JYCM

eggachecat

Dec 09, 2022

CONTENTS:

CHAPTER

ONE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

TWO

CONTENTS:

2.1 Installation

You can install `jycm` using pip:

```
pip install jycm
```

Or if you wish to install to the home directory:

```
pip install --user jycm
```

For the latest development version, first get the source from github:

```
git clone https://github.com/eggachecat/jycm.git
```

Then navigate into the local `jycm` directory and simply run:

```
python setup.py install
```

or:

```
python setup.py install --user
```

and you're done!

2.2 Usage and Examples

2.2.1 Overview

Usage

Basically all you need to diff two json with *JYCM* is as follow

```
ycm = YouchamaJsonDiffer(left, right)
ycm.diff()
result = ycm.to_dict()
```

where

- ***ycm = YouchamaJsonDiffer(left, right)* is used to setup the class;**

  - you can specify other parameters to extend the ability of default ones.
  - usage parameters are shown in below
- ***ycm.diff* is used to call the real *diff***
  - where the core diff functions are executed
- ***ycm.to_dict* is used to get the result.**
  - will return a dict
  - detail about the dict is explained right below

Return

What *ycm.to_dict* return is a dict whose keys are names of the diff-events, and values are list of dict which contains all relative info.

There's six default diff-events:

- ***dict:add***
  - keys being inserted comparing the old one
- ***dict:remove***
  - keys being removed comparing the old one
- ***list:add***
  - item being inserted comparing the old one
  - using LCS to match
- ***list:remove***
  - item being removed comparing the old one
  - using LCS to match
- ***value_changes***
  - values being changed

All the relative info at least contains these keys:

- ***left***
  - the paired old value
- ***right***
  - the paired new value
- ***left_path***
  - the paired old json path
- ***right_path***
  - the paired new json path

when it comes old/new value does not exist (for example dict:add indicting old value on the json path does not exist) a str *__NON_EXIST__* will be held.

2.2.2 Default behaviour

These examples below will show the default behaviours when it comes to comparing two json values. It will give you a sense what will be returned from the *YouchamaJsonDiffer.to_dict* after you call *YouchamaJsonDiffer.diff*.

```
left = {
    "a": 1,
    "b": 2,
    "d": "12345",
    "f": False,
    "e": [
        {"x": 1, "y": 1},
        {"x": 2, "y": 2},
        {"x": 3, "y": 3},
        {"x": 4, "y": 4},
    ]
}

right = {
    "a": 1,
    "b": 3,
    "c": 4,
    "f": True,
    "e": [
        {"x": 0, "y": 1},
        {"x": 2, "y": 2},
        {"x": 3, "y": 3},
        {"x": 5, "y": 5},
    ]
}

ycm = YouchamaJsonDiffer(left, right)
ycm.diff()

expected = {
    'dict:add': [
        {'left': '__NON_EXIST__',
         'left_path': '',
         'right': 4,
         'right_path': 'c'}
    ],
    'dict:remove': [
        {'left': '12345',
         'left_path': 'd',
         'right': '__NON_EXIST__',
         'right_path': ''}
    ],
    'list:add': [
        {'left': '__NON_EXIST__',
         'left_path': '',
         'right': {'x': 5, 'y': 5},
         'right_path': 'e->[3]'}
    ],
```

