jsonpath-rw Documentation

Release latest

September 03, 2015

Contents

https://github.com/kennknowles/python-jsonpath-rw

This library provides a robust and significantly extended implementation of JSONPath for Python. It is tested with Python 2.6, 2.7, 3.2, 3.3. *(On travis-ci there is a segfault when running the tests with pypy; I don't think the problem lies with this library)*.

This library differs from other JSONPath implementations in that it is a full *language* implementation, meaning the JSONPath expressions are first class objects, easy to analyze, transform, parse, print, and extend. (You can also execute them :-)

CHAPTER 1

Quick Start

To install, use pip:

```
$ pip install jsonpath-rw
```

Then:

```
$ python

>>> from jsonpath_rw import jsonpath, parse

# A robust parser, not just a regex. (Makes powerful extensions possible; see below)
>>> jsonpath_expr = parse('foo[*].baz')

# Extracting values is easy
>>> [match.value for match in jsonpath_expr.find({'foo': [{'baz': 1}, {'baz': 2}]})]
[1, 2]

# Matches remember where they came from
>>> [str(match.full_path) for match in jsonpath_expr.find({'foo': [{'baz': 1}, {'baz': 2}]})]
['foo.[0].baz', 'foo.[1].baz']

# And this can be useful for automatically providing ids for bits of data that do not have them (curr
>>> jsonpath.auto_id_field = 'id'
>>> [match.value for match in parse('foo[*].id').find({'foo': [{'id': 'bizzle'}, {'baz': 3}]})]
['foo.bizzle', 'foo.[1]']

# A handy extension: named operators like `parent`
>>> [match.value for match in parse('a.*.b.`parent`.c').find({'a': {'x': {'b': 1, 'c': 'number one'},
['number two', 'number one']

# You can also build expressions directly quite easily
>>> from jsonpath_rw.jsonpath import Fields
>>> from jsonpath_rw.jsonpath import Slice

>>> jsonpath_expr_direct = Fields('foo').child(Slice('*')).child(Fields('baz'))  # This is equivalent
```

CHAPTER 2

JSONPath Syntax

The JSONPath syntax supported by this library includes some additional features and omits some problematic features (those that make it unportable). In particular, some new operators such as `|` and `where` are available, and parentheses are used for grouping not for callbacks into Python, since with these changes the language is not trivially associative. Also, fields may be quoted whether or not they are contained in brackets.

Atomic expressions:

Syntax	Meaning
`$`	The root object
`` `this` ``	The "current" object.
`` `foo` ``	More generally, this syntax allows "named operators" to extend JSONPath is arbitrary ways
field	Specified field(s), described below
`[` *field* `]`	Same as *field*
`[` *idx* `]`	Array access, described below (this is always unambiguous with field access)

Jsonpath operators:

Syntax	Meaning
jsonpath1 `.` *jsonpath2*	All nodes matched by *jsonpath2* starting at any node matching *jsonpath1*
jsonpath `[` *whatever* `]`	Same as *jsonpath*`.`*whatever*
jsonpath1 `..` *jsonpath2*	All nodes matched by *jsonpath2* that descend from any node matching *jsonpath1*
jsonpath1 `where` *jsonpath2*	Any nodes matching *jsonpath1* with a child matching *jsonpath2*
jsonpath1 `|` *jsonpath2*	Any nodes matching the union of *jsonpath1* and *jsonpath2*

Field specifiers (*field*):

Syntax	Meaning
`fieldname`	the field `fieldname` (from the "current" object)
`"fieldname"`	same as above, for allowing special characters in the fieldname
`'fieldname'`	ditto
`*`	any field
field `,` *field*	either of the named fields (you can always build equivalent jsonpath using `|`)

Array specifiers (*idx*):

Syntax	Meaning
`[`*n*`]`	array index (may be comma-separated list)
`[`*start*`?:`*end*`?]`	array slicing (note that *step* is unimplemented only due to lack of need thus far)
`[*]`	any array index

CHAPTER 3

Programmatic JSONPath

If you are programming in Python and would like a more robust way to create JSONPath expressions that does not depend on a parser, it is very easy to do so directly, and here are some examples:

- `Root()`
- `Slice(start=0, end=None, step=None)`
- `Fields('foo', 'bar')`
- `Index(42)`
- `Child(Fields('foo'), Index(42))`
- `Where(Slice(), Fields('subfield'))`
- `Descendants(jsonpath, jsonpath)`

CHAPTER 4

Extensions

- *Path data*: The result of `JsonPath.find` provide detailed context and path data so it is easy to traverse to parent objects, print full paths to pieces of data, and generate automatic ids.
- *Automatic Ids*: If you set `jsonpath_rw.auto_id_field` to a value other than None, then for any piece of data missing that field, it will be replaced by the JSONPath to it, giving automatic unique ids to any piece of data. These ids will take into account any ids already present as well.
- *Named operators*: Instead of using `@` to reference the currently object, this library uses `` `this` ``. In general, any string contained in backquotes can be made to be a new operator, currently by extending the library.

CHAPTER 5

More to explore

There are way too many jsonpath implementations out there to discuss. Some are robust, some are toy projects that still work fine, some are exercises. There will undoubtedly be many more. This one is made for use in released, maintained code, and in particular for programmatic access to the abstract syntax and extension. But JSONPath at its simplest just isn't that complicated, so you can probably use any of them successfully. Why not this one?

The original proposal, as far as I know:

- JSONPath - XPath for JSON by Stefan Goessner.

CHAPTER 6

Special note about PLY and docstrings

The main parsing toolkit underlying this library, PLY, does not work with docstrings removed. For example, `PYTHONOPTIMIZE=2` and `python -OO` will both cause a failure.

CHAPTER 7

Contributors

This package is authored and maintained by:

- Kenn Knowles (@kennknowles)

with the help of patches submitted by these contributors.

CHAPTER 8

Copyright and License