poll Documentation

Release 1.0

Benjamin Hodgson

August 24, 2015

Contents

1 Polling **3**

2 Retrying **5**

3 Circuit Breaker **7**

4 Table of contents **9**
4.1 Reference 9

5 Indices and tables **13**

Python Module Index **15**

Utilities for polling, retrying, and exception handling, inspired by Polly.

CHAPTER 1

Polling

When you're waiting for a long-running process to become complete, it's often necessary to *poll* an external system to determine whether the operation is complete.

Here is a function which retries a Web request (using the `requests` library) every second until the resource exists, up to a maximum of 15 seconds:

```
from poll import poll
import requests

@poll(lambda response: response.status_code != 404, timeout=15, interval=1)
def wait_until_exists(uri):
    return requests.get(uri)
```

There's also a non-decorator form available, for when you want the *user* of a function to decide whether to poll the operation. The following code is equivalent to the function above:

```
from poll import poll_
import requests

def wait_until_exists(uri):
    poll_(
        lambda: requests.get(uri),
        lambda response: response.status_code != 404,
        timeout=15,
        interval=1
    )
```

CHAPTER 2

Retrying

When an operation may occasionally fail, it's often useful to *retry* the operation in the hope that it will succeed the next time.

Here's an approximately equivalent function to the above example, which catches the exception thrown by `raise_for_status` and retries until the response has a 2xx status code:

```
from poll import retry
import requests

@retry(requests.HTTPError, times=15, interval=1)
def wait_until_succeeds(uri):
    response = requests.get(uri)
    response.raise_for_status()
    return response
```

As with polling, you can use the 'underscored' version of `retry` to add retry logic to a function which doesn't already have it:

```
from poll import retry_
import requests

def get_or_raise(uri):
    response = requests.get(uri)
    response.raise_for_status()
    return response

def wait_until_succeeds(uri):
    retry_(
        lambda: get_or_raise(uri),
        requests.HTTPError,
        times=15,
        interval=1
    )
```

CHAPTER 3

Circuit Breaker

Simple retry logic often gets the job done, but it can cause problems. If your calls to an external service are failing because the external service is struggling under load, you don't want to exacerbate the problem by hammering it with retry attempts.

The *circuit breaker* pattern is a strategy for backing off, to avoid causing harm to external systems by retrying. If a call fails a certain number of times, the circuit breaker 'trips' and blocks any future calls.

After a time, the circuit enters the 'half-broken' state where it is ready to make one real call to test if the external service is functioning again. If this one real call fails, the circuit is broken again; otherwise, normal service is resumed.

Here's another version of our example, which blocks future attempts for sixty seconds after three calls to `attempt` fail:

```
from poll import circuitbreaker
import requests

@circuitbreaker(requests.HTTPError, threshold=3, reset_timeout=60)
def attempt(uri):
    response = requests.get(uri)
    response.raise_for_status()
    return response
```

For a more detailed explanation of Circuit Breaker, see Martin Fowler's article: http://martinfowler.com/bliki/CircuitBreaker.html

Table of contents

4.1 Reference

Utilities for polling, retrying, and exception handling.

exception `poll.`**`CircuitBrokenError`**(*message=''*, *time_remaining=0*)
Exception to indicate that the operation was not carried out because the circuit is broken.

`poll.`**`circuitbreaker`**(*ex*, *threshold*, *reset_timeout*, *on_error=<function <lambda>>*)
Decorator for functions which should 'back off' using the Circuit Breaker pattern: http://martinfowler.com/bliki/CircuitBreaker.html

This implementation of Circuit Breaker uses a 'leaky bucket' form of failure counting. For example, if *threshold* is 3 and *reset_timeout* is 60, then the circuit will be broken if the call fails three times *within a sixty-second period*. The circuit breaker is lenient towards intermittent failures.

Parameters

- **`ex`** (*class or iterable*) – The class of the exception to catch, or an iterable of classes.
- **`threshold`** (*int*) – The number of times a failure can occur before the circuit is broken.
- **`reset_timeout`** (*float*) – The length of time, in seconds, that a broken circuit should remain broken.
- **`on_error`** (*function*) – A function to be called when the decorated function throws an exception.

  If `on_error()` takes no parameters, it will be called without arguments.

  If `on_error(exception)` takes one parameter, it will be called with the exception that was raised.

  A typical use of `on_error` would be to log the exception.

Returns The final return value of the function `f`.

Raises CircuitBrokenError The operation was not carried out because the circuit is broken.