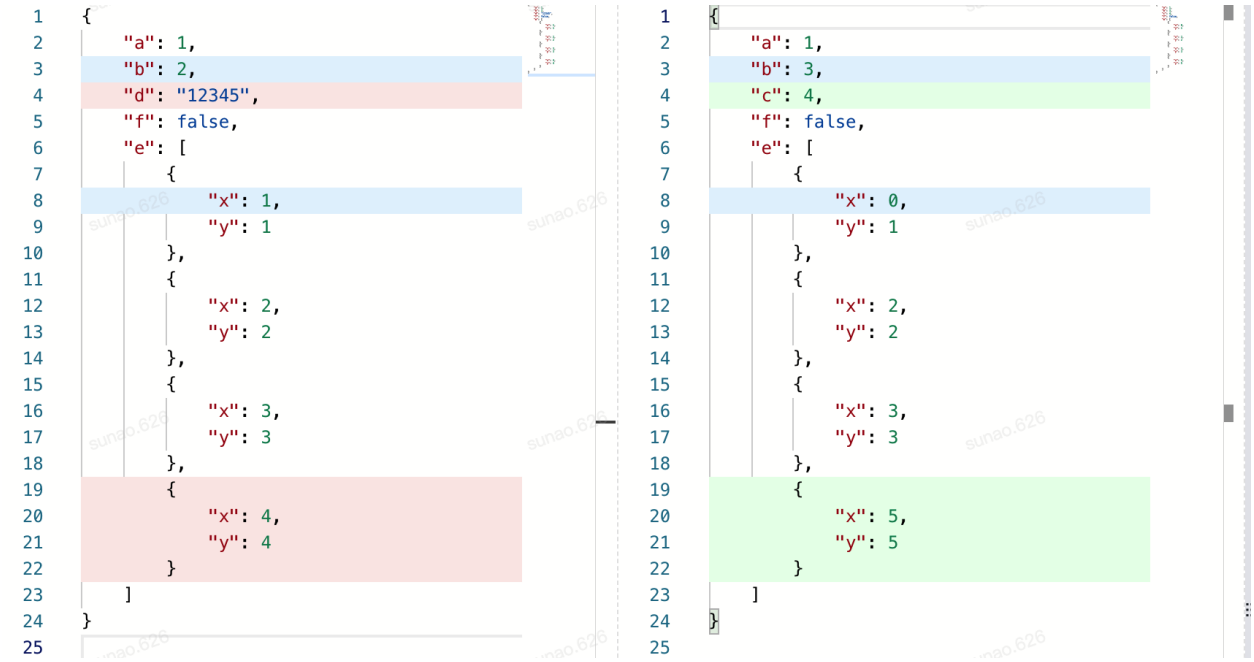
(continues on next page)

(continued from previous page)

```
    'list:remove': [
        {'left': {'x': 4, 'y': 4},
         'left_path': 'e->[3]',
         'right': '__NON_EXIST__',
         'right_path': ''}
    ],
    'value_changes': [
        {'left': 2,
         'left_path': 'b',
         'new': 3,
         'old': 2,
         'right': 3,
         'right_path': 'b'},
        {'left': 1,
         'left_path': 'e->[0]->x',
         'new': 0,
         'old': 1,
         'right': 0,
         'right_path': 'e->[0]->x'},
        {'left': False,
         'left_path': 'f',
         'new': True,
         'old': False,
         'right': True,
         'right_path': 'f'}
    ]
}
assert ycm.to_dict(no_pairs=True) == expected
```

Above is the example of default behaviours. All the default events are shown and you can check if the results are as you expected.

For better understanding here's a graph of the diff result where the red stands for "remove" and the green stands for "insert" and blue stands for "change". BTW I will release this rendering component soon



As you can see JYCM is smart: for example, it knows there's a *value_changes* on *e->[0]->[x]* and a remove on *e->[3]* of left and an insertion on *e->[3]* of right.

For more information, you can check the *Design*.

2.2.3 Ignore Order

When it comes to set, things are getting interesting. `YouchamaJsonDiffer` provides a parameter *ignore_order_func* to determine whether the values comparing should ignore order or not.

You can use the helper function **make_ignore_order_func** from **ycm.helper** to make this function which takes into a list of json path regax as parameters. Here's the example:

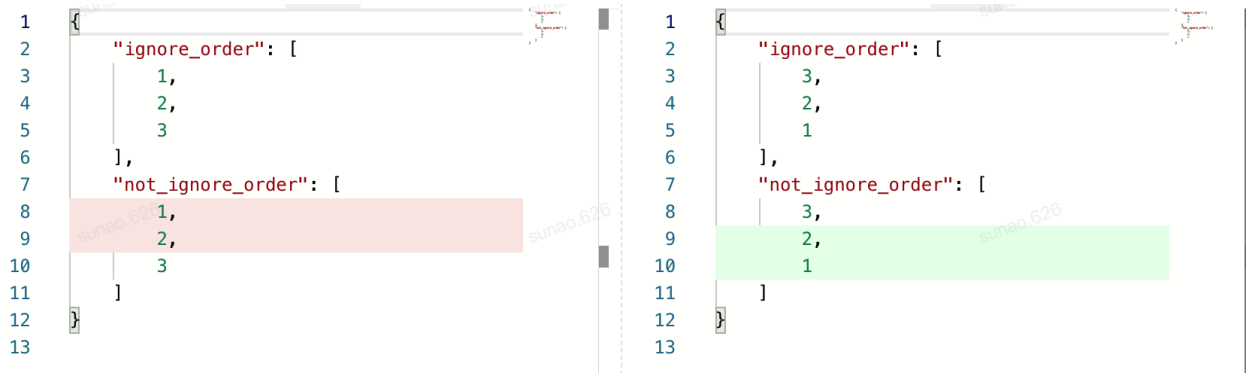
```
left = {
    "ignore_order": [1, 2, 3],
    "not_ignore_order": [1, 2, 3]
}

right = {
    "ignore_order": [3, 2, 1],
    "not_ignore_order": [3, 2, 1]
}

ycm = YouchamaJsonDiffer(left, right, ignore_order_func=make_ignore_order_func([
    "^ignore_order$"
]))
ycm.diff()
expected = {
    'list:add': [
        {'left': '__NON_EXIST__',
         'left_path': '',
         'right': 2,
```

