dictknife Documentation

Release 0.4.8

podhmo

Oct 09, 2022

Contents:

CHAPTER 1

install

```
$ pip install dictknife
```

when using commands.

```
$ pip install "dictknife[command]"
```

source

https://github.com/podhmo/dictknife

CHAPTER 2

as library (dictknife)

- pp
- deepmerge
- deepequal
- loading
- diff
- walkers
- accessor

2.1 pp

result

2.1.1 with OrderedDict

result

2.2 deepmerge

result

2.2.1 with override=True

result

2.3 deepequal

result

2.3.1 with normalize option

result

result

2.4 loading

supported formats

- yaml
- json
- toml

```
from dictknife import loading

loading.setup()

# load
d = loading.loadfile("foo.yaml")
d = loading.loadfile(None, format="yaml")  # from sys.stdin

# dump
loading.dumpfile(d, "foo.json")
loading.dumpfile(d, None, format="toml")  # to sys.stdout
```

2.5 walkers

result

Note: todo: description about chains and operator and context,...

2.6 accessor

accessor is convenience wrapper for accessing dict.

result

CHAPTER 3

as library (jsonknife)

- access by json pointer
- custom data loader

3.1 access by json pointer

access data by json pointer, you can use two functions, below.

- access_by_json_pointer
- assign_by_json_pointer

result

3.1.1 keyname within "/"

result

3.2 custom data loader

Making your custom data loader, such as below.

- *$include* keyword for including extra files's contents

So, if you want to include extra data, you can do via *$include*.

```
main:
  subdata:
    $include <extra file path>
```

how to use it.

```
$ python loader.py main.yaml > loader.output
```

input data are like below.

main.yaml

person.yaml

name.yaml

age.yaml

loaded data

3.2.1 code

loader.py

3.2.2 another example

Resolver's constructor has *onload* argument, this is the hook called when loading data. you can make another version of custom data loader(almost same behavior), using this hook.

CHAPTER 4

as command (dictknife)

- concat
- diff
- transform

4.1 concat

1. Concat dict like data(JSON, YAML).

```
$ dictknife cat --format json <(echo '{"name": "foo"}') <(echo '{"age": 20}')
{
  "name": "foo",
  "age": 20
}
```

1. convert file type (e.g. JSON to YAML)

```
# json to yaml
$ dictknife cat --output-format yaml --input-format json <(echo '{"name": "foo"}')
↪<(echo '{"age": 20}')
name: foo
age: 20

# json to toml
$ dictknife cat --output-format toml --input-format json <(echo '{"name": "foo"}')
↪<(echo '{"age": 20}')
name = "foo"
age = 20
```

4.2 diff

json diff

```
$ cat <<-EOS > person0.yaml
person:
  name: foo
  age: 20
EOS
$ cat <<-EOS > person1.yaml
person:
  age: 20
  name: foo
EOS
$ dictknife diff person{0,1}.yaml
```

```
$ cat <<-EOS > person2.yaml
person:
  age: 20
  name: bar
  nickname: b
EOS
$ dictknife diff person{0,2}.yaml
--- person0.yaml
+++ person2.yaml
@@ -1,6 +1,7 @@
 {
   "person": {
     "age": 20,
-    "name": "foo"
+    "name": "bar",
+    "nickname": "b"
   }
 }
```

4.2.1 normalize option

If input data is yaml format, the types of keys are maybe not one type.

```
$ cat <<-EOS > status.yaml
200:
  ok
default:
  hmm
EOS
$ dictknife diff status.yaml status.yaml
TypeError: unorderable types: str() < int()

$ dictknife diff --normalize status.yaml status.yaml
```

4.2.2 more normalize option

If your data is array, then, another tool something like jq, sorting is not supported.

For example, in the situation like a below.

```
$ cat <<-EOS > people0.json
[
  {
    "name": "foo",
    "age": 10
  },
  {
    "name": "bar",
    "age": 20
  }
]
EOS
$ cat <<-EOS > people1.json
[
  {
    "name": "bar",
    "age": 20
  },
  {
    "name": "foo",
    "age": 10
  }
]
EOS

# jq's -S is not working
$ diff -u <(jq -S . people0.json) <(jq -S . people1.json)
--- /dev/fd/63        2017-06-10 15:41:12.000000000 +0900
+++ /dev/fd/62        2017-06-10 15:41:12.000000000 +0900
@@ -1,10 +1,10 @@
 [
   {
-    "age": 10,
-    "name": "foo"
-  },
-  {
     "age": 20,
     "name": "bar"
+  },
+  {
+    "age": 10,
+    "name": "foo"
   }
 ]

# of cource, using sort_by is working (but it is needed that structural knowledge␣
↪about data).
$ diff -u <(jq -S "sort_by(.name)" people0.json) <(jq -S "sort_by(.name)" people1.
↪json)
```

we can check diff with `--normalize` option only.

```
dictknife diff --normalize people0.json people1.json
```

