publicaRabbitMQ Documentation

Versión 0.1

Leonardo Salvucci

29 de marzo de 2017

Índice general

Este componente tiene incorporado el uso de un cache cuando el RabbitMQ no se encuentra disponible para publicar un mensaje.

CAPÍTULO 1

Diagrama de clases

definición de clases.pdf

CAPÍTULO 2

Esquema temporal

CAPÍTULO 3

Requerimientos previos

- Redis
- RabbitMQ

CAPÍTULO 4

Instalación del componente

```
pip install git+https://github.com/LeonardoSalvucci/publicaRabbit
```

CAPÍTULO 5

Como importar el componente

```
from publicaRabbit.publicaRabbit import publicaRabbit
```

CAPÍTULO 6

Métodos

- publicaRabbit(rabbitHost='127.0.0.1',rabbitPort=5672,rabbitVHost='',rabbitUser='guest',rabbitPass='guest',configPath=None,i
- getListBaseKey()
- setListBaseKey(listBaseKey)
- iniciarSchedule()
- frenarSchedule()
- publicarMensaje(exchange,mensaje)
- _checkReportesEnCache()

CAPÍTULO 7

Configuración de Redis por defecto

El componente trae la configuración por defecto

```
{
   "redis":
       {
           "host": "127.0.0.1",
           "port": 6379
       },
   "baseListKey":"publicaRabbit:mensajes",
}
```

Para modificar la configuración por defecto se puede realizar pasando el path de un archivo de configuración en formato json al constructor, completo o parcial, por ej:

```
{
   "baseListKey":"miApp:cache",
}
```

otro ejemplo:

```
{
   "redis":
   {
       "host": "192.168.0.254",
       "port": 1000
   }
}
```

O también es posible utilizar el método `setListBaseKey('miApp:cache')` incluso dinámicamente.

Nota: Si se van a utilizar varias instancias del componente en un mismo host verificar que el baseListKey sea diferente en cada una para no generar conflictos

CAPÍTULO 8

Schedule verificador de elementos en cache

Al crear una nueva instancia del componente publicaRabbit se inicia automáticamente una tarea periódica, por defecto cada 30 segundos, que verifica si existen elementos pendientes de publicar e intenta nuevamente.

Nota: Se puede modificar el intervalo de chequeo mediante el contructor publicaRabbit

Al momento de finalizar el uso del componente es obligatorio utilizar el método ‘‘frenarSchedule()‘‘ para que no quede un hilo ejecutado en segundo plano

CAPÍTULO 9

Tests realizados

- testScheduleTask

Este test instancia la clase ScheduleTask con una funcion que imprime 'Hola Mundo' y se ejecuta cada 1 segundo hasta finalizar con Ctrl^C

- testRedisCache (unitTest)
- test_GetListName
- test_Publish_String
- test_Publish_Dict
- test_Publish_List
- test_Cache_Count
- test_Get_Multiple_Caches
- test_Publish_With_UTF8
- testRawPublicaMensaje (unitTest)
- test_publish_string
- test_publish_json_as_string
- test_publish_json
- test_publish_list_as_json
- test_publish_queue_with_ttl_parameter
- test_exchange_not_exists
- testPublicaRabbit

Para el testeo del componente es necesario generar la "caida" del servidor RabbitMQ por lo que se diseñó un test interactivo que informa y solicita ejecutar tareas para verificar que funcione correctamente en su totalidad.

```
Publica un mensaje inicial en formato String
Publica un mensaje inicial en formato Dict
Publica un mensaje inicial en formato List
Espero 10 segundos para frenar el rabbitmq-server
10
0
Publico otro mensaje sin rabbitmq
Espero 10 segundos para verificar en redis si se cacheó
10
0
Espero 35 segundos para que se ejecute al menos una vez el Scheduler sin rabbit
35
20
10
5
0
Espero 10 segundos para levantar el rabbitmq-server
10
0
Espero 35 segundos para que se ejecute al menos una vez el Scheduler sin rabbit
35
20
10
5
0
Creo un consumidor para recuperar todos los mensajes publicados y verificar si␣
↪coinciden
No hay mas mensajes
Verifico que los mensajes hayan sido correctos (cuenta de mensajesPublicados = 0)
mensajesPublicados.__len__()=0
Test Finalizado
```