(continues on next page)

(continued from previous page)

```
         'right_path': 'not_ignore_order->[1]'},
        {'left': '__NON_EXIST__',
         'left_path': '',
         'right': 1,
         'right_path': 'not_ignore_order->[2]'}
    ],
    'list:remove': [
        {'left': 1,
         'left_path': 'not_ignore_order->[0]',
         'right': '__NON_EXIST__',
         'right_path': ''},
        {'left': 2,
         'left_path': 'not_ignore_order->[1]',
         'right': '__NON_EXIST__',
         'right_path': ''}
    ]
}
assert ycm.to_dict(no_pairs=True) == expected
```

And here's the graph:



2.2.4 Default Operators

There's some operators implemented in JYCM out of the box.

IgnoreOperator

IgnoreOperator is for ignoring all the changes. Instead, a "ignore" event is reported.

```
left = {
    "ignore_me_remove": 1,
    "ignore_me_change": 1
}

right = {
    "ignore_me_add": 1,
    "ignore_me_change": 2
}
```

(continues on next page)

(continued from previous page)

```
ycm = YouchamaJsonDiffer(left, right, custom_operators=[
    IgnoreOperator("ignore_me")
])
ycm.diff()
expected = {
    'ignore': [
        {'left': '__NON_EXIST__',
         'right': 1,
         'left_path': '',
         'right_path': 'ignore_me_add',
         'path_regex': 'ignore_me',
         'pass': True},
        {'left': 1,
         'right': 2,
         'left_path':
         'ignore_me_change',
         'right_path': 'ignore_me_change',
         'path_regex': 'ignore_me',
         'pass': True},
        {'left': 1,
         'right': '__NON_EXIST__',
         'left_path': 'ignore_me_remove',
         'right_path': '',
         'path_regex': 'ignore_me',
         'pass': True}
    ]
}
assert ycm.to_dict(no_pairs=True) == expected
```

And here's the graph:

ExpectExistOperator

2.2.5 Custom Operators

Sometimes the distance function (or the function metering similarity) are more domain based.

For example, there's a json representing two points (named *distance_ok* and *distance_too_far*) on 2D space:

```
left = {
    "distance_ok": {
        "x": 1,
        "y": 1
    },
    "distance_too_far": {
        "x": 5,
```

(continues on next page)

(continued from previous page)

```
        "y": 5
    },
}

right = {
    "distance_ok": {
        "x": 2,
        "y": 2
    },
    "distance_too_far": {
        "x": 7,
        "y": 9
    },
}
```

How on earth a diff framework would know if they are the same or not ? It is where *custom operator* kicks in.

You can define an operator as following:

```
from jycm.operator import BaseOperator

class L2DistanceOperator(BaseOperator):
    __operator_name__ = "operator:l2distance"
    __event__ = "operator:l2distance"

    def __init__(self, path_regex, distance_threshold):
        super().__init__(path_regex=path_regex)
        self.distance_threshold = distance_threshold

    def diff(self, level: 'TreeLevel', instance, drill: bool) -> Tuple[bool, float]:
        distance = math.sqrt(
            (level.left["x"] - level.right["x"]) ** 2 + (level.left["y"] - level.right["y
↪"]) ** 2
        )
        info = {
            "distance": distance,
            "distance_threshold": self.distance_threshold,
            "pass": distance < self.distance_threshold
        }

        if not drill:
            instance.report(self.__event__, level, info)
        return True, 1 if info["pass"] else 0
```

Here's two things you should notice:

1. extends your class with *BaseOperator* which is imported from jycm.operator module
2. implement a diff function whose signature is as above.

