aallamaa / desir

Star 33 Fork 2

python redis client

75 commits 2 branches 0 releases 1 contributor

branch: master desir / +

code update to support passing a new url of json commands at initiali… ···

aallamaa authored on Apr 18 latest commit b6a05f512a

desir	code update to support passing a new url of json commands at initiali…	8 months ago
README.rst	code update	3 years ago
setup.py	fix incompabilities in protocol implementation with redis 2.6	3 years ago

Code
Issues 0
Pull Requests 0
Pulse
Graphs

HTTPS clone URL

https://github.com/a

You can clone with HTTPS or Subversion.

Clone in Desktop

Download ZIP

README.rst

Desir Redis Python Client

Attempt to make a minimalist redis python client. Redis commands are meta-generated based on the json commands description file "https://github.com/antirez/redis-doc/raw/master/commands.json" from redis repository. Documentation is also dynamically generated based on this file.

Plus some nice pythonic stuff that will soon come. (message passing, iterator, dict, set and list transparent implementation)

Desir is a permutation of Redis of course and related to the desire of antirez for 6379 (aka MERZ) http://antirez.com/print.php?postid=220

Install

sudo python setup.py install

Minimalist redis client

```
>>> import desir
>>> r=desir.Redis()
>>> r.keys("*")
['user', 'counter', 'user.adam', 'test', 'c', 'cnt', 'telecom-tid', 'count', 'new', 'bank-tid']
>>> r.get("test")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "desir/desir.py", line 83, in _runcmd
    return self.parent.runcmd(self.name,*args)
  File "desir/desir.py", line 158, in runcmd
    return self.Nodes[0].runcmd(cmdname,*args)
  File "desir/desir.py", line 273, in runcmd
    return self.parse_resp()
  File "desir/desir.py", line 261, in parse_resp
    raise RedisError(resp)
desir.desir.RedisError: ERR Operation against a key holding the wrong kind of value
```

```
>>> r.type("test")
'list'
>>> r.rpop("test")
'test'
>>> help(r.rpop)
Help on method rpop in module desir.desir:
rpop(self, *args) method of desir.desir.Redis instance
    Remove and get the last element in a list
    Parameters:
    Name: key,      Type: key,      Multiple parameter:False
(END)
>>> r.rpop("test")
>>> r.get("count")
'20000'
```

Pythonic sugar

1. Iterator

You can use object Counter from a Redis class instance as a unique counter across as many process, thread, server you could potentially have providing they can access the same redis instance/cluster.

```
>>> import desir
>>> r=desir.Redis()
>>> counter_name="counter" # the name of the counter is counter, meaning
>>> the key used inside redis is this name
>>> counter_seed=5 # intial value of the counter is 5
>>> c=r.Counter(counter_name,counter_seed)
>>> print "the initial value of the counter is", c
>>> for i in c:
...  print i
...  if i>10:
...   break
>>> print "the next value of the counter is", c.next()
 the initial value of the counter is 5
 6
 7
 8
 9
 10
 11
 the next value of the counter is 12
>>>
```

2. Connector

A connector is an attempt to make a message passing interface similar to the Erlang send / receive message passing functions.

A connector is defined by its name which is pointing internaly to a redis list using the connector name as the key name inside redis.

Here is how it works:

On client "toto" you do this:

```
>>> import desir
>>> n=desir.Redis()
>>> c=n.Connector("toto",timeout=5)
```

On client "tata" you do the same:

```
>>> import desir
>>> n=desir.Redis()
>>> d=n.Connector("tata",timeout=5)
```

Note that the timeout defined when instanciating the connector is used only when using the connector as an iterator. A value of 0 means timeout is never reached.

Then let´s define on client "toto" an object to send to client "tata". Note that you can send any serializable object (using pickle).

```
>>> v=[1,2,3,4,dict(a=2,b=3)]
>>> c.send("tata",v)
>>> c.send("tata",v)
>>> c.send("tata",v)
```

Now let´s go back to client "tata" and see the result:

```
>>> for v in d:
...  print v
...
['toto', 1288551730.8449249, [1, 2, 3, 4, {'a': 2, 'b': 3}]]
['toto', 1288551730.8463399, [1, 2, 3, 4, {'a': 2, 'b': 3}]]
['toto', 1288551730.8468609, [1, 2, 3, 4, {'a': 2, 'b': 3}]]
>>>
```

After 5 seconds, the for loop automatically stop trying to fetch receive from the connector as timeout was defined as 5 (seconds).

You could also use d.receive() to get a result (blocking one) or d.receive(timeout) for non blocking if timeout is not 0.

3. pub/sub

To play with pub/sub:

```
>>> import desir
>>> r=desir.Redis()
>>> r.subscribe("foo")
['subscribe', 'foo', 1]
>>>
>>> for v in r.listen():
...  print(v)
...
['message', 'foo', 'tata']
['message', 'foo', 'toto']
```

Javascript like call back function to handle messages received on a subscribed channel Python 3.2.2 (default, Feb 24 2012, 18:42:26) [GCC 4.2.1 (Based on Apple Inc. build 5658) (LLVM build 2335.15.00)] on darwin Type "help", "copyright", "credits" or "license" for more information. >>> import desir >>> def foo(p): ... print "I have received %s" % (str(p)) ... >>> a=desir.SubAsync("foo",foo)

on another redis socket i do this: publish foo toto :2

and here what i get on the console: >>> I have received ['message', 'foo', 'toto']