pyhamcrest_metamatchers Documentation

Timofey Danshin

Dec 12, 2018

Contents

So that you can match your matchers with matchers, while you are writing matchers!

But seriously: Hamcrest and hamcrest-style matchers help in writing modular and reusable tests, but for that the matchers themselves must be reliable. Now you can develop your own custom matchers and be sure that they are. All you need to do is to test them using metamatchers.

CHAPTER 1

Code Documentation

Metamatcher is a matcher that checks that another matcher behaves correctly.

When new matchers are developed, it is vital to check that they match as expected and produce helpful desriptions and mismatch_descriptions.

This metamatcher does exactly that.

Say, you have written a matcher called `is_twice_as_big_as`, and you want it to compare ints. You intend to use it like this:

```
assert_that(4, is_twice_as_big_as(2))
```

Under the hood, the following is called:

```
is_twice_as_big_as(2)._matches(4)
```

Keeping that in mind, here's how you can check your matcher with the metamatcher:

```
def test_is_twice_as_big_as(...)
    assert_that(
        # Your initialized matcher
        is_twice_as_big_as(2),
        # The metamatcher specifying the value for matching
        matches(4)
    )
```

This will fail if your `is_twice_as_big_as` matcher doesn't match.

To check that your matcher produces the correct description:

```
def test_is_twice_as_big_as(...)
    assert_that(
        is_twice_as_big_as(2),
        matches(4).with_description("An int twice as big as <2>")
    )
```

This will fail if your `is_twice_as_big_as` matcher doesn't match, if the description it produces is wrong, or both.

You can also check that your matcher doesn't match in certain situations. To do that, use the `doesnt_match` function, and to check the mismatch description, call the `with_mismatch_description` method.

Note, that you can use the `with_description` method with the `doesnt_match` metamatcher, but calling `with_mismatch_description` with the `matches` flavour of the metamatcher, will throw an exception.

1.1 matches

`pyhamcrest_metamatchers.metamatchers.`**`matches`**(*a_matcher*)

Checks that the matcher under test matches the value

Parameters **`a_matcher`** – The matcher that needs to be checked.

Returns `pyhamcrest_metamatchers.metamatchers.MetaMatcher`

1.2 doesnt_match

`pyhamcrest_metamatchers.metamatchers.`**`doesnt_match`**(*a_matcher*)

Checks that the matcher under test doesn't match the value

Parameters **`a_matcher`** – The matcher that needs to be checked.

Returns `pyhamcrest_metamatchers.metamatchers.MetaMatcher`

1.3 with_description

`MetaMatcher.`**`with_description`**(*description*)

Adds the check for the description generated by the matcher that is being tested. If this method is not called, the matcher will not check the description at all.

If this method _is_ called, then the description, generated by the matcher under test, will be checked. If the actual description doesn't match the one set here, the metamatcher will not match.

1.4 with_mismatch_description

`MetaMatcher.`**`with_mismatch_description`**(*mismatch_description*)

Adds the check for the mismatch description generated by the matcher being tested. The logic is the same as with :*`pyhamcrest_metamatchers.metamatchers.MetaMatcher.with_description()`*

The goal of this project is to write a number of utility entities to facilitate the development of proper *hamcrest* style matchers.

For now we have a metamatcher, that is a matcher that can check that a matcher under development behaves properly, that is it matches whatever it is supposed to match, and it doesn't match whatever it isn't suppsed to, and that it produces the correct descriptions and mismatch descriptions. Here is a code example to that effect:

```python
def test_is_twice_as_big_as(...)
    assert_that(
        is_twice_as_big_as(2),
        matches(4).with_description("An int twice as big as <2>")
    )

def test_is_twice_as_big_as_not_matching(...)
    assert_that(
        is_twice_as_big_as(2),
        doesnt_match(7)\
            .with_description("An int twice as big as <2>")\
            .with_mismatch_description("was <7>")
    )
```

Index