Here's more detail about the signature:

- **Parameters**
  - **level** is the instance of `TreeLevel` which is the base element being compared in JYCM

  - **instance** is the instance of `YouchamaJsonDiffer` giving you full control of the process
    * normally calling instance.report is enough
    * which is for reporting the diff event
  - **drill** is a bool function indicting whether this *diff* call is in reporting phase or not
    * drill is False when it is in the reporting phase
    * otherwise just a normal diff call (e.g. in the matching-items-in-array phase)

- **Return**
  - the first is a bool which indicting whether the diff space should be stopped from this operator
    * an example is *ignore* which returns *True* since whether the *level* or its children are different or not is irrelevant
  - the second is a float number in [0,1] mapping the similarity of the *level.left* and *level.right*. 1 is for they being are same while 0 is for they being not the same at all.
    * **TIPS** similarity maybe different in drill phase. (for example, in the matching phase, two items in array who have the same *id* field are the same even though they are different in other fields which should be reported in the reporting phase)

then you can simply init your class and pass it in to the *custom_operators*

```
ycm = YouchamaJsonDiffer(left, right, custom_operators=[
    L2DistanceOperator("distance.*", 3),
])

ycm.diff()

expected = {
    'operator:l2distance': [
        {
            'left': {'x': 1, 'y': 1},
            'right': {'x': 2, 'y': 2},
            'left_path': 'distance_ok',
            'right_path': 'distance_ok',
            'distance': 1.4142135623730951,
            'distance_threshold': 3,
            'pass': True
        },
        {
            'left': {'x': 5, 'y': 5},
            'right': {'x': 7, 'y': 9},
            'left_path': 'distance_too_far',
            'right_path': 'distance_too_far',
            'distance': 4.47213595499958,
            'distance_threshold': 3,
            'pass': False
        }
    ]
}
assert ycm.to_dict(no_pairs=True) == expected
```

Vola!

2.3 Design

2.4 Reference

2.4.1 Submodules

2.4.2 jycm.common module

2.4.3 jycm.helper module

jycm.helper.**dump_html_output**(*left*, *right*, *diff_result*, *output*, *left_title='Left'*, *right_title='Right'*)

jycm.helper.**make_ignore_order_func**(*ignore_order_path_regex_list*)

jycm.helper.**make_json_path_key**(*path_list: List[str]*)

jycm.helper.**open_url**(*index_url*)

jycm.helper.**render_to_html**(*left*, *right*, *diff_result*, *main_script_path='./index.js'*, *left_title='Left'*, *right_title='Right'*)

2.4.4 jycm.jycm module

exception jycm.jycm.**DiffLevelException**

Bases: Exception

The exception that will be threw from _diff function.

exception jycm.jycm.**DifferentTypeException**

Bases: Exception

The exception that will be threw when left and right are of different types

This maybe improved as a feature.

class jycm.jycm.**ListItemPair**(*value:* TreeLevel, *left_index*, *right_index*)

Bases: object

Pair of array items

class jycm.jycm.**Record**(*event: str*, *level:* TreeLevel, *info: Dict*)

Bases: object

To record the info made from operators and differ.

Parameters

- **event** – a unique string to describe the info
- **level** – where the info and event are described for
- **info** – the additional info attached to the event

`to_dict`()

Convert to dict

Returns
A dict

class jycm.jycm.**TreeLevel**(*left*, *right*, *left_path: List*, *right_path: List*, *up: Optional[TreeLevel]*, *diff: Optional[Callable[[TreeLevel, bool], Tuple[bool, float]]] = None*)

Bases: `object`

The base data structure for diffing.

Parameters

- **left** – left value
- **right** – right value
- **left_path** – left json path
- **right_path** – right json path
- **up** – the parent TreeLevel
- **diff** – a simple way to inject custom operators ; default None

`get_key`()

Get the key of this level

Returns
The unique key for this level

`get_path`()

Get the path of this level

Returns
The left path of this level (matching will be concerned on left mainly)

`get_type`() → type

Get the type of this level

Returns
Return the type of this level. If left and right are of different type, then a exception will be threw.

`to_dict`()

Convert TreeLevel to dict

Returns
A dict contains all info about this level.

class jycm.jycm.**YouchamaJsonDiffer**(*left*, *right*, *custom_operators: Optional[List[BaseOperator]] = None*, *ignore_order_func: Optional[Callable[[TreeLevel, bool], bool]] = None*, *debug=False*, *fast_mode=False*, *use_cache=True*)

Bases: `object`

Parameters

- **left** – a dict value.
- **right** – a dict value.
- **custom_operators** – List of operators extend from BaseOperator.

- **ignore_order_func** – the func that decides whether the current array should be ignored order or not. (level: TreeLevel, drill: boolean) => bool
- **debug** – set True then some debug info will be collected. default False.
- **fast_mode** – whether or not using LCS. default True

compare_dict(*level:* TreeLevel, *drill=False*) → float

Compare Dict

Parameters

- **level** – the tree level to be diffed
- **drill** – whether this diff is in drill mode.

Returns

A score between 0~1 to describe how similar **level.left** and **level.right** are. The score will be average score for all scores for all keys.

compare_list(*level:* TreeLevel, *drill=False*) → float

compare_list_with_order(*level:* TreeLevel, *drill=False*) → float

Compare two arrays with order

Parameters

- **level** – the tree level to be diffed
- **drill** – whether this diff is in drill mode.

