Flask Snippets

Basic Message Queue with Redis

By Armin Ronacher filed in Utilities

For all your queuing needs there is Flask-Celery but if you just want a very basic queue functionality to get started you can build yourself something on top of redis very easily.

Connecting to Redis

```
from redis import Redis
redis = Redis()
```

The Configuration

For all this to work you need to define the redis key that should be used for queuing in your app config:

```
app.config['REDIS_QUEUE_KEY'] = 'my_queue'
```

The Decorator

```
from flask import current_app
from pickle import loads, dumps


class DelayedResult(object):
    def __init__(self, key):
        self.key = key
        self._rv = None

    @property
```

```
    def return_value(self):
        if self._rv is None:
            rv = redis.get(self.key)
            if rv is not None:
                self._rv = loads(rv)
        return self._rv


def queuefunc(f):
    def delay(*args, **kwargs):
        qkey = current_app.config['REDIS_QUEUE_KEY']
        key = '%s:result:%s' % (qkey, str(uuid4()))
        s = dumps((f, key, args, kwargs))
        redis.rpush(current_app.config['REDIS_QUEUE_KEY'], s)
        return DelayedResult(key)
    f.delay = delay
    return f
```

The Queue Runner

The queue runner is a simple function that runs in a while loop and processes entries from a list key in redis. Whenever something is added on there it will pop one item off the list, deserialize it, run the function and put the result into redis for a few seconds (by default 500). If the return value is `None` we don't store anything because in that case the assumption is that the return value is not interesting.

```
def queue_daemon(app, rv_ttl=500):
    while 1:
        msg = redis.blpop(app.config['REDIS_QUEUE_KEY'])
        func, key, args, kwargs = loads(msg[1])
        try:
            rv = func(*args, **kwargs)
        except Exception, e:
            rv = e
        if rv is not None:
            redis.set(key, dumps(rv))
            redis.expire(key, rv_ttl)
```

To run the daemon you can write a simple script like this:

```
#!/usr/bin/env python
from yourapp import app
from that_queue_module import queue_daemon
queue_daemon(app)
```

Running Functions through the Queue

To define a function to be run through the queue you need to use the `@queuefunc` decorator:

```
@queuefunc
def add(a, b):
    return a + b
```

When you call it normally it will be executed synchronously and in the same process. If you however call `add.delay(a, b)` it will send off the request to call this function to the queue and return you a `QueueResult` object. This also will need an active request context. Here an example from a python shell:

```
>>> from yourapp import app, add
>>>
>>> ctx = app.test_request_context()
>>> ctx.push()
>>> rv = add.delay(1, 2)
>>> rv.return_value
3
```

`rv.return_value` will be `None` until the key is available. Usually you will want to poll for a result over HTTP or not at all, so the `return_value` attribute is only really useful for testing.

If you want to poll this result the `rv` object has a `key` attribute which is the redis key corresponding to the result:

```
>>> rv.key
'my_queue:result:7d43370c-0f98-4e98-9d4b-1cdaf7362eb5'
```

Here is how you could poll for this via HTTP:

```
from flask import session, abort, jsonify

@app.route('/add')
def add_numbers():
    a = request.args.get('a', type=int)
    b = request.args.get('b', type=int)
    if a is None or b is None:
        abort(400)
    rv = add.delay(a, b)
    session['add_result_key'] = rv.key
    return 'Waiting for result...'

@app.route('/add-result')
def add_numbers_result():
```

```
    key = session.get('add_result_key')
    if key is None:
        return jsonify(ready=False)
    rv = DelayedResult(key)
    if rv.return_value is None:
        return jsonify(ready=False)
    redis.delete(key)
    del session['add_result_key']
    return jsonify(ready=True, result=rv.return_value)
```

First you let the user access `/add` to do something, then you can use JavaScript to poll `/add-result`. Note that once successfully polled the result is deleted from the redis server.