats--*--*--avgRating": "...&    ...",
}
```

What are the possible & values?

```
 &0 = "stats--colour--196--avgRating"
 &  = (sugar) same as above
 &(0,0) = (canonical form) same as above
 &(0,1) = (1st star) "colour"
 &(0,2) = (2nd star) "196"
 &(0,3) = (3rd star) Fail
```

Shiftr WildCards 202 : Making Prefixes

```
INPUT {
  "SecondaryRatings": {
    "quality": {
      "Value": 3
    },
    "colour" : {
      "Value" : 4
} } }
```

```
SPEC {
  "SecondaryRatings": {
    "*" : {
      "Value" : "rating-&1"
} } }
```

```
OUTPUT
{
  "rating-quality": 3,
  "rating-colour": 4
}
```

Shiftr 2.0 Summary

Prefix support meant "*" and "&" wildcards could be embedded in "text".

Prevously, & was always by itself aka ".&2."

Shiftr WildCards 202 : Prefix Problem$

```
INPUT from EMO
{
  "rating-quality": 3,
  "rating-colour": 4
}
```

```
OUTPUT {
  "SecondaryRatings": {
    "quality": {
      "Value": 3
    },
    "colour" : {
      "Value" : 4
  } },
  "Sec..RatingsOrder" : [
    "quality",
    "colour"
] }
```

```
SPEC {
  "rating-*": {
     "@" : "SecondaryRatings.&(0,1).Value",
     "$(0,1)" : "SecondaryRatingsOrder.[]"
}
```

Shiftr Hangover : $ to be clear

Prefixing the JSON means that, we have two individually addressable pieces of data on the same line:

```
"rating-quality": 3 -> "quality" and 3
"rating-colour" : 4 -> "colour" and 4
```

Shiftr implicitly operates on the 3 or 4.
"$" lets you use "quality" and "colour" as data

```
SPEC
{
  "rating-*": {
    "@" : "SecondaryRatings.&(0,1).Value",
    "$(0,1)" : "SecondaryRatingsOrder.[]"
} }
```

Shiftr 301 : Explicit Arrays

```
INPUT {
  "photos": [
    "thumb.jpg",
    "normal.jpg"
  ]
}
```

```
OUTPUT {
  "Photos": [
    "normal.jpg",
    "thumb.jpg"
  ]
}
```

```
SPEC {
  "photos": {
    "0" : "Photos[1]",     // sugar
    "1" : "Photos.[0]"     // canonical
  }
}
```

```
Will fail with "NumberFormatException"
    "A" : "Photos.[0]"     // fail
```

Shiftr 301 : Arrays to be clear

```
INPUT Array {
  "photos": [
    "thumb.jpg",
    "normal.jpg"
  ]
}
```

Equivalent

```
INPUT Map {
  "photos": {
    "0" : "thumb.jpg",
    "1" : "normal.jpg"
  }
}
```

```
SPEC {
  "photos": {
    "0" : "Photos[1]",     // sugar
    "1" : "Photos.[0]"     // canonical
  }
}
```

Shiftr treats array indices as keys.

Shiftr 302 : Reference Arrays

```
INPUT {
 "photos": [
  {
   "caption" : "Bat!"
   "url": "normal.jpg"
} ] }
```

```
OUTPUT {
 "Photos": [
  {
   "Cap" : "Bat!"
   "URL": "normal.jpg"
} ] }
```

```
SPEC {
  "photos": {
    "*" : {
      "caption" : "Photos.[&1].Cap",
      "url" :     "Photos.[&1].URL",
} } }
```

Shiftr Final Exam : What is this doing?

```
Polloi to DevApi Spec for Reviews
{
  "~id": "Id",
  "~lastUpdateAt": "LastModificationTime",
  "about": {
    "0": {
      "externalId": "ProductId"
    }
  },
  "cdv-*": {
    "@":       "ContextDataValues.&(0,1).Value",
    "$(0,1)": "ContextDataValues.&.Id"
  },
  "photos": {
    "*": {
      "mediumImageLegacyId":   "Photos[&1].Sizes.medium.Id",
      "thumbnailImageLegacyId":"Photos[&1].Sizes.thumbnail.Id",
      "largeImageLegacyId":    "Photos[&1].Sizes.large.Id",
      "caption":               "Photos[&1].Caption",
      "largeImageExternalUrl": "Photos[&1].Sizes.large.Url"
    }
  },
```

Jolt Extras : Tools for your Custom Transform

```
ElasticSearch to DevApi Spec for Reviews

Object input = ...
{
  "Rating": 3,
  "RatingRange": 5,
  "SecondaryRatings": {
    "quality": {
      "Id": "quality",
      "Value": 3,
      "Range": 7
} } }

SimpleTraversal<Integer> traversal = SimpleTraversal
   .newTraversal( "SecondaryRatings.quality.Value" );

