src

queue_logic package

Submodules

queue_logic.client module

class **queue_logic.client.KeyDBQueue**(*keydb_url*, *service*)

Базовые классы: `KeyDBOperations` , `KeyDBFields`

Класс клиент для работы с KeyDB (Redis).

Содержит методы управления очередями ответов и задач.

async **check_queue**()

Проверка на существования задачи в очередях задач

Проверка в очередях вида {service}_queue и {service}_reestr.

1. Если очередь задач не пуста, то берем задачу из этой очереди, иначе -> return None
2. Проверяем очередь ответов на наличие уже обработанного кеша задачи.
3. Если задача не была обработана, возвращаем эту задачу как ответ функции
4. Если задача уже была обработана, то обновляем ей TTL и return None

За счет TTL очередь ответов не копится и обновляется примерно раз в сутки (зависит от параметров).

Результат:
None | str - задача или ничего

async **check_exists**(*payload*)

Проверка на существование обработанной задачи в очереди ответа :type payload: `str` :param payload: задача :rtype: `str` :return: ответ hget

async **check_exists_with_update_ttl**(*payload*)

Функция проверки на существование обработанной задачи в очереди ответа с обновлением TTL

Параметры:

payload (`str`) – задача

Тип результата:

`bool`

Результат:

boolean - существует ли задача

async **set_answer**(*payload*, *answer*)

Сохранение в очередь ответов ответа на задачу. Ключ - задача, значение - ответ

Параметры:

- **payload** (`str`) – задача
- **answer** (`dict`) – ответ на задачу, либо json, либо контейнер (dict, list)

async **return_to_queue**(*payload*)

Вернуть задачу из очереди задач обратно. Добавление в начало очереди

Используется в случае, если задача не успешно обработалась

Параметры:

payload (`str`) – задача

Результат:

ответ lpush

async **add_task**(*payload*)

Добавить задачу в очередь задач. Добавление в конец очереди

Используется для генерации тестовых заданий. Схожая логика с `return_to_queue` :type payload: `str` :param payload: задача :return: ответ rpush

queue_logic.connect module

class **queue_logic.connect.KeyDBConnectInterface**(*keydb_url*, *service*)

Базовые классы: `object`

async **connect**()

async **close**()

queue_logic.keydb_fields module

class **queue_logic.keydb_fields.KeyDBFields**

Базовые классы: `object`

Вспомогательный класс для хранения значений по умолчанию класса клиента KeyDB

property **task_ttl_ok**: *int*

Геттер на получение времени хранения успешного ответа в очереди KeyDB

Результат:
время в секундах

property **task_ttl_error**: *int*

Геттер на получение времени хранения неуспешного ответа в очереди KeyDB

Результат:
время в секундах

property **queue_main**: *str*

Получение имени очереди в KeyDB, из которой будут браться задачи.

Например, `callapp_queue` - очередь задач

Результат:
название очереди

property **queue_registry**: *str*

Получение имени очереди в KeyDB, из которой будут браться задачи обработки реестров.

Например, `callapp_reestr` - очередь задач реестра

Результат:
название очереди

property **queue_response**: *str*

Получение имени очереди в KeyDB, в которую будут класться ответы на задачи из предыдуших выше очередей

Например, `callapp` - очередь ответов

Результат:
название очереди

queue_logic.logger module

queue_logic.operations module

queue_logic.operations.timeout(*func*)

Декоратор для задания таймаута на операции, перехватывания ошибок и форматирования их в isphere-exceptions

class **queue_logic.operations.KeyDBOperations**(*keydb_url*, *service*)

Базовые классы: `KeyDBConnectInterface`

Класс-обертка над основными операциями с redis.

Используется для задания декораторов и прямого подключения к redis

async **hget**(***kwargs*)

async **hset**(***kwargs*)

async **lpush**(***kwargs*)

async **rpush**(***kwargs*)

async **lpop**(***kwargs*)

async **execute_command**(***kwargs*)

Module contents

setup module

tests package

Module contents