deepmerge Documentation

Release 0.1

Yusuke Tsutsumi

Apr 25, 2021

Contents

1 Example **3**
- 1.1 User Guide 4
- 1.2 Strategies 5
- 1.3 API Reference 7

2 Indices and tables **9**

Python Module Index **11**

Index **13**

Deepmerge is a flexible library to handle merging of nested data structures in Python (e.g. lists, dicts).

It is available on pypi, and can be installed via pip:

```
pip install deepmerge
```

CHAPTER 1

Example

Generic Strategy

```
from deepmerge import always_merger

base = {"foo": ["bar"]}
next = {"foo": ["baz"]}

expected_result = {'foo': ['bar', 'baz']}
result = always_merger.merge(base, next)

assert expected_result == result
```

Custom Strategy

```
from deepmerge import Merger

my_merger = Merger(
    # pass in a list of tuple, with the
    # strategies you are looking to apply
    # to each type.
    [
        (list, ["append"]),
        (dict, ["merge"]),
        (set, ["union"])
    ],
    # next, choose the fallback strategies,
    # applied to all other types:
    ["override"],
    # finally, choose the strategies in
    # the case where the types conflict:
    ["override"]
)
base = {"foo": ["bar"]}
next = {"bar": "baz"}
```

(continues on next page)

(continued from previous page)

```
my_merger.merge(base, next)
assert base == {"foo": ["bar"], "bar": "baz"}
```

Want to get started? see the *User Guide*

Contents:

1.1 User Guide

1.1.1 Simple Usage

deepmerge was written as a library to help construct merge functions, eliminating some of the boilerplate around recursing through common data structures and joining results. Although it's recommended to choose your own strategies, deepmerge does provided some preconfigured mergers for a common situations:

- deepmerge.always_merger: always try to merge. in the case of mismatches, the value from the second object overrides the first o ne.
- deepmerge.merge_or_raise: try to merge, raise an exception if an unmergable situation is encountered.
- deepmerge.conservative_merger: similar to always_merger, but in the case of a conflict, use the existing value.

Once a merger is constructed, it then has a merge() method that can be called:

```
from deepmerge import always_merger

base = {"foo": "value", "baz": ["a"]}
next = {"bar": "value2", "baz": ["b"]}

always_merger.merge(base, next)

assert base == {
    "foo": "value",
    "bar": "value2",
    "baz": ["a", "b"]
}
```

1.1.2 Merges are Destructive

You may have noticed from the example, but merging is a destructive behavior: it will modify the first argument passed in (the base) as part of the merge.

This is intentional, as an implicit copy would result in a significant performance slowdown for deep data structures. If you need to keep the original objects unmodified, you can use the deepcopy method:

```
from copy import deepcopy
result = deepcopy(base)
always_merger.merge(result, next)
```

1.1.3 Authoring your own Mergers

The *deepmerge.merger.Merger* class enacts the merging strategy, and stores the configuration about the merging strategy chosen.

The merger takes a list of a combination of strings or functions, which are expanded into strategies that are attempted in the order in the list.

For example, a list of [“append”, “merge”] will attempt the “append” strategy first, and attempt the merge strategy if append is not able to merge the structures.

If none of the strategies were able to merge the structures (or if non exists), a `deepmerge.exception.InvalidMerge` exception is raised.

Strategies

The merger class alone does not make any decisions around merging the code. This is instead deferred to the strategies themselves.

1.1.4 Built-in Strategies

If you name a strategy with a string, it will attempt to match that with the merge strategies that are built into deepmerge. You can see a list of which strategies exist for which types at *Strategies*

1.1.5 Custom Strategies

Strategies are functions that satisfy the following properties:

- have the function signature (config, path, base, nxt)
- return the merged object, or None.

Example:

```python
def append_last_element(config, path, base, nxt):
    """ a list strategy to append the last element of nxt only. """
    if len(nxt) > 0:
       base.append(nxt[-1])
       return base
```

If a strategy fails, an exception should not be raised. This is to ensure it can be chained with other strategies, or the fall-back.

1.2 Strategies

1.2.1 Authoring your own Strategy

Your function should take the arguments of (`merger`, `path`, `base_value`, `value_to_merge_in`).

Strategies are passed as a list, and the merge runs through each strategy in the order passed into the merger, stopping at the first one to return a value that is not the sentinel value deepmerge.STRATEGY_END.