`poll.`**`exec_`**(*f*, *ex*, *until*, *times=3*, *timeout=15*, *interval=1*, *on_error=<function <lambda>>*, **args*, ***kwargs*)
General function for polling, retrying, and handling errors.

Parameters

- **`f`** (*function*) – The function to retry
- **`ex`** (*class or iterable*) – The class of the exception to catch, or an iterable of classes

- **until** (*function*) – The success condition. `until` should be a function; it will be called with the return value of the function. `until(x)` should return `True` if the operation was successful (and retrying should stop) and `False` if retrying should continue.
- **times** (*int*) – The maximum number of times to retry
- **interval** (*float*) – How long to sleep in between attempts in seconds
- **on_error** (*function*) – A function to be called when `f` throws an exception.

  If `on_error()` takes no parameters, it will be called without arguments.

  If `on_error(exception)` takes one parameter, it will be called with the exception that was raised.

  If `on_error(exception, retry_count)` takes two parameters, it will be called with the exception that was raised and the number of previous attempts (starting at 0).

  A typical use of `on_error` would be to log the exception.

Any other arguments are forwarded to `f`.

Returns The final return value of the function `f`.

Raises TimeoutError The call did not succeed within the specified timeout.

`poll.poll`(*until*, *timeout=15*, *interval=1*)

Decorator for functions that should be repeated until a condition or a timeout.

Parameters

- **until** (*function*) – The success condition. `until` should be a function; it will be called with the return value of the function. `until` should return `True` if the operation was successful (and retrying should stop) and `False` if retrying should continue.
- **timeout** (*float*) – How long to keep retrying the operation in seconds
- **interval** (*float*) – How long to sleep between attempts in seconds

Returns The final return value of the decorated function

Raises TimeoutError The condition did not become true within the specified timeout.

`poll.poll_`(*f*, *until*, *timeout=15*, *interval=1*, **args*, ***kwargs*)

Repeatedly call a function until a condition becomes true or a timeout expires.

Parameters

- **f** (*function*) – The function to poll
- **until** (*function*) – The success condition. `until` should be a function; it will be called with the return value of the function. `until` should return `True` if the operation was successful (and retrying should stop) and `False` if retrying should continue.
- **timeout** (*float*) – How long to keep retrying the operation in seconds
- **interval** (*float*) – How long to sleep in between attempts in seconds

Any other arguments are forwarded to `f`.

Returns The final return value of the function `f`.

Raises TimeoutError The condition did not become true within the specified timeout.

`poll.retry`(*ex*, *times=3*, *interval=1*, *on_error=<function <lambda>>*)

Decorator for functions that should be retried upon error.

Parameters

- **ex** (*class or iterable*) – The class of the exception to catch, or an iterable of classes
- **times** (*int*) – The maximum number of times to retry
- **interval** (*float*) – How long to sleep in between attempts in seconds
- **on_error** (*function*) – A function to be called when the decorated function throws an exception.

  If `on_error()` takes no parameters, it will be called without arguments.

  If `on_error(exception)` takes one parameter, it will be called with the exception that was raised.

  If `on_error(exception, retry_count)` takes two parameters, it will be called with the exception that was raised and the number of previous attempts (starting at 0).

  A typical use of `on_error` would be to log the exception.

Returns The return value of the decorated function

Raises TimeoutError The function did not succeed within the specified timeout.

`poll.`**`retry_`**(*f*, *ex*, *times=3*, *interval=1*, *on_error=<function <lambda>>*, **args*, ***kwargs*)

Call a function and try again if it throws a specified exception.

Parameters

- **f** (*funciton*) – The function to retry
- **ex** (*class or iterable*) – The class of the exception to catch, or an iterable of classes
- **times** (*int*) – The maximum number of times to retry
- **interval** (*float*) – How long to sleep in between attempts in seconds
- **on_error** (*function*) – A function to be called when `f` throws an exception.

  If `on_error()` takes no parameters, it will be called without arguments.

  If `on_error(exception)` takes one parameter, it will be called with the exception that was raised.

  If `on_error(exception, retry_count)` takes two parameters, it will be called with the exception that was raised and the number of previous attempts (starting at 0).

  A typical use of `on_error` would be to log the exception.

Any other arguments are forwarded to `f`.

Returns The final return value of the function `f`.

Raises TimeoutError The function did not succeed within the specified timeout.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

p

poll, 9

C

circuitbreaker() (in module poll), 9
CircuitBrokenError, 9

E

exec_() (in module poll), 9

P

poll (module), 9
poll() (in module poll), 10
poll_() (in module poll), 10

R

retry() (in module poll), 10
retry_() (in module poll), 11