4.3 transform

```
$ cat status.yaml
200:
  ok
default:
  hmm

$ cat status.yaml | dictknife transform --code='lambda d: [d,d,d]'
- 200: ok
  default: hmm
- 200: ok
  default: hmm
- 200: ok
  default: hmm
```

CHAPTER 5

as command (jsonknife)

Handling JSON data espencially swagger like structure.

- bundle
- cut
- deref
- examples

5.1 deref and cut

```
$ tree src
src
└── colors.yaml
```

src/colors.yaml

5.1.1 deref

deref is unwrap function.

```
mkdir -p dst
jsonknife deref --src src/colors.yaml --ref "#/rainbow/yellow" > dst/00dref.yaml
jsonknife deref --src src/colors.yaml --ref "#/rainbow/yellow@yellow" > dst/01dref.
↪yaml
jsonknife deref --src src/colors.yaml --ref "#/rainbow/yellow@yellow" --ref "#/
↪rainbow/indigo@indigo" > dst/02dref.yaml
```

dst/00deref.yaml with *–ref "#/rainbow/yellow"*

dst/01deref.yaml with *–ref "#/rainbow/yellow@yellow"*

dst/02deref.yaml with *–ref "#/rainbow/yellow@yellow" –ref "#/rainbow/indigo@indigo"*

5.1.2 cut

```
$ jsonknife cut --src ./dst/02deref.yaml --ref "#/yellow" > ./dst/00cut.yaml
```

dst/00cut.yaml

5.2 bundle and deref

```
$ tree src
src/
├── api
│   ├── me.json
│   └── user.json
├── definitions
│   ├── primitive.json
│   └── user.json
└── main.json
```

src/main.json

src/api/me.json

src/api/user.json

src/definitions/primitive.json

src/definitions/user.json

5.2.1 bundle output

bundle output is this.

```
$ jsonknife bundle --src src/main.json --dst bundle.yaml

# if you want json output
$ jsonknife bundle --src src/main.json --dst bundle.json
```

bundle.yaml

5.2.2 deref output

deref output is this.

```
$ jsonknife deref --src src/main.json --dst deref.yaml

# if you want json output
$ jsonknife deref --src src/main.json --dst deref.json
```

deref.yaml

5.3 examples

```
$ tree src
src/
├── person.yaml
└── primitive.yaml

$ jsonknife deref --src src/person.yaml --dst dst/extracted.yaml --ref "#/definitions/
↪person"
$ jsonknife examples dst/extracted.yaml --format yaml > dst/data.yaml
```

src/person.yaml

src/primitive.yaml

dst/extracted.yaml

dst/data.yaml

CHAPTER 6

as command (swaggerknife)

- json2swagger
- flatten

6.1 json2swagger

Generating swagger spec from data.

input data

```
$ swaggerknife json2swagger config.json --name config --dst config-spec.yaml
```

config-spec.yaml

6.1.1 with multiple sources

with multiple sources, required option detection is more accrurately.

input data

person-foo.json

person-bar.json

```
$ swaggerknife json2swagger person-foo.json person-bar.json --name person --dst␣
↪person-spec.yaml
```

person-spec.yaml

01person-bar.json doesn't have nickname and nickname is not reuired in generated spec.

6.1.2 with *—annotate* option

with annotation file.

with-annotations.yaml

annotations.yaml

```
swaggerknife json2swagger with-annotations.yaml --annotate=annotations.yaml --name␣
↪Top --dst with-annotations-spec.yaml
```

with-annotations-spec.yaml

6.2 flatten

only swagger like structure (toplevel is *#/definitions*).

```
$ tree src
src/
└── abc.yaml

$ mkdir -p dst
$ jsonknife flatten --src src/abc.yaml --dst dst/abc.yaml
```

src/abc.yaml

dst/abc.yaml

CHAPTER 7

Indices and tables

- genindex
- modindex
- search