ROH

Table of Contents

Overview

What is a ROH?

ROH is a distributed Python tasks manager.

It is possible to distribute python tasks across the network to different nodes

When to Use ROH?

To distribute loads

Let’s see some python code

The following snippet shows some internals.

the AMQP module

```
channel_.exchange_declare(exchange='test_exchange_1', durable=True,
                          exchange_type="fanout") ①
result = channel_.queue_declare(queue='my_queue', durable=True)
queue_name = result.method.queue
channel_.queue_bind(exchange="test_exchange_1", queue=queue_name,
                    routing_key="") ②
channel_.basic_qos(prefetch_count=1)
channel_.basic_consume(on_message, queue=queue_name,
                       no_ack=False) ③
channel_.start_consuming()
```

① We define an exchange

② We bind a queue

③ We send the ACK

Name	Description

`exchange`	The exchange to bind to
`queue`	The queue to bind to
`routing_key`	The routing key to bind with
`arguments`	Other properties (construction arguments) for the binding
`nowait`	Do not wait for the response
`callback`	A callback method taking one argument, the bound queue
`ticket`	The ticket number

Let's see some erlang code

The following snippet shows some internals.

erlang module

```
-spec(handle_call(Request :: term(), From :: {pid(), Tag :: term()}, % ①
    State :: #state{}) ->
    {reply, Reply :: term(), NewState :: #state{}} |
    {reply, Reply :: term(), NewState :: #state{}, timeout() | hibernate} |
    {noreply, NewState :: #state{}} |
    {noreply, NewState :: #state{}, timeout() | hibernate} |
    {stop, Reason :: term(), Reply :: term(), NewState :: #state{}} |
    {stop, Reason :: term(), NewState :: #state{}}).
handle_call({add_task, Task}, _From,
    State = #state{running_workers = MRW, waiting_queue = QWQ, supervisor = Sup}) ->
    case is_watermark_processes(MRW) of % ②
        true ->
            QWQ2 = queue:in(Task, QWQ),
            roh_console_log:info("Added in waiting list, current size: ~w",
[queue:len(QWQ2)]),
            {reply, ok, State#state{waiting_queue = QWQ2, global = State#state.global
+ 1}};
        false -> MRW2 = execute_new_worker(Task, Sup, MRW),
            {reply, ok, State#state{running_workers = MRW2, global =
State#state.global + 1}} % ③
    end;
```

① Define the type for the analyzer

② check the watermark

③ execute the task