Returns

A score between 0~1 to describe how similar level.left and level.right are

compare_list_without_order(*level:* TreeLevel, *drill=False*) → float

compare_primitive(*level:* TreeLevel, *drill=False*) → float

Compare primitive values

Parameters

- **level** – the tree level to be diffed
- **drill** – whether this diff is in drill mode.

Returns

A score between 0~1 to describe how similar **level.left** and **level.right** are.

diff()

Entry function to be called to diff

diff_level(*level:* TreeLevel, *drill: bool*) → float

Diff level function It is a wrapper of _diff_level with cache mechanism.

Parameters

- **level** – the tree level to be diffed
- **drill** – whether this diff is in drill mode.

Returns

A score between 0~1 to describe how similar **level.left** and **level.right** are.

generate_lcs_pair_list(*level:* TreeLevel) → List[*ListItemPair*]

Generate all ListItemPair

Use LCS algorithm to match arrays with taking order into consideration

Parameters
level – a tree level

Returns
List of pairs

get_diff(*no_pairs=False*)

Do the diff and return the json diff

Normally to_dict is enough to collect all the info.

Parameters
no_pairs – boolean to decide whether to report pairs of json path

Returns
a dict

report(*event: str*, *level:* TreeLevel, *info: Optional[Dict] = None*)

Report any useful info

Parameters

- **event** – a unique string to describe the info
- **level** – where the info and event are described for
- **info** – the additional info attached to the event

report_pair(*level:* TreeLevel)

Report pair of json path

In order to save space, only the level whose left and right path are different will be reported

Parameters
level – TreeLevel

to_dict(*no_pairs=False*) → dict

Convert this to a dict

Normally to_dict is enough to collect all the info.

Parameters
no_pairs – boolean to decide whether to report pairs of json path

Returns
a dict

use_custom_operators(*level:* TreeLevel, *drill=False*) → Tuple[bool, float]

Compare with custom operators

Parameters

- **level** – the tree level to be diffed
- **drill** – whether this diff is in drill mode.

Returns
A boolean to determine whether diff should be early stopped. A score between 0~1 to describe how similar **level.left** and **level.right** are.

jycm.jycm.**gather_serial_pair**(*target_index*, *indices*, *list_*, *container*)

Match parts between LCS index

Parameters

- **target_index** – LCS index that being collected before
- **indices** – all candidates index
- **list** – list of values
- **container** – reference of collector

Returns:

2.4.5 jycm.km_matcher module

class jycm.km_matcher.**KMMatcher**(*weights*)

Bases: object

add_to_tree(*x*, *prevx*)

do_augment(*x*, *y*)

find_augment_path()

solve(*verbose=False*)

update_labels()

2.4.6 jycm.operator module

class jycm.operator.**BaseOperator**(*path_regex: str*)

Bases: object

diff(*level:* TreeLevel, *instance*, *drill: bool*) → Tuple[bool, float]

match(*level:* TreeLevel) → bool

class jycm.operator.**ExpectChangeOperator**(*path_regex: str*)

Bases: *BaseOperator*

diff(*level:* TreeLevel, *instance:* YouchamaJsonDiffer, *drill: bool*) → Tuple[bool, float]

class jycm.operator.**ExpectExistOperator**(*path_regex: str*)

Bases: *BaseOperator*

diff(*level:* TreeLevel, *instance:* YouchamaJsonDiffer, *drill: bool*) → Tuple[bool, float]

class jycm.operator.**FloatInRangeOperator**(*path_regex*, *interval_start*, *interval_end*)

Bases: *BaseOperator*

diff(*level:* TreeLevel, *instance:* YouchamaJsonDiffer, *drill: bool*) → Tuple[bool, float]

class jycm.operator.**IgnoreOperator**(*path_regex: str*, **args*, ***kwargs*)

Bases: *BaseOperator*

diff(*level:* TreeLevel, *instance:* YouchamaJsonDiffer, *drill: bool*) → Tuple[bool, float]

class jycm.operator.**ListItemFieldMatchOperator**(*path_regex*, *field*)

Bases: *BaseOperator*

diff(*level:* TreeLevel, *instance:* YouchamaJsonDiffer, *drill: bool*) → Tuple[bool, float]

jycm.operator.**get_operator**(*name: str*)

jycm.operator.**register_operator**(*operator_class: Type[*BaseOperator*]*)

2.4.7 Module contents

jycm codes. .

PYTHON MODULE INDEX

j

INDEX

L

M

O

R

S

T

U

Y