Designing a Pythonic Interface

@honzakral

Illustrated guide to `this`

elastic

```
import this
```

Disclaimer

- Personal opinions
- Do as I say, not as I do

API

API is a service for "code"

Fulfills a "contract"

Vaguer

Simplifying Access

Simple is better than complex.
Complex is better than complicated.

Hiding complexity

```
response = client.search(
    index="my-index",
    body={
      "query": {
        "bool": {
          "must": [{"match": {"title": "python"}}],
          "must_not": [{"match": {"description": "beta"}}]
          "filter": [{"term": {"category": "search"}}]
        }
      },
      "aggs" : {
        "per_tag": {
          "terms": {"field": "tags"},
          "aggs": {
            "max_lines": {"max": {"field": "lines"}}
          }
        }
      }
    }
)

for hit in response['hits']['hits']:
    print(hit['_score'], hit['_source']['title'])
```

Hiding complexity

```
s = Search(using=client, index="my-index")
s = s.filter("term", category="search")
s = s.query("match", title="python")
s = s.query(~Q("match", description="beta"))

s.aggs.bucket('per_tag', 'terms', field='tags') \
    .metric('max_lines', 'max', field='lines')

for hit in s:
    print(hit.meta.score, hit.title)
```

Be explicit!

Explicit is better than implicit.

Hide mechanics,

not meaning!

Mechanics

```
response = client.search(
    index="my-index",
    body={
      "query": {
        "bool": {
          "must": [{"match": {"title": "python"}}],
          "must_not": [{"match": {"description": "beta"}}]
          "filter": [{"term": {"category": "search"}}]
        }
      },
      "aggs" : {
        "per_tag": {
          "terms": {"field": "tags"},
          "aggs": {
            "max_lines": {"max": {"field": "lines"}}
          }
        }
      }
    }
)

for hit in response['hits']['hits']:
    print(hit['_score'], hit['_source']['title'])
```

Meaning

```
s = Search(using=client, index="my-index")
s = s.filter("term", category="search")
s = s.query("match", title="python")
s = s.query(~Q("match", description="beta"))

s.aggs.bucket('per_tag', 'terms', field='tags') \
    .metric('max_lines', 'max', field='lines')

for hit in s:
    print(hit.meta.score, hit.title)
```

Admit to leakiness

```
s = Search(using=client, index="my-index")
s = s.filter("term", category="search")
s = s.query("match", title="python")
s = s.query(~Q("match", description="beta"))

s.aggs.bucket('per_tag', 'terms', field='tags') \
    .metric('max_lines', 'max', field='lines')

response = client.search(index="my-index", body=s.to_dict())
```

Be familiar!

In the face of ambiguity,
refuse the temptation to guess.

Copy shamelessly

```
q = Entry.objects.filter(headline__startswith="What")
q = q.exclude(pub_date__gte=date.today())
q = q.filter(pub_date__gte=date.today())
```

```
curl -XGET localhost:9200/my-index/_search -d '{
  "query": {
    "bool": {
      "must": [{"match": {"title": "python"}}],
      "must_not": [{"match": {"description": "beta"}}]
      "filter": [{"term": {"category": "search"}}]
    }
  },
  "aggs" : {
    "per_tag": {
      "terms": {"field": "tags"},
      "aggs": {
        "max_lines": {"max": {"field": "lines"}}
      }
    }
  }
}'
```

Be consistent!

Special cases aren't special enough
to break the rules.

If it makes sense

Although practicality beats purity.

```
s = Search(using=client, index="my-index")
s = s.filter("term", category="search")
s = s.query("match", title="python")
s = s.query(~Q("match", description="beta"))
```

```
s.aggs.bucket('per_tag', 'terms', field='tags') \
    .metric('max_lines', 'max', field='lines')
```

Be friendly!

Python is interactive

dir, __repr__, __doc__

```
>>> for hit in Search().query("match", title="pycon"):
...     dir(hit)
...
["meta", "title", "body", ...]
>>>
>>>
>>> Q({
...   "bool": {
...     "must": [{"match": {"title": "python"}}],
...     "must_not": [{"match": {"description": "beta"}}]
...   }
... })
Bool(must=[Match(title='python')], must_not=[Match(description='beta')])
>>>
>>>
>>> help(Search.to_dict)
Help on function to_dict in module elasticsearch_dsl.search:

to_dict(self, count=False, **kwargs)
    Serialize the search into the dictionary that will be sent over as th
    requests body.

    :arg count: a flag to specify we are interested in a body for count -
        no aggregations, no pagination bounds etc.

    All additional keyword arguments will be included into the dictionary
```

Iterative build

Flat is better than nested.
Sparse is better than dense.

Iterative build

```
s = Search(using=client, index="my-index")
# filter only search
s = s.filter("term", category="search")
# we want python in title
s = s.query("match", title="python")
# and no beta releases
s = s.query(~Q("match", description="beta"))

# aggregate on tags
s.aggs.bucket('per_tag', 'terms', field='tags')
# max lines per tag
s.aggs['per_tag'].metric('max_lines', 'max', field='lines'
```

Safety is friendly

Errors should never pass silently.
Unless explicitly silenced.

Fail by default

allow ignore

Tests? Tests!

Be flexible!

API is still code

Things change

Adapt!

No code is perfect

Now is better than never.
Although never is often better than *right* now.

Thanks!

@honzakral