Integer qualityRating = traversal.get( input );

AssertEquals( 3, qualityRating );

// Unlike JsonPath we can set values too
traversal.set( input, 5 );
```

Future

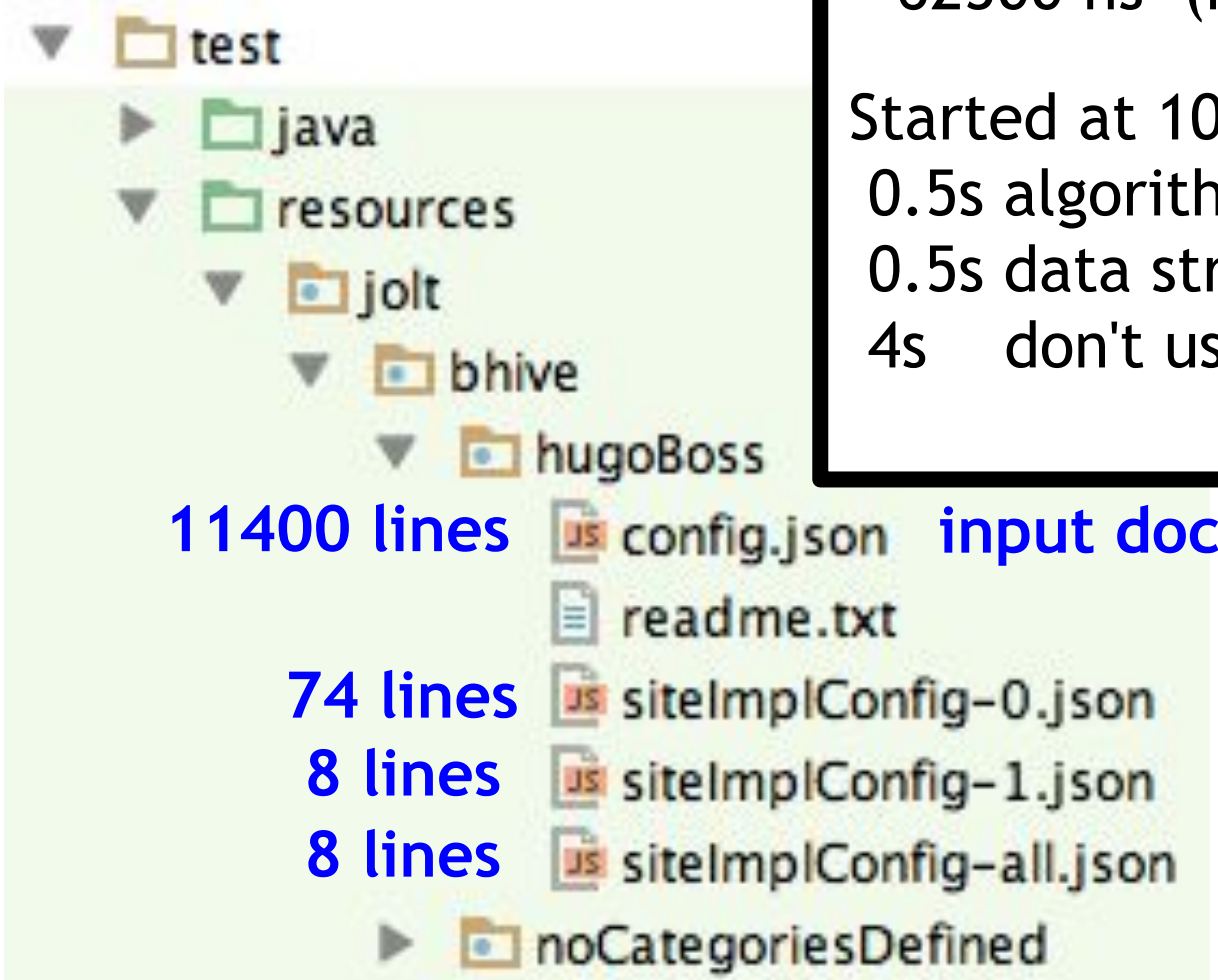
- Make Jackson just a "test" dependency.
- Move JOLT out to it's own project.
- Remove apache.commons dependency
- 0 dependencies
-
- Open Source
- Add new wildcard "#"
- Performance

Questions

A Note on Testing and Performance :

Poor Man's perf test
80k in 5 seconds
0.0625 ms (milliseconds)
62500 ns (nanoseconds)

Started at 10 seconds, dropped
0.5s algorithm improvement
0.5s data structure improvement
4s don't use Regex for "*"



Performance Takeaway : Pacman Looks Right

Totally Unscientific Concierge Performance Stats

Local Concierge / Local ElasticSearch
Small Dataset
"Includes" queries, but no facets

Requests

- 52
- 162 ms avg

Transforms

- 932
- 0.7 ms avg
- 13 ms per Request