boolrule Documentation

Release 0.3.0

Steve Webster

Jan 15, 2018

Contents

boolrule is a simpe boolean expression evaluation engine.

boolrule was built by the team at tails.com to evaluate conditional edges between nodes in a graph-like structure, though we've found numerous uses for it since.

Contents:

CHAPTER 1

boolrule

Simple boolean expression evaluation engine.

- Free software: MIT license
- Documentation: https://boolrule.readthedocs.io.

1.1 Features

Compare simple boolean statements:

```
>>> rule = BoolRule('5 > 3')
>>> rule.test()
True
>>> rule = BoolRule('5 < 3')
>>> rule.test()
False
```

Evaluate boolean statements against a context dict:

```
>>> can_buy_beer = BoolRule('user.age_years >= 18')
>>> can_buy_beer.test({'user':{'age_years': 12}})
False
>>> can_buy_beer.test({'user':{'age_years': 20}})
True
```

Combine conditions with and and or operators to produce complex expressions:

```
>>> is_hipster = BoolRule('address.postcode.outcode in ("E1","E2") or user.has_beard␣
↪= true')
>>> address = {
>>>   'postcode': {
>>>      'outcode': 'E1'
>>>   }
```

```
>>> }
>>> is_hipster.test({'has_beard': False, 'address': address})
True
```

1.2 Credits

Made possible by the excellent pyparsing library.

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install boolrule, run this command in your terminal:

```
$ pip install boolrule
```

This is the preferred method to install boolrule, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for boolrule can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/tailsdotcom/boolrule
```

Or download the tarball:

```
$ curl  -OL https://github.com/tailsdotcom/boolrule/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

The entirety of boolrule's functionality is encapsulated in the `BoolRule` class.

3.1 Getting started

The simplest use case is evaluating simple, self-contained expressions:

```
from boolrule import BoolRule

expression = '5 > 10'
rule = BoolRule(expression)
rule.test()  # False
```

However, the real power of boolrule comes when the expression makes use of values from the context dict passed to the *test()* method:

```
from boolrule import BoolRule

expression = 'content.is_published = true and user.level in content.allowed_levels'
rule = BoolRule(expression)

context = {
    'user': {
        'level': 'super',
    },
    'content': {
        "is_published": True,
        'allowed_levels': [
            'admin',
            'super'
        ]
    },
}
```

```
if rule.test(context):
    # Let the user see the content
    pass
```

3.2 Lazy compilation

By default the expression is compiled when you create a new `BoolRule` object. If you're instantiating a lot of `BoolRule` instances but are only likely to call `test` on a few of them (because you're looking for just the first match, for example) then you can use the optional `lazy`` argument in the call to `BoolRule`` to defer compilation until the first call to `test()`:

```
rules = [
    BoolRule(expression, lazy=True)
    for expression in expressions
]

if any(r in rules.test(context)):
    # Do a thing
    pass
```

CHAPTER 4

boolrule API

4.1 BoolRule

class boolrule.**BoolRule**(*query*, *lazy=False*)

Represents a boolean expression and provides a *test* method to evaluate the expression and determine its truthiness.

Parameters

- **query** – A string containing the query to be evaluated
- **lazy** – If `True`, parse the query the first time it's tested rather than immediately. This can help with performance if you instantiate a lot of rules and only end up evaluating a small handful.

test(*context=None*)

Test the expression against the given context and return the result.

Parameters **context** – A dict context to evaluate the expression against.

Returns True if the expression succesfully evaluated against the context, or False otherwise.

4.2 Exceptions

class boolrule.**MissingVariableException**

Raised when an expression contains a property path that's not supplied in the context.

class boolrule.**UnknownOperatorException**

Raised when an expression uses an unknown operator.

This should never be thrown since the operator won't be correctly parsed as a token by pyparsing, but it's useful to have this hanging around for when additional operators are being added.

CHAPTER 5

Writing boolean expressions

The grammar supported by boolrule is fairly simple but powerful.

5.1 Whitespace

Except within string literals, all whitespace is ignored.

5.2 Literals

Numeric literls are written as bare numbers. Floating point and exponent-based numbers are supported:

```
10       # int
-10      # int with optional sign
10.5     # float (without optional sign)
10.5E-3  # equivalent to 0.0105
```

String literls can be single or double quoted:

```
"Hello, world"
'boolrule rulez"
```

Boolean literals are the bare values `true` and `false`

`None` type is the bare value `none`

5.3 Property paths

In order to reference values from the context passed into the `test()` method you specify the path to the property as a dot-separated identifier:

```
foo
foo.bar
foo.bar.baz
```

At evaluation time, these will map to either object attributes or dict keys in that order.

5.4 Basic comparison operators

Operator	Description	Example
=, ==, eq	Equality	`foo == 5`
!=, !==, ne	Inequality	`bar != 5`
>, gt	Greater than	`foo > 5`
>=, gte	Greater than or equal to	`foo >= 5`
<, lt	Less than	`foo < 5`
<=, lte	Less than or equal to	`foo <= 5`
is	Identity	`foo is True`
isnot	Inverse identity	`foobar isnot True`

5.5 Logical operators

Operator	Description	Example
and	Logical and	`foo == 5 and bar < 10`
oe	Logical or	`bar == 5 or bar < 10`

5.6 Membership operators

Operator	Description	Example
in	Is a member of	`foo in ("a","b","c")`
notin	Is not a member of	`foo not in ("a","b")`

5.7 Nested expressions

You can use parentheses to next expressions:

```
foo > 5 and (10 < bar or bar > 20)
```

CHAPTER 6

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/tailsdotcom/boolrule/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

6.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

6.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

6.1.4 Write Documentation

boolrule could always use more documentation, whether as part of the official boolrule docs, in docstrings, or even on the web in blog posts, articles, and such.

6.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/tailsdotcom/boolrule/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

6.2 Get Started!

Ready to contribute? Here's how to set up *boolrule* for local development.

1. Fork the *boolrule* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/boolrule.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv boolrule
$ cd boolrule/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 boolrule tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

6.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/tailsdotcom/boolrule/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ py.test tests.test_boolrule
```

CHAPTER 7

Credits

7.1 Development Lead

- Steve Webster <spjwebster@gmail.com>

7.2 Contributors

None yet. Why not be the first?

CHAPTER 8

History

8.1 0.2.0 (2016-10-27)

- Fixed error caused by refactor from internal codebase that was preventing deep context level values from being referenced in a substitution value

8.2 0.1.2 (2016-09-30)

- Improved documentation

8.3 0.1.1 (2016-09-30)

- Made `context` optional
- Improved documentation

8.4 0.1.0 (2016-09-30)

- First release on PyPI.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Index