For example, this function would not be considered valid for any base value besides the string “foo”:

```python
from deepmerge import STRATEGY_END

def return_true_if_foo(config, path, base, nxt):
  if base == "foo":
```

(continues on next page)

(continued from previous page)

```
        return True
    return STRATEGY_END
```

Note that the merger does not copy values before passing them into mergers for performance reasons.

1.2.2 Builtin Strategies

These are the built in strategies provided by deepmerge.

class `deepmerge.strategy.type_conflict.`**`TypeConflictStrategies`**(*strategy_list*)
contains the strategies provided for type conflicts.

`NAME = 'type conflict'`

static strategy_override(*config*, *path*, *base*, *nxt*)
overrides the new object over the old object

static strategy_override_if_not_empty(*config*, *path*, *base*, *nxt*)
overrides the new object over the old object only if the new object is not empty or null

static strategy_use_existing(*config*, *path*, *base*, *nxt*)
uses the old object instead of the new object

class `deepmerge.strategy.fallback.`**`FallbackStrategies`**(*strategy_list*)
The StrategyList containing fallback strategies.

`NAME = 'fallback'`

static strategy_override(*config*, *path*, *base*, *nxt*)
use nxt, and ignore base.

static strategy_use_existing(*config*, *path*, *base*, *nxt*)
use base, and ignore next.

class `deepmerge.strategy.dict.`**`DictStrategies`**(*strategy_list*)
Contains the strategies provided for dictionaries.

`NAME = 'dict'`

static strategy_merge(*config*, *path*, *base*, *nxt*)
for keys that do not exists, use them directly. if the key exists in both dictionaries, attempt a value merge.

static strategy_override(*config*, *path*, *base*, *nxt*)
move all keys in nxt into base, overriding conflicts.

class `deepmerge.strategy.list.`**`ListStrategies`**(*strategy_list*)
Contains the strategies provided for lists.

`NAME = 'list'`

static strategy_append(*config*, *path*, *base*, *nxt*)
append nxt to base.

static strategy_override(*config*, *path*, *base*, *nxt*)
use the list nxt.

static strategy_prepend(*config*, *path*, *base*, *nxt*)
prepend nxt to base.

1.3 API Reference

class `deepmerge.merger.`**`Merger`**(*type_strategies*, *fallback_strategies*, *type_conflict_strategies*)

> **Parameters** **`List[Tuple]`** (*`type_strategies,`*) – a list of (Type, Strategy) pairs that should be used against incoming types. For example: (dict, "override").

exception `deepmerge.exception.`**`DeepMergeException`**

exception `deepmerge.exception.`**`InvalidMerge`**(*strategy_list_name*, *merge_args*, *merge_kwargs*)

exception `deepmerge.exception.`**`StrategyNotFound`**

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

d

deepmerge.exception, 7

Index

D

deepmerge.exception (*module*), 7
DeepMergeException, 7
DictStrategies (*class in deepmerge.strategy.dict*), 6

F

FallbackStrategies (*class in deepmerge.strategy.fallback*), 6

I

InvalidMerge, 7

L

ListStrategies (*class in deepmerge.strategy.list*), 6

M

Merger (*class in deepmerge.merger*), 7

N

NAME (*deepmerge.strategy.dict.DictStrategies attribute*), 6
NAME (*deepmerge.strategy.fallback.FallbackStrategies attribute*), 6
NAME (*deepmerge.strategy.list.ListStrategies attribute*), 6
NAME (*deepmerge.strategy.type_conflict.TypeConflictStrategies attribute*), 6

S

strategy_append() (*deepmerge.strategy.list.ListStrategies static method*), 6
strategy_merge() (*deepmerge.strategy.dict.DictStrategies static method*), 6
strategy_override() (*deepmerge.strategy.dict.DictStrategies static method*), 6
strategy_override() (*deepmerge.strategy.fallback.FallbackStrategies static method*), 6
strategy_override() (*deepmerge.strategy.list.ListStrategies static method*), 6
strategy_override() (*deepmerge.strategy.type_conflict.TypeConflictStrategies static method*), 6
strategy_override_if_not_empty() (*deepmerge.strategy.type_conflict.TypeConflictStrategies static method*), 6
strategy_prepend() (*deepmerge.strategy.list.ListStrategies static method*), 6
strategy_use_existing() (*deepmerge.strategy.fallback.FallbackStrategies static method*), 6
strategy_use_existing() (*deepmerge.strategy.type_conflict.TypeConflictStrategies static method*), 6
StrategyNotFound, 7

T

TypeConflictStrategies (*class in deepmerge.strategy.type_conflict*), 6